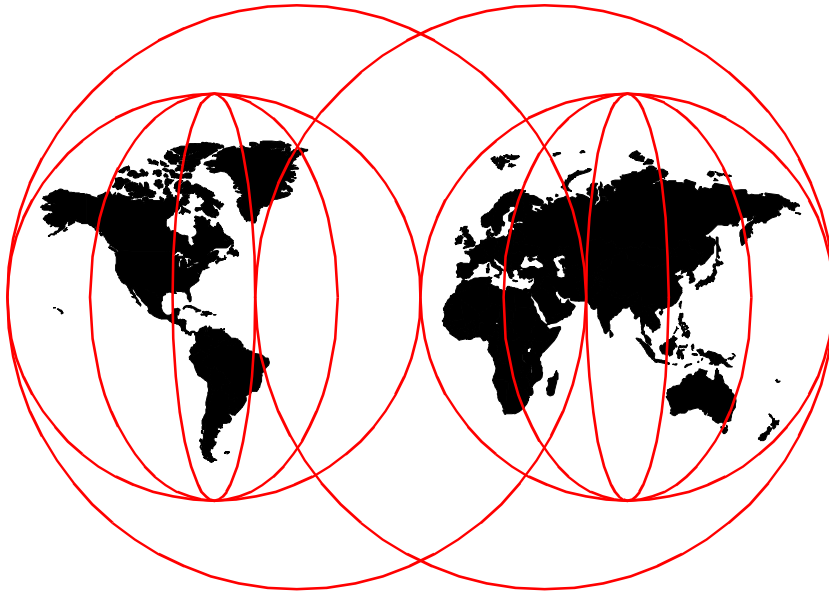


A DB2 Enterprise Query Environment — Build It with QMF for Windows !

Joerg Reinschmidt, Catalin Comsia, Andre Roberto Santos



International Technical Support Organization

www.redbooks.ibm.com

SG24-5746-00



International Technical Support Organization

**A DB2 Enterprise Query Environment —
Build It with QMF for Windows !**

December 1999

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix E, "Special notices" on page 393.

First Edition (December 1999)

This edition applies to Version 6, Release 1 Refresh of QMF for Windows, Program Number 5655-DB2 for use with the Windows NT Operating System.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. QXXE Building 80-E2
650 Harry Road
San Jose, California 95120-6099

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

© Copyright International Business Machines Corporation 1999. All rights reserved.

Note to U.S Government Users – Documentation related to restricted rights – Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Figuresxi
Tables	xv
Prefacexix
The team that wrote this redbook	xx
Comments welcome	xxii
Chapter 1. Introduction to query management	1
1.1 Basics about queries and data	1
1.2 Enterprise query and reporting environment	3
1.3 A short history of the QMF family	6
Chapter 2. Wouldn't it be nice?	9
2.1 Typical IS Needs	10
2.2 The Enterprise Query Environment	11
2.2.1 IT issues	11
2.2.2 User issues	13
2.3 QMF for Windows: the solution	13
2.3.1 Product Positioning	14
2.4 Enterprise Benefits	17
2.5 Implementation examples	20
2.5.1 Financial industry	20
2.5.2 Public services institution	21
Chapter 3. Getting started	25
3.1 The networking environment	25
3.2 TCP/IP basics	26
3.2.1 TCP/IP architecture	28
3.2.2 IP addressing	29
3.2.3 Subnets	30
3.2.4 IP datagram	32
3.2.5 IP routing	33
3.3 Configuring your TCP/IP	36
3.3.1 OS/390 OpenEdition	37
3.3.2 AS/400	42
3.3.3 AIX	44
3.3.4 Windows	46
3.4 SNA basics	48
3.4.1 SNA layers	49
3.4.2 APPC basics and terminology	50

3.5	Configuring your SNA (LU 6.2, APPC, and CPI-C)	55
3.5.1	Windows NT	56
3.6	Data exchange protocols	62
3.6.1	DRDA remote unit of work (RUW)	62
3.6.2	DRDA distributed unit of work (DUW)	63
3.6.3	Distributed request (DR)	63
3.6.4	Private protocols	63
3.6.5	Nonrelational access	64
3.7	Connecting via Call Level Interface (CLI)	64
3.8	Installing QMF for Windows	65
3.8.1	Advanced installation	65
3.8.2	Unattended installation	67
Chapter 4. DBA's guide		69
4.1	Working with QMF for Windows Administrator	69
4.1.1	Configure database connections	72
4.1.2	Test the server connection	78
4.1.3	Create QMF for Windows objects	81
4.1.4	Bind QMF for Windows packages	88
4.1.5	Granting Permissions	89
4.1.6	Creating sample tables	90
4.1.7	Delete a database server	92
4.2	Governing and administration	92
4.2.1	Creating resource limits groups	92
4.2.2	Creating schedules	94
4.2.3	Assigning users to the resource group	110
4.3	Security	111
4.3.1	Change password capability	112
4.3.2	Lists	112
4.4	Other DBA Tasks	115
4.4.1	Convert dynamic SQL to static SQL	115
4.4.2	DB2 UDB for OS/390 predictive governor support	118
4.4.3	Large Object (LOB)	119
4.4.4	QMF linear procedures	120
4.4.5	Command line mode	120
4.4.6	Scheduling with Windows NT	122
Chapter 5. Developer's guide		125
5.1	Application development concepts using QMF for Windows	125
5.1.1	Application Program Interface (API)	126
5.1.2	QMF for Windows APIs or ODBC Applications?	126
5.1.3	Synchronization Aspects	128
5.1.4	Database Connectivity	129

5.1.5	Web Development	130
5.2	Main QMF for Windows APIs	131
5.2.1	GetServerList()	131
5.2.2	InitializeServer()	131
5.2.3	GetQMFObjectList()	132
5.2.4	InitializeQuery()	132
5.2.5	GetQueryText()	132
5.2.6	GetQueryVerb()	133
5.2.7	SaveQMFQuery()	133
5.2.8	Open()	133
5.2.9	GetColumnCount()	134
5.2.10	GetColumnHeadings()	134
5.2.11	FetchNextRow()	134
5.2.12	Close()	134
5.3	Using Visual Basic with QMF for Windows	135
5.3.1	Getting Started	135
5.3.2	Application examples	136
5.3.3	Example 1 — Execute a query stored on the server	138
5.3.4	Example 2 — Execute a query stored in a file	142
5.3.5	Example 3 — Execute an SQL statement	147
5.4	Using Delphi with QMF for Windows	154
5.4.1	Getting Started	154
5.4.2	Delphi application example	156
5.5	Using C++ with QMF for Windows	172
5.5.1	Getting started	173
5.5.2	C++ specifics	173
5.6	General programming considerations	173
5.6.1	Two phase commit	173
5.6.2	Editing prompted queries	175
5.6.3	Other QMF APIs	175
Chapter 6.	User's guide	177
6.1	Product Installation and configuration	177
6.2	Basic concepts	181
6.3	Accessing existing objects	183
6.3.1	Objects stored at a server	183
6.3.2	Objects stored in a file	186
6.4	Working with objects	187
6.4.1	Tables	188
6.4.2	Queries	188
6.4.3	Forms and Reports	193
6.4.4	Procedures	196
6.5	Create new objects	197

6.5.1	Create new tables	198
6.5.2	Create new queries	199
6.5.3	Create new form and report	211
6.5.4	Create new procedures	227
6.6	Using Data Snap-Ins for QMF for Windows	231
6.6.1	Lotus 123	232
6.6.2	Microsoft Excel	235
6.6.3	Microsoft Access	238
6.7	Converting dynamic SQL to static SQL	242
6.7.1	Dynamic versus static SQL	243
6.8	Checking your resource limits	243
6.9	Security	246
6.9.1	Change password capability	246
6.9.2	Lists	247
6.10	Customizing the interface	249
6.11	Migrating from OS/2 Query Manager	251
Chapter 7. Web considerations		253
7.1	Web presence basics	254
7.1.1	How does it work?	254
7.2	Static reports	255
7.2.1	Convert a standard QMF Form to HTML	256
7.2.2	Report preview feature	259
7.2.3	Scheduling	260
7.3	Dynamic reports	262
7.3.1	CGI	263
Chapter 8. Summary		271
8.1	Future directions	272
8.2	QMF Personal Portal	273
Appendix A. Working with variables		277
A.1	Substitution variables	279
A.2	Global variables	281
A.2.1	User defined global variables	281
A.2.2	Pre-loaded global variables	283
A.3	Form variables	290
Appendix B. QMF for Windows APIs		293
B.1	AddDecimalHostVariable	293
B.2	AddHostVariable()	294
B.3	BindDecimalHostVariable()	295
B.4	BindHostVariable()	296
B.5	BindSection()	297

B.6	CancelBind()	298
B.7	ChangePassword()	298
B.8	ClearList()	299
B.9	Close()	299
B.10	Commit()	300
B.11	CompleteQuery()	301
B.12	CopyToClipboard()	301
B.13	DeleteQMFObject()	303
B.14	EndBind()	303
B.15	Execute()	304
B.16	ExecuteEx()	304
B.17	ExecuteStoredProcedure()	305
B.18	ExecuteStoredProcedureEx()	307
B.18.1	Export()	308
B.19	ExportForm()	311
B.20	ExportReport()	311
B.21	FastSaveData()	313
B.22	FetchNextRow()	314
B.23	FetchNextRowEx()	316
B.24	FetchNextRows()	316
B.25	FetchNextRowsEx()	318
B.26	FlushQMFCache()	318
B.27	GetColumnCount()	319
B.28	GetColumnDataValue()	319
B.29	GetColumnHeader()	320
B.30	GetColumnHeaderEx()	320
B.31	GetColumnHeadings()	321
B.32	GetColumnValue()	322
B.33	GetColumnValueEx()	323
B.34	GetDefaultServerName()	324
B.35	GetGlobalVariable()	324
B.36	GetHostVariableNames()	325
B.37	GetHostVariableTypeNames()	325
B.38	GetHostVariableTypes()	326
B.39	GetLastErrorString()	326
B.40	GetLastErrorType()	327
B.41	GetLastSQLCode()	328
B.42	GetLastSQLError()	329
B.43	GetLastSQLState()	330
B.44	GetOption()	331
B.45	GetOptionEx()	332
B.46	GetProcText()	333
B.47	GetProcVariables()	333

B.48	GetQMFObjectInfo()	334
B.49	GetQMFObjectInfoEx()	336
B.50	GetQMFObjectList()	338
B.51	GetQMFObjectListEx()	339
B.52	GetQMFPProcText()	340
B.53	GetQMFPQueryText()	341
B.54	GetQueryText()	341
B.55	GetQueryVerb()	342
B.56	GetResourceLimit()	343
B.57	GetResourceLimitEx()	348
B.58	GetRowCount()	348
B.59	GetServerList()	349
B.60	GetServerListEx()	350
B.61	GetStoredProcedureResultSets()	351
B.62	GetVariables()	352
B.63	GetVariablesEx()	353
B.64	InitializeProc()	353
B.65	InitializeQuery()	354
B.66	InitializeServer()	355
B.67	InitializeStaticQuery()	356
B.68	IsStatic()	357
B.69	Open()	358
B.70	Prepare()	359
B.71	PrintReport()	359
B.72	ReinitializeServer()	360
B.73	Rollback()	360
B.74	RunProc()	361
B.75	SaveData()	361
B.76	SaveQMFPProc()	363
B.77	SaveQMFPQuery()	364
B.78	SetBindOption()	365
B.79	SetBindOwner()	367
B.80	SetBusyWindowButton()	368
B.81	SetBusyWindowMessage()	369
B.82	SetBusyWindowMode()	369
B.83	SetBusyWindowTitle()	370
B.84	SetGlobalVariable()	371
B.85	SetHostVariable()	371
B.86	SetOption()	372
B.87	SetParent()	373
B.88	SetProcVariable()	374
B.89	SetVariable()	375
B.90	ShowBusyWindow()	375

B.91 StartBind()	376
Appendix C. QMF for Windows tables and views	379
C.1 Tables	379
C.1.1 Q.OBJ_ACTIVITY_DTL	379
C.1.2 Q.OBJ_ACTIVITY_SUM	380
C.1.3 Q.OBJECT_DATA	381
C.1.4 Q.OBJECT_DIRECTORY	381
C.1.5 Q.OBJECT_REMARKS	382
C.1.6 Q.RAA_SUBTYPE	382
C.1.7 RDBI.AUTHID_TABLE	382
C.1.8 RDBI.PROFILE_TABLE	383
C.1.9 RDBI.RESERVED	383
C.1.10 RDBI.RESOURCE_TABLE	384
C.2 Views	384
C.2.1 Q.RAA_OBJECT_VIEW	384
C.2.2 RDBI.ADMIN_VIEW	385
C.2.3 RDBI.AUTHID_VIEW	385
C.2.4 RDBI.PROFILE_VIEW	386
C.2.5 RDBI.RESOURCE_VIEW	386
C.2.6 RDBI.TABLE_VIEW	387
C.2.7 RDBI.USER_ADMIN_VIEW	388
C.2.8 RDBI.USER_AUTHID_VIEW	388
C.3 Table and view relationships	388
Appendix D. Using the additional material	391
D.1 Using the CD-ROM or diskette	391
D.1.1 System requirements for using the CD-ROM or diskette	391
D.1.2 How to use the CD-ROM or diskette	391
D.2 Locating the additional material on the Internet	391
Appendix E. Special notices	393
Appendix F. Related publications	397
F.1 International Technical Support Organization publications	397
F.2 Redbooks on CD-ROMs	397
How to get ITSO redbooks	399
IBM Redbook fax order form	400

Glossary	401
List of abbreviations	403
Index	405
ITSO redbook evaluation	413

Figures

1. Evolution from queries to knowledge discovery	2
2. A business intelligence environment	3
3. A query and reporting environment	4
4. QMF and DB2 development history	6
5. Decision making process	9
6. The IT infrastructure	12
7. The Enterprise Query Environment	14
8. Decision support evolution	15
9. QMF family of products	17
10. TCP/IP architecture model: layers and protocols	28
11. Class A address without subnets	30
12. Class A address with subnet mask and subnet address	31
13. Format of an IP datagram header	33
14. Direct and indirect routing	34
15. Routing table scenario	35
16. IP routing table entries example	35
17. IP routing algorithm (with subnets)	36
18. AIX TCP/IP SMIT panel	44
19. Defined network interface on AIX	45
20. AIX minimum configuration	45
21. AIX hosts file	46
22. Windows NT TCP/IP properties	47
23. Personal communication - configuration window	57
24. Configure node	58
25. Configure connection	59
26. Adjacent node definition	60
27. Partner LU 6.2	61
28. CPI-C side information	62
29. Server definition file	70
30. TCP/IP Connection definition	74
31. Provider DLL	75
32. CPI-C connection definition	76
33. DB2 datajoiner using cli: server parameters	77
34. Test the server connection	78
35. Communication error	79
36. Tracing the server connection	80
37. Create objects	83
38. Granting permissions	90
39. Create the sample QMF tables	91
40. Create resource limit group	94

41. Main: parameters	95
42. Timeouts: parameters	97
43. Limits: parameters	99
44. SQL Verbs: parameters	101
45. Options: parameters	102
46. Save data: parameters.	105
47. Binding tab parameters	106
48. Object tracking: parameters.	108
49. Assign users to resource group	111
50. Change password	112
51. Lists	114
52. Create static SQL.	116
53. Input variables	117
54. Bind complete	118
55. Procedure to be scheduled	121
56. Windows NT scheduler	124
57. Application architecture when using ODBC.	127
58. QMF database access.	129
59. Visual basic configuration	135
60. Visual basic — execute query stored on the server.	142
61. Visual basic — execute query stored on a file.	147
62. Visual basic — execute SQL statement.	154
63. Delphi configuration	155
64. Delphi example, query list	157
65. Delphi example, new query	158
66. Delphi example, executing query.	159
67. Installation options	177
68. Custom installation.	178
69. Connection window	179
70. Setting up the server definition file.	180
71. Database management systems	181
72. Tables in a database	182
73. Object links	183
74. Open object from server	184
75. List of objects	185
76. Open object from file	187
77. Query result	189
78. Find	191
79. Export data.	193
80. Convert form to HTML	194
81. HTML form	195
82. HTML form result	196
83. Set server window	198

84. Save data into new table	199
85. New SQL query menu	200
86. New query window	201
87. Save new SQL query at server	202
88. Substitution variable.	203
89. New prompted query window.	204
90. Select tables to a new prompted query,.	205
91. Adding join condition to a new prompted query	206
92. Adding columns to a new prompted query.	207
93. Add sort condition to a new prompted query	208
94. Add row condition to a new prompted query	209
95. Displaying result of the new prompted query.	210
96. Saving new prompted query	211
97. New blank form	213
98. New blank form main window	214
99. Adding columns to a new blank form	215
100.Saving new blank form	219
101.Ways of creating new form	220
102.Using the sum function	224
103.Result of group and sum functions	225
104.Modify column order in a form	226
105.Final result of the modified form	227
106.New procedure	228
107.Saving new procedure.	231
108.Lotus 123 Snap-In	233
109.Formatting data using Lotus 123 Snap-In	234
110.Result of Lotus 123 Snap-In	235
111.Excel Snap-In	236
112.Formatting data using Excel Snap-In	237
113.Result of Excel Snap-In.	238
114.Access Snap-In	239
115.Selecting a table to place the data	240
116.Select and report list	241
117.Result of access Snap-In	242
118.Retrieve the most current resource limits	245
119.Resource limits	246
120.Change password	247
121.Lists	248
122.Customize toolbar	249
123.Move the toolbar	250
124.Web environment	255
125.Convert form to HTML.	256
126.Form main	257

127.	Form main window	258
128.	Form detail	259
129.	HTML form preview	260
130.	Publishing procedure	261
131.	Dynamic reports	262
132.	CGI example — first screen	264
133.	CGI example — second screen	265
134.	An Enterprise Query Environment	272
135.	Rocket personal portal	274
136.	Properties screen	275
137.	Copy to favorites	276
138.	Windows registry	278
139.	Variable structure	279
140.	Substitution variable	280
141.	Global variables	281
142.	Adding global variables	282
143.	New global variable created	283
144.	Object activity detail table	379
145.	Object activity summarization table	380
146.	Object data table	381
147.	Object directory table	381
148.	Object remarks table	382
149.	RAA subtype table	382
150.	AuthID table	382
151.	Profile table	383
152.	Reserved table	383
153.	Resource table	384
154.	RAA object view	385
155.	Admin view	385
156.	AuthID View	385
157.	Profile view	386
158.	Resource view	387
159.	Table view	387
160.	User admin view	388
161.	User AuthID view	388
162.	Q.RAA_OBJECT_VIEW relations	389
163.	RDBI.TABLE_VIEW relations	390

Tables

1. SNA configuration parameters	56
2. Installation parameters	67
3. Required privileges	89
4. Sample QMF tables	90
5. Columns on query PROMPTED_QUERY	212
6. Edit Codes	216
7. Usage codes	221
8. Procedure commands	229
9. Global variables naming convention	284
10. DSQAO global variables	284
11. DSQCP global variables	285
12. DSQCP global variables	286
13. DSQEC global variables	286
14. DSQQW global variables	287
15. Form variables	291
16. AddDecimalHostVariable parameters	293
17. AddHostVariable parameters	294
18. Valid values for the parameter type	294
19. BindDecimalHostVariable parameters	295
20. BindHostVariable parameters	296
21. Valid values for the parameter DataType	296
22. BindSection Parameters	297
23. CancelBind Parameters	298
24. ChangePassword parameters	299
25. ClearList parameters	299
26. Close parameters	300
27. Completequery parameters	301
28. CopyToClipboard parameters	302
29. DeleteQMFOBJECT parameters	303
30. EndBind parameters	303
31. Execute parametes	304
32. ExecuteEx parameters	305
33. ExecuteStoredProcedure parameters	306
34. ExecuteStoredProcedureEx parameters	307
35. Export parameters	309
36. Valid values for the parameter format	310
37. Valid values for the parameter StringDelimiter	310
38. Valid values for the parameter ColumnDelimiter	310
39. ExportForm parameters	311
40. ExportReport parameters	312

41. Valid values for the parameter SourceType.	313
42. FastSaveData parameters.	314
43. FetchNextRow parameters	315
44. FetchNextRowEx parameters	316
45. FetchNextRows parameters	316
46. FetchNextRowsEx parameters	318
47. GetColumnCount parameters	319
48. GetColumnDataValue parameters.	319
49. GetColumnHeader parameters	320
50. GetColumnHeaderEx parameters	321
51. GetColumnHeadings parameters	321
52. GetColumnValue parameters	323
53. GetColumnValueEx parameters	323
54. GetGlobalVariable parameters	324
55. GetHostVariableNames parameters	325
56. GetHostVariableTypeNames parameters	325
57. GetHostVariableTypes parameters	326
58. GetLastErrorType - error types	327
59. Format of the array returned by the GetLastSQLError API	329
60. GetOption parameters	331
61. Valid values for the parameter option.	331
62. GetOptionEx parameters	332
63. GetProcText parameters	333
64. GetProcVariables parameters	334
65. GetQMFObjectInfo parameters	335
66. Valid values for the parameter type	335
67. Valid values for the parameter Time	336
68. GetQMFObjectInfoEx parameters	337
69. Valid values for the parameter Type	337
70. Valid values for the parameter Time	338
71. GetQMFOBJECTList parameters	338
72. Valid values for the parameter Type	339
73. GetQMFOBJECTListEx parameters	340
74. Valid values for the parameter Type	340
75. GetQMFProcText parameters	341
76. GetQMFQueryText parameters	341
77. GetQueryText parameters.	342
78. GetQueryVerb parameters.	342
79. GetResourceLimit parameters	343
80. Valid values for the parameter Resource.	343
81. GetResourceLimitEx parameters.	348
82. GetRowCount parameters.	349
83. GetServerList parameters.	350

84. GetServerListEx parameters	351
85. GetStoredProceduresResultSets parameters	351
86. GetVariables parameters	352
87. GetVariablesEx parameters	353
88. InitializeProc parameters	354
89. Valid values for the parameter SourceType.	354
90. InitializeQuery parameters	354
91. Valid values for the parameter sourcetype	355
92. InitializeServer parameters	355
93. InitializeStaticQuery parameters	357
94. IsStatic parameters	357
95. Open parameters	358
96. Prepare parameters	359
97. RunProc parameters	361
98. SaveData parameters	362
99. SaveQMFPProc parameters	364
100. SaveQMFPQuery parameters	364
101. SetBindOption parameters	365
102. Meanings and Values for the options	366
103. SetBindOwner parameters	368
104. SetBusyWindowButton parameters	368
105. SetBusyWindowMessage parameters	369
106. SetBusyWindowMode parameters	369
107. Valid values for the parameter mode	370
108. SetBusyWindowTitle parameters	370
109. SetGlobalVariable parameters	371
110. SetHostVariable parameters	372
111. SetOption parameters	372
112. Valid values for the parameter Option	372
113. SetParent parameters	374
114. SetProcVariable parameters	374
115. SetVariable parameters	375
116. ShowBusyWindow parameters	376
117. StartBind parameters	376

Preface

Database management system implementations within a single company are no longer restricted to a single data source that is easy to administer and easy to access from a single front end application. Today, the typical database infrastructure has become very diverse, with relational database management systems from various vendors, and may also be installed on different operating systems. In addition, these RDBMSs are not the only data sources; a considerable amount of data can also be stored in hierarchical databases such as Information Management System (IMS) and Virtual Storage Access Method (VSAM).

Administering and accessing these diverse data sources can easily become a nightmare for people who need to work with them. Each database management system may provide a different user interface, and this fact alone makes end users and companies unhappy, because they must either learn different user interfaces or must invest extensively in education.

In this redbook we describe how to set up an Enterprise Query Environment, where users can access different database management systems using the same front end application. Also, they can share the queries they created with other users, by either storing the queries at the appropriate database server or in a LAN accessible file. The database administrator will have the opportunity to restrict the resources available to each user, or user groups, thereby balancing the expected workload.

This redbook will help you design and implement a solution to build an Enterprise Query Environment using QMF for Windows. We discuss specifics for three typical users of this product: database administrators, application developers, and end-users. We describe the WWW functionality and other specifics of the product, such as its capabilities for Business-to-Business and Customer-to-Business applications. The information provided particularly applies to Version 6, Release 1 Refresh of QMF for Windows, Program Number 5655-DB2, for use with the Windows NT Operating System.

First, we introduce Query Management in general and give a short history of the QMF product family. We then discuss typical problems and possible solutions when dealing with a distributed database and query environment. We also describe the idea of the Enterprise Query Environment to be outlined in the rest of the book, and explain how to get started with setting up the Enterprise Query Environment. Included are the basics of the networking protocol used in our scenario (TCP/IP and SNA), configuration of these protocols, and installation of the QMF for Windows product.

After discussing QMF for Windows from the perspective of a database administrator (DBA), we present the necessary information for a software developer who wants to use the QMF for Windows application programming interfaces (APIs), with examples of Microsoft Visual Basic, Delphi, and C++ application development. For the end user of QMF for Windows, we cover the main tasks that the user of this product will typically need to perform. We conclude with a summary of topics not previously covered, and offer some hints on future directions for this versatile product.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization San Jose Center.

Joerg Reinschmidt is an Information Mining and Knowledge Management Specialist at the International Technical Support Organization, San Jose Center. He writes extensively and teaches IBM classes worldwide on Information Mining, Knowledge Management, DB2 Universal Database, and Internet access to legacy data. Before joining the ITSO in 1998, Joerg worked in the IBM Solution Partnership Center (SPC) in Germany as a DB2 Specialist, supporting independent software vendors (ISVs) to port their applications to use IBM data management products.

Catalin Comsia is a Senior Database Administrator with Bank of America in Seattle. He has 15 years of experience in Relational Database Management Systems, especially DB2. His areas of expertise include data analysis, systems architecture, database administration, and data warehouse. Catalin has worked extensively with relational database technology building Information Services projects for companies in the US, Canada, Italy, and Romania.

Andre Roberto Santos is a System Specialist working for IBM Global Service in Brazil. He has 5 years of experience in application development and has worked with IBM for 2 years. His areas of expertise includes programming languages such as Visual Basic and Delphi, and application development methods such as Object Modeling Technique (OMT), Unified Modeling Language (UML), and Structured Analysis. He holds a Master's degree in Database Human Computer Interfaces from the University of Sao Paulo, Sao Paulo, Brazil.

Thanks to the following people for their invaluable contributions to this project:

Michael Biere
IBM Cincinnati

Mark Flores
IBM Santa Teresa Lab

Tom Iglehart
Rocket Software

Vasilis Karras
IBM ITSO, Poughkeepsie Center

Matthew G. Kelley
Rocket Software

Andrea Reid
IBM Santa Teresa Lab

Mark Romano
IBM Santa Teresa Lab

Jon A. Scheer
Life Cycle Consulting Inc., Torrance, CA

Shawn Sullivan
Rocket Software

Dave Ward
Washington Mutual, Seattle, WA

Shirley Worthington
Multnomah County, Portland, OR

Andy Youniss
Rocket Software

Comments welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in “ITSO redbook evaluation” on page 413 to the fax number shown on the form.
- Use the online evaluation form found at <http://www.redbooks.ibm.com/>
- Send your comments in an Internet note to redbook@us.ibm.com

Chapter 1. Introduction to query management

In the current global economy, organizations must respond quickly to market opportunities and competitive pressures. In order to achieve this, they must empower managers and other knowledge workers to make rapid front-line decisions. This requires fast and easy access to information about the key factors driving the business — capabilities provided by business intelligence tools in general and data analysis tools in specific. A fundamental group of business intelligence products is comprised of query and reporting tools.

This redbook is about implementing a DB2 query and reporting tool, the Query Management Facility (QMF), specifically using QMF for Windows, into an existing enterprise environment. The audience of the redbook is intended to be database administrators, application developers, end-users, and other IT professionals with a need for an enterprise query and reporting environment.

In this chapter we investigate the basic requests of query and reporting management in an enterprise environment, and how these requests are answered by the QMF family of integrated tools available on OS/390, VM, VSE, and Windows.

1.1 Basics about queries and data

Most companies store large amounts of data about their day-to-day business processes. This data is a rich source of information about their business, its processes, and its customers. Increasing competitive pressures may be one of the most important drivers for seriously trying to use the information contained in all that data and using it to gain the knowledge needed to stay in business. If the economic situation is favorable, they probably do not have to worry about this potential. But once the competition moves ahead, or the environment deteriorates, it may be too late to change.

Today's Relational Database Management Systems (RDBMSs) capture, manipulate, and analyze much of the data that run the world. The majority of the world enterprises store their data in relational databases. To retrieve, analyze, and manipulate this data from the database, the first step is the *query*. A query is a statement that indicates only **what** subset of the data in the database should be retrieved and **how** this data is to be manipulated; the RDBMS controls and directs **where** the data is.

The fundamental difference between a *navigational* database system like Information Management System (IMS) and a *relational* database system like Database 2 (DB2) is that the RDBMS has the power to store and retrieve the access path to the data, making access to the data easier and quicker. The result of that query is a set of data that, formatted in a *report*, provides value and gives useful information for the company. Figure 1 shows the evolution of query and reporting over the last years.

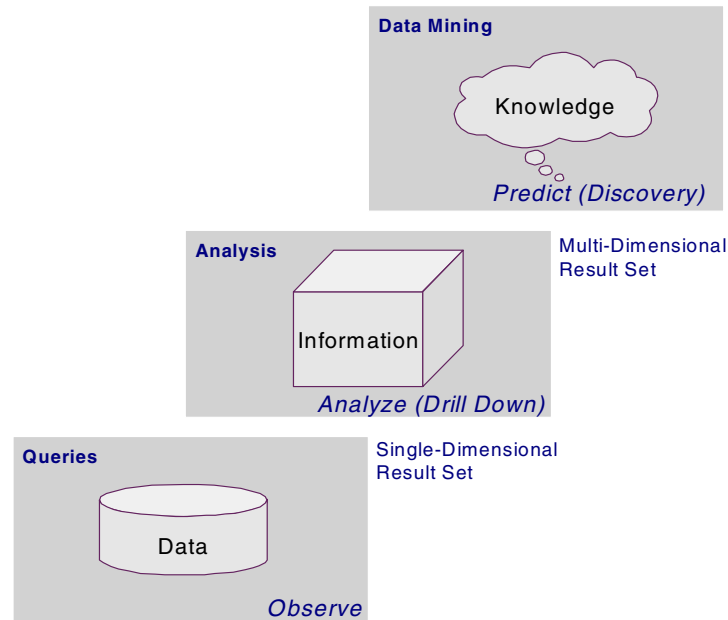


Figure 1. Evolution from queries to knowledge discovery

Queries can be made using Structured Query Language (SQL), a standard language to query relational database systems, but each RDBMS vendor provides its own specific variation. Almost every RDBMS is provided also with an ad-hoc query interface where the user can enter the SQL statement and receive the result. These interfaces require knowledge of SQL and are intended to be used mainly by power users. An alternative is provided by query and reporting tools that provide an easy means of generating SQL without knowledge of the SQL language itself.

1.2 Enterprise query and reporting environment

Until recently, query and reporting tools were typically software installed on mainframe systems and workstations, and it was only cost-effective to install and maintain them for dedicated analysts and other power users — typically about 20 percent of knowledge workers in an organization. Sharing and dissemination of results could be awkward and tedious as well.

Now, with the emergence of World Wide Web technologies and the acceptance of the Internet as a primary vehicle for distributing information, there is an opportunity to maximize the return on investment in corporate information. It is now possible to cost-effectively deliver reporting capabilities to every user, from senior executives and production line workers, to sales forces and branch offices across an entire organization and beyond, to customers, suppliers, and other business partners. Query and reporting tools should have some typical characteristics, such as being able to:

- Provide a common user interface.
- Centralize the administration.
- Give control over the resources used by the users.
- Prevent users from unwanted data access.

Today we are in the era of democratization of information, in which accessibility to data can make cost-effective enterprise query and reporting a reality. Figure 2 shows an overview of the components of a typical business intelligence environment.

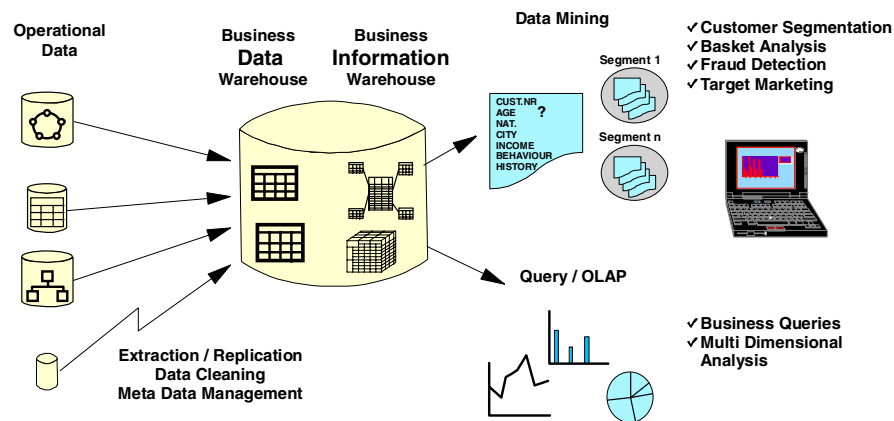


Figure 2. A business intelligence environment

Until recently, the primary focus of Information Technology (IT) query and reporting solutions was to address departmental needs, which meant deploying tools to power users within the client/server environment. With the emergence of enterprise data warehousing and a global economy, today's enterprise reporting requirements are more complex and diverse than ever before, and present IT professionals with a new set of challenges:

- Meeting enterprise reporting needs without creating an IT reporting backlog.
- Delivering reports to the organization in a cost-effective way that is easy to manage on a global basis.
- Delivering a solution that will grow with evolving user needs and that won't lock the organization into a particular vendor's infrastructure as technology changes.

The optimal approach to enterprise query and reporting is to provide organizations with a complete solution (one that addresses reporting requirements for both running and improving the business) that meets the needs of power users, casual users, and everyone in between. An example of this environment is shown in Figure 3.

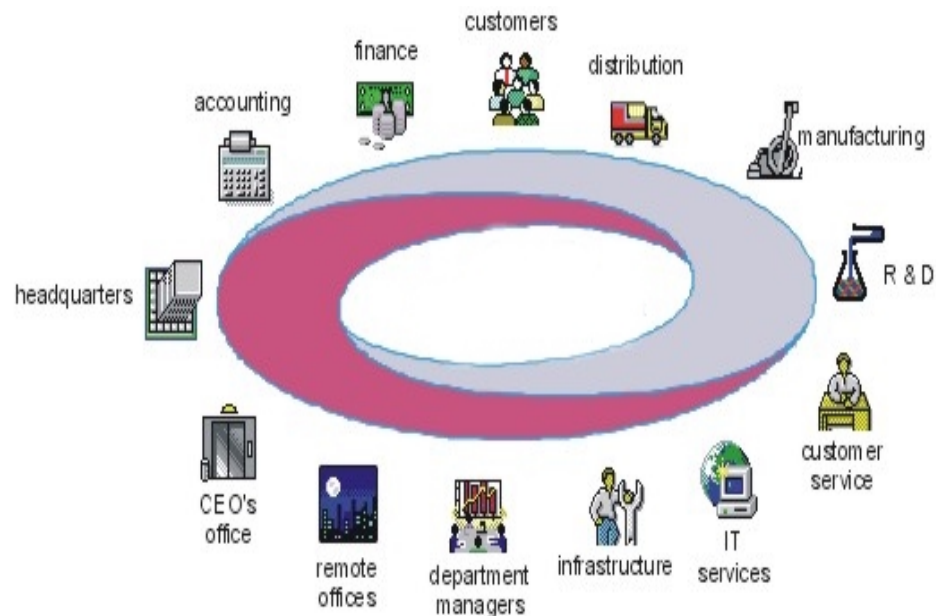


Figure 3. A query and reporting environment

Enterprise Query Environment solutions should also be built to address the new challenges facing IT as reporting emerges from departmental coverage to full-enterprise coverage. This can be achieved through:

- **Common Interface.** Providing a common user interface, independent of the platform a product is installed on, will reduce the cost of training the users as well as improve the end user acceptance of a tool.
- **Distributed reporting.** Organizations need to move as much reporting as possible to the end user community across many functional areas in the company.
- **Centralized report management and distribution.** To be cost-effective and timely, IT needs a central place to store, update, manage, and distribute reports in order to insure the integrity of the data.
- **Open and scalable Web architecture.** As we move into the next millennium, enterprise query and reporting architectures are increasingly moving to support the World Wide Web based architecture. This enables them to be open to support existing enterprise infrastructures, and to be scalable to accommodate all users both within the enterprise and outside the enterprise.

A significant portion of the cost of implementing software solutions is in deployment, training, and maintenance. Enterprise query and reporting solutions that are completely server based and have a browser interface ensure maximum deployability and the lowest possible deployment and maintenance burden for IT. At the time of this writing, QMF does not yet fully support this Web based approach, but it has already added the first steps in this direction by allowing reports to be published through a Web server automatically.

The typical enterprise database environment is heterogeneous, where the data sources usually are both relational and non-relational. Therefore, one big requirement from the users is to have a common access tool to allow this kind of distributed access. The QMF family of tools provides querying, analyzing, and reporting specifically designed for accessing data stored in DB2, but through DB2 DataJoiner is also capable to concurrently access non-relational data like IMS and Virtual Storage Access Method (VSAM), and multi-vendor relational databases.

1.3 A short history of the QMF family

QMF's history follows closely the evolution of DB2. Since Version 1.1 in 1984, QMF is architected specifically for — and developed in close parallel with — DB2 on the OS/390 platform (MVS, VM, and VSE), as shown in Figure 4.

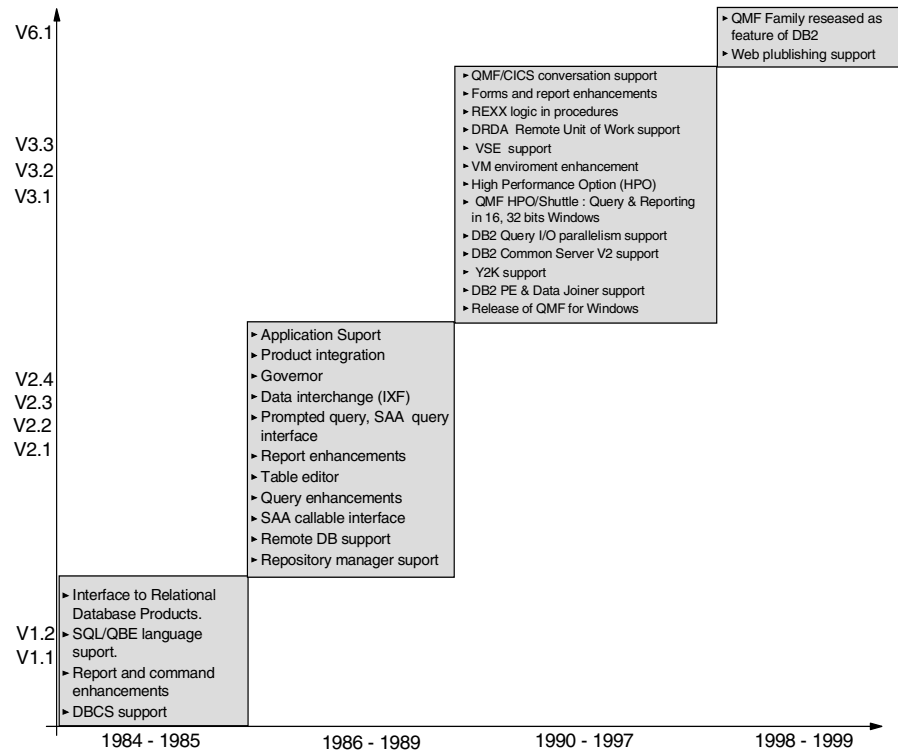


Figure 4. QMF and DB2 development history

The QMF family is an integrated toolset for querying, reporting, and updating data stored in the entire DB2 family as well as, when combined with DB2 DataJoiner, in nonrelational IBM data sources and other vendor relational data sources. QMF is available in 15 national languages.

The QMF Family has moved from being an independent product for use with DB2 for OS/390 and DB2 for VM and VSE to become a priced feature of the DB2 product. QMF for Windows can also be ordered as a separate product to access workstation DB2s only (AIX, OS/2, Windows NT, and so on).

The products featured in the QMF toolset are:

- **QMF for MVS, VM, and VSE** (QMF host) enables multiple types of users, novice, analyst, expert, application developers and DBAs, to access enterprise-wide data and produce reports (including HTML), charts or customized applications (via QMF API), as well as prototyping. QMF host allows for access to large amount of data, run batch processing, execute QMF procedures (including REXX conditional logic), and sharing central repositories of queries and reports across the enterprise.
- **QMF High Performance Option (HPO) for OS/390** is a set of integrated tools that helps manage a DB2 query environment and protects resources with extensive permissions and restrictions based on user resource groups and schedules. It includes QMF for Windows, the Windows based version of the query, reporting, and updating tool, as well as a management component to allow the administration of a QMF-only environment with such features as the ability to restrict user access based on different time schedules. Another component allows you to compile queries and reports into standard COBOL programs.

Three separate features have been consolidated into one QMF V6 feature called QMF HPO that includes:

- **QMF HPO/Manager** streamlines the administration and maintenance of QMF for OS/390 and QMF for MVS reporting environments with object tracking, query analysis, governing, and job canceling capabilities, that monitor and cancel currently executing QMF queries at will, plus many more capabilities not addressed in this book.
- **QMF HPO/Compiler** converts QMF for OS/390 and QMF for MVS queries, forms, and procedures into COBOL applications by automatically generating (and customizing as needed) COBOL code that executes static SQL and runs compiled reports from lists within TSO/ISPF and CICS (allowing it to be run in pseudo-conversational mode), or port them to other environments.
- **QMF for Windows** complements DB2 and the QMF Family by offering query, updating, and reporting from the Windows environment, providing access to existing QMF objects (queries, forms, procedures) without any migration necessary, execution of QMF commands, and governing control over user actions and DB2 resource consumption via its administrative component. QMF for Windows V6 can be ordered as a feature of DB2 for OS/390 V6, DB2 for VSE and VM V6, as a feature of QMF V6 on each of the aforementioned S/390 platforms, or as a stand-alone product licensed to access DB2 workstation databases only.

Note:

Any QMF for Windows host licensed version (OS/390, VM, VSE, and AS/400) can access all members of the DB2 family from mainframe to workstation.

The workstation-only license cannot access mainframe databases.

The development and support of QMF HPO Manager and Compiler and QMF for Windows is the responsibility of **Rocket Software**, an IBM business partner since 1991.

This chapter summarizes the typical IS department's list of needs for query & reporting tools, how QMF addresses these needs, and product positioning and platform functionalities of the QMF family with an accent on QMF for Windows.

2.1 Typical IS Needs

Today's enterprises are taking innovative approaches to information processing and delivery. Their approaches meld *delivery technologies* with the world of *business intelligence technologies*, providing the personalization we all desire in our working environments. The Information Services (IS) department in each of these companies is faced everyday with more demands for information processing and with the increasing need to offer creative solutions to deliver that information.

Let's take a look at the notes of an IS specialist at the end of a typical brainstorming meeting on query and reporting needs with the business partners/departments of their enterprise:

- "We can't afford to spend months setting up our warehouse front end."
- "We need a tool that gets us up and running quickly."
- "I want to look at my business data as soon as I install the product."
- "We need one tool that can access all our DB2 database platforms."
- "We use our spreadsheet applications all day long; that is where we need to work with our data."
- "We currently produce nightly, printed reports; we need to publish these reports on our intranet."
- "Who is using what queries?"
- "Which queries are not being used and can be removed from our reporting warehouse?"
- "Which queries are being run most frequently?"
- "We have to make use of stored procedures and static SQL to optimize performance."
- "Idle cursors and idle connections impact our database performance."
- "I need to connect to multiple database servers at the same time and run multiple queries at the same time."
- "We need to process big queries that return hundreds of thousands of rows of data."

In the rest of this book we will explore in detail how all these typical IS demands are answered by the QMF Family of Tools, specifically by QMF for Windows.

2.2 The Enterprise Query Environment

To address most of the questions mentioned above, there are certain requirements that need to be covered when thinking about a query and reporting environment. The requirements do not end with the list of concerns the IT specialist will arise, but also need to take into consideration the requirements of the user who will finally be the person to work with the installed tools. In general, we can talk about these aspects when thinking about an Enterprise Query Environment:

- IT issues
- User issues

2.2.1 IT issues

As mentioned previously, most of the typical business environments today deal with a wide variety of databases, tools, and even operating systems. One of the main challenges will be to get all these different products and environments to work together in a way that is transparent to the user, who is doing a day's work.

A fundamental requirement to accomplish this will be to implement a common data access infrastructure to all the various data sources. This requirement starts with an evaluation of the supported networking protocols, and covers all aspect, up to the point of implementing a common query and reporting tool to access the different database systems.

Today, much of this is easy to implement; for example, TCP/IP has evolved to become the standard for connectivity to different computer systems. In terms of database access, there are some standards available that allow access to different data sources, for example, Distributed Relational Database Architecture (DRDA) or Open Database Connectivity (ODBC).

Figure 6 shows an overview of different database systems and networking protocols used in a typical IT infrastructure in today's businesses.

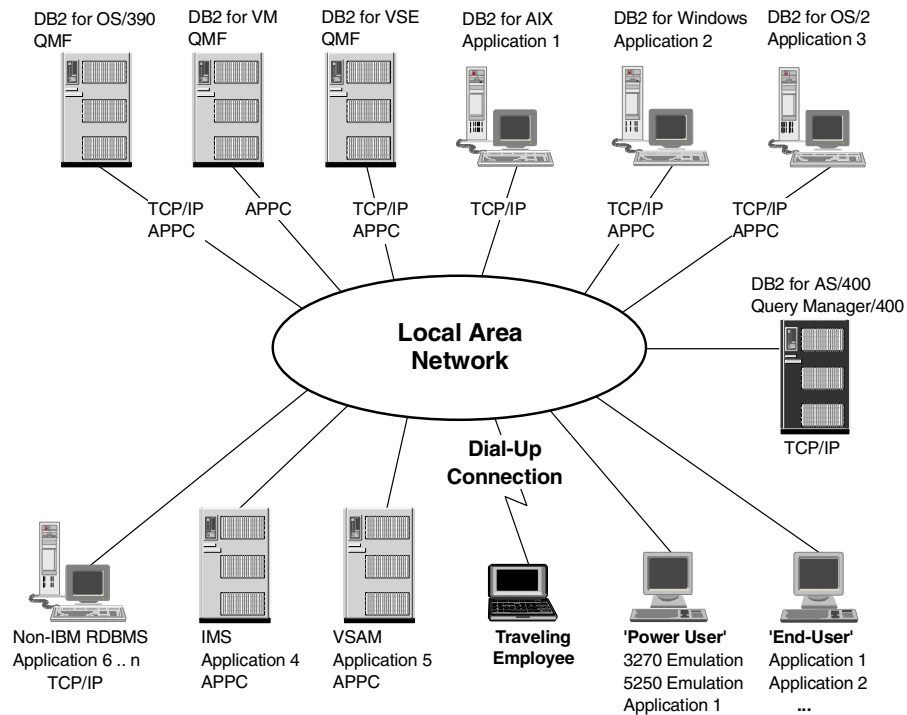


Figure 6. The IT infrastructure

This picture illustrates some concerns with this situation. For example:

- Not all of the data sources allow access using the same networking protocol.
- Administering and tracking the application installation will get very *complex*.
- The “Power User” needs access to all of the data sources available. Therefore, he needs to know how to handle all of the proprietary front ends. Once new queries or reports have been created, they should be made available to a broad number of employees, requiring them to use the same front end the query has been created with.
- The “End User” may only need access to predefined queries and reports, but most of the tools are too complex to handle, as they also allow the creation of new queries. Very often this kind of user even wants to use their standard spreadsheet application or personal database to access all the company data sources.

- The “Travelling User” does not want to have a full-function database client on his computer, but rather, would prefer to have a “thin client” installed.

To solve some of these problems, businesses develop in-house applications to support their employees. However, these applications need to be maintained, and are also very costly to be kept up-to-date for new technologies.

2.2.2 User issues

As mentioned above, in an environment like that shown in Figure 6 on page 12, the user must be able to use different applications to perform his work. Many of these applications are too complex, or they allow more options than most of the users require.

The majority of users may run the same query again and again. Once a query has been redefined due to new requirements, the user needs to find a way (or is told a way) to transfer this new query to his own system. Normally, the end user would be happier if someone else who is better skilled in these matters could perform this task. Providing a central shared query repository would allow the queries to be edited at a single source and be available to the users without any further action required.

Finally, the database administrator is increasingly confronted with resource and performance problems resulting from many more users accessing the databases. For example, imagine that a highly tuned database suddenly gets requests from many users, all using their favorite applications, executing new queries that cause enormous amounts of data to be sent across the network.

2.3 QMF for Windows: the solution

QMF for Windows complements the QMF family by offering query and reporting from the Windows environment, access to existing QMF host objects (queries, forms, procedures), execution of QMF commands, and governing control over user actions and DB2 resource consumption.

Figure 7 shows a way to solve most, if not all, of the concerns raised previously. It shows the addition of another product, DB2 DataJoiner, to the existing environment to access non-DB2 data sources.

While DB2 DataJoiner acts like any other DB2 database management system, it allows you to map a non-IBM relational database such as Oracle, Sybase, Informix, MS SQL Server, to be accessed from the application, just as if it were another DB2 database. DB2 DataJoiner, in connection with Classic Connect, also allows non-relational data sources, such as IMS and VSAM, to be accessed as if they were a DB2 relational database.

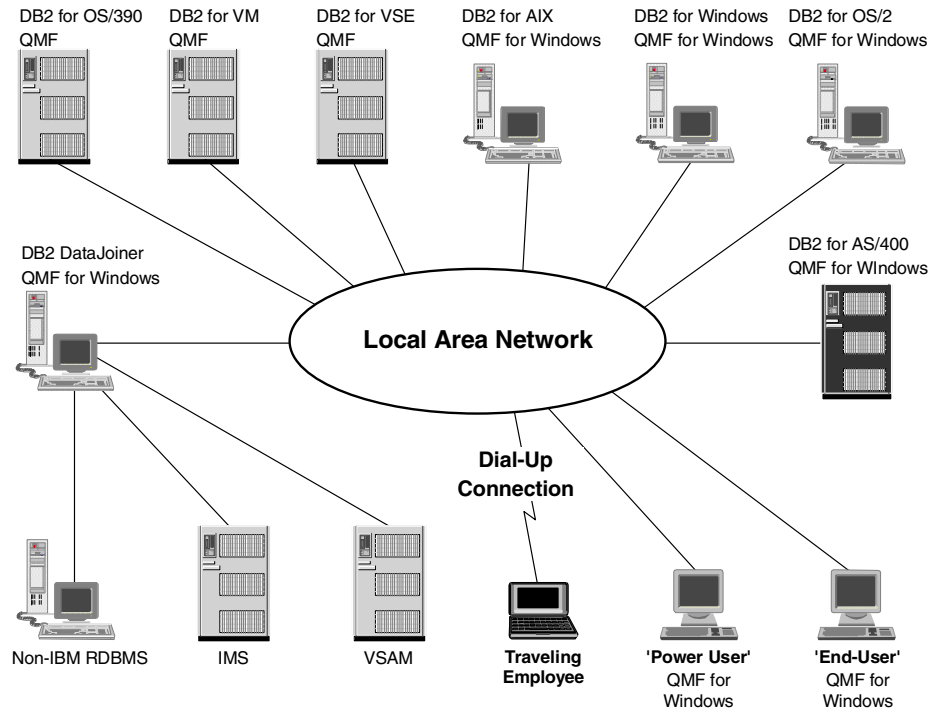


Figure 7. The Enterprise Query Environment

2.3.1 Product Positioning

The best way to describe query and reporting tools is to simply say that "business intelligence starts with query and reporting tools". Figure 8 shows the evolution of decision support and its fundamental place in business intelligence.

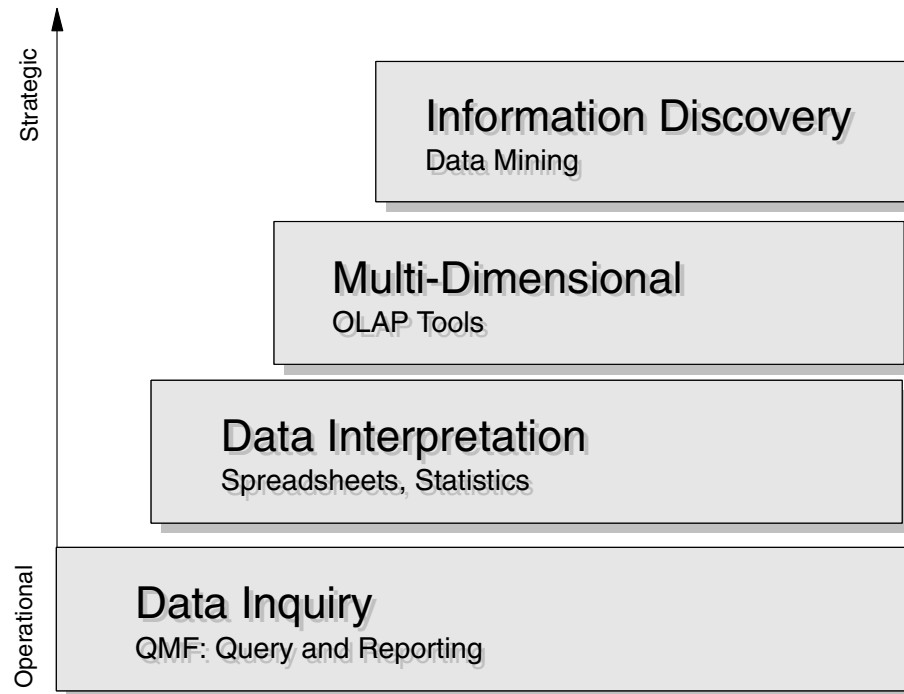


Figure 8. Decision support evolution

QMF for Windows delivers the most essential business reporting requirements for corporate situations where the broadest range of needs must be met with the minimum number of tools.

QMF for Windows Version 6 is a multi-purpose Enterprise Query Environment for large scale business reporting, data sharing, server resource protection, rapid application development (RAD), and native connectivity to all of the DB2 host (OS/390, MVS, VM, and VSE) and DB2 workstation platforms. It provides native support for TCP/IP connectivity to DB2 Version 5 and above, and is coupled with DB2 across platforms by the enterprise data sharing standard Distributed Relational Database Architecture (DRDA). Support for multi-vendor database environments through DB2 DataJoiner enables users to access data outside DB2.

Version 6 includes support for QMF linear procedures, DB2 stored procedures, a command line interface for enhanced automation, and new levels of governing available to administrators. QMF Form calculations (requiring IBM Object REXX and 32-bit) and specialized form variables for Web publishing enable you to build advanced features into QMF reports.

End users can work from the QMF for Windows "quick start" interface to build queries and business reports, share them with other QMF users, and publish them on the Web. QMF for Windows can also be used as the data manipulation engine behind the most important commercial or custom Windows applications — including spreadsheets, charting and analysis tools, executive information systems, and desktop databases.

DBAs will appreciate QMF for Windows as a single product with the capabilities to open all of the DB2 host and DB2 workstation database platforms to Windows users (3.x, 9x, NT, and WIN-OS/2), without having database gateways, middleware, or ODBC drivers to manage. QMF for Windows provides resource management functions to block waste or abuse through detailed permissions organized on each server by group, by schedule, or by combinations of group and schedule.

Developers obtain a robust Windows application programming interface (API) with QMF for Windows that lets them rapidly build DB2 applications. QMF for Windows allows them to fully exploit DB2 performance, SQL syntax, and advanced database performance techniques such as static SQL, uncommitted read, and DB2 stored procedures.

QMF for Windows V6 can be ordered as a feature of DB2 for OS/390 V6, DB2 for VSE and VM V6, as a feature of QMF V6 on each of the aforementioned S/390 platforms, or as a stand-alone product licensed to access DB2 workstation databases only, as shown in Figure 9.

Note:

QMF for Windows has a "server based licensing", which means that no matter how many users use the product, the licensing is based on the DB2 servers (or subsystems) accessed. So, for each DB2 accessed, no matter by how many users, only one QMF for Windows license is needed.

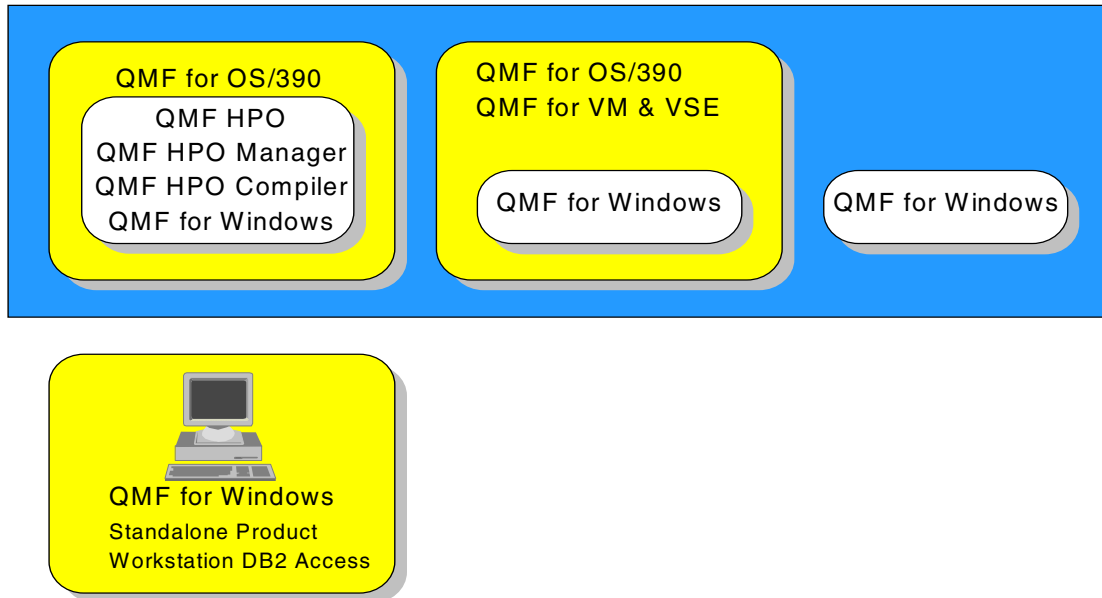


Figure 9. QMF family of products

2.4 Enterprise Benefits

QMF for Windows provides the following benefits for the user, the developer, the database administrator, and the enterprise.

For the user

QMF provides these benefits:

- Build new queries easily with the standard prompted query builder.
- Use existing QMF objects stored on any server.
- Automate DB2 query from Windows applications.
- Integrate with Lotus 1-2-3, Excel, Lotus Approach, Access, Delphi, or many other OLE 2.0 applications.
- Create and share QMF forms.
- Edit DB2 data directly in the Table Editor.
- Use the QMF for Windows GUI or a favorite application interface.
- Edit table rows from a query result, or a row at a time.
- Query multiple servers simultaneously.

- Get outstanding DB2 performance and reliability.

For the developer

QMF provides these benefits:

- Bring industrial strength DB2 applications to desktop tools.
- Integrate DB2, QMF objects, and commands with Windows 3.x, Windows 9x, or Windows NT OLE 2.0 automation controller application.
- Easily build Windows applications that:
 - Retrieve QMF queries from servers.
 - Launch QMF commands.
 - Integrate existing QMF forms.
- Create new or select existing QMF forms from the Windows desktop.
- Use the table editor to create test data.
- Convert heavily used queries to static SQL for better performance.
- Shield users from the complexity of connecting to databases.
- Control QMF for Windows in the background with its own API.

For the database administrator

QMF provides these benefits:

- Execute static SQL from Windows.
- Centralized install/elimination of database gateways, middleware, and ODBC driver.
- Protect DB2 from runaway queries and novice users.
- Build governing into Windows applications.
- Use existing DB2 security.
- Centralize control over server resources.
- Adjust governing limits by the following:
 - Time of day
 - Day of week
 - User groups
 - Server
- Set governing thresholds to do the following:
 - Warn users
 - Cancel queries and threads

- Limit by:
 - Rows fetched
 - Idle query time-outs
 - Server response time-outs
 - Idle connection time-outs
- Allow or disallow 14 different SQL verbs.
- Turn on/off table editor and other features by group.

For the enterprise

QMF provides these benefits:

- TCP/IP support for DB2 V5, including DB2 Universal Database.
- Large scale retrieval with outstanding performance--from Windows.
- Full 16-bit and 32-bit support.
- Query local or remote databases.
- Maintain full DB2 security and authorizations.
- Fully exploit DB2 system integrity.
- Maximize return on server investment, minimize waste.
- Eliminate TSO, CMS, and CICS HOST logon.
- Make enterprise database resources available, yet more protected.
- Gain from the ease of use of desktop languages and availability of skills.
- Develop business solutions rapidly and flexibly.
- Minimize complexity.

Existing host QMF users

QMF provides these benefits:

- Extends QMF capabilities to the desktop.
- Extends QMF capabilities to Windows applications.
- No learning curve.
- Extends QMF reports to the Web.
- Converts existing QMF reports to Web reports.

2.5 Implementation examples

In the following sections we give two examples of QMF for Windows implementations that have been performed for IBM customers.

2.5.1 Financial industry

The first example concerns the implementation of QMF for Windows in a company working in the financial market segment.

2.5.1.1 Platforms

QMF for Windows is used to access two DB2 subsystems: a production system and a test system, which both run MVS on an IBM ES9000-982.

2.5.1.2 Background

The bank is a large financial institution and has the task of providing accurate and timely information to different areas of the company. After the decision had been made to eliminate VM, a solution was needed to provide the users with the same level of support they were receiving before. All of the databases and associated tables and queries on VM were migrated to a decision support DB2 subsystem.

2.5.1.3 The problem and the solution

QMF for Windows is now used to provide access to the DB2 tables, and only a minimum of modification has been necessary to help the users adapt to the new environment. Users who were familiar with using QMF on VM before have had no problems getting used to the new implementation, and users who had never used QMF before did not need to learn anything about TSO. The QMF for Windows Graphical User Interface (GUI) has provided the users with an easy-to-use solution to access the data in the format they needed. It has also allowed printing the reports locally and in a timely manner. The data can be downloaded from the host system and loaded directly into the user's Windows based database or into a spreadsheet application. This overall approach has eliminated many of the steps previously required to get the data from the host to the PC application.

2.5.1.4 Product functionality

The QMF for Windows front end provides the users with easy access to the company's data by providing them with lists of queries and forms available to them, and allowing them to run those queries and forms with a simple click of the mouse. New queries can be created and stored on the mainframe system or in a file stored on a PC. Downloading data is now as simple as executing a query and saving the data to the end user's PC based application.

2.5.1.5 Strengths and weaknesses

The main strength of the new environment is that the end users no longer have to log on to TSO in order to gain access to the required data, which has been an important issue when eliminating VM.

One of the few weaknesses has been that the error messages, which appear with communication problems using SNA as the protocol between the end users workstation and the host, have been hard to interpret.

2.5.1.6 Selection criteria

There have been three main reasons that the customer chose QMF for Windows:

1. Most of the users were already familiar with QMF on VM, so the transition to the new environment was very easy.
2. Using a Windows based product eliminated the requirement of many users having TSO authority.
3. The ability to interact with standard Windows based applications was helpful.

2.5.1.7 Deliverables

The new environment allows the end users to create new queries, and to download data for report processing or for import into Windows applications.

2.5.1.8 Required support

The only support required during the implementation of the new environment was the need for assistance to configure the SAN Network Gateway.

2.5.2 Public services institution

The second example concerns the implementation of QMF for Windows in a public services institution.

2.5.2.1 Platforms

DB2 V 4.1 for MVS runs on an IBM 9672-R32 under OS/390. QMF runs on this 9672 under both TSO and CICS, and is available to any user who has security on the CICS region under which QMF runs. QMF for Windows is installed on a Windows NT Server and is connected via LU 6.2 through an SNA server.

2.5.2.2 Background

This institution supports local government's information processing, networking, software, and telecommunications needs.

2.5.2.3 The problem and the solution

The institution needed a more efficient way to analyze information that could help them solve certain problems. There was a tremendous amount of information stored in an operational system and VSAM files. In the past, in order for the data to be analyzed, on-line CICS queries were used, or a request would be sent to the application development department to create or run a COBOL report with various parameters. Therefore, a new report request could take days, weeks, or even months to be completed, due to an ever-increasing backlog of requests. A few years ago, the IS department set up a data warehouse to allow ad-hoc queries to be executed. Many users had been trained to use the prompted feature of QMF on CICS. Today the users can easily look at trends and find similarities in events by using QMF for Windows.

2.5.2.4 Product functionality

The QMF for Windows front end provides the users with easier access to the data. Printing and downloading capabilities have been improved as well. It is easy to use, plus it allows for the utilization of existing work already generated with QMF under CICS. The download add-ins of QMF for Windows ability to directly import data into the Windows based application also enhances its functionality. Now it is possible, with a click of a button, to execute a query on QMF that joins several tables, each having between 300,000 to over 2 million rows of data, and transfer the small result set into a Windows spreadsheet application automatically.

2.5.2.5 Strengths and weaknesses

The QMF for Windows strengths have been the ease of installation and use, as well as giving people access to existing QMF for OS/390 queries from a Windows based workstation.

No weaknesses have been reported.

2.5.2.6 Selection criteria

As the customer had already used QMF on the OS/390 system, the ability of QMF for Windows to utilize all of the efforts invested on the host platform without any modifications was the justification for its implementation.

2.5.2.7 Deliverables

The new environment allows the end users to create new queries easily by using the prompted query feature of QMF for Windows, and to download and utilize the data from the host easily.

2.5.2.8 Required Support

The only support required during the implementation of the new environment was the need for assistance to configure the SAN Server on the Windows NT server.

Chapter 3. Getting started

This chapter explains the basic architecture behind the operation of QMF for Windows and describes the prerequisite software and configuration that it requires. It provides a step-by-step guide to install QMF for Windows and the QMF for Windows Administrator. The intended audience is the person (system programmer, or database administrator) responsible for installing QMF for Windows.

3.1 The networking environment

The primary function of QMF for Windows is to access data stored in any database in the DB2 family of databases, including DB2 DataJoiner. There are two ways in which QMF for Windows can connect to DB2: via the Open Group's Distributed Relational Database Architecture (DRDA), and via the DB2 UDB for Windows, OS/2 or AIX Call Level Interface (CLI).

In data communication, several layers of communication are used in a very complex environment connecting different operating systems and data sources of completely different architectures.

From the application point of view, the base for all communication and data transfer consists of *communication protocols*. Several protocols are available for inter-system communication. Especially in the workstation area, you have several options:

- The connectivity between the workstation environment and the host systems is based, for example, on Ethernet or on Token Ring.
- The communication protocols mostly used in the workstation environment are NetBIOS and TCP/IP. The protocol TCP/IP is used for communication to host systems as well.
- The most common protocol in the host environment is still APPC (Advanced Program to Program Communication) based on Shared Network Architecture (SNA). The use of Advanced Peer-to-Peer Networking (APPN) simplifies the configuration.

The next layer is the protocol or architecture supplying the base for the *data exchange* between the data sources and targets.

- For the data transfer between the DB2 family products and other relational database software in different operating environments (host and workstation), this is Distributed Relational Database Architecture (DRDA). This architecture is a detailed blueprint that specifies all of the layers and functions required in a client/server distributed database application.
- Between like platforms (workstations or VM and VSE) there are *private* protocols available as well. For example, between the Universal Databases (UDBs) on the workstations, there is a private data exchange protocol available, which has the same functionality as DRDA.
- For the nonrelational data sources, there are special “private” solutions (like CrossAccess or Classic Connect) used to provide these sources in a relational view.

QMF for Windows and DB2 are both distributed relational database applications that operate together in a client/server relationship. Both QMF for Windows and DB2 implement and adhere to DRDA. Each component plays a separate and distinct role in this relationship:

- QMF for Windows is the **DRDA application requester**.
- DB2 acts as the **DRDA application server**.

One layer of DRDA describes the communication protocol that must be used by the participants in the architecture. Specifically, it defines that requesters and servers must communicate via the TCP/IP or the SNA LU 6.2 protocol.

Before installing, configuring, or using QMF for Windows you must verify that the required network infrastructure is properly configured.

CLI is an application programming interface for relational database access, included as a part of the DB2 UDB for Windows, OS/2, or AIX client. Using CLI, client applications (like QMF for Windows) can connect to DB2 UDB servers and execute SQL statements. In addition to DB2 Connect, CLI applications can also connect to the mainframe databases (OS/390, VM, and VSE).

3.2 TCP/IP basics

The need to interconnect networks that use different protocols was recognized early in the 1970s during a period when both the use and development of networking technology were increasing. Even though the rapid growth in networking over the past three decades has allowed users much greater access to resources and information, it has caused significant problems with merging, or interconnecting, different types of networks.

Open protocols and common applications were required, leading to the development of a protocol suite known as *Transmission Control Protocol/Internet Protocol* (TCP/IP), which originated with the U.S. Department of Defense (DoD) in the mid-1960s, and took its current form around 1978.

An interesting article about the history of the Internet can be found at:

<http://www.isoc.org/internet-history/>

In the early 1980s, TCP/IP became the backbone protocol in multivendor networks such as ARPANET, NFSNET, and regional networks. The protocol suite was integrated into the University of California at Berkeley's UNIX operating system and became available to the public for a nominal fee. From that point TCP/IP became widely used. Its spread to other operating systems resulted in increasing use in both local area network (LAN) and wide area network (WAN) environments.

Today, TCP/IP enables corporations to merge differing physical networks while giving users a common suite of functions. It allows interoperability between equipment supplied by multiple vendors on multiple platforms, and it provides access to the Internet. In fact, the Internet, which has become the largest computer network in the world, is based on TCP/IP.

So why has the use of TCP/IP grown at such a rate? The reasons include the availability of common application functions across differing platforms and the ability to access the Internet, but the primary reason is that of interoperability. The open standards of TCP/IP allow corporations to interconnect or merge different platforms. An example is the simple case of allowing file transfer capability between an MVS/ESA host and, perhaps, a Hewlett Packard workstation.

TCP/IP also provides for the routing of multiple protocols to and from diverse networks. For example, a requirement to connect isolated networks using IPX, AppleTalk, and TCP/IP protocols using a single physical connection can be accomplished with routers using TCP/IP protocols.

One further reason for the growth of TCP/IP is the popularity of the socket programming interface between the TCP/IP transport protocol layer and TCP/IP applications. A large number of applications today have been written for the TCP/IP socket interface.

3.2.1 TCP/IP architecture

TCP/IP, as a set of communications protocols, is based on layers. Unlike System Network Architecture (SNA) or Open Systems Interconnection (OSI), which distinguish seven layers of communication, TCP/IP has only four layers. See Figure 10. The layers enable heterogeneous systems to communicate by performing network-related processing such as message routing, network control, and error detection and correction.

- Application layer

The application layer is provided by the program that uses TCP/IP for communication. Examples of applications are Telnet, File Transfer Protocol (FTP), e-mail, Gopher and SMTP. The interface between the application and transport layers is defined by port numbers and sockets.

- Transport layer

The transport layer provides communication between application programs. The applications may be on the same host or on different hosts. Multiple applications can be supported simultaneously. The transport layer is responsible for providing a reliable exchange of information. The main transport layer protocol is TCP. Another is User Datagram Protocol (UDP), which provides a connectionless service in comparison to TCP, which provides a connection-oriented service. That means that applications using UDP as the transport protocol have to provide their own end-to-end flow control. Usually, UDP is used by applications that need a fast transport mechanism.

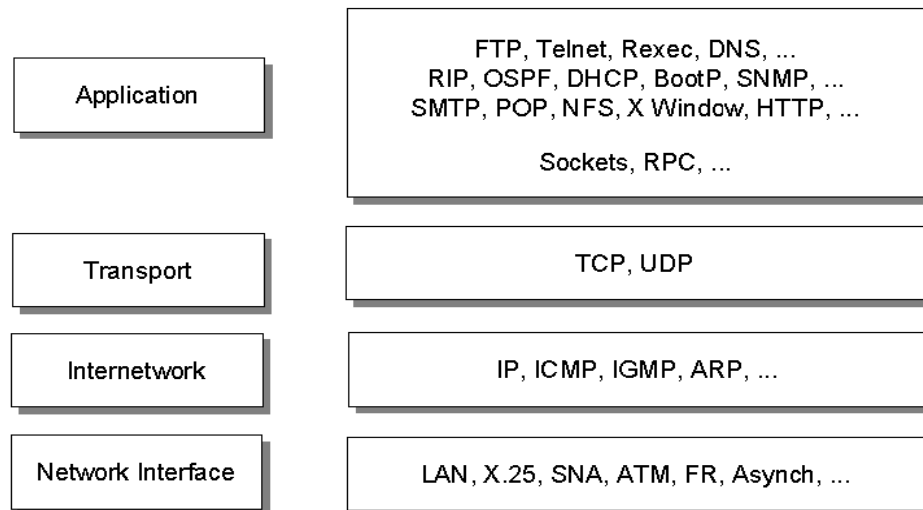


Figure 10. TCP/IP architecture model: layers and protocols

- Internet layer

The Internet layer provides communication between computers. Part of communicating messages between computers is a routing function that ensures that messages will be correctly delivered to their destination. The Internet Protocol (IP) provides this routing function. Examples of Internet layer protocols are IP, ICMP, IGMP, ARP, and RARP.

- Network interface layer

The network interface layer, sometimes also referred to as link layer, data link layer or network layer, is implemented by the physical network that connects the computers. Examples are LAN (IEEE 802.x standards), Ethernet, X.25, ISDN, ATM, Frame Relay, or async.

Note that the request for changes (RFCs) actually do not describe or standardize any network layer protocols by themselves, they only standardize ways of accessing those protocols from the Internet layer.

3.2.2 IP addressing

IP uses *IP addresses* to specify source and target hosts on the Internet. (For example, we can contrast an IP address in TCP/IP with a fully qualified NETID.LUNAME in SNA). An IP address consists of 32 bits and is usually represented in the form of four decimal numbers, one decimal number for each byte (or octet). For example:

00001001	01000011	00100110	00000001	a 32-bit address
9	67	38	1	decimal notation

An IP address consists of two logical parts: a network address and a host address. An IP address belongs to one of four classes.

Some values for these host IDs and network IDs are preassigned and cannot be used for actual network or host addressing:

- All bits 0

Stands for *this*: this host (IP address with <host address>=0) or this network (IP address with <network address>=0). When a host wants to communicate over a network, but does not know the network IP address, it may send packets with <network address>=0. Other hosts on the network interpret the address as meaning *this network*. Their reply contains the fully qualified network address, which the sender records for future use.

- All bits 1

Stands for *all*: all networks or all hosts. For example:

128.2.255.255

Means all hosts on network 128.2 (class B address).

This is called a directed broadcast address because it contains both a valid <network address> and a broadcast <host address>.

- Loopback

Class A network 127.0.0.0 is defined as the loopback network. Addresses from that network are assigned to interfaces that process data inside the local system and never access a physical network (loopback interfaces).

3.2.3 Subnets

Because of the explosive growth of the Internet, the principle of assigned IP addresses became too inflexible to facilitate changes to local network configurations. Such changes might occur when:

- A new type of physical network is installed at a location.
- Growth of the number of hosts requires splitting the local network into two or more separate networks.
- Growing distances require splitting a network into smaller networks, with gateways between them.

To avoid having to request additional IP network addresses in these cases, the concept of subnets was introduced. The assignment of subnets can be done locally, as the whole network still appears to be one IP network to the outside world.

Recall that an IP address consists of a pair <network address> and <host address>. For example, let us take a class A network; the address format is shown in Figure 11.

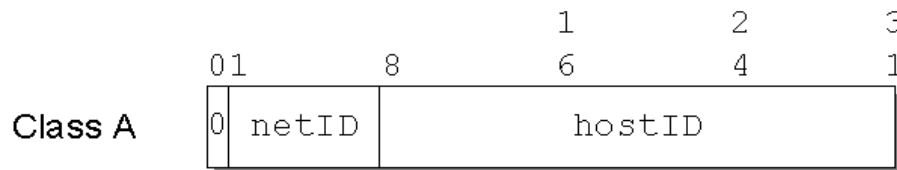


Figure 11. Class A address without subnets

Let us use the following IP address:

00001001 01000011 00100110 00000001 a 32-bit address
 9 67 38 1 decimal notation (9.67.38.1)

9.67.38.1 is an IP address (class A) having:
 9 as the <network address>
 67.38.1 as the <host address>

Subnets are an extension to this concept, by considering a part of the <host address> to be a subnetwork address. IP addresses are then interpreted as <network address><subnetwork address><host address>.

We may, for example, want to choose the bits from 8 to 25 of a class A IP address to indicate the subnet addresses, and the bits from 26 to 31 to indicate the actual host addresses. Figure 12 shows the subnetted address that has thus been derived from the original class A address.

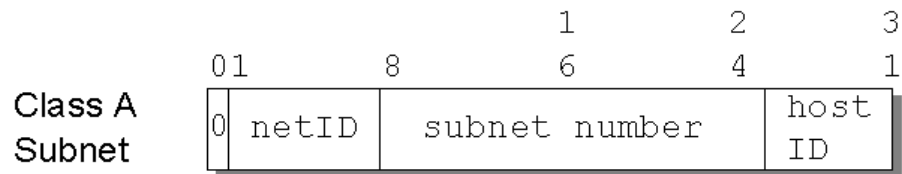


Figure 12. Class A address with subnet mask and subnet address

We usually use a bit mask, known as the *subnet mask*, to identify which bits of the original host address field to indicate the subnet number. In the above example, the subnet mask is 255.255.255.192 in decimal notation (or 11111111 11111111 11111111 11000000 in bit notation). Note that, by convention, the <network address> is masked as well.

For each of these subnet values, only $(2^{18})-2$ addresses (from 1 to 262143) are valid because of the all bits 0 and all bits 1 number restrictions. This split will therefore give 262142 subnets, each with a maximum of $(2^6)-2$ or 62 hosts.

The value applied to the subnet number takes the value of the full byte, with nonsignificant bits set to zero. For example, the hexadecimal value 01 in this subnet mask assumes an 8-bit value, 01000000, and gives a subnet value of 64 (and not 1, as it might seem).

Applying this mask to our sample class A address 9.67.38.1 would break the address down as follows:

```
00001001 01000011 00100110 00000001 = 9.67.38.1 (class A address)
11111111 11111111 11111111 11----- 255.255.255.192 (subnet mask)
===== logical_AND
00001001 01000011 00100110 00----- = 9.67.38 (subnet base address)
```

and leaves a host address of:

```
----- ----- ----- --000001 = 1 (host address)
```

IP recognizes all host addresses as being on the local network for which the logical_AND operation described above produces the same result. This is important for routing IP datagrams in subnet environments (see “IP routing” on page 33).

The actual number would be:

```
----- 01000011 00100110 00----- = 68760 (subnet number)
```

The subnet number shown above is a relative number, that is, it is the 68760th subnet of network 9 with the given subnet mask. This number bears no resemblance to the actual IP address that this host has been assigned (9.67.38.1) and has no meaning in terms of IP routing.

The division of the original <host address> part into <subnet> and <host> parts can be chosen freely by the local administrator; except that the values of all zeros and all ones in the <subnet> field are reserved for special addresses.

3.2.4 IP datagram

The unit of transfer of a data packet in TCP/IP is called an *IP datagram*. It is made up of a header containing information for IP and data that is only relevant to the higher-level protocols. IP can handle fragmentation and reassembly of IP datagrams. The maximum length of an IP datagram is 65,535 bytes (or octets). There is also a requirement for all TCP/IP hosts to support IP datagrams of up to 576 bytes without fragmentation.

The IP datagram header is a minimum of 20 bytes long (see Figure 13).

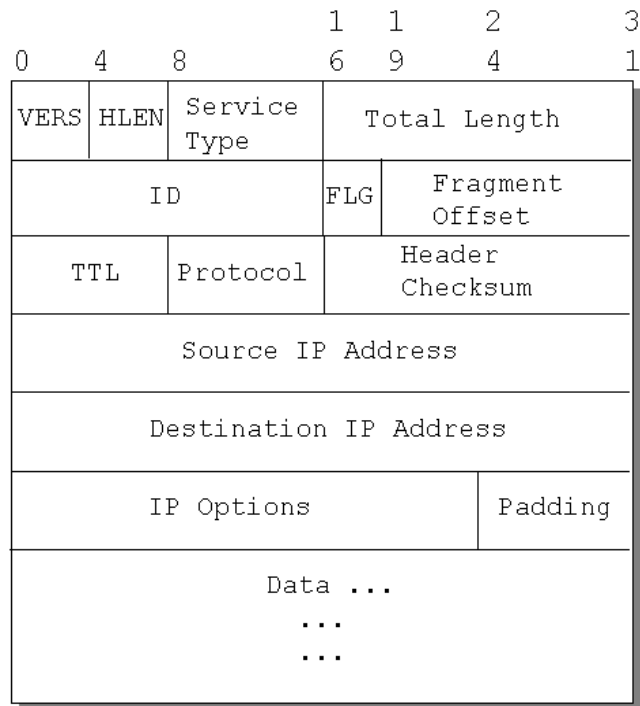


Figure 13. Format of an IP datagram header

3.2.5 IP routing

There are two types of IP routing: direct and indirect. For example, Figure 14 shows that the host C has a direct route to hosts B and D, and an indirect route to host A through gateway B.

3.2.5.1 Direct routing

If the destination host is attached to a physical network to which the source host is also attached, an IP datagram can be sent directly, simply by encapsulating the IP datagram in the physical network frame. This is called *direct delivery* and is referred to as *direct routing*.

3.2.5.2 Indirect routing

Indirect routing occurs when the destination host is not on a network directly attached to the source host. The only way to reach the destination is through one or more IP gateways. In TCP/IP terminology, the terms *gateway* and *router* are used interchangeably for a system that actually performs the duties of a router. The address of the first of these gateways (the first hop) is called

an indirect route in the context of the IP routing algorithm. The address of the first gateway is the only information needed by the source host.

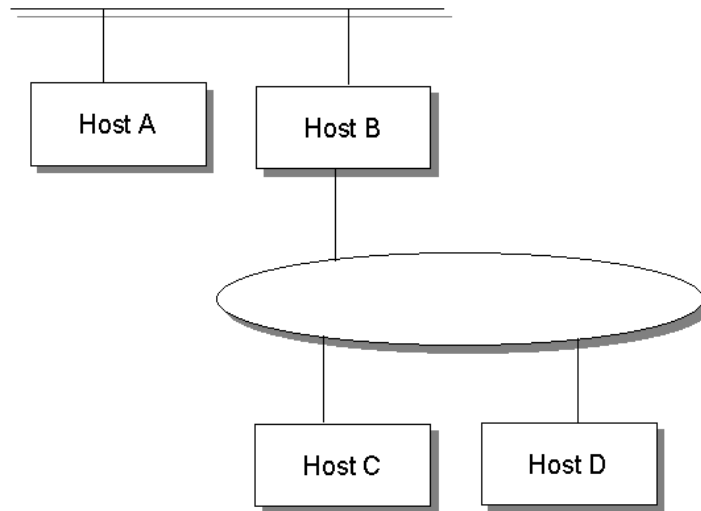


Figure 14. Direct and indirect routing

3.2.5.3 IP routing table

The determination of available direct routes is derived from the list of local interfaces available to IP and is composed by IP automatically at initialization. A list of networks and associated gateways (indirect routes) must be configured to be used with IP routing if required.

Each host keeps the set of mappings between the:

- Destination IP network addresses
- Routes to next gateways

The mappings are stored in the IP routing table (see Figure 15). Three types of mappings can be found in this table:

- Direct routes, for locally attached networks
- Indirect routes, for networks reachable through one or more gateways
- Default route, which contains the (direct or indirect) route to be used if the destination IP network is not found in the mappings of type 1 and 2 above

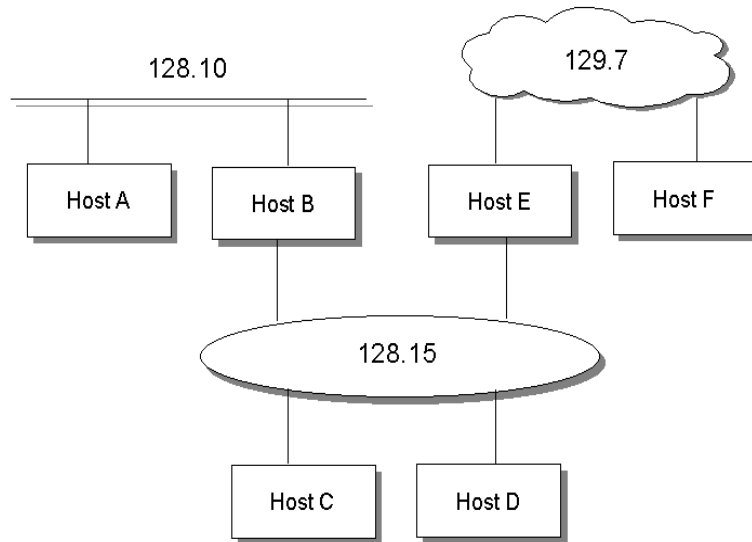


Figure 15. Routing table scenario

The routing table of host F might contain the symbolic entries shown in Figure 16.

destination	router	interface
129.7.0.0	F	lan0
128.15.0.0	E	lan0
128.10.0.0	E	lan0
default	B	lan0
127.0.0.1	loopback	lo

Figure 16. IP routing table entries example

3.2.5.4 IP routing algorithm

IP uses a unique algorithm to route an IP datagram. Figure 17 shows an IP routing algorithm with subnets.

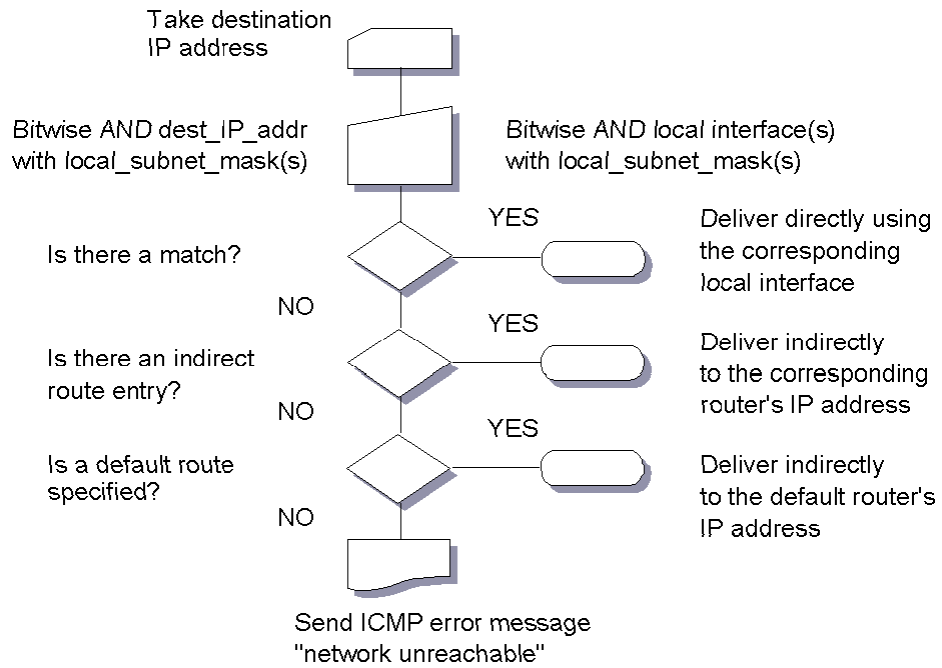


Figure 17. IP routing algorithm (with subnets)

Note:

- IP routing is an iterative process. It is applied by every host handling a datagram, except for the host to which the datagram is finally delivered.
- Routing tables and the routing algorithm are local to any host in an IP network. To forward IP datagrams on behalf of other hosts, routers must exchange their routing table information with other routers in the network, using special routing protocols.

3.3 Configuring your TCP/IP

To access a DB2 server using TCP/IP, QMF for Windows must be able to establish a TCP/IP connection from the **local host** (the system on which QMF for Windows is running) to the **remote host** (the system on which DB2 is running) and **remote port** (the port on which DB2 is listening).

The process of implementing TCP/IP connectivity between Microsoft Windows and DB2 is generally a much easier task than the corresponding SNA connectivity. You still must rely on your in-house TCP/IP networking staff and your TCP/IP software vendor's technical support services to implement and support your network configuration.

QMF for Windows requires a WinSock 1.1 interface to the installed TCP protocol stack.

3.3.1 OS/390 OpenEdition

If OpenEdition MVS and TCP/IP are used for the first time, it is highly recommended that a local OS/390 specialist with skills in OpenEdition MVS and TCP/IP be enlisted to assist. The redbook *Accessing OS/390 OpenEdition MVS from the Internet*, SG24-4721, provides guidance to configure TCP/IP with emphasis on accessing OpenEdition.

If TCP/IP V3R1 for MVS is already installed on your system, it will in many cases be useful to set up a separate TCP/IP started task for OpenEdition MVS services.

Note

You can run one or more TCP/IP address space copies on a single MVS system. To concurrently connect more than one TCP/IP instance to OpenEdition MVS, you need to configure the OpenEdition MVS to use Common I-Net (C-INET) AF_INET PFS, instead of integrated sockets AF_INET PFS, which does not support multiple TCP/IP instances connected to OpenEdition MVS.

When you have DB2 using TCP/IP, however, you can have only one TCP/IP instance connect to OpenEdition MVS on the same MVS system, because DB2 uses asynchronous I/O services, and C-INET AF_INET PFS does not support asynchronous I/O. Therefore, the integrated sockets AF_INET PFS must be used. In other words, you must configure OpenEdition MVS to use the integrated sockets AF_INET PFS when using DB2 with TCP/IP. Refer to the *TCP/IP V3R1 for MVS: Customization and Administration Guide*, SC31-7134, for further consideration of multiple copies of TCP/IP.

3.3.1.1 Verifying the TCP/IP on OS/390

Normally, the name of the transport providers implemented as started tasks are TCPIPMVS (the default transport provider for traditional MVS process) and TCPIPOE (the default transport provider for OpenEdition MVS process). Contact your system administrator to know the implemented transport provided on your system. For example, you can see the started tasks for the TCP/IP on the **System Display and Search Facility (SDSF)**.

Type `DA` on the `COMMAND INPUT` line as shown below.

```
Display Filter View Print Options Help
-----
HOX1900----- SDSF PRIMARY OPTION MENU -- INVALID COMMAND
COMMAND INPUT ==> DA                                SCROLL ==> CSR

LOG          - Display the system log
DA           - Display active users in the sysplex
I           - Display jobs in the JES2 input queue
O           - Display jobs in the JES2 output queue
H           - Display jobs in the JES2 held output queue
ST          - Display status of jobs in the JES2 queues
PR          - Display JES2 printers on this system
INIT        - Display JES2 initiators on this system
MAS         - Display JES2 members in the MAS
LINE        - Display JES2 lines on this system
NODE        - Display JES2 nodes on this system
SO          - Display JES2 spool offload for this system

Licensed Materials - Property of IBM

5647-A01 (C) Copyright IBM Corp. 1981, 1997. All rights reserved.
US Government Users Restricted Rights - Use, duplication or
F1=HELP     F2=SPLIT     F3=END      F4=RETURN   F5=IFIND    F6=BOOK
F7=UP       F8=DOWN      F9=SWAP    F10=LEFT   F11=RIGHT   F12=RETRIEVE
```

The following screen lists all the currently active jobs on the system.

```

Display Filter View Print Options Help
-----
SDSF DA SC53 SC53 PAG 0 SIO 0 CPU 12/ 6 LINE 69-79 (79)
COMMAND INPUT ==>> SCROLL ==>> CSR
NP JOBNAME STEPNAME PROCSTEP JOBID OWNER C POS DP REAL PAGING SIO
DBC2SPAS DBC2SPAS IEFPROC STC23276 STC NS FE 730 0.00 0.00
DB2RES4 IKJACCNT SCGPVM14 TSU24491 DB2RES4 IN F9 845 0.00 0.00
TCPIPOE TCPIPOE TCPIP STC24509 TCPIPOE NS FE 1793 0.00 0.00
KARRAS IKJACCNT SCGSA065 TSU24503 KARRAS LO FF 775 0.00 0.00
KMT2 IKJACCNT TCP66006 TSU24500 KMT2 LO FF 702 0.00 0.00
DB2IMSTR DB2IMSTR IEFPROC STC24517 STC NS FE 807 0.00 0.00
MVSNFSC5 MVSNFSC5 MVSCINT STC22804 STC NS FE 5398 0.00 0.00
DB2IDIST DB2IDIST IEFPROC STC24521 STC NS FE 1878 0.00 0.00
DB2IDBM1 DB2IDBM1 IEFPROC STC24520 STC NS FE 4721 0.00 0.00
IRLIPROC IRLIPROC STC24518 STC NS FE 216 0.00 0.00
TCPIPMVS TCPIPMVS TCPIP STC24508 TCPIPOE NS FE 2287 0.00 0.00

F1=HELP F2=SPLIT F3=END F4=RETURN F5=IFIND F6=BOOK
F7=UP F8=DOWN F9=SWAP F10=LEFT F11=RIGHT F12=RETRIEVE

```

You can see the multiple transport driver support TCPIP environment (**TCPIPOE**, **TCPIPMVS**) on the sample screen above.

After completing the steps above, you should verify your system's host name and address configuration for TCP/IP OpenEdition (**TCPIPOE**) as follows:

The status of a TCP/IP transport provider is monitored with the TSO NETSTAT command. You can specify the name of the transport providers started task to be monitored if you have multiple transport providers installed. You can check your home IP address from the TSO command panel by typing NETSTAT HOME TCP TCPIPOE, which is for TCP/IP OpenEdition.

```
Menu List Mode Functions Utilities Help

                                ISPF Command Shell
Enter TSO or Workstation commands below:

====> NETSTAT HOME TCP TCPIPOE

Place cursor on choice and press enter to Retrieve command

=>
=>
=>
=>
=>
=>
=>
=>
=>
=>

F1=Help      F3=Exit      F10=Actions  F12=Cancel
```

The following screen shows the result of NETSTAT HOME TCP TCPIPOE command.

```
MVS TCP/IP NETSTAT CS/390 V2R5          TCPIP NAME: TCPIPOE          19:32:24
Home address list:
Address          Link          Flg
-----          ----          ---
9.9.9.18        OSAL2160      P
127.0.0.1       LOOPBACK
***
```

Alternatively you can check for the host name and IP address on the **OpenMVS ISPF Shell** panel.


```

File Directory Special_file Tools File_systems Options Setup Help
-----
                                OpenMVS ISPF Shell

Enter a pathname and do one of these:

- Press Enter.
- Select an action bar choice.
- Specify an action code or command on the command line.

Return to this panel to work with a different pathname.
                                                    More:      +

=====
=====
=====

Command ==> sh cat /etc/hosts
F1=Help      F3=Exit      F5=Retrieve  F6=Keyshelp F7=Backward F8=Forward
F10=Actions  F11=Command F12=Cancel

```

Type a shell command `sh cat /etc/hosts` or `ex cat /etc/hosts` to view the hosts list. The following screen shows the result of the `sh cat /etc/hosts` command.

```

BROWSE -- /tmp/DB2RES4.15:00:01.121320.ishell ----- Line 00000000 Col 001 080
Command ==>                                         Scroll ==> PAGE
***** Top of Data *****
-----
Set up environment variables for Java and Servlets for OS/390 -
-----
PATH reset to /usr/lpp/java/J1.1/bin:/bin:.
-----
CLASSPATH reset to ./usr/lpp/java/J1.1/lib/classes.zip:/usr/lpp/internet/server
-----
--> Path set for JAVA Servlet support
--> LD PATH set for JDBC support
9.9.9.18 wtsc53oe wtsc53oe.itso.ibm.com
***** Bottom of Data *****

F1=HELP      F2=SPLIT      F3=END      F4=RETURN      F5=RFIND      F6=RCHANGE
F7=UP        F8=DOWN       F9=SWAP     F10=LEFT       F11=RIGHT     F12=RETRIEVE

```

3.3.2 AS/400

Before trying to connect to your AS/400 DB2 database, check your server configuration by typing `CHGTCPDMN` and pressing **F4**.

```
Change TCP/IP Domain (CHGTCPDMN)

Type choices, press Enter.

Host name . . . . . 'AS400'

Domain name . . . . . 'ibm.com'

Host name search priority . . . *REMOTE      *REMOTE, *LOCAL, *SAME
Domain name server:
  Internet address . . . . . '127.0.0.1'
                          '9.9.9.9'

Bottom
F3=Exit  F4=Prompt  F5=Refresh  F10=Additional parameters  F12=Cancel
F13=How to use this display  F24=More keys
```

If there is no configuration information, contact your system administrator. Check your TCP/IP setup by trying to ping your server from one of the supported clients. In the case shown above, you would try ping `as400.ibm.com`.

The RPC server is also needed for client/server communication. To start it you need `*IOSYSCFG` authority. You can check for this authorization by using the `WRKUSRPRF` command. Select your own username, choose option 5 and check the second page of the profile settings under special authority. Also, check if your username is listed in the system distribution directory with `WRKDIRES`. You can add your profile to this directory with `ADDDIRE` if you are not listed. Start the RPC server with `STRNFSSVR *RPC`.

In addition to the TCP/IP protocol itself, you should check your system configuration with `DSPNETA` to see if the database has been configured properly. The entries listed as current system name, local control point name, and default local location must be the same. In the example below, this name is `AS400`.

```

Display Network Attributes
                                     System:  AS01
Current system name . . . . . : AS400
Pending system name . . . . . :
Local network ID . . . . . : ITSCNET
Local control point name . . . . . : AS400
Default local location . . . . . : AS400
Default mode . . . . . : BLANK
APPN node type . . . . . : *NETNODE
Data compression . . . . . : *NONE
Intermediate data compression . . . . . : *NONE
Maximum number of intermediate sessions . . . . . : 200
Route addition resistance . . . . . : 128
Server network ID/control point name . . . . . : *LCLNETID  *ANY

                                     More...

Press Enter to continue.

F3=Exit  F12=Cancel

```

To check if local databases are configured for access as relational databases, use the WRKRDBDIRE command.

```

Work with Relational Database Directory Entries

Position to . . . . .

Type options, press Enter.
  1=Add  2=Change  4=Remove  5=Display details  6=Print details

      Relational      Remote
Option Database      Location      Text

      AS400          *LOCAL

                                     Bottom

F3=Exit  F5=Refresh  F6=Print list  F12=Cancel
(C) COPYRIGHT IBM CORP. 1980, 1998.

```

There must be an entry with the same name as the current system name from the previous screen with *LOCAL listed as remote location.

3.3.3 AIX

Before you can access a DB2 UDB database at the AIX system, TCP/IP must be configured to allow the QMF client to connect to the AIX system. To check the configuration:

Enter `smit` on the AIX command line. Following the selections shown below leads to a screen like the one shown in Figure 18.

System Management -> Communication Applications and Services -> TCP/IP

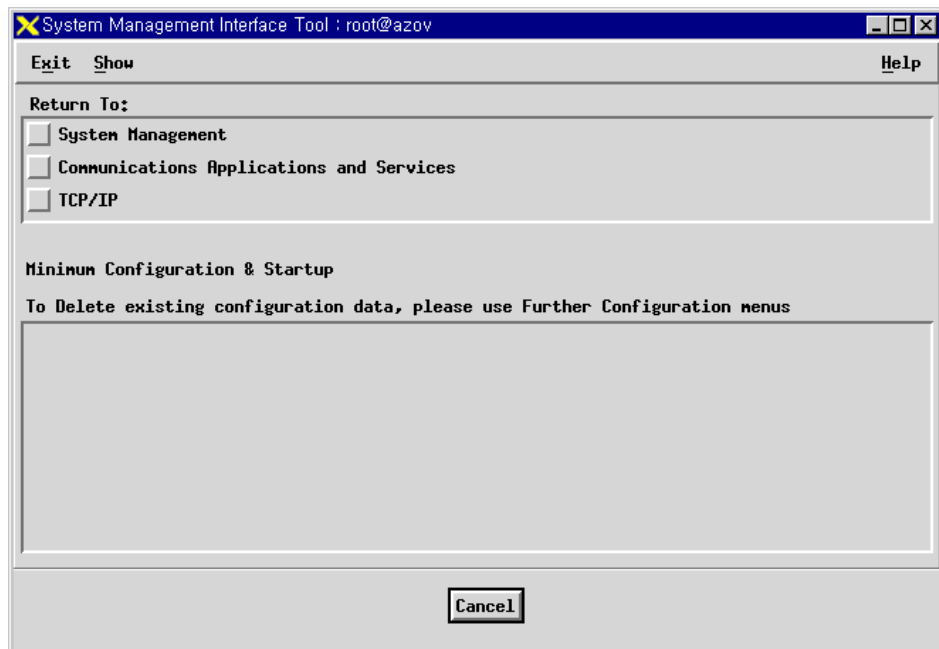


Figure 18. AIX TCP/IP SMIT panel

Select the defined network interface (Figure 19). If you don't know the network interface to be used, you need to ask your network administrator.

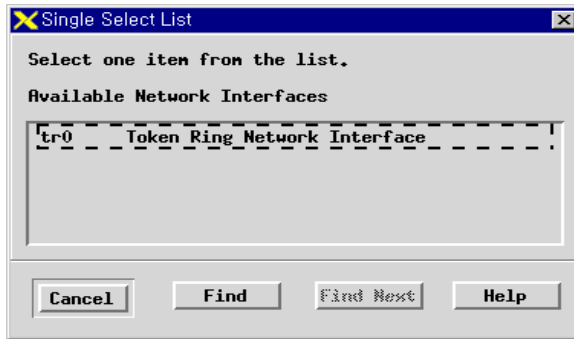


Figure 19. Defined network interface on AIX

Check for the host name, domain name, and so on, as shown in Figure 20. You will need to know the hostname or IP address in order to configure the connectivity on the IM client.

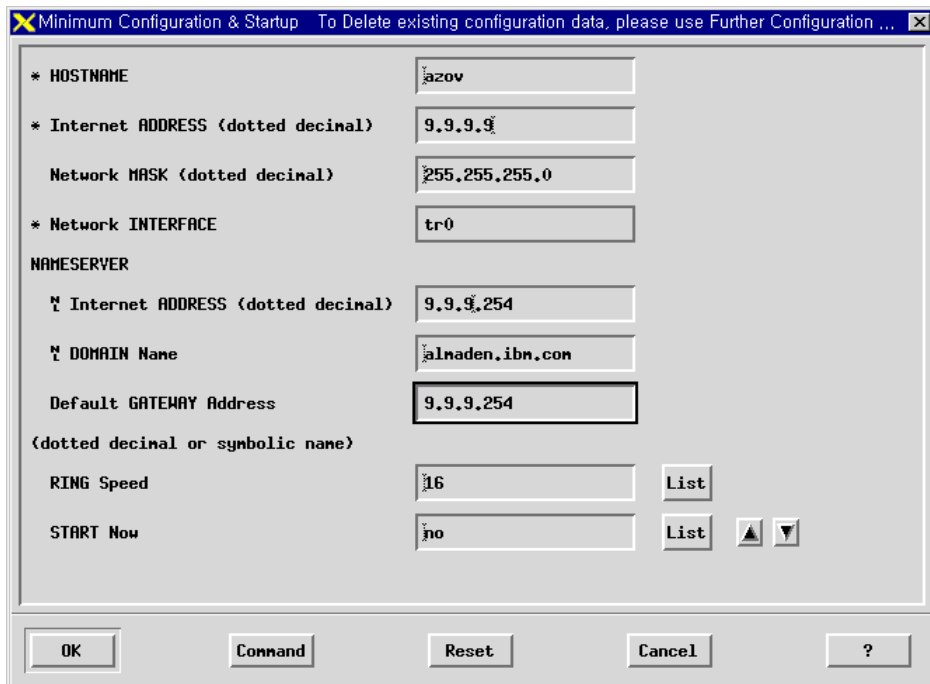
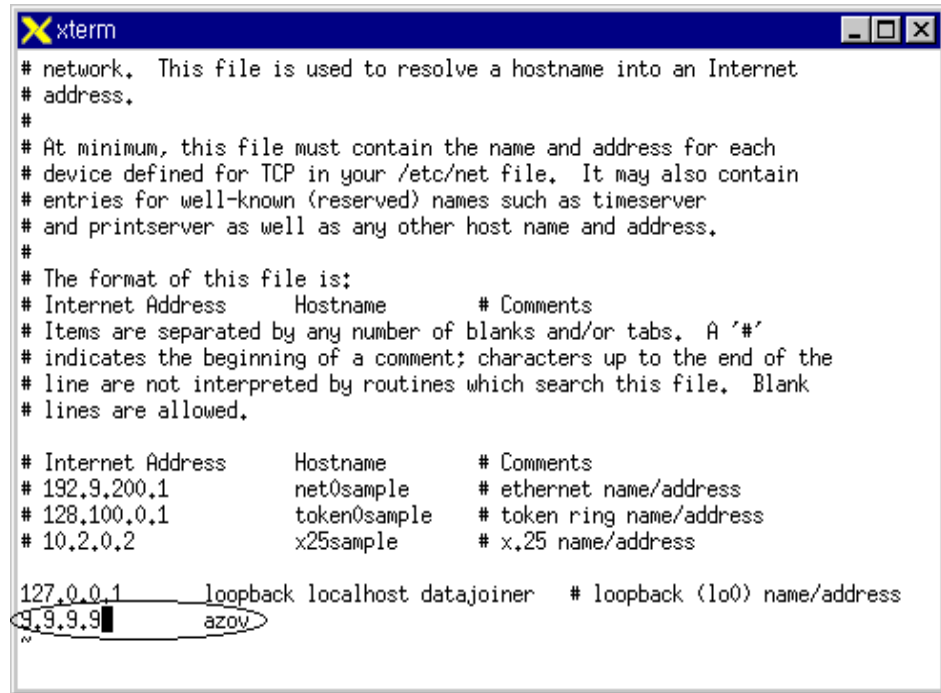


Figure 20. AIX minimum configuration

Once this is done, you may want to check the TCP/IP status on the IM server system.

You can check all TCP/IP related subsystems with the `lssrc -g tcpip` command on the AIX command line. Be sure that the `inetd` daemon is active. You can also check for `inetd` only with `lssrc -s inetd` command.

Add the dotted decimal address and host name into the `/etc/hosts` file of the IM client system (see Figure 21).



```
xterm
# network. This file is used to resolve a hostname into an Internet
# address.
#
# At minimum, this file must contain the name and address for each
# device defined for TCP in your /etc/net file. It may also contain
# entries for well-known (reserved) names such as timeserver
# and printserver as well as any other host name and address.
#
# The format of this file is:
# Internet Address      Hostname      # Comments
# Items are separated by any number of blanks and/or tabs. A '#'
# indicates the beginning of a comment; characters up to the end of the
# line are not interpreted by routines which search this file. Blank
# lines are allowed.
#
# Internet Address      Hostname      # Comments
# 192.9.200.1          net0sample    # ethernet name/address
# 128.100.0.1          token0sample  # token ring name/address
# 10.2.0.2              x25sample     # x.25 name/address
#
127.0.0.1              loopback localhost datajoiner # loopback (lo0) name/address
9.9.9.9                azov
```

Figure 21. AIX hosts file

Verify the TCP/IP connection from the client.

You can check the connection from the client to the server by typing `PING HOSTNAME` on the command line.

3.3.4 Windows

We describe all actions in this section using Windows NT as a sample system. Using Windows 95 as the operating system for the IM client might have different screen names, but is basically the same procedure as shown here for Windows NT.

To check if TCP/IP is configured on your system, double-click the **Network** icon in the NT **Control Panel**. Select the **Protocol** tab, click **TCP/IP** and then **Properties**, as shown in Figure 22. The window must show an adapter and either have DHCP selected or an IP address specified.

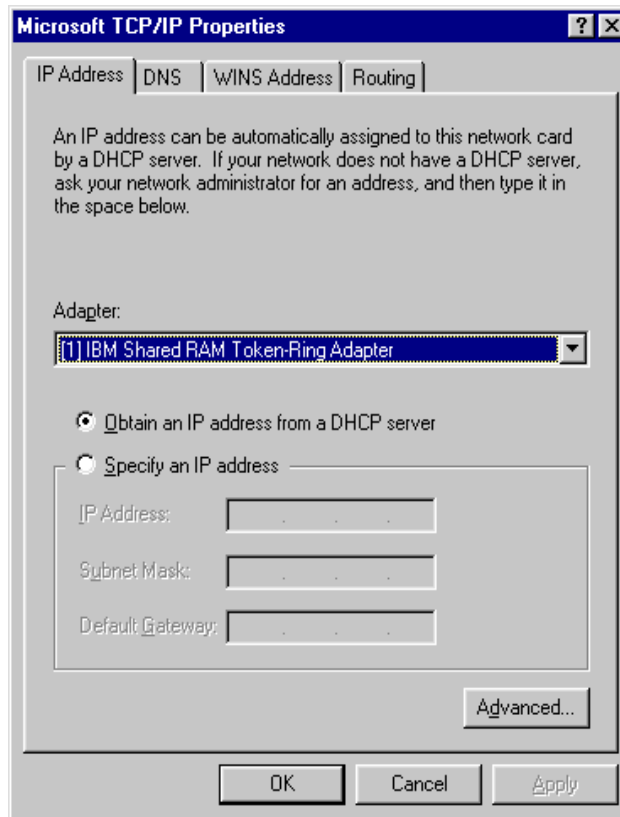


Figure 22. Windows NT TCP/IP properties

Try to find out if you can reach the server machine through the TCP/IP protocol from the machine you intend to use as the client. To test this, open an NT Command Prompt window and type `ping hostname` for the desired hostname.

If this fails, try to find out the IP address of the host. Unless your system is configured using DHCP, you may find an entry in your local hosts file. Try to ping the address instead of the hostname. If this also fails, your network setup is incorrect and you should contact your network administrator.

Otherwise, you need to edit the file that is used to translate hostnames into IP addresses. This file, called HOSTS, is located in the directory WINNT\SYSTEM32\DRIVERS\ETC (normally found on drive C:). Add a line to that file with any text editor as shown below for `examplehost` with address 9.9.9.9.

```
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows NT.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#       102.54.94.97       rhino.acme.com           # source server
#       38.25.63.10      x.acme.com             # x client host

127.0.0.1       localhost
9.9.9.9         examplehost
```

Never change the entry or address for `localhost`. Test your new setup by saving your hosts file and then `ping examplehost`. This should now work.

You can use DHCP on your clients without any problems. If you need to use DHCP on your IM server, the server's hostname must be registered with a DNS server, because you cannot specify a fixed IP address in the client's hosts file.

3.4 SNA basics

SNA is a communications architecture that was originally designed to address the increasing complexity of data processing and communications needs as technology advanced. SNA is not a product, it is an architecture that provides a blueprint or specification of how diverse products can connect and communicate with each other. It defines a set of rules that describe the transport of data and specify, for example, what the data looks like, how it is to be packaged, and who the sender and receiver are.

3.4.1 SNA layers

SNA is a structured architecture that consists of seven well-defined layers, each of which performs a specific network function. As a result of this structure, SNA products have compatible interfaces that facilitate interconnection and communication. Their new functions and technologies can be added to the network with no disruption to the flow of data. The layers defined in SNA are:

- Transaction services
Provides application services, such as distributed database access or distributed file management.
- Presentation services
In general, formats the data for different media and sharing of devices.
- Data flow control
Synchronizes the flow of data within a network, correlates data exchanges, and groups related data into units.
- Transmission control
Regulates the rate at which data is exchanged to fit within the processing capacity of the participants to avoid congestion in the transport network. This layer is also responsible for encryption, if required.
- Path control
Routes data between source and destination and controls data traffic within the network.
- Data link control
Transmits data between adjacent network nodes.
- Physical control
Connects adjacent nodes physically and electrically and is the hardware part of the connection.

SNA provides function subsetting at a logical unit (LU) level. Thus, the architecture provides standard option sets, in the form of profiles, which are defined in the transmission control, data flow control, and presentation services levels. A specific LU supports a subset of architected profiles that reduces the number of options specific network products have to implement. End users (for example, terminal users, applications) access the network through the LU component of a network product. The LU manages the exchange of data and acts as the intermediary between the end user and the network.

Network products only need to implement the rules or protocols that support the type of communication appropriate to the product. Therefore, printers implement printer protocols, (for example, LU1), and display devices implement LU2.

With the advent of personal computers, programmable workstations, and LANs, SNA has led the move toward peer-to-peer networking. Within SNA, the LU 6.2 protocol is the defined standard for communications between functionally equivalent LUs. It provides a standard set of formats and communication rules that allow programs to directly communicate across multiple hardware and software environments.

3.4.2 APPC basics and terminology

LU 6.2 describes the standard functions that programs can use to communicate with each other. The implementation of this protocol is called APPC. Sometimes, LU 6.2 and APPC are used synonymously.

3.4.2.1 The LU 6.2 protocol

The LU 6.2 protocol provides a consistent method for programs to:

- Identify and negotiate the communications options to be used by each partner program
- Provide the name of the partner destination and program
- Supply end-user security parameters to be associated with the request on the remote platform
- Control the transmission of messages
- Synchronize the processing between the partner programs
- Perform coordinated commit processing

The actual functions that can be used may vary from one APPC implementation to another. Programs communicate using options that are supported and agreed to by the APPC support on either side.

3.4.2.2 Logical units

The APPC LU implements the LU 6.2 protocol and provides the means by which transaction programs (TPs) access the network. The SNA software accepts requests from the TPs, executes them, and routes the data packet while hiding the physical details of the underlying network. The term *local* LU refers to the LU to which a TP issues its APPC calls. The Local LU then communicates with a *partner* LU on either the same or a different node.

3.4.2.3 Transaction programs

When an APPC application establishes a connection with another LU in the network, the application must identify the name of the TP it wants to execute. The TP name is the logical network name that is used by a program to communicate over the network, and can be up to 64 bytes long.

There are two basic types of TPs:

- Application TPs, which perform tasks for end users
- Service TPs, which perform tasks related to system services, such as changing passwords

3.4.2.4 Sessions

The logical connection between LUs is called a *session*. Sessions are established when one LU sends another LU the request to BIND. During this BIND process, both partners exchange information about the characteristics of the connection to be established. The LU that starts the session, the initiator, is known as the primary LU, and the recipient of the session is known as the secondary LU.

LUs have either single-session capability, that is, they allow only one active LU-to-LU session with a partner LU at a time; or they have parallel-session capability. With parallel-session support it is possible to have more than one session with the same partner LU at the same time. Most LUs have parallel-session capability.

3.4.2.5 Conversations

To avoid the overhead of session setup every time two LUs want to communicate, another concept, that of a conversation, has been defined. A conversation is the logical connection between TPs and is carried out over a session. Conversations serially reuse a session to exchange end user information. When a conversation ends, another conversation can be allocated and use the same session. Sessions are typically long-lived links, whereas conversations exist for the duration of the exchange of information, for example, the time it takes to process a transaction.

The basic elements of a conversation are:

- **Starting a conversation.** When a program wants to start an LU 6.2 conversation with another program, it issues an allocate request. The ALLOCATE identifies the partner and requests a connection. The APPC LU makes the connection, if possible, and attempts to reuse a session that was previously established. If none exists, it creates a session. The partner can accept or reject the conversation request. If the request is

accepted and the conversation established, the caller is put into send state, the partner is put into receive state, and information can now be exchanged between the two.

- **Exchange of information.** The program that is in send state sends information to its partner, using the SEND_DATA verb. This action causes the data to be put into the LU's buffer (the LU that is local to the program). The LU does not actually send the data until either the buffer is full (max RU for the session on which the conversation is allocated) or the program explicitly issues a verb (that is, FLUSH, CONFIRM, or one that changes state) that explicitly causes the LU to transmit the buffered data. The partner LU accepts the data and buffers it. The TP gets the data and/or status indicators as a result of issuing a verb such as RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE.
- **Requesting confirmation.** TPs can optionally request synchronization of communications by requesting and granting confirmations, so that they can determine whether their partners have successfully received data that they have sent. Requesting confirmation allows programs to agree that processing completed without error. The level of synchronization (in this case, CONFIRM) is specified during allocation with the SYNC_LEVEL parameter. When a TP issues the CONFIRM verb after a SEND_DATA, the TP actually waits (is suspended) until it receives the response from the partner. The partner responds that it has either received the data (CONFIRMED) or that an error has occurred.
- **Sending an error notification.** When an error occurs, either partner may inform the other of the condition by issuing a SEND_ERROR or DEALLOCATE_ABEND.
- **Ending conversations.** A conversation is ended by issuing a DEALLOCATE verb. The partner receives both a deallocate indicator and any remaining data and then may be asked to confirm the deallocation or just terminate. The partner is put into reset state and completes its own processing. When a conversation is ended, the session is then available for use by another conversation.

When TPs communicate over an LU 6.2 conversation, there are a few areas that they both agree on for compatibility.

Synchronization level provides a means by which two TPs (if they choose to do so) reach a consistent state of processing with respect to the data exchanged. The programs on either end synchronize their actions by requesting and granting confirmations. Confirmation could be used, for example, when one program wants to ensure that its partner received the

data it sent before deleting the source of the transmitted data. There are three possible SYNC_LEVELS.

- **NONE** — Specifies no synchronization.
- **CONFIRM** — Allows a TP to request specific acknowledgment from a partner that it has received a message. This SYNC_LEVEL enforces confirmation exchanges and error reporting. There are two flavors of confirmation. The first is one wherein a TP issues a CONFIRM verb, to which the partner replies either positively with CONFIRMED, or reports an error condition with SEND_ERROR. The second one is where a TP requests confirmation from its partner by issuing a verb such as DEALLOCATE with type=SYNC_LEVEL.
- **SYNCPOINT** — Allows all transaction programs in a distributed environment to commit or back out changes to protected resources (that is, databases). The LU takes responsibility for syncpoint processing.

A synchronous conversation is characterized by a transaction program issuing an ALLOCATE of a conversation, a SEND of data, and a RECEIVE for the reply. When the partner detects that the input message is complete, that is, the partner program enters the RECEIVE state, the conversation is considered to be synchronous. Replies are sent back on the same conversation.

An asynchronous conversation is characterized by a transaction program issuing an ALLOCATE of a conversation, a SEND of data, and a DEALLOCATE. This conversation allows the originating TP to do other work while the partner processes the request. The conversation is in RESET state. If an output reply is to be sent back, a new conversation to send the reply is allocated.

The choice of a MAPPED or BASIC conversation type affects the format of data transmission. An APPC logical record is a sequence of length "LL" (2 bytes) and "data" fields. A typical pattern is "LLdataLLdata..." where the LL fields define how much data follows before the next length field. The choices for preparing the data for transmission in this manner are:

- **MAPPED** — Lets APPC format the data into and out of this pattern for each transmission. The handling of the details of the underlying data stream is the responsibility of the APPC LU and not the application. Thus, applications are easier to code because they only need to prepare the data in the format that the partner expects.
- **BASIC** — Requires the TPs to be responsible for formatting the data and including the LL field. This puts the burden on the TP rather than the APPC LU.

3.4.2.6 Common Programming Interface for Communications

Because the LU 6.2 architecture provides freedom of syntax, products such as APPC/MVS, OS/400, and OS/2 have the option of creating unique implementations of the APPC API as long as they adhere to the semantics (services) as defined by the architecture. As a result of this freedom, the different products have in fact done so. This makes it very complicated for APPC programmers who have to program in different environments, because they are faced with having to learn the unique API appropriate to an implementation. For example, to issue an allocate verb in the APPC/MVS environment, a programmer codes ATBALLC; in an OS/2 environment, a programmer would use the MC_ALLOCATE verb, and in an OS/400 environment, a programmer would use the ACQUIRE and EVOKE verbs.

To alleviate this problem, a standard and consistent API has been defined for applications that need to communicate in the APPC arena. The Common Programming Interface for Communications (CPI-C) is common across multiple environments and differs from the product-specific APIs in that it provides an exact common syntax to specify the function calls and parameters. CPI-C is designed to eliminate the mismatch of the verb specifications in the different environments. This capability allows an application programmer to learn the syntax on one platform and port the knowledge or even part of the program to another supporting platform. Application programs can use CPIC calls in the major languages such as COBOL, C, FORTRAN, and PL/I. Pseudonym files are provided in the different operating environments (in MVS, check SYS1.SAMPLIB). The files can be "included" or "copied" into programs that call CPI-C.

It is up to an application programmer to decide whether to use CPI-C or the native interface unique to the specific APPC implementation.

Advanced Program to Program Communication (APPC) is the System Network Architecture (SNA) protocol, on which Distributed Relational Data Architecture (DRDA) is based. APPC is used for the communication between the host system and the workstation servers. The necessary definitions for the APPC protocol are much easier to handle using VTAM/Advanced Peer to Peer Networking (APPN) on the host side. That means some of the definitions are made automatically during startup of communication.

For the time being, TCP/IP is not supported by DB2 for VSE V5, one of the host data servers for this project, which just has support for APPC protocol.

To set up DRDA in VM, we need APPC/VM VTAM Support (AVS), which handles the communications between VM and non-VM systems in the network. DB2 communicates by using Inter User Communication Vehicle

(IUCV) with the AVS virtual machine. Coordinated Resource Recovery (CRR) provides the synchronization services for two-phase commit processing. A VM CMS application can act as a DRDA client to access any DRDA server.

On VSE, CICS handles the DRDA connectivity and also supports the two-phase commit processing. VSE itself uses Cross Partition Communication (XPCC) to talk to CICS.

APPC communication can be used for communication with CrossAccess Servers as well. CrossAccess is able to work as a VTAM application, not requiring use of the CICS. This allows the possibility of using native TCP/IP for VSE with this products. Details about this configuration will be described later in this book.

3.5 Configuring your SNA (LU 6.2, APPC, and CPI-C)

LU 6.2 is an SNA communications architecture. APPC (Advanced Program-to-Program Communication) is a language based on the LU 6.2 architecture. A developer of SNA transaction programs has to choose from many different implementations of APPC. This could potentially lead to product incompatibility. Even though each implementation of APPC adheres to the LU 6.2 architecture, two implementations of APPC might not be exactly the same. Therefore, programs that rely on one vendor's APPC implementation might not work with another vendor's implementation.

This problem is solved by a standard, common programming interface, CPI-C, which implements the APPC verb set. Therefore, applications that require the use of the APPC verb set can instead be written using CPI-C in order to achieve SNA vendor independence. QMF for windows is an application that is written using CPI-C.

In an SNA network, QMF for Windows' basic requirement is that it must be able to establish an LU 6.2 session with DB2, using the CPI-C interface. This connectivity is not provided with QMF for Windows; you must have a third party tool that implements it. Whatever product you use to provide SNA connectivity, it must be installed and configured before you proceed with installing or using QMF for Windows.

The process of implementing LU 6.2 connectivity between Microsoft Windows and DB2 can be a complex task, depending on your SNA environment. The Windows-based SNA products that can be used, and the different ways to use them, are far too numerous and complex to be described in detail in this redbook. You must rely on your in-house SNA networking staff and your SNA

software vendor's technical support services to implement and support your network configuration.

The first step is to identify the appropriate values for the following definitions in Table 1.

Table 1. SNA configuration parameters

Definition	Sample value	Your value
Network ID	USIBMSC	
Local node name	SC02242	
Local node ID	05D 02242	
Local LU	SC02242I	
Destination address	400022160011	
Adjacent CP name	SCG20	
Partner node name	SCPDB2X	
Partner network ID	USIBMSC	
Mode name	IBMRDB	

3.5.1 Windows NT

The following section guides you through the configuration of the CPI-C for a Windows client using IBM Personal Communications Version 4.3.

1. To start the configuration, select the Windows NT *Start* Menu. Click on **Programs**, then on **IBM Personal Communication**, and finally on **SNA Node Configuration**.
2. Pull down the File menu and select **New**.

If you already have a valid Personal Communication configuration file and you want to modify this, simply select your configuration file from the File menu instead.

The window as shown in Figure 23 will open, showing the possible APPC definitions required to define the CPI-C communication. In order to configure Personal Communications for CPI-C, the following parameters need to be configured:

- Configure Node
- Configure Devices
- Configure Connections
- Configure Partner LU 6.2

- Configure Modes
- Configure CPI-C Side Information

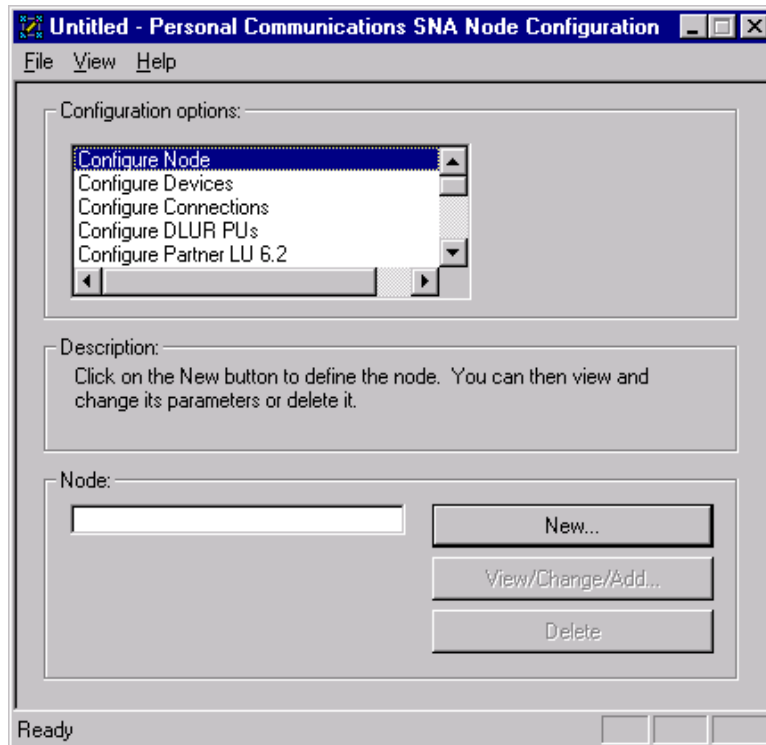


Figure 23. Personal communication - configuration window

3.5.1.1 Configure Node

In the main configuration window, select the **Configure Node** entry and click **New**. Enter the data in the window shown in Figure 24 and click **OK**. Leave the entries in the Advanced and DLU Requester with their default values.

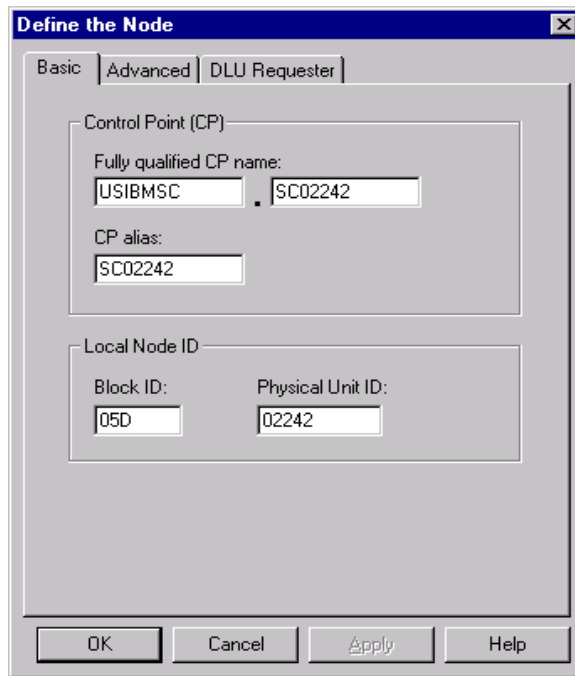


Figure 24. Configure node

3.5.1.2 Configure Devices

Select the **Configure Devices** and the Data Link Control (DLC) required and click **New**. All of the entries should be done automatically. Click **OK** to confirm the entries.

3.5.1.3 Configure Connections

In the main configuration window select the **Configure Connections** as well as the **DLC** defined in the previous step and again click **New**. The window shown in Figure 25 will appear.

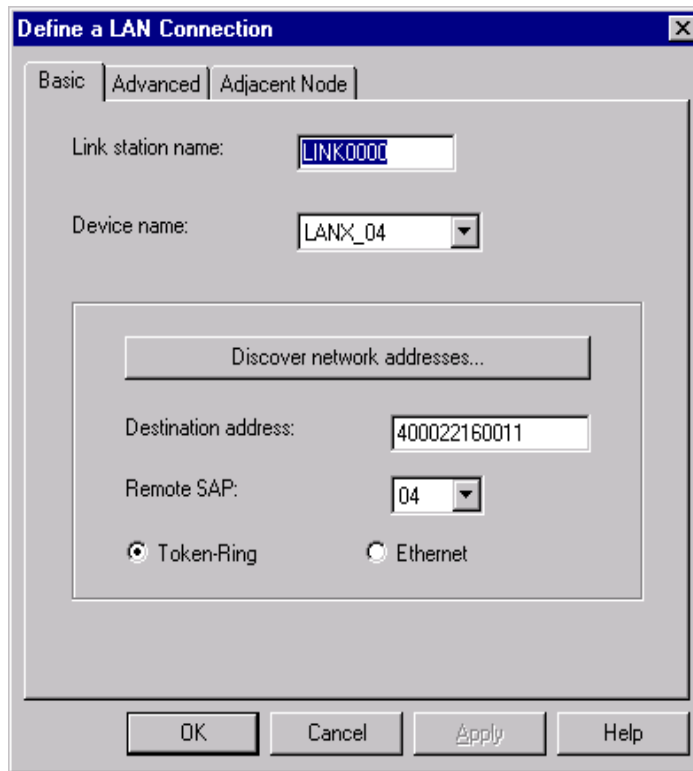


Figure 25. Configure connection

After all the entries are made, select the Advanced tab and make sure the following parameters are set to on (checked):

- Activate Link at Start
- Solicit SSCP Sessions

The entry for the **PU name** can be left with its default value, DIRPU000. Make sure that your local node's **Physical Unit ID** is entered correctly.

Go on to the Adjacent Node tab and enter the values as shown in Figure 26, then click **OK** to return to the main configuration window.

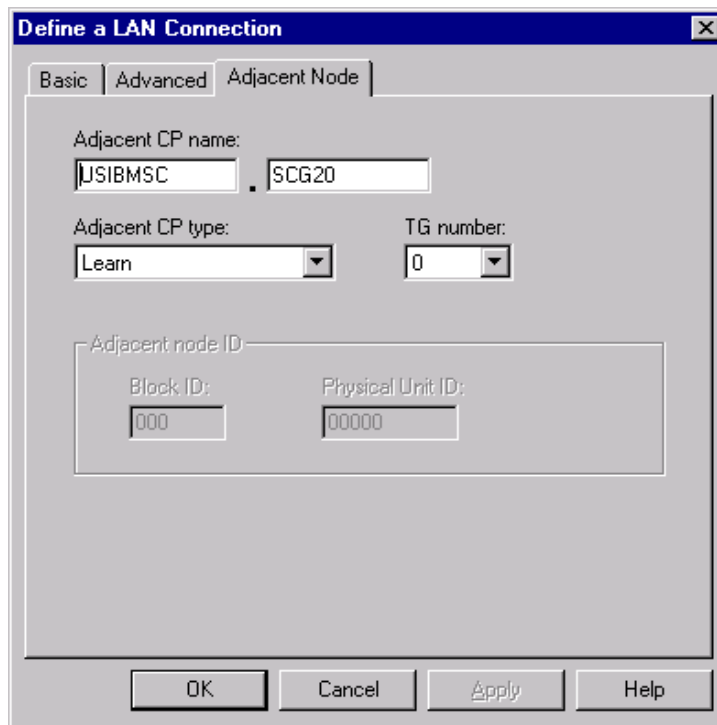


Figure 26. Adjacent node definition

3.5.1.4 Configure Partner LU 6.2

To configure the Partner LU 6.2, select the corresponding line in the configuration windows and click **New**. Enter all your data into the screen, as shown in Figure 27 and click **OK** to confirm the entries.

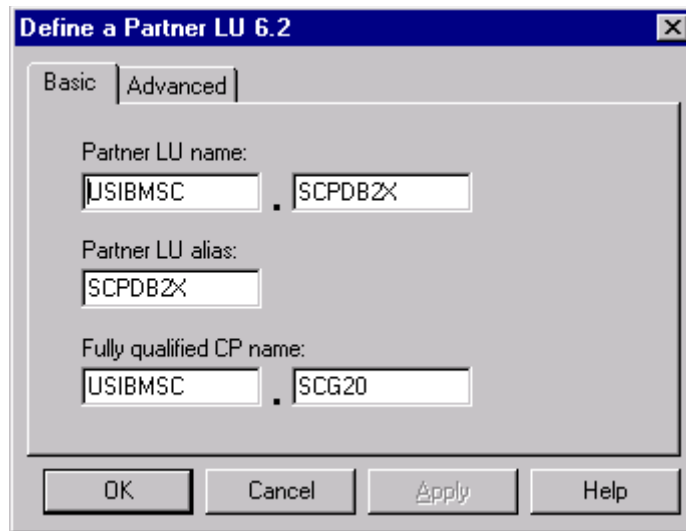


Figure 27. Partner LU 6.2

3.5.1.5 Configure Modes

To create a new Mode to be used with the CPII-C communication, select Configure Mode, click New and type the mode name into the entry field. All other entries might be kept with their defaults, if you create the mode named **IBMRDB** that is typically used when connecting to the DB2 system on the host. Click OK to return to the main configuration window.

3.5.1.6 Configure CPI-C Side Information

The last step is to finally create the definitions for the **CPI-C Side Information**. Figure 28 shows the entries to be made. The **Symbolic Destination Name** is the value that QMF for Windows requires when defining a server to be accessed using CPI-C.

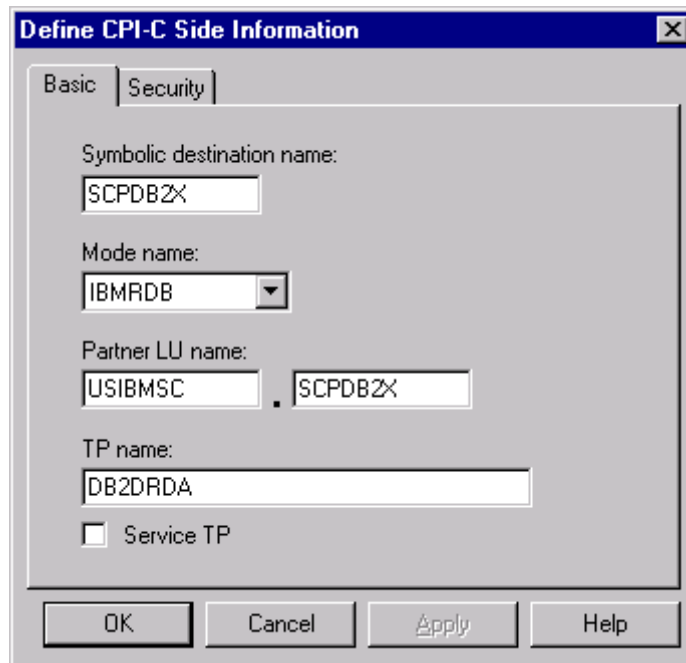


Figure 28. CPI-C side information

Clicking **OK** will confirm the parameters and the configuration file needs to be saved by selecting **Save As** from the File menu.

3.6 Data exchange protocols

This section discusses the different options and capabilities for accessing and transferring data between different systems. More detailed information about this topic can be found in *From Multiplatform Operational Data to Data Warehousing and Business Intelligence*, SG24-5174.

3.6.1 DRDA remote unit of work (RUW)

The remote unit of work, also referred to as DRDA level 1, offers remote data access from an *Application Requester* (AR) to an *Application Server* (AS).

RUW means that one *Database Management Subsystem* (DBMS) is accessed with one logical unit of work. After changed data has been committed, the application might switch to another DBMS, even on another platform.

The AR functionality is not only available on the workstation side; an application running on OS/390 or VM might access a workstation DBMS as well. The way to define the remote access is different on the different platforms. On OS/390 all definitions are held in DB2 tables, whereas the VM side requires some definitions on the operating system side as well. The PC systems (Windows and OS/2) offer a GUI for all required definitions.

3.6.2 DRDA distributed unit of work (DUW)

The distributed unit of work represents the DRDA level 2, giving access to more than one DBMS within one logical unit of work. That is, the application might switch to another DBMS before committing the data on all sites in parallel. A funds transfer from one DBMS to another is the best sample for such an application. If the update on one side fails, the changes on the other side are rolled back as well. So there is no risk of an inconsistency in between. The DUW requires a *Two Phase Commit* functionality on all participating DBMSs.

3.6.3 Distributed request (DR)

The two phase commit function is the base for the distributed request as well. DR means access to several DBMS within one *select* statement. With this functionality, a join of data from multiple systems is possible. The IBM DataJoiner (DJ) offers this function when providing several (remote) data sources as one single database image. The SELECT SQL statement (join of two or more tables) is issued against the DJ database, and the middleware splits it up to several DUW requests or transfers the data and performs the join locally.

3.6.4 Private protocols

Private protocols are used within the DB2 family for communication between the *Universal Database* (UDB) products on the workstation side. This private protocol provides same functionality as DRDA.

The SQLDS private protocol is used for communication between DB2 Server for VM and DB2 Server for VSE through *Guest Sharing* only. It provides no DRDA functionality. With the setting of the DB2 for VSE or VM start-up parameter PROTOCOL to either AUTO or SQLDS, you may decided whether the database server accepts a DRDA request or private protocol only.

3.6.5 Nonrelational access

For the access to nonrelational data from a "relational" requester, there are no specific protocols to be named; only products.

OS/390

A feature of DB2 DataJoiner called Classic Connect provides access to IMS and VSAM data on OS/390 in a relational view. A server application running on the host, maps the nonrelational data to the specified table image using *meta data*. This meta data is generated from the COBOL copybooks or the database descriptor block (DBD for IMS), which describe the format of the data records.

VSE/ESA

A similar function is provided for access to VSAM, DL/I, and sequential files on VSE by CrossAccess from CROSS ACCESS Corporation. The Server is running permanently on the host to be accessed either through TCP/IP or APPC. For every request, a Database Management System Interface (DMSI) is started in another (dynamic) partition. The DMSI accesses the nonrelational data, while the server maps it to the relational table image by using the meta data, as described for Classic Connect.

So far, the access to the nonrelational data is read-only. Joining of *tables* is possible for "like" data; that means, for example, two VSAM tables.

3.7 Connecting via Call Level Interface (CLI)

In order for QMF for Windows to connect to DB2 via CLI, you first use the DB2 UDB facilities to define your database servers and how to connect to them; this configuration is outside the control of QMF for Windows. The network configuration is performed as part of the DB2 UDB client configuration, rather than as a part of QMF for Windows configuration.

To access a DB2 UDB server using CLI, the 32-bit version of QMF for Windows must be able to establish a CLI connection from the local host (the system on which QMF for Windows is running) to the remote host (the system on which DB2 UDB is running) via the DB2 UDB client.

QMF for Windows requires the DB2 UDB client Version 5.2 or later to access the database via CLI and supports CLI connections only to the DB2 UDB database servers on the workstation environment, including DB2 DataJoiner.

3.8 Installing QMF for Windows

QMF for Windows is made up of three components:

1. Client (PC code that users use to connect to the DB2 server.)
2. Administrator tool (PC code that administrators use to configure QMF for Windows).
3. Host enabler (code that administrators upload to the database server via the administrator's tool, enabling the client to connect to DB2). This is more a post-installation task that needs to be performed by the database administrator or systems programmer. In this step the QMF objects (Tables and Views) are created on each server and the QMF packages are bound to the database.

When you insert the product CD-ROM you have the option of installing QMF for Windows only, or also install the QMF for Windows Administrator. The main installation process is very straight forward as long as the database administrator installs the product using the distribution media. But this requires the installation CD-ROM to be passed among the potential users and each user has to perform the installation by himself and in addition will install the *full version*, rather than the *thin client*. The following section describes this installation process and its options in more detail.

3.8.1 Advanced installation

In order to simplify the installation process, QMF for Windows allows the installation files to be placed on a central system and the users can start the installation process from there. This type of installation will also allow the administrator to set certain variables for the users in advance, thus simplifying the post installation process for the end users.

Preparing for the advanced installation will require a couple of steps to be performed and these steps are described in the following.

3.8.1.1 Copying disk images

Before you can perform an advanced installation, you must copy the disk images from the QMF for Windows CD. For information on locating the correct language and version (16- or 32-bit) of QMF for Windows, refer to the `readme.txt` file in the QMF for Windows CD root directory.

Using Windows Explorer or File Manager, copy each disk image directory to a centrally accessible computer.

3.8.1.2 Predefining user options

These are two common options that you can predefine for your users by editing the **rdbi.ini** or **qmfwin4.reg** file.

- Server definition file

The server definition file specifies the database servers that QMF for Windows uses. You must enter the name and the path of the server definition file to be used QMF for Windows.

- CPI-C provider DLL

The CPI-C provider DLL defines which SNA product you are using. If you are using a CLI or TCP/IP connection only, you do not need to declare this setting.

- Edit predefinition file for 16-bit installations:

User options are stored in the file named **rdbi.ini**. This file is on Disk1 of the QMF for Windows installation diskettes. The user options, found in the [Options] section of this file, are ServerDefinitionsFile and CPICDLL.

Example

```
[Options]
ServerDefinitionsFile=f:\windows\sdf.ini
CPICDLL=c:\windows\system\wincpic.dll
```

- Edit predefinition file for 32-bit installations

User options are stored in the *registry*. The registry is updated during QMF for Windows installation based on the settings in the file named **qmfwin4.reg**.

This file is on Disk1 of the QMF for Windows installation diskettes. The user options, in the following section of this file: [HKEY_CURRENT_USER\Software\IBM\RDB\Options], are ServerDefinitionsFile and CPICDLL. When you specify values for these options in this file, make sure that all backslashes (\) are doubled. For example, specify c:\\dir (not c:\dir). Also, make sure that both the option name and value are enclosed in double quotes.

Example

```
"ServerDefinitionsFile"="f:\\windows\\sdf.ini"
"CPICDLL"="c:\\windows\\system\\wincpic.dll"
```

3.8.1.3 Save options file on Disk1

After you edit and save the appropriate predefinition file, copy it to Disk1 of the QMF for Windows installation diskettes.

3.8.2 Unattended installation

An unattended installation allows you to select the installation options for your QMF for Windows users before beginning the installation process. The advantage of this method is that you can designate all the options of an installation rather than having to select the same options repeatedly for each installation. The options are defined when you edit **setup.ini**.

Using a simple text editor, edit the **setup.ini** file. This file, on Disk1 of the installation diskettes, controls the installation process and determines the settings used for the installation. The variables you can set are shown in Table 2.

Table 2. Installation parameters

Option	Settings	Explanation
AutoInstall =	0, 1	To enable unattended installation set to 1.
FileServerInstall =	0, 1	0 means all QMF files will be installed to the path specified in InstallPath. This results in a full installation. 1 means QMF is already installed on the fileserver in the path specified by InstallPath. This will create only the program group entries on the user desktop, resulting in a thin client installation. No code is transferred to the user system.
SetupType =	0, 1, 2	0=Typical, 1=Compact, 2=Custom (requires Components to be specified)
InstallPath =	<directory>	Directory where QMF will be installed (if FileServerInstall=1)
OverwriteINI =	0, 1	Valid only for 16-bit installations: Existing rdbi.ini will be overwritten if set to 1
ProgramGroup =	<name>	Name of the Program Group that will be created.

If the installation type Custom (SetupType=2) is specified, the following components have to be specified whether they have to be installed or not:

- Base= will install QMF for Windows
- Admin= will install QMF for Windows Administrator
- Excel= will install MS Excel Add-In
- 1-2-3= will install Lotus 1-2-3 Add-In

For each of those components, 1 means it will be installed, 0 means it will not be installed.

Example 1: Setup.ini

```
[Options]
AutoInstall=1
FileServerInstall=0
SetupType=2
InstallPath=C:\Programs\QMFWin
ProgramGroup=QMFWin
Base=1
Admin=0
Excel=0
1-2-3=1
```

This setup.ini file specifies an unattended installation. A custom installation is performed, installing the QMF for Windows product and the Lotus 1-2-3 Add In. The files will be copied to the C:\Programs\QMFWin directory, and a program group or program folder named **QMFWin** will be created.

Example 2: Setup.ini

```
[Options]
AutoInstall=1
FileServerInstall=1
SetupType=0
InstallPath=H:\QMFWin
ProgramGroup=QMFWin
```

This setup.ini file specifies an unattended installation. A typical installation is performed, but no files will be physically transferred to the client workstation, as QMF for Windows already is installed on a network drive H in the QMFWin directory. Only a program group or program folder named **QMFWin** will be created, resulting in a *thin client* installation.

Chapter 4. DBA's guide

This chapter provides a step-by-step guide to effectively configure and administer QMF for Windows, starting with the Administrator module. It also provides detailed examples regarding the execution of the DBA functions. The intended audience is the database administrator responsible for configuring and maintaining QMF for Windows.

4.1 Working with QMF for Windows Administrator

QMF for Windows Administrator is the administrative component of QMF for Windows and using this module is strictly an administrator task. There are four basic tasks that you perform with QMF for Windows Administrator:

- Defining and configuring the database servers that QMF for Windows will access.
- Creating QMF for Windows database objects, binding packages and granting permissions in each database server that QMF for Windows will access.
- Creating QMF for Windows sample tables in each database server that QMF will access.
- Administer QMF users, governing, and object tracking.

When you are using QMF for Windows Administrator, you are always editing a particular **server definition file** (SDF). The SDF contains all of the technical information needed by QMF for Windows to access any number of database servers. There are two ways (shown in Figure 29) to use server definition files:

1. You can allow each user to have his or her own SDF.
2. You can create a single SDF that is shared by multiple users over a file-sharing network. This approach has the advantage that it centralizes administration of the SDF; you only need to create and maintain a single file, and your users need only point to that file when they run QMF for Windows.

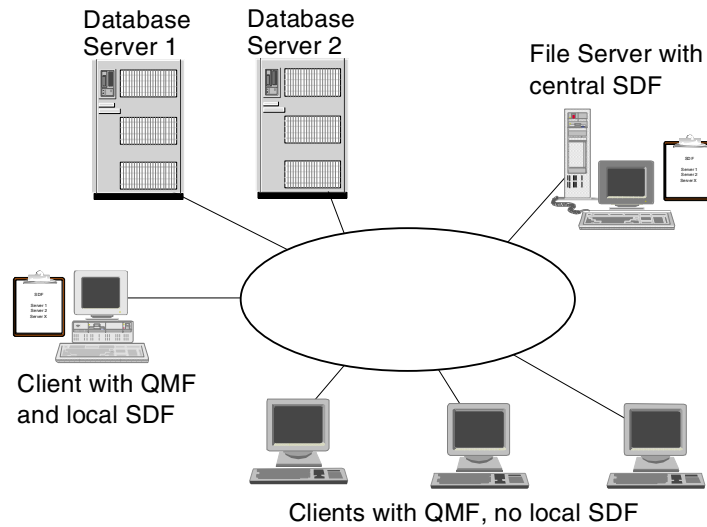


Figure 29. Server definition file

You can create a new SDF for a user or group of users by selecting the **New...** command or the **Save As...** command from the **File** menu. To open and work with a different SDF, select the **Open...** command from the **File** menu.

Example of SDF:

```
[Server Parameters]
Server1=QMF Demo
Server2=DB2NT
Server3=DB2AIX
[QMF Demo]
RDBName=SAMPLE
HostName=qmfdemo.rocketsoftware.com
Port=40000
SymDestName=*TCP/IP*
DecimalDelimiter=Period
StringDelimiter=Apostrophe
RDBI-CollectionID-0000000000000012=QMF611
QMFWin-CollectionID-0000000000000011=QMF611
DefaultSchedule1=Y00000078000003840009601710000012C0000000000000000
00A0000000007FDFFFE00002710XX
[DB2NT]
Timestamp=19990830205032
RDBName=SAMPLE
HostName=78-axfxb
```

```

Port=50000
SymDestName=*TCP/IP*
DriverName=
DefaultSchedule1=Y00000078000003840009601710000012C0000000000000000
00A0000000007FDFFFE00002710XX
DefaultSchedule2=@
0000000000000000
DefaultSchedule3=2
DefaultSchedule4=1
QueryBlockSize=32500
DecimalDelimiter=Period
StringDelimiter=Apostrophe
IsolationLevel=CursorStability
BindReplace=1
BindKeep=1
BindOwner=IMRES2
EnterpriseType=
DatabaseType=DB2
xSingleCCSID=1252
xMixedCCSID=0
xDoubleCCSID=0
xQMFCCSID=37
QMFWin-CollectionID-000000000000011=NULLID
QMFWin-CollectionID-000000000000014=NULLID
[DB2AIX]
Timestamp=19990831165455
RDBName=SAMPLE
HostName=AZOV
Port=60000
SymDestName=*TCP/IP*
DriverName=
DefaultSchedule1=Y00000078000003840009601710000012C0000000000000000
00A0000000007FDFFFE00002710XX
DefaultSchedule2=@
0000000000000000
DefaultSchedule3=2
DefaultSchedule4=1
QueryBlockSize=32500
DecimalDelimiter=Period
StringDelimiter=Apostrophe
IsolationLevel=CursorStability
BindReplace=1
BindKeep=1
BindOwner=db2inst1
EnterpriseType=
DatabaseType=DB2
xSingleCCSID=819

```

```
xMixedCCSID=0
xDoubleCCSID=0
xQMFCCSID=37
QMFWin-CollectionID-000000000000011=NULLID
QMFWin-CollectionID-000000000000014=NULLID
```

Note:

The server definition file is created and edited using the QMF for Windows Administrator application. Editing this file using any other method is not recommended, as it may corrupt the file.

4.1.1 Configure database connections

The primary function of QMF for Windows is to access data stored in any database in the DB2 family of databases. There are three ways in which QMF for Windows can connect to DB2:

- Using DRDA via TCP/IP
- Using DRDA via CPI-C
- Using CLI

Because QMF for Windows implements the **DRDA requester** specification, it is capable of connecting to any database that adheres to and implements the **DRDA server** component. The IBM database products that contain a DRDA server component and are capable of communicating directly with QMF for Windows are:

- DB2 UDB for OS/390, DB2 for OS/390, and DB2 for MVS
- DB2 Server for VSE&VM, and SQL/DS
- DB2 UDB for AS/400
- DB2 Universal Database and DB2 Common Server
- DB2 Parallel Edition
- DB2 DataJoiner

You use QMF for Windows Administrator to define each server, giving it a name and also specifying the technical information that QMF for Windows needs to access it. This process is analogous to defining a data source in ODBC. In order to define a new database server to QMF for Windows, from the Administrator main window you must click **New...** and enter all of the following required values on the **Server Parameters** dialog box:

- Specify a server name
- Specify an RDB name (not necessary for CLI connections)
- Specify the network connection

The following describes these steps in detail:

4.1.1.1 Server name

You must define each licensed database server that you or your users access with QMF for Windows. When you do so, you give each database server a **server name**. There are no restrictions on what this name can be; it is intended to be a descriptive, user-friendly label for the server, used only by the users in QMF for Windows and by the DBA in QMF for Windows Administrator. This name is all that the user of QMF for Windows needs to know in order to access that server; all of the technical details about how to access the server are hidden behind the server name in the SDF.

4.1.1.2 RDB name

The next step is to define the RDB name, also known as the *location name* in DB2 for OS/390 or MVS terminology, or simply the *database name* in DB2 Universal Database or DB2 Common Server technology.

Hint:

For DB2 OS/390 DBAs: If you are not sure of the value to enter here, there is an easy way to determine the correct value; with a tool other than QMF for Windows, run the following query at the server:

```
SELECT DISTINCT CURRENT SERVER FROM SYSIBM.SYSTABLES
```

The resulting value is the RDB name for the server.

4.1.1.3 Network connection

As mentioned previously, there are three different ways for QMF for Windows to connect to a database:

- DRDA via TCP/IP
- DRDA via CPI-C
- CLI

After selecting the radio button for the required connection, you need to enter the following values, depending on the selection made:

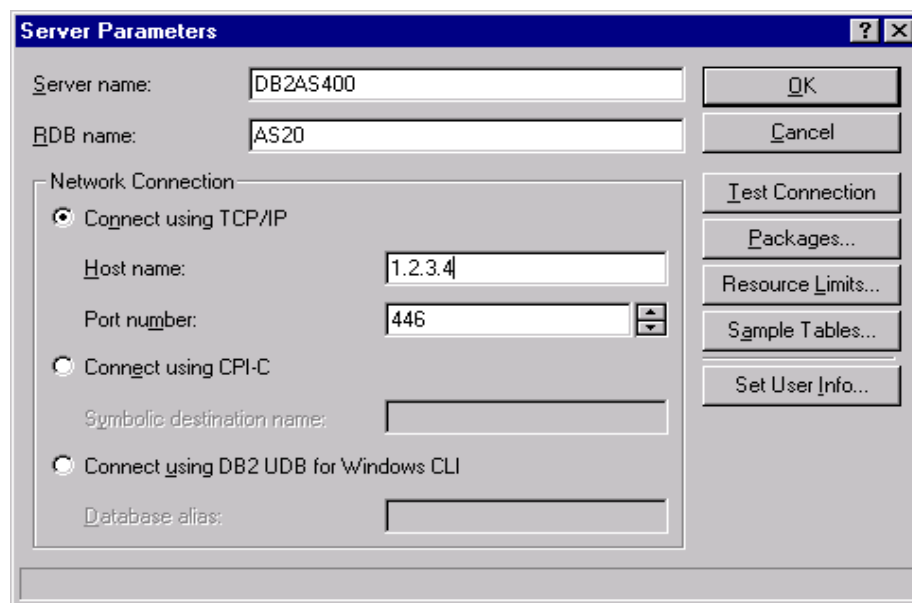
TCP/IP

Not all the DB2 host servers (DB2 UDB for OS/390, DB2 for OS/390, and DB2 for MVS) support TCP/IP connections. If you don't know if your DB2 OS/390 subsystem is configured to use TCP/IP, see the Sync Port and host name in DB2xMSTR. If DB2x is not configured to support TCP/IP, see the *DB2 Installation Guide* provided with your DB2 license for instructions on how to configure your subsystem.

Host name: If you enter a *TCP domain name* for the host name, QMF for Windows Administrator resolves that name to an address using the *GetHostByName* socket call. Alternatively, you can directly specify the host address in dotted decimal notation (for example, "1.2.3.4")

Port number: This is the TCP/IP port the database server is configured for with its listener port in the services file.

Figure 30 shows a sample Server Parameters screen for the connection definition to a DB2 for AS/400 database using DRDA via TCP/IP as the network connection.



The screenshot shows a dialog box titled "Server Parameters" with a standard Windows window control bar (minimize, maximize, close). The dialog is divided into several sections:

- Server name:** A text box containing "DB2AS400".
- RDB name:** A text box containing "AS20".
- Network Connection:** A section with a title bar and a group box containing:
 - Connect using TCP/IP**: This option is selected. Below it are:
 - Host name:** A text box containing "1.2.3.4".
 - Port number:** A spin box containing "446".
 - Connect using CPI-C**: This option is unselected. Below it is:
 - Symbolic destination name:** An empty text box.
 - Connect using DB2 UDB for Windows CLI**: This option is unselected. Below it is:
 - Database alias:** An empty text box.

- Buttons:** On the right side of the dialog, there is a vertical stack of buttons: "OK", "Cancel", "Test Connection", "Packages...", "Resource Limits...", "Sample Tables...", and "Set User Info...".

Figure 30. TCP/IP Connection definition

Note:

You can connect to the current version of DB2 DataJoiner via DRDA using only the SNA protocol at this time. The DRDA connection using the TCP/IP protocol is not supported by the current version of the DB2 DataJoiner.

CPI-C

Before configuring this connection, you must specify the SNA software that you are using to implement CPI-C in your Windows environment. This software has to be configured, and the **CPI-C symbolic destination name** for the database server has to be defined before the QMF for Windows installation. The CPI-C symbolic destination name is defined in your SNA software as described in detail in 3.5, “Configuring your SNA (LU 6.2, APPC, and CPI-C)” on page 55.

From the main window in QMF for Windows Administrator, select the **Options...** command from the **Edit** menu. In the Options dialog box in the CPI-C Options group, specify the name of the DLL that your SNA software provides for CPI-C applications as shown in Figure 31 on page 75. The name of the provider DLL typically is **wcpic32.dll**.

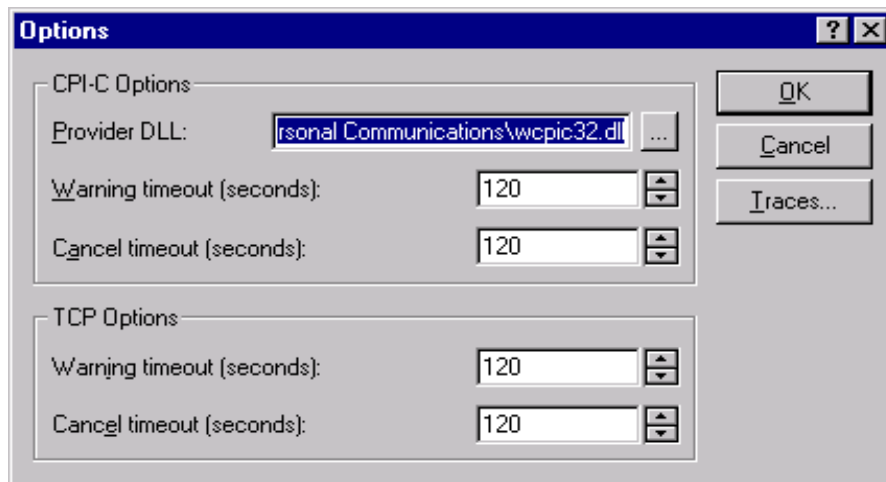


Figure 31. Provider DLL

Once the SNA software is configured, you can proceed with the QMF for Windows Administrator.

Using DRDA via CPI-C requires only one single entry in the QMF for the Windows Administrator screen, the *symbolic destination name*, as shown in Figure 32.

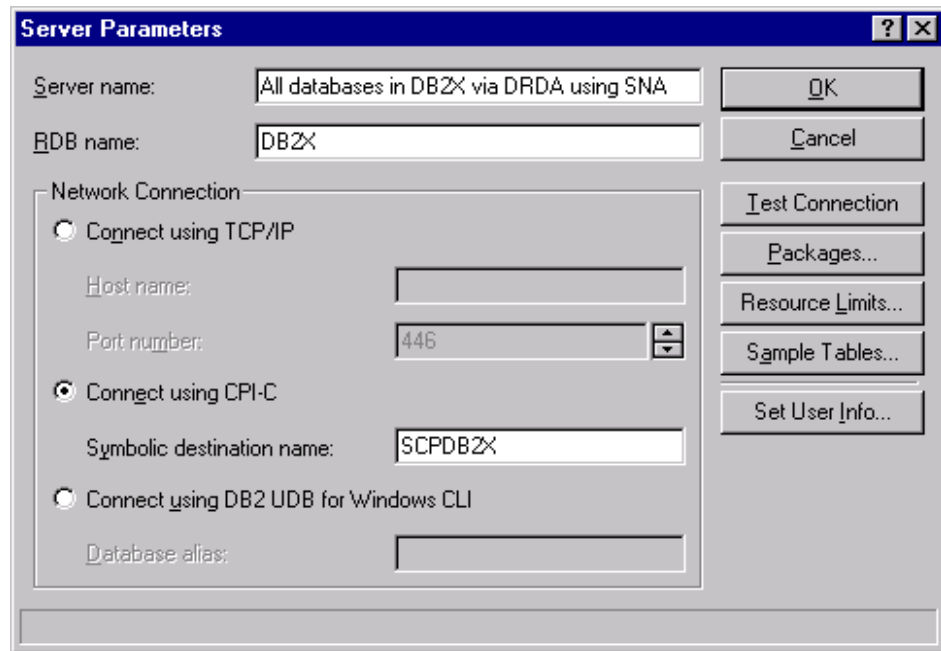


Figure 32. CPI-C connection definition

CLI

In order for QMF for Windows to connect to DB2 via CLI, you first use the DB2 UDB facilities to define your database servers and how to connect to them; this configuration is outside the control of QMF for Windows. Once this configuration is completed at the DB2 UDB client, in order to create the connection inside QMF for Windows Administrator you need only to specify the *alias* defined for a particular database.

The primary advantage of using CLI is simplified configuration: if a database is already defined in the DB2 UDB client, its alias is the only information needed in QMF for Windows. The network configuration is performed as part of the DB2 UDB client configuration, rather than as a part of QMF for Windows configuration.

The disadvantages of using CLI are:

- QMF for Windows performance using CLI and DB2 Connect is generally slower than using DRDA connection to connect to OS/390 servers directly.
- QMF for Windows only supports connecting to workstation databases with CLI: you must use DRDA connections to access any host DB2 database.
- Restrictions on using QMF for Windows via CLI to execute DB2 Stored Procedures that return multiple result sets.
- Restrictions on using QMF for Windows via CLI to bind static SQL.

Figure 33 shows a connection to an Oracle database using DB2 DataJoiner via CLI.

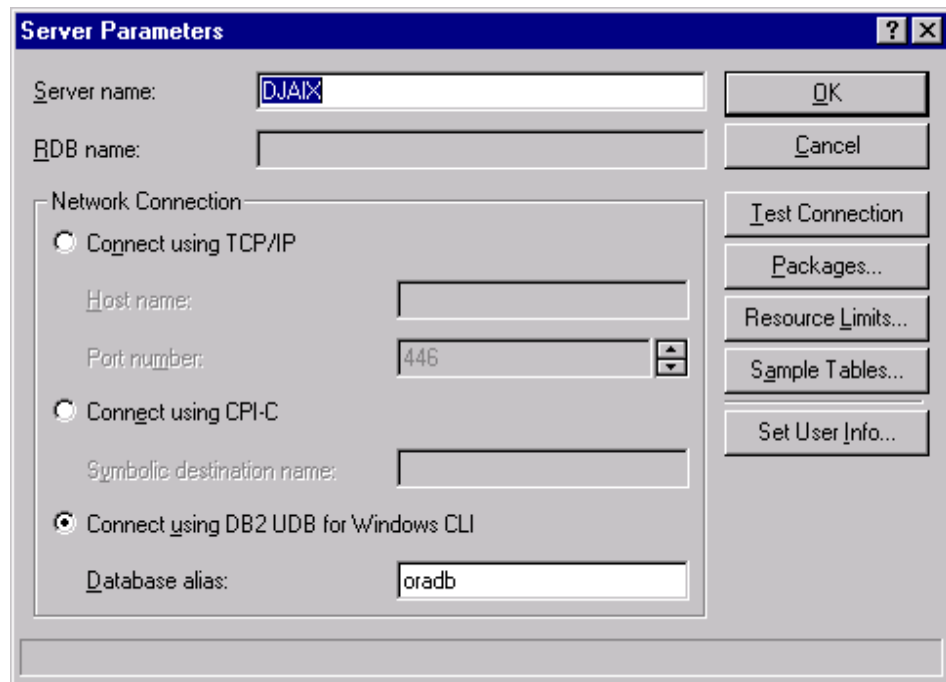


Figure 33. DB2 datajoiner using cli: server parameters

To access a DB2 UDB server using CLI, the 32-bit version of QMF for Windows must be able to establish a CLI connection from the local host (the system on which QMF for Windows is running) to the remote host (the system on which DB2 UDB is running) via the DB2 UDB client.

QMF for Windows requires the DB2 UDB client Version 5.2 or later to access the database via CLI and supports CLI connections to the following database servers: DB2 UDB, DB2 Parallel Edition, and DB2 DataJoiner.

To connect to DB2 for MVS, DB2 for OS/390 and DB2 UDB for OS/390 using CLI, you must have DB2 Connect installed locally or as a gateway. Although this connectivity is possible, for performance reasons we recommend to connect to those platforms using a DRDA connection to avoid a possible performance bottleneck at the DB2 Connect gateway. However, if DB2 Connect is already installed in the existing environment and its performance is sufficient for the amount of users using this gateway, this will allow for a very easy setup for QMF for Windows.

4.1.2 Test the server connection

To ensure that QMF for Windows can establish a connection to the database server, select the server in the QMF for Windows Administrator main window and click **Edit...**The **Server Parameters** box opens; click **Test Connection** to test the connection to the selected database server as shown in Figure 34.

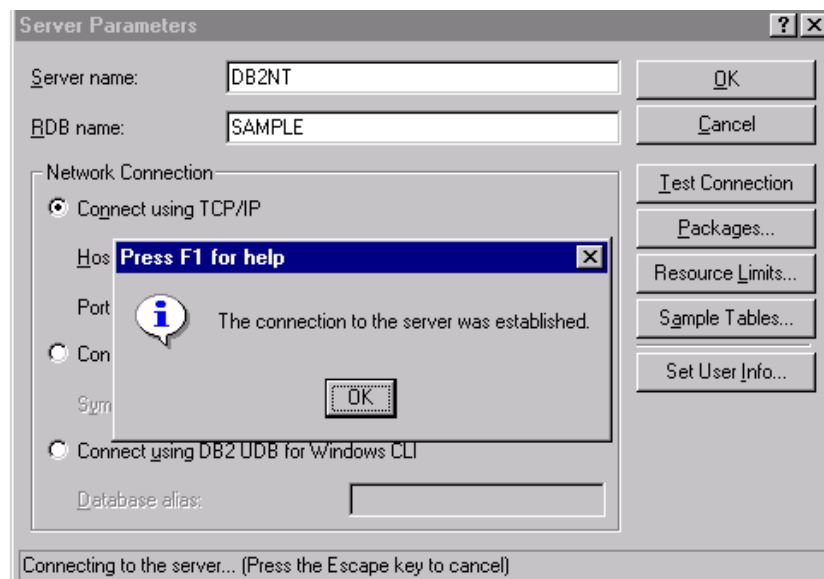


Figure 34. Test the server connection

If there are any problems with your network configuration, QMF for Windows Administrator displays an error message as shown in Figure 35.

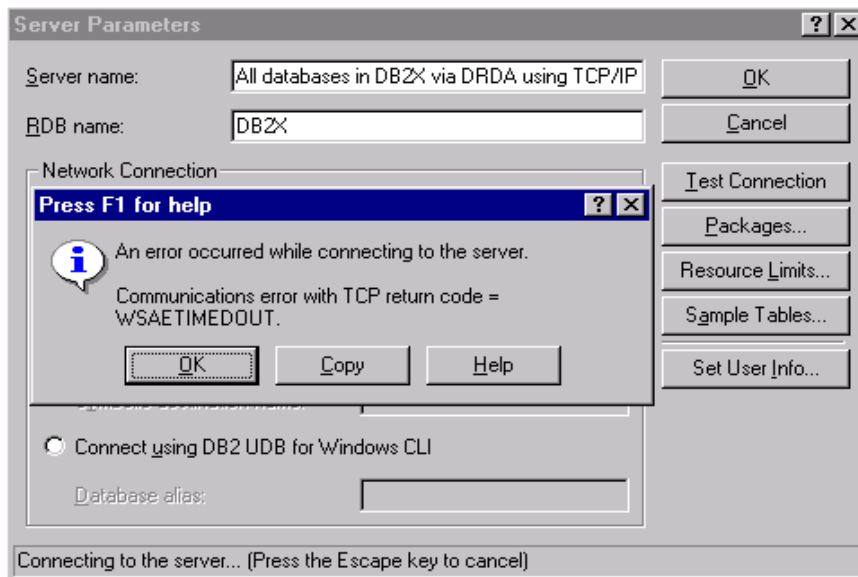


Figure 35. Communication error

The **Copy** button allows you to copy the error message on the clipboard to be then saved for further investigation.

4.1.2.1 Troubleshooting the server connection

If an error occurs, the displayed return code should be used to diagnose the problem with your network technical support services. If you need to further investigate the problem, you can turn on tracing in QMF for Windows Administrator as follows:

1. In the main window, select **Options...** from the **Edit** menu.
2. In the Options dialog box, click **Traces...**
3. In the Traces dialog box, review the listed trace file names; the defaults are probably acceptable.
4. If the problem occurs when connecting using TCP/IP or CPI-C, in the appropriate combo box, select **Calls with parameters and buffers** to record all the communication information.

Important: Be aware that the user id, password, and other sensitive data will be written in clear text in the trace files.

5. Click OK in the Trace dialog box and then in the Options dialog box.
6. Test the connection that caused the error again.

A detailed trace of the calls that QMF for Windows Administrator made to the SNA or TCP/IP software is written to the specified trace files. See Figure 36.

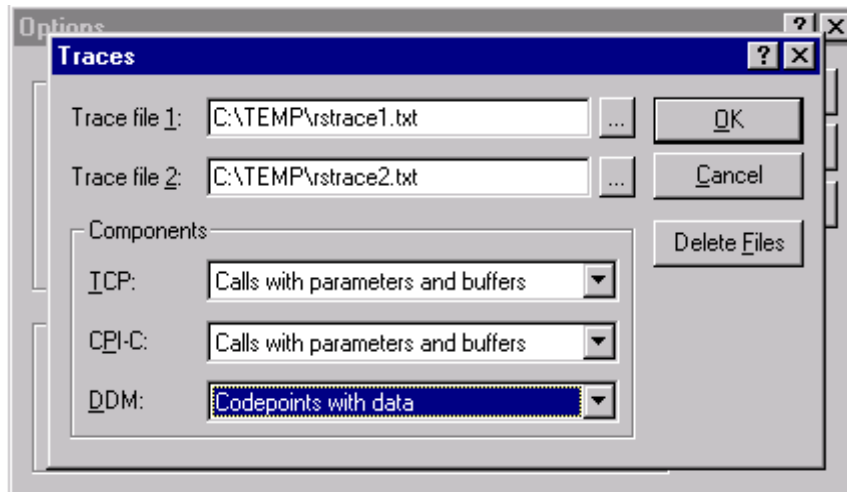


Figure 36. Tracing the server connection

Note:

Turn tracing off when the problem is resolved; tracing can have a significant adverse impact on the performance of QMF for Windows.

If you are connecting to DB2 via an SNA network:

The SNA link and the LU 6.2 session between the QMF for Windows LU and the DB2 LU must be active in order to establish the connection.

There are very few errors that can occur in QMF for Windows Administrator when trying to establish a connection to the server. The problem at this point almost certainly indicates a problem with the network configuration rather than with the QMF for Windows Administrator. These few errors are:

- Failure to activate the SNA software or to start the SNA node.
- Failure to activate the SNA link.
- Failure to properly configure an LU 6.2 session between the QMF for Windows LU and the DB2 LU.

4.1.3 Create QMF for Windows objects

You must create the QMF for Windows objects at each database server you want to connect. Some of these objects might already exist at the server from a previous QMF installation (such as OS/390). The Administrator module can automatically determine which objects need to be created, and it will allow you to automatically create them. Before proceeding, verify if you have the necessary DB2 privileges to create objects at the target server.

To create the QMF installation objects, use the following steps:

1. In the QMF for Windows Administrator main window, select the server and click **Edit...**
2. On the Server Parameters dialog box, click **Packages...**
3. Enter all of the required values on the resulting Packages dialog box.
4. Click the **Create objects** button.

The required values or selections are:

1. Collection Name

This is the user defined name that will group the packages that are created for use by QMF for Windows and QMF for Windows Administrator. The collection name is limited to eight characters.

In this collection, the following five packages will be bound: RAARDBI1, RAARDBI2, RAARDBIA, RAASHUT2, RAASHUT3.

Note

When you define multiple server entries within the SDF, accessing the same database but using different network connection options, be sure to use different Collection Names for each network option. If you do NOT do so, only those users accessing the server using the last bound packages will be able to access the server.

2. Owner ID

To bind those packages, you must have authority at the database server to run the SQL that they contain. If your primary authorization ID has the required privileges, you can leave this field blank. If you have a secondary authorization ID that you usually use for administrative tasks, enter it in this field.

Note:

When defining a connection to DB2 UDB for OS/390, DB2 for OS/390, or DB2 for MVS, use upper-case for the Owner ID. QMF for Windows does not translate lower-case to upper-case.

3. Replace existing packages (if any)

In most cases, you should make sure this box is checked. When installing a new version or service release of QMF for Windows, this will replace all of the existing packages, thus all the clients with a full QMF for Windows installation need to upgrade to the new version as well. All QMF for Windows installations with older versions will no longer be able to connect to the server.

4.1.3.1 Keep existing authorizations on packages

In most cases, you should make sure this box is checked.

4.1.3.2 Decimal delimiter

Select the decimal delimiter that you and your users enter when writing SQL.

4.1.3.3 String delimiter

Select the string delimiter that you and your users enter when writing SQL.

Figure 37 shows the screen to create the QMF for Windows objects.

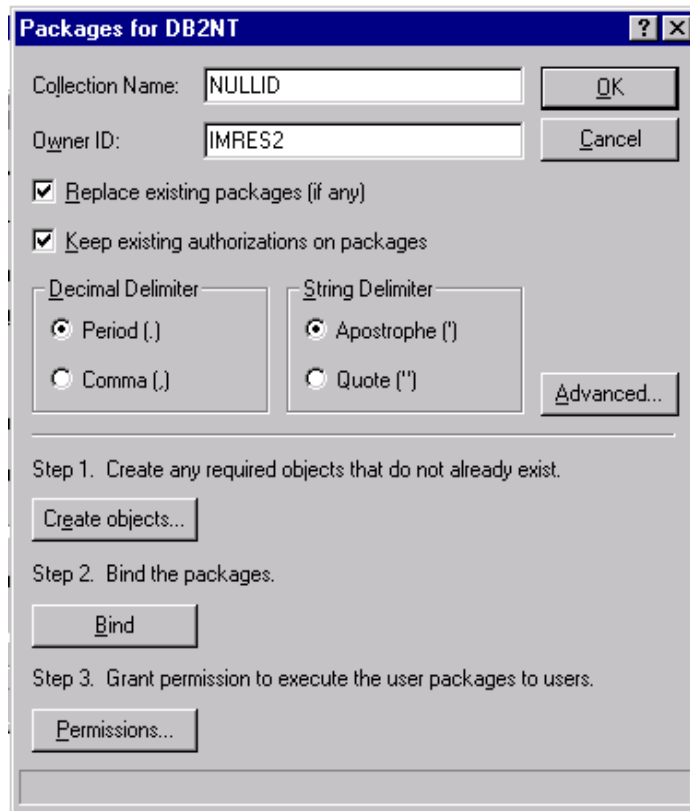


Figure 37. Create objects

When you click **Create objects...**, QMF for Windows Administrator opens a connection to the database server to bind the installation package RAARDBI1. Your authorization ID must have the BINDADD privilege at the database server in order to bind package RAARDBI1. After the installation package is bound, QMF for Windows Administrator asks you whether you want to automatically check which objects need to be created (and which ones already exist). Click **YES** at this prompt.

In order to check for the existence of the required objects, the SELECT authority is required on the following tables, depending on the type of database server.

DB2 UDB for OS/390, DB2 for OS/390, DB2 for MVS:

```

SYSIBM.SYSDATABASE
SYSIBM.SYSTABLESPACE
SYSIBM.SYSTABLES

```

```
SYSIBM.SYSINDEXES  
SYSIBM.SYSCOLUMNS
```

DB2 for VM & VSE, SQL/DS:

```
SYSTEM.SYSDBSPACES  
SYSTEM.SYSCATALOG  
SYSTEM.SYSINDEXES  
SYSTEM.SYSCOLUMNS
```

DB2 UDB for AS/400

```
QSYS2.SYSTABLES  
QSYS2.SYSINDEXES  
QSYS2.SYSCOLUMNS
```

DB2 UDB, DB2 Common Server, DB2 Parallel Edition, DB2 DataJoiner

```
SYSCAT.TABLESPACES (except DB2 Parallel Edition and DB2 DataJoiner  
Version 1)  
SYSCAT.TABLES  
SYSCAT.INDEXES  
SYSCAT.COLUMNS
```

For the complete list of the QMF tables and views that are created in this step, see Appendix C, “QMF for Windows tables and views” on page 379.

If all of the objects already exist, a message to this effect will be displayed and you do not need to do anything further.

If one or more objects need to be created, the Create Objects dialog box opens, displaying a series of SQL statements (separated by semicolons). These are the SQL statements that QMF for Windows Administrator needs to execute to create the required tables. Review this statements carefully for correct syntax and naming conventions. If you are satisfied with the statements, click **OK** to run them at the selected database server.

Hint:

Save the SQL statements by copying the text to the clipboard and save it to a file for future references.

A return SQL code of -551 or -552 indicates that the authorization ID on the bind (either your primary user ID or the specified *owner ID*) does not have all the privileges required to create database objects at the selected database server.

Encoding Schemas

If you have different encoding schemas, QMF for Windows produces an SQL code = -873 SQL state = 53090 error. If you are using Host QMF, before running the **Create Objects** step within the QMF for Windows Administrator, check your database encoding schema. The DB2 system tables must have the same encoding schema as the Host QMF control tables. The encoding schema is either EBCDIC or ASCII. The encoding schema must be the same in both applications to run the Create Objects step from within the QMF for Windows Administrator. The change of the DB2 encoding schema (installation parameter DEF ENCODING SCHEME in panel DSNTIPF) is not recommended.

Special consideration should be used for DB2 OS/390 servers defined as CCSID ASCII, since you have to modify the DDL provided by the QMF for Windows Administrator's automatic detection. QMF for Windows does not check for the CCSID and you could run into a -873 error when the product tries to create a QMF view from one table defined as ASCII and one as EBCDIC. We suggest you modify the DDL as follows:

```
CREATE DATABASE .....CCSID EBCDIC
```

Existing QMF installations

If you already have QMF on an S/390 platform (OS/390, VM, or VSE) or any QMF for Windows versions prior to V.6.1 installed: QMF for Windows creates some of its own tables and views in the already present databases. The naming convention adopted by QMF for Windows uses **RDBI** as the *creator* of the majority of the QMF objects.

Version 6.1 and later of QMF for Windows stores *user profile*, *resource limits*, and *authorization ID* information in different tables, and uses different views than previous versions and host QMF.

The new tables and views used to access these tables are created when you click the **Create Objects** button on the **Packages** dialog in QMF for Windows Administrator. If you look at the SQL that is generated to create these tables and views you'll notice that:

1. If any of the previous host QMF tables are detected to exist, INSERT statements will be created to copy all of the data stored in the old tables to the new tables.

```
INSERT INTO RDBI.PROFILE_TABLE  
SELECT * FROM Q.PROFILES;
```

```

INSERT INTO RDBI.RESOURCE_TABLE
SELECT * FROM Q.RESOURCE_VIEW;

```

2. In the CREATE VIEW statements for each of RDBI.PROFILE_VIEW and RDBI.RESOURCE_VIEW views, there are two versions of the FROM clause, one referring to a table owned by RDBI, and one referring to a table owned by Q (which is commented out by default). To continue sharing information between host QMF and QMF for Windows, comment out the line referring to RDBI, and uncomment the line referring to Q. If you make no changes, the information in the tables owned by Q will be unaffected, but also will not be used by QMF for Windows.

So, if you want to use your existing QMF objects, only the above-mentioned modification during installation has to be done. Once this is set up correctly, no changes need to be made to the existing objects to make them available through QMF for Windows.

User profiles

By default, QMF for Windows V6.1 stores and accesses user profile information in the table named RDBI.PROFILE_TABLE. QMF for Windows V6.1 always accesses this table through the view named RDBI.PROFILE_VIEW. If you want to continue using your existing Q.PROFILES table (host QMF), modify the DDL used to create RDBI.PROFILE_VIEW and point it to Q.PROFILES as follows:

```

CREATE VIEW RDBI.PROFILE_VIEW
(
    CREATOR,
    "CASE",
    DECOPT,
    CONFIRM,
    WIDTH,
    LENGTH,
    LANGUAGE,
    SPACE,
    TRACE,
    PRINTER,
    TRANSLATION,
    PFKEYS,
    SYNONYMS,
    RESOURCE_GROUP,
    MODEL,
    ENVIRONMENT
) AS SELECT CREATOR, "CASE", DECOPT, CONFIRM,
           WIDTH, LENGTH, LANGUAGE, SPACE,
           TRACE, PRINTER, TRANSLATION,

```

```

        PFKEYS, SYNONYMS, RESOURCE_GROUP,
        MODEL, ENVIRONMENT
    FROM Q.PROFILES;
-- FROM RDBI.PROFILE_TABLE;

```

Resource groups

By default, QMF for Windows V6.1 stores and accesses resource group (governing) information in the table named RDBI.RESOURCE_TABLE. QMF for Windows V6.1 always accesses this table through the view named RDBI.RESOURCE_VIEW. If you want to continue using your existing Q.RESOURCE_TABLE table (host QMF), modify the DDL used to create RDBI.RESOURCE_VIEW and point it to Q.RESOURCE_VIEW (the view already created on Q.RESOURCE_TABLE) as follows:

```

CREATE VIEW RDBI.RESOURCE_VIEW
(
    RESOURCE_GROUP,
    RESOURCE_OPTION,
    INTVAL,
    FLOATVAL,
    CHARVAL
) AS SELECT RESOURCE_GROUP, RESOURCE_OPTION,
           INTVAL, FLOATVAL, CHARVAL
    FROM Q.RESOURCE_VIEW;
-- FROM RDBI.RESOURCE_TABLE;

```

Primary/secondary IDs

By default, QMF for Windows V6.1 stores and accesses primary/secondary authid relationship information in the table named RDBI.AUTHID_TABLE. QMF for Windows V6.1 always accesses this table through the view named RDBI.AUTHID_VIEW. If you want to continue using your existing table Q.RAA_AUTHID_TABLE (older version of the QMF for Windows product), modify the DDL used to create RDBI.AUTHID_VIEW and point it to Q.RAA_AUTHID_TABLE as follows:

```

CREATE VIEW RDBI.AUTHID_VIEW
(
    PRIMARY_ID,
    SECONDARY_ID
)
AS
SELECT PRIMARY_ID, SECONDARY_ID
    FROM Q.RAA_AUTHID_TABLE;
-- FROM RDBI.AUTHID_TABLE;

```

4.1.4 Bind QMF for Windows packages

To run distributed SQL at any database server, you must bind the QMF for Windows packages at that database server. The QMF for Windows packages refer to the set of objects that the product uses. These packages might already exist at the server; in this case, they will be replaced.

You use QMF for Windows Administrator to choose the *collection name* and bind options (owner id, etc.....) for the packages that it requires and to automatically bind the packages at the server. In this collection, the following five packages will be bound: RAARDBI1, RAARDBI2, RAARDBIA, RAASHUT2, RAASHUT3.

- RAARDBI1 is used only in the server configuration phase to create the database objects required by QMF for Windows in each connected database server.
- RAARDBIA is used only by the Administrator module and contains the SQL required for administrative functions.
- The other three packages: RAARDBI2, RAASHUT2, and RAASHUT3, are used by the QMF for Windows module.

To create the packages at the target server, follow this procedure:

1. In the QMF for Windows Administrator main window, select the server and click **Edit...**
2. On the Server Parameters dialog box, click **Packages...**
3. Enter all of the required values on the resulting Packages dialog box.
4. On the Packages dialog box, you bind the user and administrator packages by clicking **Bind**. The status line will indicate the progress of the bind operation. If any errors occur, you must correct them and repeat the bind. The most common errors include:
 - SQL code of -204 that indicates that a required table does not exist. To create it, click **Create Objects...**
 - SQL codes of -551 or -552 indicate that the authorization ID on the bind (either your primary user ID or the specified *owner ID*) does not have all of the following privileges required to bind the packages as shown in Table 3.

Table 3. Required privileges

Table/View Name	Privileges
RDBI.RESERVED	SELECT
RDBI.PROFILE_VIEW	SELECT, INSERT, UPDATE
RDBI.TABLE_VIEW	SELECT
RDBI.USER_AUTHID_VIEW	SELECT
RDBI.USER_ADMIN_VIEW	SELECT
RDBI.RESOURCE_VIEW	SELECT, INSERT, UPDATE, DELETE
Q.RAA_SUBTYPE	SELECT, INSERT, UPDATE, DELETE
Q.RAA_OBJECT_VIEW	SELECT
Q.OBJECT_DATA	SELECT, INSERT, DELETE
Q.OBJ_ACTIVITY_SUMM	SELECT, INSERT, UPDATE, DELETE
Q.OBJECT_DIRECTORY	SELECT, INSERT, UPDATE, DELETE
Q.OBJECT_REMARKS	SELECT, INSERT, UPDATE, DELETE
Q.OBJ_ACTIVITY_DTL	SELECT, INSERT, UPDATE, DELETE

4.1.5 Granting Permissions

Finally, after you bind the QMF for Windows packages, you must grant permissions to your users to execute the user packages and work with the product. This grant is automatically done by the Administrator module; you only have to specify the users' IDs and the product will construct and execute the grant SQL at the server:

1. Click **Permissions...** on the Packages dialog box to display the Package Permissions dialog box.
2. In the Package Permissions dialog box enter the user IDs to which you want to grant the authority. (See Figure 38 on page 90).
3. Click **Grant** to make this change at the selected database server.

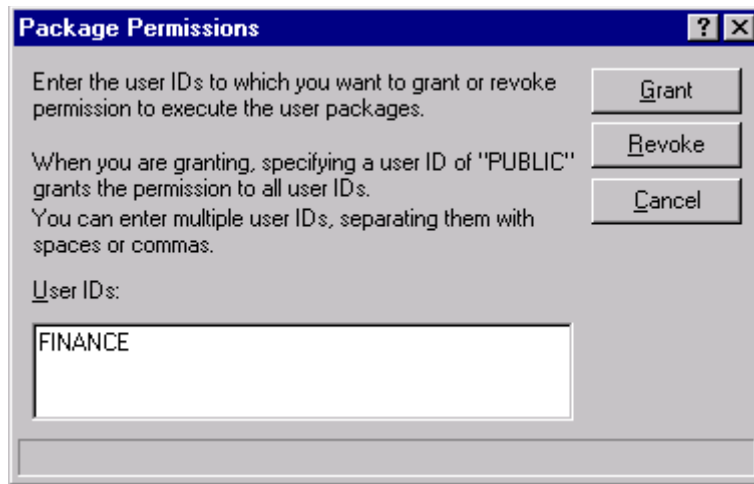


Figure 38. Granting permissions

4.1.6 Creating sample tables

QMF for Windows provides nine sample tables that you can use while learning the product, before you begin working with your own tables. The sample tables are used throughout Chapter 6, “User’s guide” on page 177 and the online help as examples. They contain information about an imaginary electrical parts company. See Table 4.

Table 4. Sample QMF tables

Sample table name	Contains information about
Q.APPLICANT	The prospective employees of the company.
Q.INTERVIEW	The interview schedule for prospective employees.
Q.ORG	Organization of the company by department (within division).
Q.PARTS	Materials supplied to company.
Q.PRODUCTS	Products produced by the company.
Q.PROJECT	Company projects.
Q.STAFF	The employees of the company.
Q.SALES	Sales information for the company.
Q.SUPPLIER	Other companies who supply materials to the company.

To create the sample tables, use the following procedure:

1. Select the server at which you want to create the samples tables and click the **Edit...** button. The Server Parameter dialog box opens.
2. Click the **Sample Tables...** button. A dialog box opens, warning you that the sample tables will overwrite any previous versions of the sample tables that exist on the server. Figure 39 shows the screen that appears when the sample tables are going to be created.

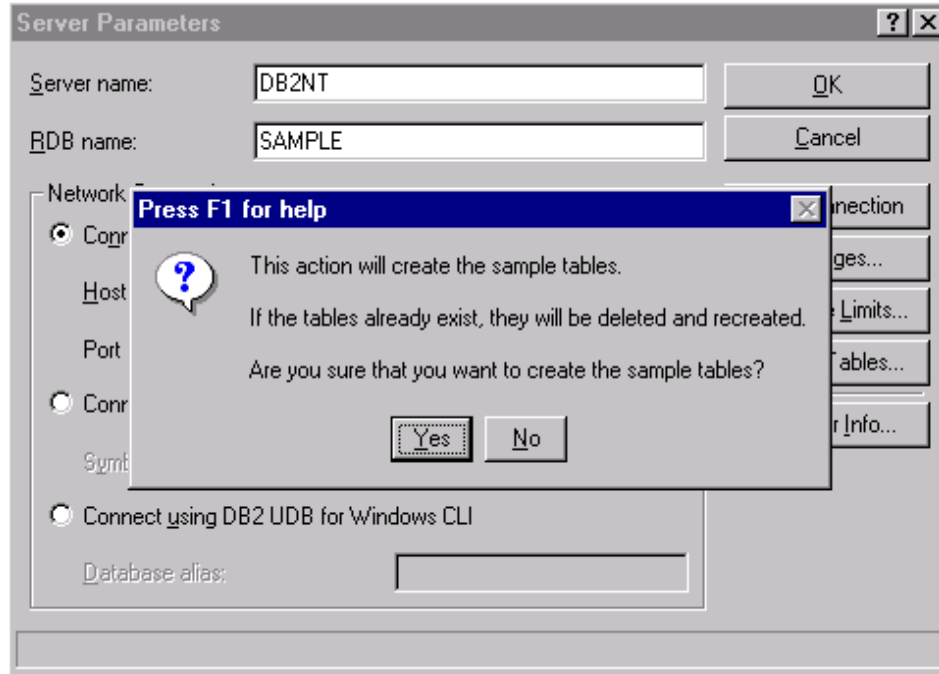


Figure 39. Create the sample QMF tables

Note:

Special considerations for existing QMF installations:

The sample tables supplied with each of the QMF family of products are identical. If they exist already at the specified server, QMF for Windows will delete and recreate them.

3. Click **Yes** to create the sample tables.

4.1.7 Delete a database server

If you decide to delete a server connection, from the main window in QMF for Windows Administrator, select the database server you want to delete and click the **Delete...** button. If you answer Yes, the selected server entry will be deleted from the list of available servers.

The QMF for Windows installation objects created at that database server are not deleted by this action; if you want to clean-up the QMF for Windows objects, you have to explicitly delete them at the server. For a list of the QMF for Windows installation objects, use Appendix C, “QMF for Windows tables and views” on page 379.

4.2 Governing and administration

QMF for Windows incorporates a resource governor, restricting *what actions* a user can perform in QMF for Windows and placing *limits on the resources* a user can consume. This governing feature allows the DBA to provide the users with distributed access to DB2, with confidence that doing so does not have an adverse impact on the overall database or network performance.

The governing function of QMF for Windows is always active. If you do not explicitly set up resource limits, governing based on default limits still happens. **We strongly recommend you define your own sets of limits before you enable the users to access QMF for Windows.**

Using the Administrator module, the DBA can define sets of limits and restrictions, which are called *resource limits groups*. The DBA can then assign users to resource limits groups, according to the governing level that you want performed for those users.

4.2.1 Creating resource limits groups

A resource limits group is a collection of limits and controls on the resources that are governed by QMF for Windows. You can control resource consumption *by user, by day of week, and by time of day*. For example, a resource limits group can contain one set of limits that is in effect weekdays between 8 AM and 6 PM, and another that is in effect on weekends and off-hours. QMF for Windows Administrator is used to maintain resource limits groups.

To prevent users from circumventing limits that you establish, resource limits groups are securely stored in a database table at the database server. Specifically, resource limits groups are stored in the table named RDBI.RESOURCE_TABLE. A view named RDBI.RESOURCE_VIEW is

defined on this table because QMF for Windows users access the resource limits information through that view.

To use QMF for Windows Administrator to maintain resource limits groups, you must have the authorization to execute the QMF for Windows Administrator package. This prevents unauthorized users from changing the limits that are established by the DBA.

Users who are not explicitly assigned to a resource limits group are governed by the limits defined in the *default resource limits group*. The DBA is responsible for creating and maintaining the default resource limits group, which is named **<Default>**.

Note:

If you want to prevent the access to users not explicitly registered in any resource limit group, we strongly suggest that you update the **<Default>** resource limit group and uncheck all the boxes in the **SQL Verbs, Options, Save Data, Binding, and Object Tracking** folders in the **Edit Resource Limits Group Schedule** window.

In order to create a new resource limits group, use the following procedure:

1. Select the server you are currently working with in the QMF for Windows Administrator window and click the **Edit...** button. The **Server Parameters** dialog box opens.
2. Click the **Resource Limits...** button. The **Resource limits groups** list dialog box opens, showing a list of all the resource groups defined at the server.
3. Select the resource limits group from which you want to model the new group and click the **New...** button. The **New Resource Limits Group** dialog box opens.
4. Type a name for the group in the **Group name** field. There are no restrictions on the name you enter.
5. Type any comments, up to 80 characters in length, that describe the new resource limits group. Optionally, you can leave this field blank.
6. If the **Create this group using schedules from...** check box is enabled (see Figure 40), the group you selected as a model has schedules that you can copy into the new group. Check this box if you want to create the new group with copies of all of the schedules contained in the model group. Otherwise, the new group contains no schedules.

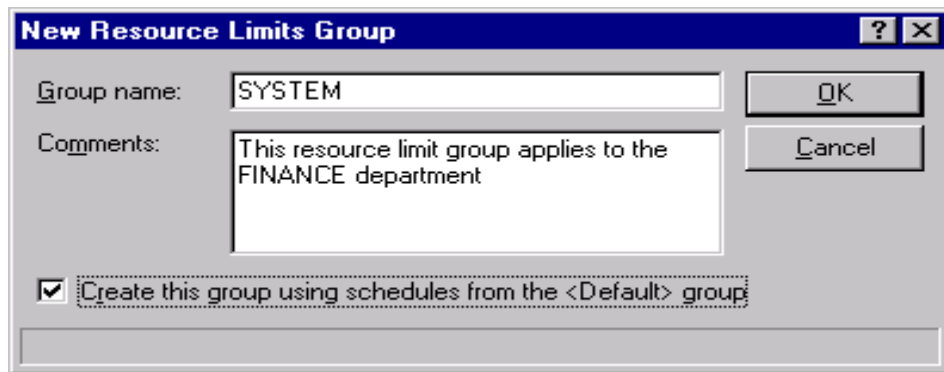


Figure 40. Create resource limit group

7. Click OK to create the resource limits group.

4.2.2 Creating schedules

After the resource limits group is determined, QMF for Windows determines which schedule in the group is in effect. This schedule is in effect at the designated database server and checks for the server Time Zone; **if the database server is accessed by users located in different Time Zones than the server, the resource limit schedule to which they are assigned will determine their access rights in function of the server time.**

A schedule is uniquely identified by a schedule *number*. In addition to specifying a unique number, you must also specify an effective day of week and time of day range. That is, the **From** and **To** Time and the **From** and **To** Day values for the schedule define when the limits and controls are in effect. All ranges are inclusive.

In order to create a new schedule in the resource limits group, use the following procedure:

1. Select the resource limits group for which you want to create schedules in the Resource Limits Group List dialog box and click the **Edit...** button. The **Edit Resource Limits Group** dialog box opens.
2. Select a schedule in the Schedule List if you want it to be used as a model for the new schedule.
3. Click the **New...** button. The **New Resource Limits Group Schedule** dialog box (**Main** Tab) opens so you can create a new schedule. If you have selected a schedule in the Schedule List, the selected schedule is used as a model for the new schedule.

4. Enter the required values on each of the following eight tabs and click OK to create the new schedule.
 - **Main tab:** In the main tab you need to specify the times and days this schedule will be in effect. Figure 41 shows the Main tab to create a new schedule.

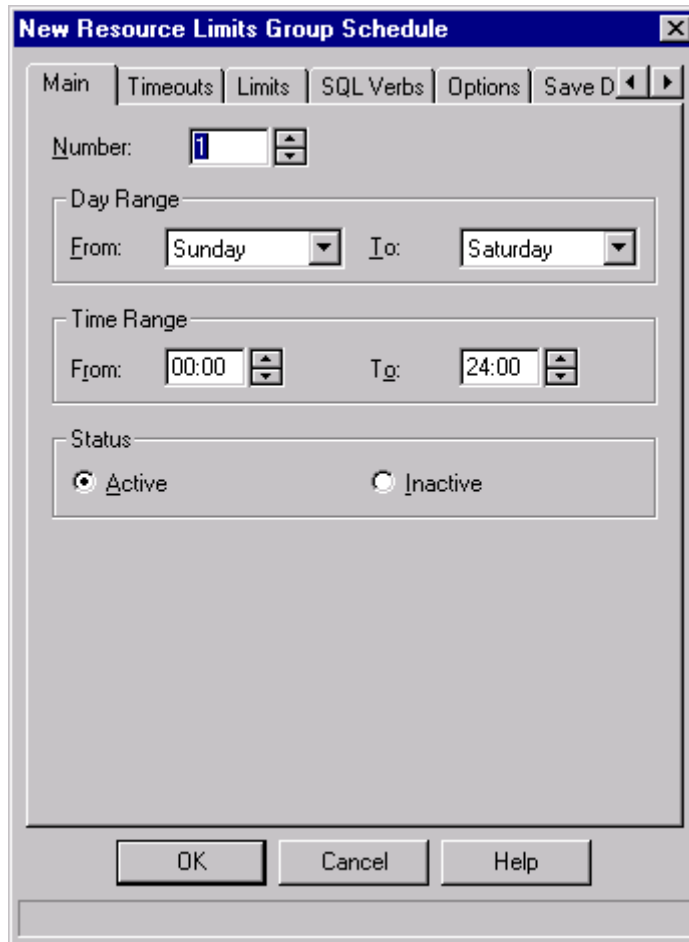


Figure 41. Main: parameters

- **Number** — Indicates the priority of the schedule. For example, if two schedules in the resource limits group cover the same time period (or overlap), the one with the lower number is used. Specify a number greater than zero for this field.

- **Day Range** — The schedule is active between the From Day and the To Day, inclusive. You can specify a range that wraps around the end of the week. For example, if you select Friday as the From Day and Monday as the To Day, the schedule is active on Friday, Saturday, Sunday, and Monday (subject to time of day scheduling).
- **Time Range** — The schedule is active between the From Time and the To Time, inclusive. You can specify a range that wraps around midnight. For example, if you select 20:00 as the From Time and 8:00 as the To Time, the schedule is active from 8:00 p.m. to 12:00 midnight and from 12:00 midnight to 8:00 am (subject to day of week scheduling).
- **Status** — Select *Active* to enable the schedule, subject to day of week and time of day scheduling. The status of the schedule is also subject to the Active or Inactive status of the resource limits group as a whole. Select *Inactive* to disable the schedule, regardless of day of week or time of day scheduling.

Hint:

If more than one schedule is defined to be in effect at the same time, the QMF for Windows governor will use the one with the lowest schedule number.

- **Timeouts tab:** This is used to specify the limits for this schedule in order to gain control over the resource consumption for this group. Entries with a value of “0” mean that no limit will be defined. Figure 42 shows the Timeout tab of the schedule definition.

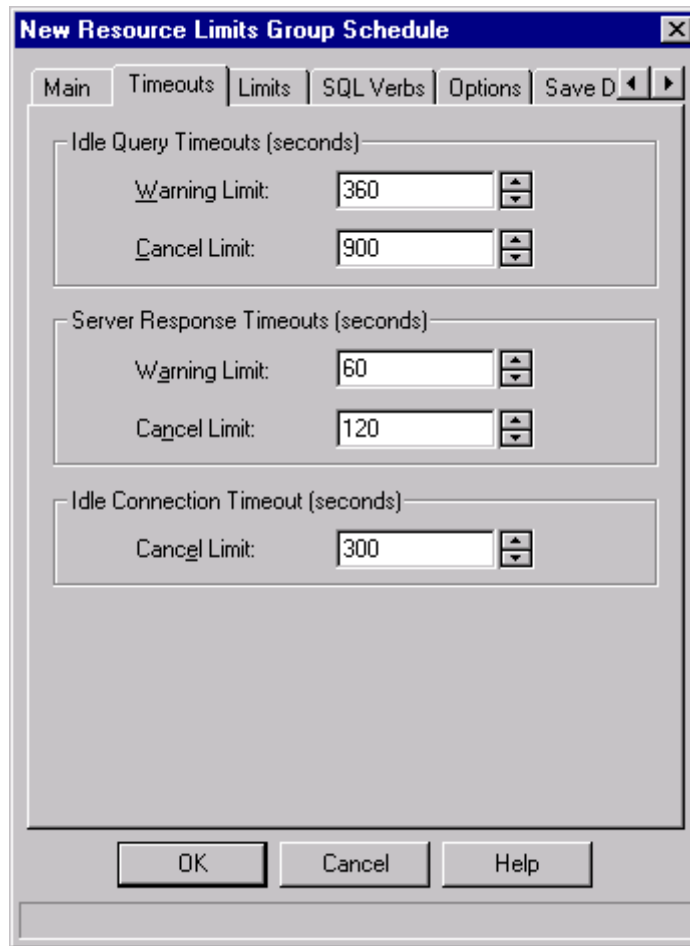


Figure 42. Timeouts: parameters

- Idle Query Timeouts — Limits the amount of time a query can remain idle You may set two different timeouts:
 - Warning Limit — When this timeout expires, QMF for Windows reminds the user that the query is idle and prompts the user whether or not to cancel the query.
 - Cancel Limit — When this timeout expires, QMF for Windows automatically cancels the query.

A query might be idle when the first buffer of data has been returned to the user, and QMF for Windows is waiting for that user to go to the bottom of the data before it fetches the next set of data.

- **Server Response Timeouts** — Limits the amount of time QMF for Windows waits for a response from the database server before cancelling a request. QMF for Windows waits asynchronously for a response each time it sends a request to the database server. For example, when you run a query, QMF for Windows sends the request to the database server and waits asynchronously for the query results to return from the database server.

Hint:

A lower timeout limit prevents long-running (runaway) queries. A higher limit allows database requests to complete when the database server is slow due to resource contention or other reasons.

- **Warning Limit** — When this timeout expires, QMF for Windows prompts the user whether or not to cancel the request.
- **Cancel Limit** — When this timeout expires, QMF for Windows automatically cancels the request.
- **Idle Connection Timeout Cancel Limit** — Limits the amount of time QMF for Windows retains an idle connection to a database server. This limit balances the trade-off between connection establishment overhead and idle connection resource consumption.

Hint:

A lower timeout limit helps minimize the resources consumed at servers by idle connections. A higher timeout limit helps minimize the overhead of establishing connections.

When this timeout expires, QMF for Windows automatically closes the idle connection to the database server.

- **Limits tab:** In this window (shown in Figure 43), you specify other resource limits, like maximum rows allowed to fetch by this resource limit group.

It is important to keep in mind the way that QMF for Windows retrieves the data. For example, if a resource limit group has a defined fetch limit of 10,000 rows, QMF for Windows fetches the first buffer full of data, let's say 8,000. It then checks and sees that the maximum of 10,000 has not yet been reached and fetches the next buffer of data. As this second buffer full of data might again return 8,000 rows, the user will see 16,000 rows, even if the limit has been defined to be 10,000.

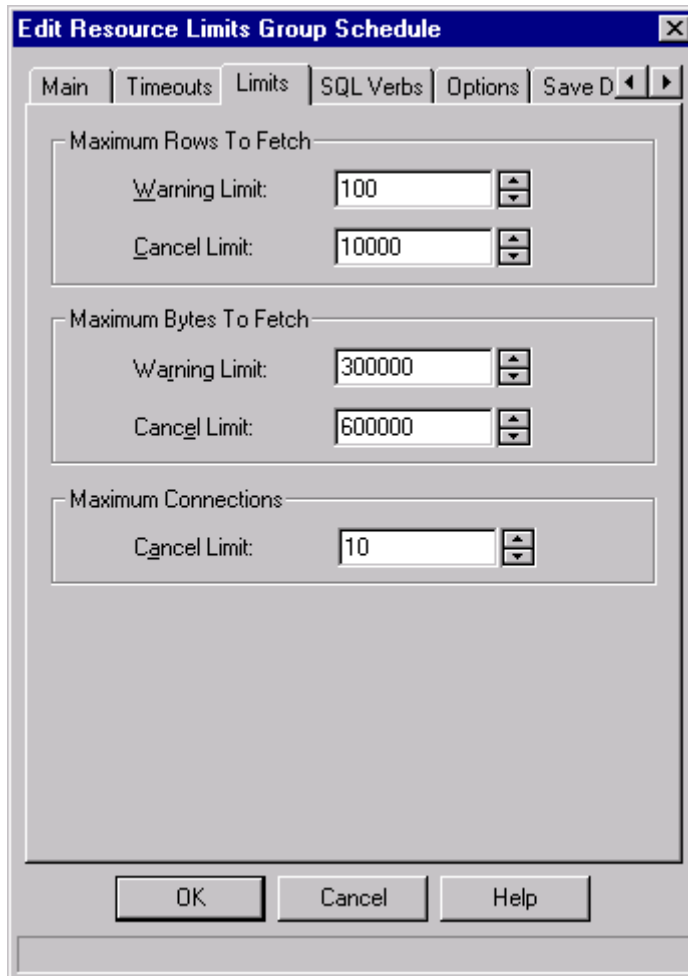


Figure 43. Limits: parameters

- Maximum Rows To Fetch — Limits the number of rows of data QMF for Windows retrieves from a database server when running a query.
 1. Warning Limit — When this limit is reached, QMF for Windows prompts the user whether or not it should continue to fetch more data.
 2. Cancel Limit — When this limit is reached, QMF for Windows automatically cancels the query.

- Maximum Bytes To Fetch — Limits the number of bytes of data QMF for Windows retrieves from a database server when running a query. See Row Limits, Byte Limits, and Query Buffers for more information.
 - Warning Limit — When this limit is reached, QMF for Windows prompts the user whether or not it should continue to fetch more data.
 - Cancel Limit — When this limit is reached, QMF for Windows automatically cancels the query.
- Maximum Connections Cancel Limit — Limits the number of simultaneous connections that QMF for Windows establishes to the database server. In general, connections are reused, so that if you run one query at a server and then run another query at the same server, only one connection is required. However, if you run those two queries simultaneously, then two connections are required. If QMF for Windows requires another connection to a server and the Maximum Connections limit is reached, an error is returned and the operation is not performed.
- **SQL Verbs Tab:** QMF for Windows delivers the most essential business reporting requirements for corporate situations where the broadest range of needs must be met with the minimum number of tools. You can allow or disallow the use of certain SQL verbs when a user is accessing a database server from QMF for Windows. If a user attempts to run a query that contains a disallowed verb, QMF for Windows cancels the query without sending it to the database server. If a user attempts to run a query that contains an allowed verb, QMF for Windows sends the query to the database server, and the database server's security authorization validation takes place. Figure 44 shows the screen where the permission for the use of certain SQL verbs are defined.

Note:

Turning off the permissions for update, delete, and insert does not affect the ability to perform these actions using the table editor.

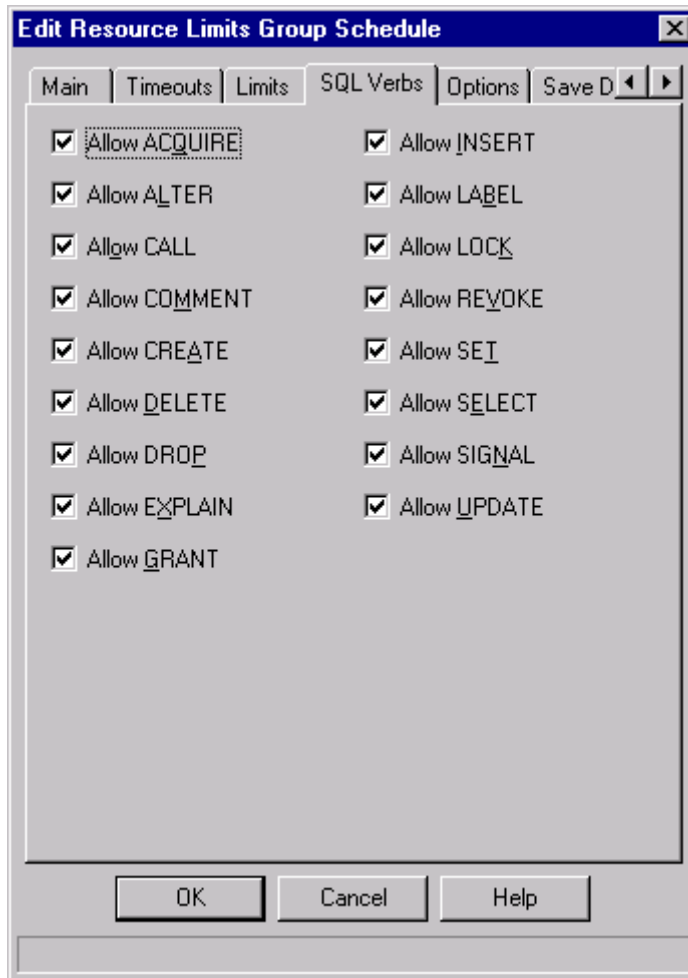


Figure 44. SQL Verbs: parameters

You can allow/disallow the use of the following verbs from QMF for Windows: ACQUIRE, ALTER, CALL, COMMENT, CREATE, DELETE, DROP, EXPLAIN, GRANT, INSERT, LABEL, LOCK, REVOKE, SET, SELECT, SIGNAL, UPDATE.

- **Options tab:** Using the Options tab allows to control the access to the database objects for the resource limit group (see Figure 45).

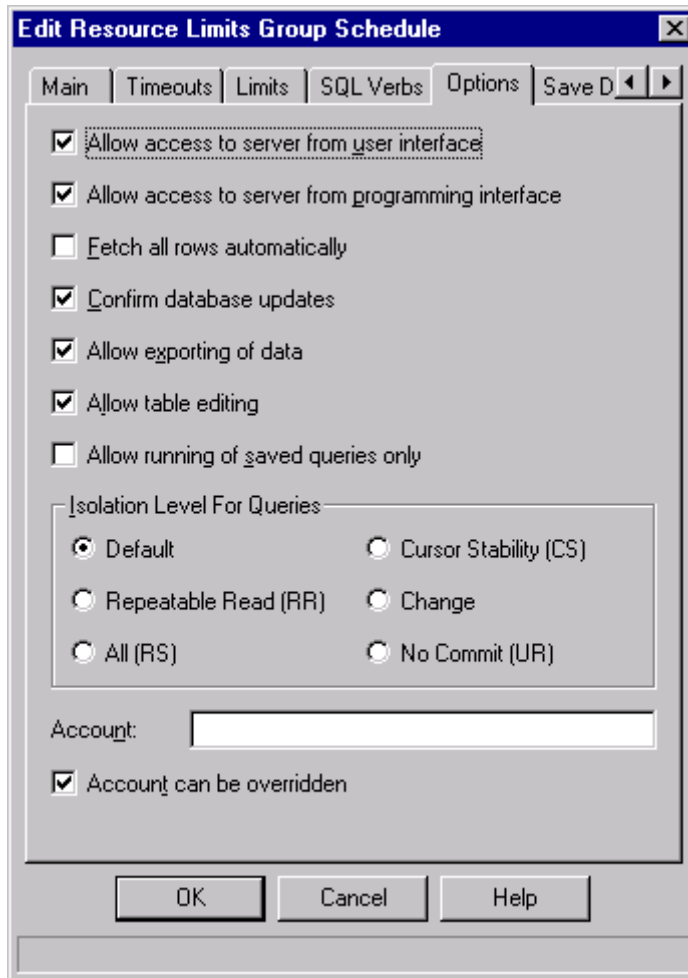


Figure 45. Options: parameters

- Allow access to server from user interface — Permits user access to the server from the QMF for Windows user interface.
- Allow access to server from programming interface — Permits user access to the server from the QMF for Windows programming interface.
- Fetch all rows automatically — Determines how the database server sends query results to QMF for Windows. QMF for Windows typically requests data from the database server only as required to display query results. For example, if 20 rows fill up the query window, QMF for Windows requests only 20 rows. When the user scrolls down to make

the 21st row visible, QMF for Windows requests more data. However, if the user runs the query and then waits a long time before scrolling down, the query remains active for that entire period of time. This is undesirable because it consumes resources at the database server the entire time the query is active.

Hint:

If you enable this parameter, QMF for Windows repeatedly requests data until it receives all of the data, independent of the user's scrolling requests.

- Confirm database updates — Determines whether QMF for Windows prompts the user to confirm database changes resulting from the queries they run or actions they perform when editing tables. Enable this option if you want QMF for Windows to prompt users to confirm database changes. Disable this option if you want database changes to happen without confirmation.
- Allow exporting of data — Permits use of the Export data command on the File menu or in procedures.
- Allow table editing — Permits use of the table editor with QMF for Windows.
- Allow running of saved queries only — When checked, limits the user to running only queries that have been previously saved at the database server. In addition, the user is prohibited from saving new queries at the database server.
- Isolation level for queries — Sets the isolation level for queries run by users. This option only has an effect at the following types of servers: DB2 for MVS Version 4, DB2 for OS/390 Version 5, DB2 UDB for OS/390 Version 6, and DB2 Server for VM & VSE Version 5 or higher.
- Account — Sets the default string specifying accounting information that is sent to the database server when users in the resource limits group connect to it.
- Account can be overridden — When selected, allows the user to override the default account by entering a new one on the **Set User Information** dialog box. If access to database objects other than those owned by members of the resource limit group should be prevented, the checkbox will not be selected.

- **Save Data tab:** Figure 46 shows the screen where the parameters for the permission to save data is defined for the resource limit group. The parameters are:
 - Allow Save Data command — Enables the user to save data at the database server. Saving data can be an extremely resource-intensive action, and can have a significant impact on your database server and network performance.
 - Default table space — This option is available only if you select the Allow Save Data command option. It specifies a table space as the default target for tables created by the save data process. The syntax of the table space name you enter must conform to the database server's rules for table space names. Any value you specify is used as part of a CREATE TABLE SQL statement executed when the user saves data to a new table.
 - Default table space can be overridden — This option is available only if you select the Allow Save Data command option. It specifies whether the user is forced to use the table space specified in Default table space or can specify any table space (subject to database security authorizations). Select this check box to allow the user to specify any table space.

Note:

If you select this option but do not specify a default table space, the user cannot specify a table space, and the database server uses a default.

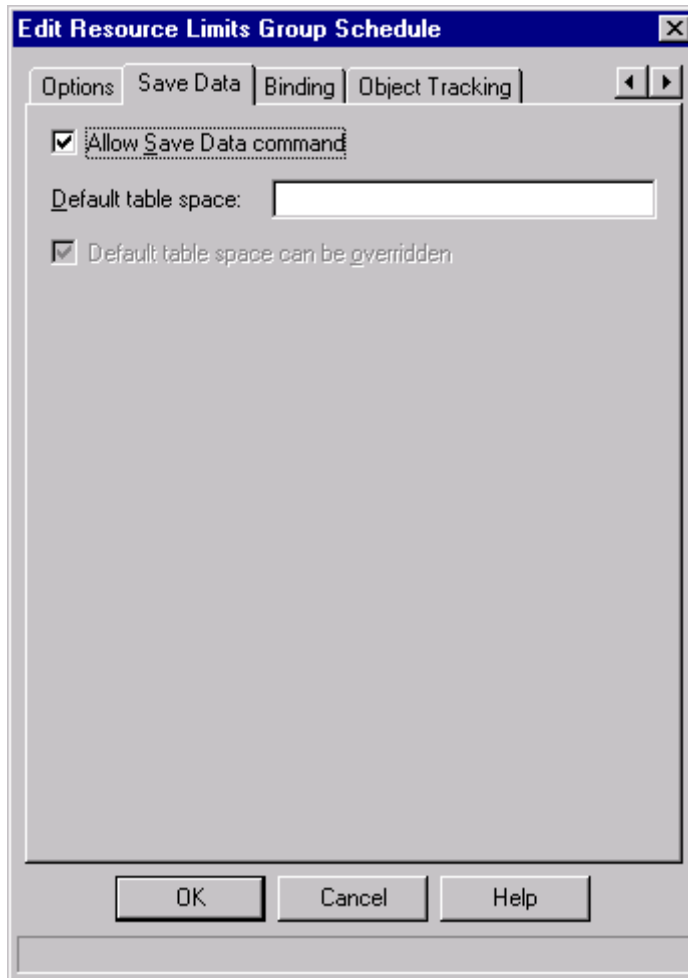


Figure 46. Save data: parameters

- **Bindings tab:** In this screen parameters for the bind process are defined. (See Figure 47 on page 106).
 - Allow binding of packages — Enables users to bind static packages for their queries.
 - Allow dropping of packages — Enables users to drop static packages from the database server.
 - Default collection name — Specifies the default collection ID for static packages bound by users.

- Default collection name can be overridden — Specifies whether a user is forced to use the Default collection name or can specify any collection ID (subject to database security authorizations).

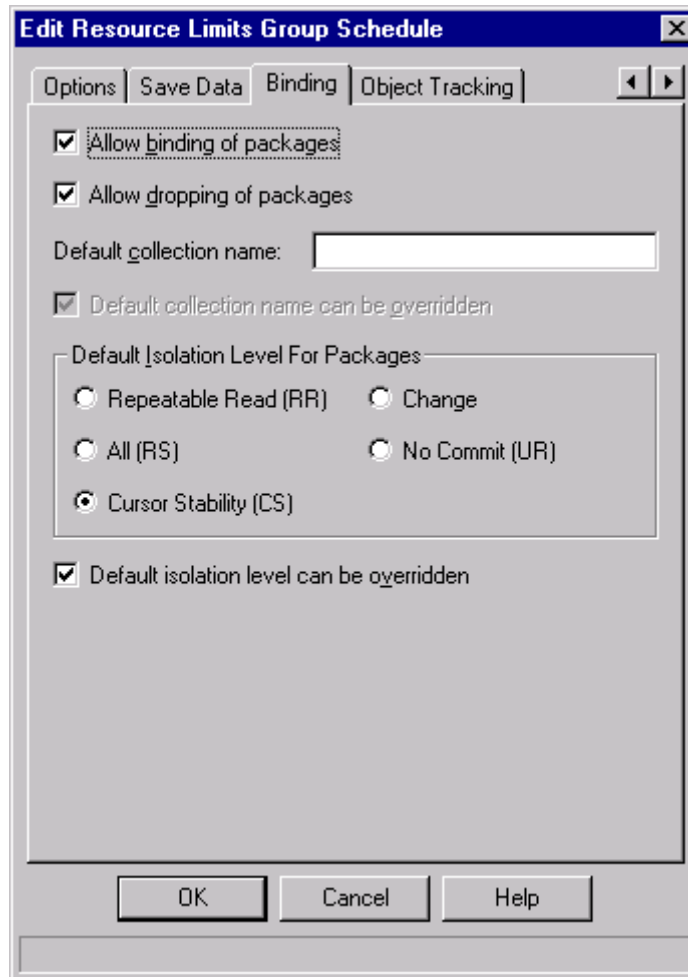


Figure 47. Binding tab parameters

- Default Isolation Level For Packages — Specifies the default isolation level for static packages bound by users:
 - Repeatable Read (RR) — Specifies that the execution of SQL statements in the package is isolated (protected) from the actions of concurrent users for rows the requester reads and changes, as well as phantom rows.

- All (RS) — Specifies that the execution of SQL statements in the package is isolated (protected) from the actions of concurrent users for rows the requester reads and changes.
 - Cursor Stability (CS) — Specifies that the execution of SQL statements in the package and the current row to which the database cursor is positioned are isolated (protected) from the actions of concurrent users for changes the requester makes.
 - Change — Specifies that the execution of SQL statements in the package is isolated (protected) from the actions of concurrent users for changes the requester makes.
 - No Commit (UR) — Specifies that the execution of SQL statements in the package is not isolated (protected) from the actions of concurrent users for changes the requester makes.
- Default isolation level can be overridden — Specifies whether a user is forced to use the Default isolation level or can specify any isolation level.
- **Object Tracking tab:** QMF for Windows 6.1 contains detailed object tracking abilities that are managed through the QMF for Windows Administrator. During the installation QMF for Windows creates two tables for Object Tracking:
 - **Q.OBJ_ACTIVITY_DTL** (the detail table) holds all of the detailed tracking options determined by the **Object Tracking** Tab within the Resource Limits for your particular Resource Group.
 - **Q.OBJ_ACTIVITY_SUMM** (the summary table) holds the summary information for the objects.

Using this Object Tracking capability allows the database administrator to perform such tasks as:

- Running detailed history report for all QMF objects.
- Locating unused objects
- Locating frequently accessed data sources (table/columns)
- Spotting potential problem areas.

Figure 48 shows the Object Tracking screen where this function will be enabled and defined.

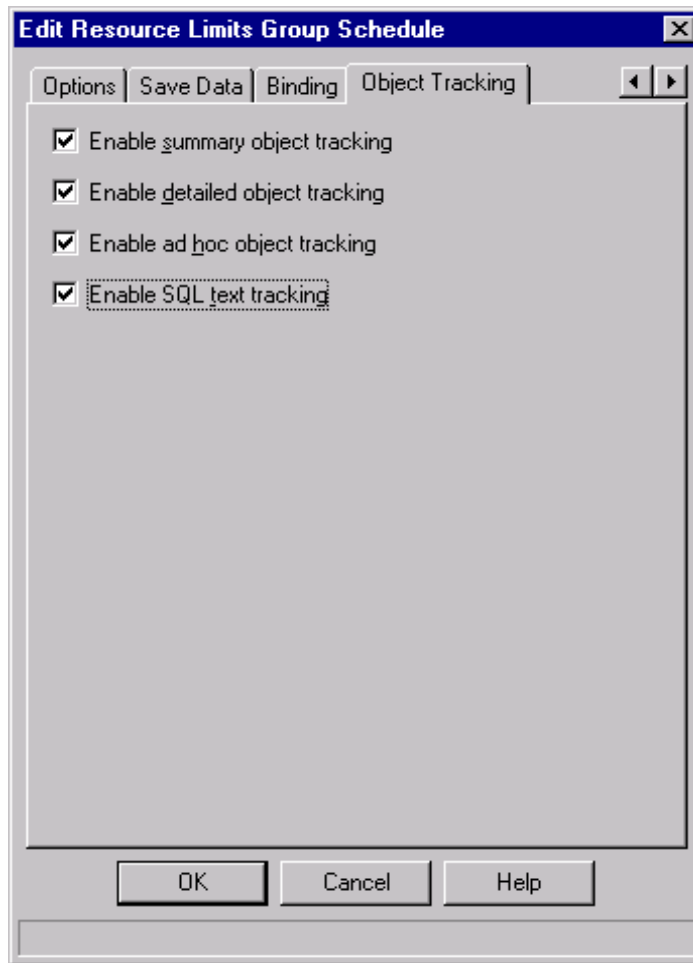


Figure 48. Object tracking: parameters

- Enable summary object tracking — Determines whether or not object use is tracked. Enable this option if you want to track the number of times an object has been run, and the most recent times it was run and modified.
- Enable detailed object tracking — Determines whether or not a detailed record of each action taken using the object is kept. Enable this option if you want to keep a record of each time the object was run, who ran it,

and the results. This option must be enabled if you want to use ad-hoc object tracking or SQL text tracking.

Note:

If you enable this option, a large amount of data can be inserted into the Q.OBJ_ACTIVITY_DTL table. See “Object tracking: maintenance” on page 109 for more information.

- Enable ad-hoc object tracking — Determines whether or not a record of each ad-hoc query is kept. Enable this option if you want to keep a record of each ad-hoc query that is run, and the SQL text of that query. Ad-hoc object tracking requires detailed object tracking.
- Enable SQL text tracking — Determines whether or not a record of the SQL text of each query is kept. Enable this option if you want to keep a record of the text of the SQL each query that is run. SQL text tracking requires detailed object tracking.

Object tracking: maintenance

Once you have enabled the tracking above you must perform maintenance on the Q.OBJ_ACTIVITY_DTL table that QMF for Windows creates during the installation process. You can create a scheduled task to run a DELETE statement which will do most of your cleanup for this table.

With these tracking features enabled you can maintain a record of all queries and their SQL text. You can track how long queries take, how often they run, and who runs them.

Object tracking data is added to the Q.OBJ_ACTIVITY_DTL table. Periodically, you must perform maintenance on the table and its associated table space (RAADB.RAATS2) and index (Q.RAAIX2). The following recommendations will assist you in maintaining efficient performance:

- To maintain performance of table Q.OBJ_ACTIVITY_DTL:
- Make sure RUNSTATS is run for the appropriate table space and index.
- As the data volume increases, rebind the QMF for Windows packages to ensure the use of the index.
- Delete old data periodically. For example, you can use this query to delete all rows older than 30 days:

```
DELETE FROM Q.OBJ_ACTIVITY_DTL WHERE "DATE" < (CURRENT DATE - 30
DAYS)
```

NOTE

The first-used, last-used, and last-modified summary statistics stored in the Q.OBJ_ACTIVITY_SUMM table are not affected when you delete detailed data.

4.2.3 Assigning users to the resource group

The relationship between a QMF for Windows user and a resource limits group is stored in a table at the database server, specifically, the table named **RDBI.PROFILE_TABLE**, accessed via the view named **RDBI.PROFILE_VIEW**. QMF for Windows Administrator maintains user and resource limits group relationships in this table.

When QMF for Windows connects to a database server, the user must provide user information (user ID and password), which is validated by the database server. If the user information is valid, QMF for Windows determines which resource limits group to use by first locating the correct profile for the user; this is done by searching the CREATOR, ENVIRONMENT, and TRANSLATION columns in the RDBI.PROFILE_VIEW table.

To assign a user to a certain resource limit group, follow these steps:

1. Click the **Assign...** button on the **Resource Limits Group List** dialog box. The **Assign User Profiles** dialog box opens.
2. Type the first user ID you want to assign in the Show user profiles with "creator" matching field or a matching pattern if you want to work with multiple user IDs, and click the **Refresh List** button. QMF for Windows Administrator retrieves all the user profiles stored in the RDBI.PROFILE_VIEW table that match the value you entered and displays them in the Not Assigned or Assigned lists.

Note:

If the user ID you want to assign does not have an entry in the RDBI.PROFILE_VIEW table, click the Create New... button to create the new user profile.

3. Select the appropriate user IDs and use the Assign and Unassign buttons to move them to either list.
4. Click OK.

Figure 49 shows the QMF for Windows screen used to make these definitions.

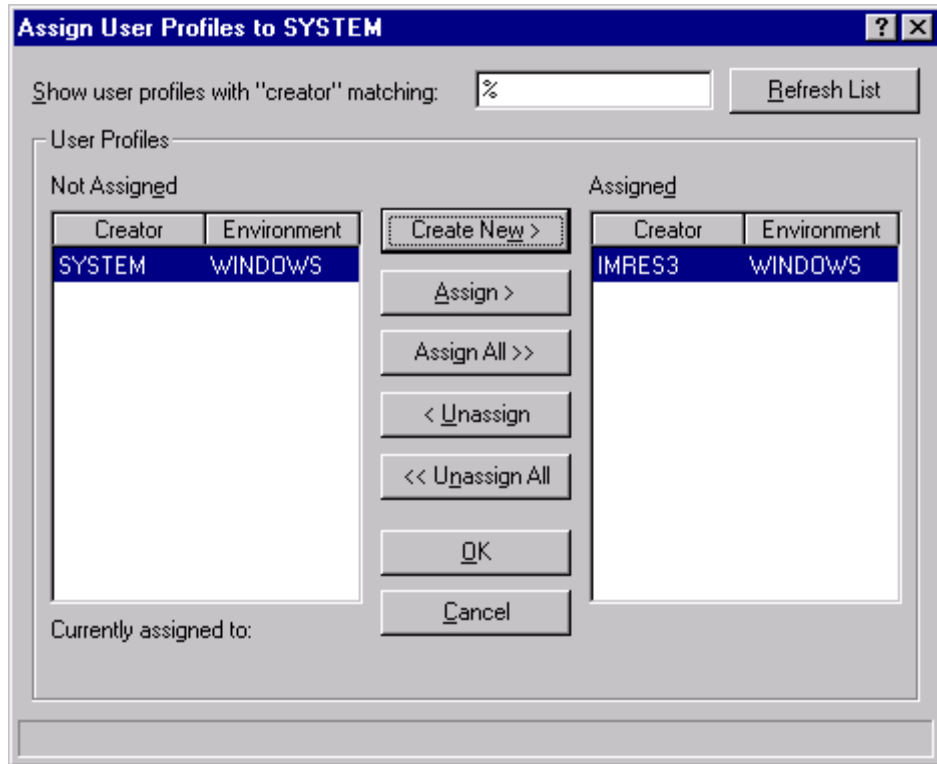


Figure 49. Assign users to resource group

4.3 Security

QMF for Windows Administrator is the administrative component of QMF for Windows. Using the Administrator module is strictly an administrator task. There should not be any need for an end user to run QMF for Windows Administrator. However, there is no security risk if an end user does run QMF for Windows Administrator; the existing database and file-sharing security mechanisms still apply and restrict what a user can or cannot do.

To prevent users from circumventing limits that you establish, resource limits groups are securely stored in a database table at the database server. Specifically, resource limits groups are stored in the table named **RDBI.RESOURCE_TABLE**. A view named **RDBI.RESOURCE_VIEW** must

be defined on this table because QMF for Windows accesses that view, not the table.

QMF for Windows Administrator is used to maintain resource limits groups. To use QMF for Windows Administrator to maintain resource limits groups, you must have the authorization to execute the QMF for Windows Administrator package. This prevents unauthorized users from changing the limits that are established by the administrator.

4.3.1 Change password capability

Users can change their host and workstation passwords from within QMF for Windows. See Figure 50.

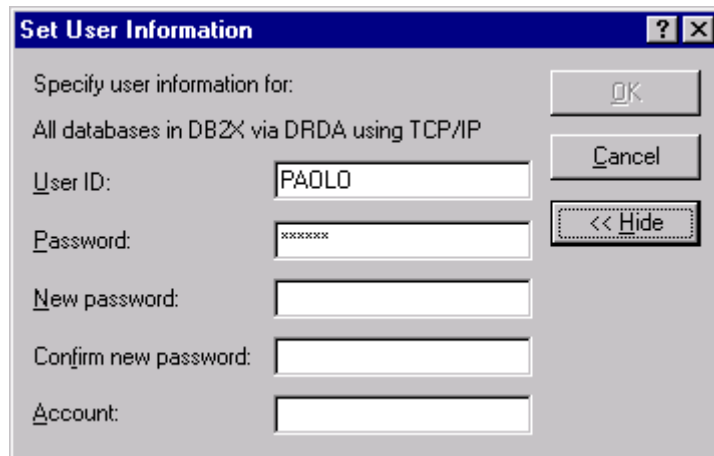


Figure 50. Change password

4.3.2 Lists

As mentioned previously, QMF for Windows knows four different types of objects:

- Queries
- Forms
- Procedures
- Tables

The database administrator might want to restrict the visibility of these objects to a certain number of users. To do this QMF for Windows allows the creation of predefined lists that the users will see by default when working with the product. Lists are also useful to simplify the days work of the users by providing them with a tailored set of QMF for Windows objects by default.

If a user starts working with predefined queries and so on, they would typically use the **File -> Open from Server** menu. This would open a window that would then need to be modified if this user does not need to see all objects from all users. Creating a predefined list provides an alternative way to simplify the work, by using the **File -> Open** menu to see a list of tailored objects.

The following steps need to be performed to create and save a predefined list.

1. On the **File** menu of QMF for Windows, click **New -> List** to open the windows shown in Figure 51. Make sure to select the correct server to create the list from. If the window does not show the required server (like the `DB2AIX` in the example shown), go to the **List -> Set Server** menu to set the active server accordingly.
2. You may now specify the owner of the objects that need to be in the list, the object directly by name, and the type of objects to be included in the list.
3. Click the **Refresh List** button to create the list. This list may then be modified by removing certain objects individually from this list.
4. Save the newly created list using the **File -> Save As** menu. This will create a file in the default installation directory that can then be opened using the **File -> Open** menu.

Note:

Using this method does not prevent the user from being able to see all objects by clicking the **Refresh List** button in the list window.

For true restriction, the **Create View** permission for **Object View** or **Table View** should be edited at the creation of a collection. Different collections can then be used by different user groups using different server definition files.

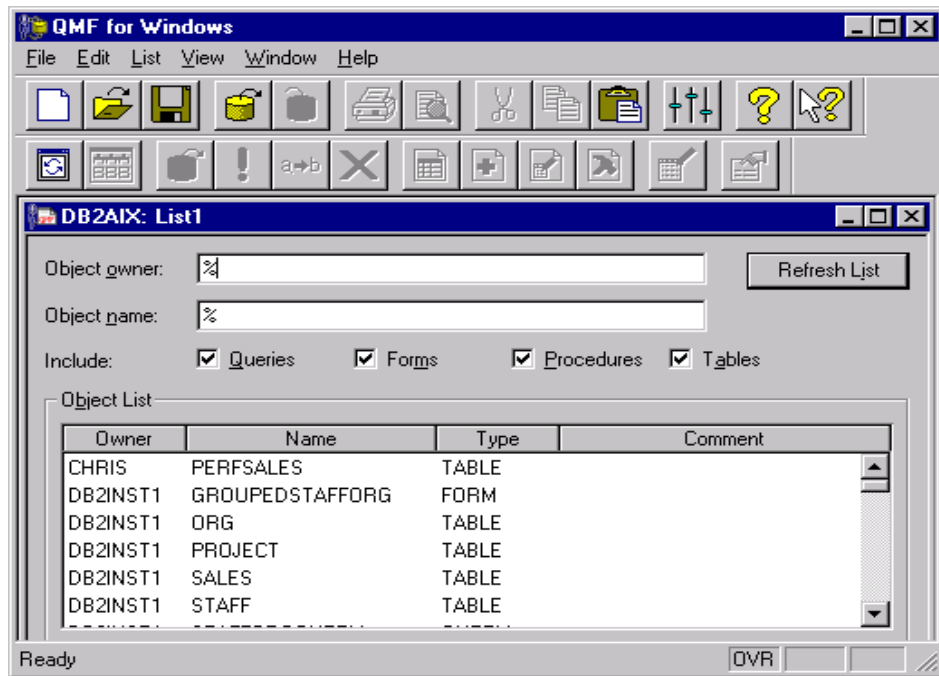


Figure 51. Lists

Within the List window, several options are available using the icons in the Toolbar or the right mouse button. These options are:

- **Display object** to view the selected object. This function is available for Queries, Forms, Procedures, and Tables.
- **Run object** to execute the selected object. This is only available for Queries and Procedures.
- **Draw object** will create a query based on a selected table. The type of query may be either a SELECT query, an SQL UPDATE query, an SQL INSERT query, or a prompted query. This options only works for tables.
- **Edit object** is available only for tables and will open the Table Editor for this table.
- **Properties** again is available for all four types of objects and displays the properties of the selected object, including comments, attributes, and historical usage information.

QMF for Windows, as of this writing, has one problem when working with Lists:

When logging on to a DB2 on OS/390 with a user with SYSADM authorization, using Lists will not show the system catalog table and views. The issue is that all tables with at least one row in SYSIBM.SYSTABAUTH will show up in lists. If there are no rows in SYSIBM.SYSTABAUTH (which can be the case for some of the system tables), it will not show up. The work around is to grant at least one permission to at least one user on each of the system tables.

4.4 Other DBA Tasks

In addition to the functionality of the QMF for Windows Administrator already mentioned, there are more activities that the database administrator can perform. The following section covers some of these activities.

4.4.1 Convert dynamic SQL to static SQL

Static queries are SQL queries that have previously been passed through the database servers preprocessor and the access plan to the data has been stored within a package. When this static query is executed later, the database server no longer uses the query text and its preprocessor to determine the optimal access path, but it can use the access path stored within the package directly. This reduces resource consumption at the server and improves the execution of the query.

All queries created through QMF for Windows use dynamic SQL. If the database administrator, by using the object tracking capabilities of the product, identifies certain queries that are executed very frequently, they might be candidates for a conversion to static SQL. QMF for Windows allows this conversion to be performed for SQL queries only. If you have any prompted query that is supposed to be converted to a static query, the following steps have to be performed first:

1. Open the prompted query.
2. Select **Convert to SQL** from the **Query** menu.
3. Save the new SQL query.

Once this is done, select the query and convert it to a static query using the following steps:

1. Open the SQL query.
2. Select **Bind Static Package** in the **File** menu of QMF for Windows.

3. Select the **Package** tab, enter the collection ID and a package name, and change any of the available options as shown in Figure 52.

Clicking the Advanced button will allow for a more detailed bind option definition than using the main screen itself. Parameters related to the Date and Time Format, Blocking and Degree of Parallelism, using Snapshot and Explain, and Dynamic rules as well as rules related to character subtypes such as FOR BIT DATA all can be set there.

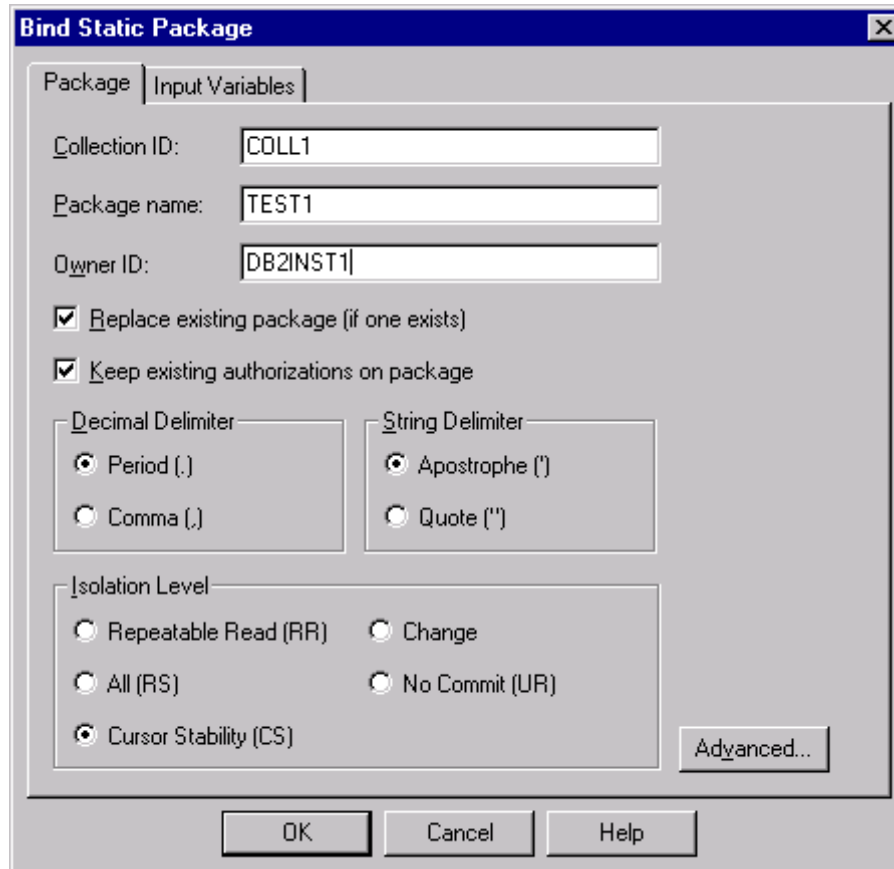


Figure 52. Create static SQL

4. If the original query makes use of substitution variables, select the **Input Variable** tab. Here all the substitution variables used have to be translated to host variables for the static query. Not all of the substitution variables are easy to map to host variables, as they provide direct text substitution within the query text before being send to the database server, whereas host variables are sent to the database server as part of the query.

5. Valid data types for host variables are:

- CHAR(n)
- VARCHAR(n)
- INTEGER
- SMALLINT
- FLOAT
- DECIMAL(p,s)
- DATE
- TIME
- TIMESTAMP

Figure 53 shows the Input Variable screen.

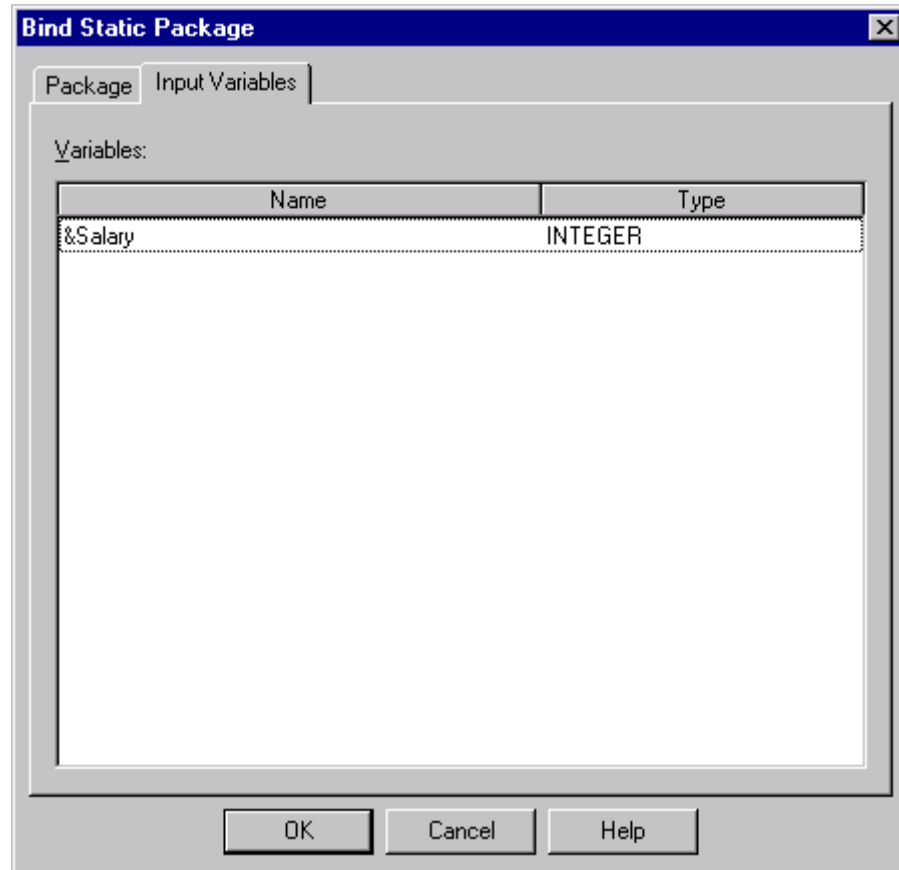


Figure 53. Input variables

6. Click OK to convert the query to a static query. A window such as the one shown in Figure 54 will appear.

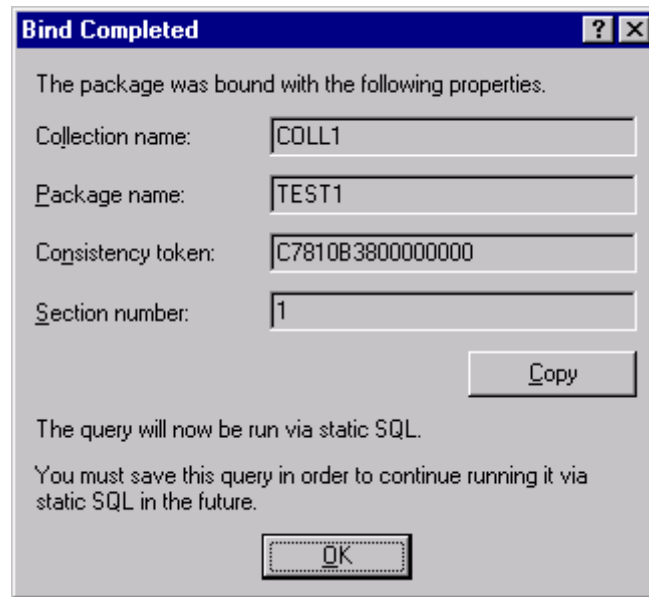


Figure 54. Bind complete

Note the comment at the bottom of the screen. If you do not save the query that just has been translated into a static SQL query, it will not be able to be executed as static SQL later.

4.4.2 DB2 UDB for OS/390 predictive governor support

In DB2 UDB for OS/390, the *Resource Limit Facility* (or RLF), also known as the *DB2 Governor* since DB2 V.2, limits the amount of CPU time an SQL statement can take, which prevents SQL statements from making excessive requests. Up until V.5, the function of RLF was mainly to prevent dynamic SQL statements from consuming too much CPU by specifying a maximum number of service units a query could use before it is stopped. This function is known as *reactive governing*. However, the resources used to get to the point where the query is stopped by RLF, are wasted. In addition, DB2 may use additional resources to back out the changes done so far.

With V.6 of DB2 UDB for OS/390, the *predictive governing* function of the RLF provides an estimate of the processing cost of SQL statements *before they run*. The cost estimate is expressed as a number of CPU milliseconds and service units (SUs).

Note

Statement cost estimation is supported for both dynamic and static SELECT, INSERT, UPDATE or DELETE, but predictive governing is only for dynamic statements.

4.4.2.1 Cost estimation

To predict the cost of an SQL statement, you execute EXPLAIN to put information about the statement cost in **DSN_STATEMNT_TABLE**.

The governor controls only the dynamic SQL manipulative statements SELECT, UPDATE, DELETE, and INSERT. Each dynamic SQL statement used in a program is subject to the same limits. The limit can be a *reactive governing limit* or a *predictive governing limit*. If the statement exceeds a reactive governing limit, the statement receives an *error SQL code*. If the statement exceeds a predictive governing limit, it receives a *warning or error SQL code*.

Creating, populating, and interpreting the contents of **DSN_STATEMNT_TABLE** is done by the DBA or system administrator; you can establish the limits for individual plans or packages, for individual users, or for all users who do not have personal limits.

Each company has its own procedures defined by your installation for adding, dropping, or modifying entries in the Resource Limit Specification table.

4.4.2.2 How QMF for Windows handles predictive governing

If your installation uses predictive governing, QMF for Windows will check for the +495 and -495 SQLCODEs that predictive governing can generate after a PREPARE statement executes.

- Warning prompt (+495 SQL Error)
- Error message - Exceeded Limit (-495 SQL Warning)

4.4.3 Large Object (LOB)

DB2 UDB has a data type called Large Object (LOB), able to store “non-traditional” data such as images, video, and sound inside a database table. QMF for Windows Version 6.1 does not support this data type.

If the database that will be accessed using QMF for Windows contains tables that have LOB fields defined, the database administrator might decide either to prevent access to these tables in general, or to create a view of these tables, omitting the LOB column.

4.4.4 QMF linear procedures

With QMF for Windows, you can create only one type of procedure to run QMF commands: a *linear procedure* to run a series of QMF commands with a single RUN command.

A QMF linear procedure is a QMF for Windows object that, instead of containing SQL commands, contains procedure commands. Where SQL manipulates data, procedure commands manipulate QMF objects (tables, queries, forms and even other procedures). One way to define procedures is that the user is automating actions that would normally be done by clicking buttons and menu selections.

Procedures are sets of commands that enable the DBA or other users to run queries, print reports, import and export data, as well as perform other functions. Like any other QMF object, procedures can be stored at the database server, or saved in a file locally or on a file server. All commands issued through procedures are governed by the resource limits you already configured.

NOTE

In QMF for Windows you cannot create a *procedure with logic* to run a series of QMF commands like in the host QMF where the commands are run based on REXX logic you add to the procedure. QMF for Windows does not support REXX procedures, but only forms calculations using IBM Object REXX.

For more information on QMF for Windows Procedures, please see 6.4.4, “Procedures” on page 196.

4.4.5 Command line mode

Settings and actions can be defined to take effect when QMF for Windows is started. These parameters are defined on the QMF for Windows command line. They can be used to preset settings, or to run unattended sessions. The possible parameters and their function are:

- **/IServer:servername:** The /IServer parameter defines the server where the startup procedure specified on the /IProcName parameter is stored.
- **/IProcName:procedurename:** The /IProcName parameter defines the name of a procedure stored at a database server to run after starting QMF for Windows. To use the /IProcName parameter, you must also specify the /IServer parameter, if the server is different than the server the procedure has been created on.

- **/IProcFile:procedurefile:** The /IProcFile parameter defines the location and name of a locally stored procedure file to run after starting QMF for Windows.
 - **/UserID:userID:** The /UserID parameter defines the user ID to use when running a procedure specified with the /IProcName or /IProcFile parameters. It is used in conjunction with the /IPassword parameter.
 - **/IPassword:password:** The /IPassword parameter defines the password of the user specified with the /UserID parameter.
- Note:** The /IPassword parameter includes the user's password in plain text.
- **/Batch:** The /Batch parameter ends the current session of QMF for Windows and closes the application, after running any procedure specified on the command line.
 - **&variablename=variablevalue:** The &variablename= parameter defines or updates global variable values for use in any procedure or query being run. Any number of variables can be defined.

Following is a simple example to show the command line functionality of QMF for Windows:

1. Create a procedure such as the one shown in Figure 55.

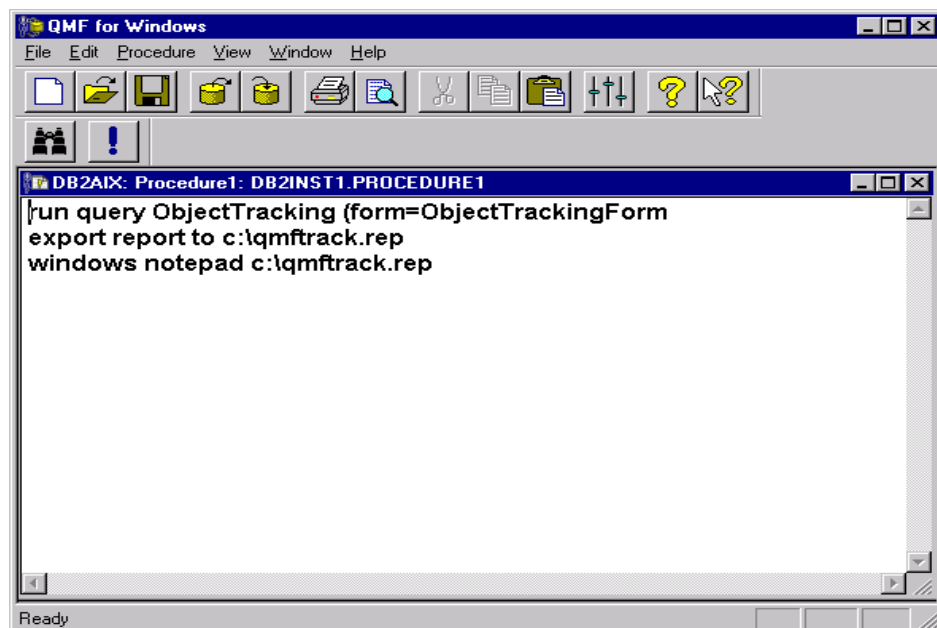


Figure 55. Procedure to be scheduled

2. Save the procedure at the server, or as a local file on your system (for example, C:\procedure1.prc). The sample shown above will execute a query called ObjectTracking using the Form called ObjectTrackingForm for report generation. It will then export the report to a file called qmftrack.rep at the local system in the c:\ directory. The last statement, windows notepad c:\qmftrack.rep, will open the saved report file using the Windows Notepad application.
3. You may now either execute the procedure saved at the server, or the one saved in a local file.

- To execute the procedure saved at the server, issue the following statement from the system command prompt:

```
"C:\Program Files\IBM\QMF for Windows\qmfwin.exe" /IServer:DB2AIX  
/IProcname:DB2INST1.Procedure1 /IUserID:db2inst1 /IPassword:db2inst1
```

Make sure to include the double quotes around the path specification within the statement. They are only required if some of the statements within the path specification have spaces in their names.

- To execute the procedure saved in a local file issue the following statement from the system command prompt:

```
"C:\Program Files\IBM\QMF for Windows\qmfwin.exe" /IServer:DB2AIX  
/IProcfile:"c:\Procedure1.prc" /IUserID:db2inst1 /IPassword:db2inst1
```

Make sure to include the double quotes around the path specification within the statement if necessary. They are required if some of the names within the path specification contain spaces.

4.4.6 Scheduling with Windows NT

As QMF for Windows is a very good administrative tool, the database administrator might find it useful to automate certain queries and procedures to be performed automatically every night. In order to do this, a procedure needs to be created and then be scheduled to execute at a predefined point in time. QMF for Windows does not have its own schedule, but it does support the Windows NT native scheduler. The following shows how to schedule certain queries and procedures to be executed using the Windows NT scheduler.

1. **Create the procedure to be executed:** The first step will be to either create a new procedure containing all the tasks that are planned to be scheduled to run automatically, or select an existing one. A sample procedure is shown in Figure 55.

2. **Schedule with the Windows NT AT command:** The Windows NT operating system has its own scheduling mechanism that is usually started with the AT command. The AT command schedules commands and programs to run on a computer at a specified time and date. The Schedule service must be running to use the AT command. The following shows the AT command syntax:

```
AT [\\computername] [ [id] [/DELETE] | /DELETE [/YES]]
AT [\\computername] time [/INTERACTIVE]
    [ /EVERY:date[,...] | /NEXT:date[,...]] "command"
```

- **\\computername:** Specifies a remote computer. Commands are scheduled on the local computer if this parameter is omitted.
- **id:** Is an identification number assigned to a scheduled command.
- **/delete:** Cancels a scheduled command. If id is omitted, all the scheduled commands on the computer are canceled.
- **/yes:** Used with cancel all jobs command when no further confirmation is desired.
- **time:** Specifies the time when command is to run.
- **/interactive:** Allows the job to interact with the desktop of the user who is logged on at the time the job runs.
- **/every:date[,...]:** Runs the command on each specified day(s) of the week or month. If date is omitted, the current day of the month is assumed.
- **/next:date[,...]:** Runs the specified command on the next occurrence of the day (for example, next Thursday). If date is omitted, the current day of the month is assumed.
- **"command":** Is the Windows NT command, or batch program to be run.

To run the previous mentioned procedure, save the command in a file called C:\Procedure1.BAT and the syntax for the scheduler to run the procedure every Monday at 9 PM would be:

```
AT 9:00PM /EVERY:Monday "C:\Procedure1"
```

3. **Using the Windows NT scheduler:** Windows NT also has a graphical interface to perform the scheduling tasks. Figure 56 shows an example screen to schedule a procedure using the Windows NT graphical scheduling application.

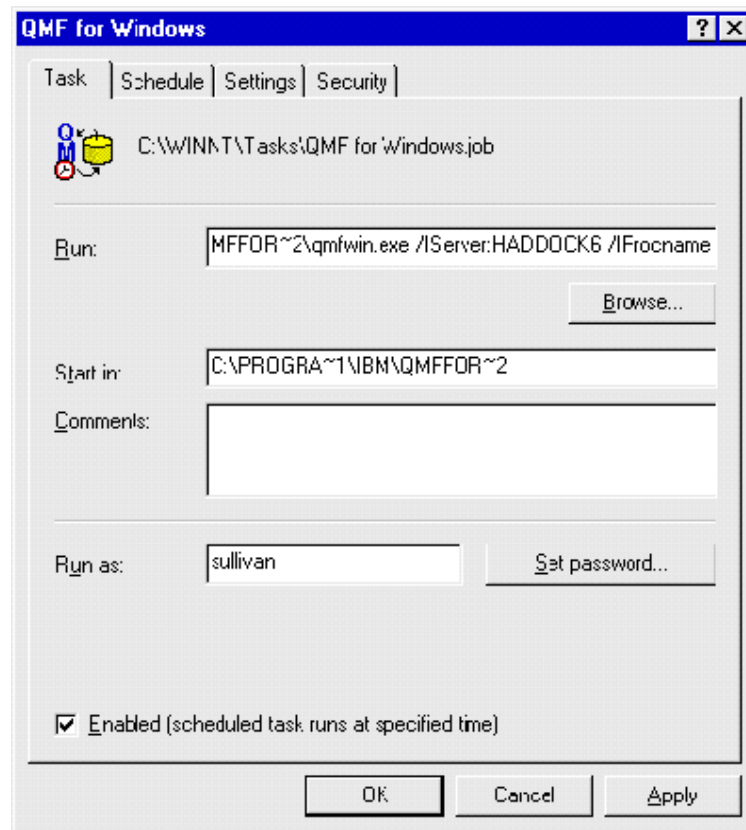


Figure 56. Windows NT scheduler

Chapter 5. Developer's guide

As seen in the prior chapters, QMF for Windows is a tool for building Enterprise Query Environments. We saw that in an Enterprise Query Environment, all queries are stored in a query repository, so that the vehicle for accessing the data and reports is centralized. This results in better organization, control, and reliability of the data.

But one problem is that most companies have their own applications for solving special needs such as calculations, marketing, human resources, and many others. Many of these applications submit queries to the database server to retrieve some data. Now, using QMF for Windows and storing all queries in a central repository, the applications no longer need to have their own SQL statements inside the code. Keeping the SQL text inside the application would not allow the query execution to be controlled by QMF for Windows. A better way to do this is to have the application use the queries stored in the query repository. That way, when the QMF administrator changes a query, the application is automatically changed and starts to retrieve the data using the new query.

This chapter shows how an application can access the QMF queries stored at a central repository and manipulate them. QMF for Windows has a set of Application Program Interfaces (APIs) that enable the application to execute the functions necessary to use the Enterprise Query Environment. For a complete understanding of how application development is done using the QMF for Windows API, this chapter shows some concepts of application development using QMF for Windows, and provides examples of programs using the APIs listed. For a complete API reference, see Appendix B, "QMF for Windows APIs" on page 293.

It is important to keep in mind that the examples presented in this chapter have no performance or interface considerations. These examples are for demonstration purposes only.

5.1 Application development concepts using QMF for Windows

Using the QMF for Windows APIs, we demonstrate some basic concepts that will help programmers of different programming languages to implement their applications.

5.1.1 Application Program Interface (API)

An API is a function with a pre-defined functionality that allows an application to execute that function without any knowledge of how it is performed. APIs can be seen as black boxes, that is, with well-defined inputs and outputs, but no vision of the process itself. APIs are very useful when dealing with device drivers for hardware and software packages.

Most of the programming languages designed for Windows can access the APIs very easily. The way this is done differs a lot from one programming language to another, but all of them have a way of calling the APIs. Once these APIs are included in the programming language environment, using them is like using a regular function in the program.

Any programming language that access the Windows API can also access the QMF APIs. Some examples of programming languages that can access the Windows APIs are Microsoft Visual Basic, Microsoft Visual C++, Borland C++, Borland Delphi, IBM Visual Age for Java, and many others.

In order to develop an application using QMF APIs, no extra middleware is necessary besides the one required for the end user. That means that there are no special prerequisites for developers. The basic middleware is, briefly, the middleware necessary for the network environment, the installation of QMF for Windows on the computer, and the programming language of your preference. For more details on installation and configuration of QMF for Windows, see Chapter 3, "Getting started" on page 25.

The same thing happens with the final application that you will distribute to your users. The only middleware necessary for the application to run is the middleware for the network and QMF installed within the environment.

5.1.2 QMF for Windows APIs or ODBC Applications?

Open Database Connectivity (ODBC) is a standard or open API for accessing databases. When using ODBC statements in a program, you can access data in a number of different databases, including Access, dBase, DB2, Excel, and text files. In addition to the ODBC software, a separate module or driver is needed for each database to be accessed. These drivers are provided by each database. The overall architecture when using ODBC is shown in Figure 57.

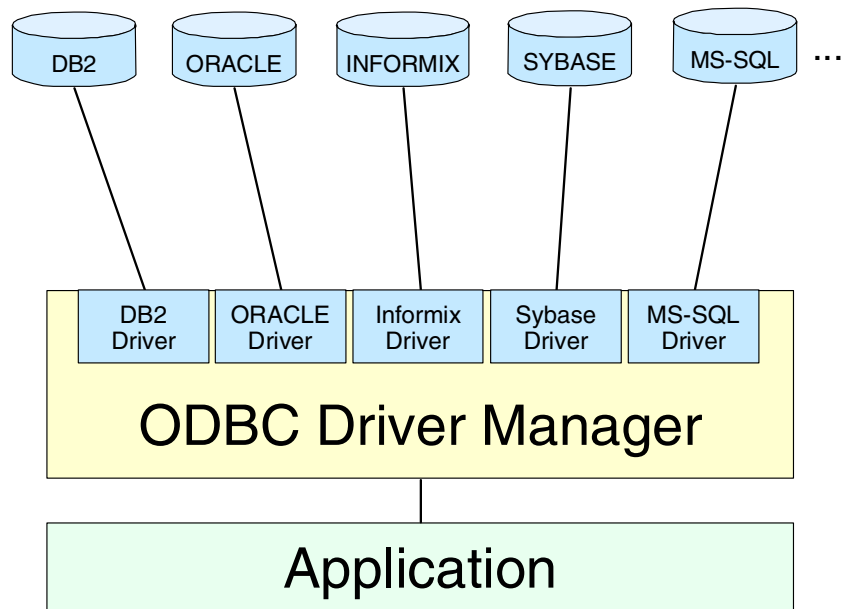


Figure 57. Application architecture when using ODBC

ODBC is based on and closely aligned with the Open Group standard SQL Call Level Interface. It allows programs to use SQL requests that will access databases without having to know the proprietary interfaces to the databases. ODBC handles the SQL request and converts it into a request that the individual database system understands.

In the newer distributed object architecture called Common Object Request Broker Architecture (CORBA), the Persistent Object Service (POS) is a superset of both the Call Level Interface and ODBC. When writing programs in the Java language and using the Java Database Connectivity (JDBC) application program interface, you can use a product that includes a JDBC-ODBC "bridge" program to reach ODBC-accessible databases.

As with many other standards, ODBC solves some problems and creates others. Some problems you may find when using ODBC include performance issues and error handling. Rather than using ODBC, you might want to connect directly to the database — QMF for Windows does that. It provides a direct access from the client desktop to the database server, meaning that there is no need to install the database client (CLI) or ODBC in order to access the database.

Resulting from that native access, the performance of the application is considerably better than ODBC applications. Other problems like obtaining the error code from the database are also solved so the application becomes more powerful and organized.

5.1.3 Synchronization Aspects

When you develop an application, one aspect that has to be very clear is synchronization. An application can be synchronous or asynchronous, which means the same as saying that an application is single-threaded or multi-threaded.

A thread is a single stream of execution within a program. *Multi-tasking* refers to the computer's ability to perform multiple jobs concurrently. For example, operating systems can run two or more programs at the same time. *Multi-threading*, on the other hand, is an extension of multi-tasking. But rather than multiple programs, multi-threading involves multiple threads of control running within a single program.

The advantages of multi-threading are that, without threads, an entire program can be held up by one CPU-intensive task. With threads the other tasks can continue processing without waiting for the CPU-intensive task to finish. For example, using a database administrative application, the user might be able to export tables (which can take several minutes or even hours) within one thread, but still be able to carry out other tasks such as creating new tables or retrieving information from other tables within another thread. The user does not need to wait for the first thread to finish. Note, however, that there might be a problem of controlling these threads — because the thread that was exporting the tables does not know that another thread may be just in the process of creating a new table, or, even worse, the second thread may try to drop a table while it is being exported from the first thread.

All of the QMF for Windows API functions are *synchronous*. This means that when a API is called in your application, it blocks, or does not return until the requested action completes. In other words, the code line just below the line calling the API will not be executed until the API completes its function. This implementation is desirable because it simplifies programming the application. However, if your application is single-threaded, it will not be able to respond to user input or perform screen refreshes while it is waiting for a QMF for Windows API function to return.

If it is necessary for the application to be *asynchronous*, the programmer has to create a new thread within the application and call the API from the new thread. This new thread will then be blocked, but all other threads will continue to execute normally. The programmer has to manage the timing and execution of the threads. When developing multi-threaded applications, keep in mind that the QMF for Windows API responds to one function call at a time from a client. That means you must wait for one function call to complete before making another, or create multiple instances of the QMF for Windows API (one for each thread using the API).

5.1.4 Database Connectivity

The QMF for Windows family is a set of tools that can connect directly only to the DB2 database family. In order to connect to other databases such as Oracle, Sybase, and MS-SQL, DB2 DataJoiner needs to be used as represented in Figure 58. It is even possible to access VSAM files or IMS files on the OS/390 environment using QMF for Windows and DB2 DataJoiner, but these data sources can only be opened for reading.

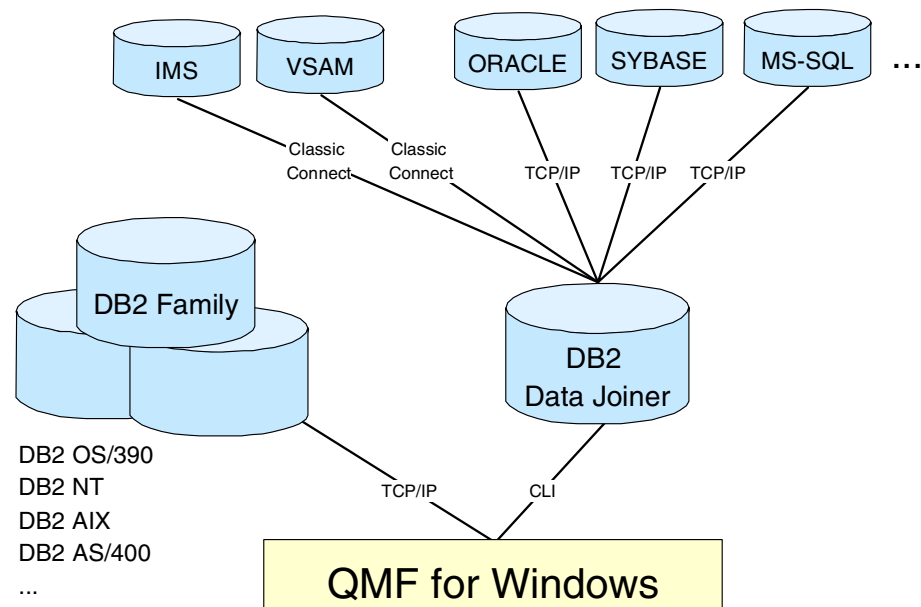


Figure 58. QMF database access

Each instance of the QMF for Windows API object represents a single unit of work and can only connect to a single database server at a time. This unit of work creates and uses the same connection to the database server for all database activities that are subject to a subsequent rollback or commit, including opening a query, fetching data, and executing other SQL statements.

To connect to multiple database servers simultaneously, you have to create multiple instances of the QMF for Windows API object, one for each database server.

To connect to multiple database servers serially (one after the other) using the same QMF for Windows API object, call `Commit()` or `Rollback()` and then `InitializeServer()`. The `Commit()` and `Rollback()` functions close the current connection to the database server while the `InitializeServer()` function opens a new connection. Calling these functions ends the current unit of work, and subsequent calls start a new unit of work.

Due to the fact that all database activities occur within the same connection, creating more than one query in a given instance of the QMF for Windows API object by calling `InitializeQuery()` two or more times, they will share the same single connection.

QMF for Windows creates and uses a second connection to the database in order to handle administrative database activities, for example, retrieving QMF information. This second connection is necessary to support a consistent rollback and commit mechanism for client applications. The QMF for Windows API object automatically handles these connections to the database. However, if your system administrator has established a limit for the number of connections allowed, remember that each instance of the QMF for Windows API object may use two connections.

5.1.5 Web Development

When building an Enterprise Query Environment, it is very likely that some of the queries have to be executed throughout the company, and using the Web architecture (Internet or intranet) is a nice way of doing that. QMF for Windows provides various ways of doing this (see Chapter 7, “Web considerations” on page 253 for more information on this subject).

One way of allowing the queries to be accessed through the Web is by developing a Web application, using for example CGI, ASP, Servlet, or one of many other technologies. To develop such an application, the programmer can still use the QMF APIs, so that the interface between the application and QMF is the same as when developing a regular application.

5.2 Main QMF for Windows APIs

QMF for Windows provides a total of 91 APIs. This section describes the most important ones. For a complete API reference, see Appendix B, “QMF for Windows APIs” on page 293.

5.2.1 GetServerList()

The first important API is GetServerList(). This API goes to the QMF for Windows server definition file (SDF) and lists the database servers that QMF is configured to access. If there is only one database server in the environment, or if the database server name is already known, it is possible to directly call the InitializeServer() API.

5.2.2 InitializeServer()

Once you have your database server name, it is necessary to initialize the connection to that server. That is what the InitializeServer() API does. To execute this API, it is necessary to pass the following parameters:

- **ServerName** — This name has to match the same name in the QMF for Windows SDF.
- **UserID** — This is the user ID for accessing the database.
- **Password** — This is the password of that user ID.
- **ForceDialog** — If the value of this parameter is not zero, then a dialog prompting the user ID and password will be shown. Otherwise, the dialog will only be shown if necessary, that is, if there are no values in the UserID and Password parameters.
- **SuppressDialog** — If the value of this parameter is not zero, then the dialog box prompting for the user information will never be shown. This is especially useful when developing applications for the Web, where the user ID and password are to be prompted for in a different way.

5.2.3 GetQMFOBJECTLIST()

Another important API is GetQMFOBJECTLIST(). After connecting to the server, it might be very important to know what queries, procedures, forms and tables are available. This API allows the objects to be listed. The parameters of this API are:

- Owner — The name of the owner of the objects. To include all objects regardless of the owner, this parameter must be left blank or contain the "%" character.
- Name — Name of the object. To include all objects regardless of the name, this parameter must be left blank or contain the "%" character.
- Type — Select the type of objects to be listed. There are four types of objects: Queries (2048), Forms (1024), QMF Procedures (512) and Tables (256). To list more than one type at the same time, it is necessary to sum the values of the desired types.
- List — That is the pointer in which the result will be stored. There are various ways of passing a pointer as a parameter, depending on the programming language.

5.2.4 InitializeQuery()

This API initializes a query for execution. It has two parameters:

- SourceType — There are three types of queries: the ones passed directly to the API (0), the ones stored on the server (1), and the ones stored into text files (2).
- Source — If the SourceType is 0, then the Source parameter must have the SQL statement; if the SourceType is 1, then the Source parameter must have the owner and the name (Owner.Name) of the query stored in the server; and if the SourceType is 2, then the Source parameter must have the file name.

The result of this API is the QueryID, a number that will identify the Query exclusively. Many other APIs require the QueryID as input parameter.

5.2.5 GetQueryText()

This API returns the SQL statement from a QMF query passed in the input parameter QueryID. It is very important for applications which have to manipulate or show the SQL statement. In an Enterprise Query Environment, all the queries will be stored together and all applications should be able to retrieve any query at any time.

5.2.6 GetQueryVerb()

This API returns the verb of the query passed in the input parameter QueryID. The verb of a query might be SELECT, INSERT, UPDATE, DELETE, CALL, and others. In some cases it is extremely important that the application knows what kind of action the query is taking, because queries that return values may have to be manipulated, and those that only execute a certain action may not return any value at all.

5.2.7 SaveQMFQuery()

This API allows the application to create new queries or to modify existing queries on the server.

It has five input parameters:

- OwnerAndName — A string containing the owner and name, separated by a period, of the query you want to save; for example, "John.Query2".
- Text — The SQL statement.
- Comment — A string with any comment necessary.
- Replace — If there already exists a query with the same owner and name as on the OwnerAndName parameter, the query might be replaced or not. If this parameter is zero the existing query will not be replaced, and an error will occur. Otherwise, if this parameter is not zero, the existing query will be replaced.
- Share — Indicates if the query is to be shared with other users. Zero indicates that this query is not to be shared. If not zero, the query can be shared.

5.2.8 Open()

This API will execute the query initialized by the InitializeQuery() API. Note that this API will only execute queries with the SELECT verb. Queries that use the UPDATE, INSERT, DELETE, CALL and others verbs have to be executed using the Execute() API.

This API has three input parameters:

- QueryID — The QueryID as returned from InitializeQuery().
- RowLimit — Maximum number of rows allowed to be retrieved. Zero indicates that no limit is enforced. This number will not overwrite the row limit established by the QMF Administrator.

- **FetchAllRows** — If TRUE, indicates that all rows will be fetched at once, the result will be stored in the client, and the database will be free for other users. If FALSE, the cursor stays open until the Close() API is executed.

5.2.9 GetColumnCount()

This API returns the number of columns that the query passed in the input parameter QueryID has. This value is very useful for handling the return value of the FetchNextRow() API within loops.

5.2.10 GetColumnHeadings()

This API returns the column names in the Headings parameter as a pointer for the query specified in the input parameter QueryID. The pointer has to be handled after the API is executed. The way that pointers are handled depends a lot depending on the programming language used. This function is especially useful to display the columns names for the end user or for manipulating columns values within the application.

5.2.11 FetchNextRow()

This API has to be used after the Open() API. It fetches the next result row of a query specified in the QueryID input parameter. The result data in the row is returned in the pointer specified in the second input parameter Row. This pointer has to be handled after the API executes. The way that pointers are handled differs a lot, depending on the programming language. If the API executes successfully, the result value is zero.

When the end of the result set has been reached (there are no more rows to fetch) the result is empty and the return value of the API is -1.

One important aspect of this API is the data types supported. This types include string, float, double, short, long, and binary.

5.2.12 Close()

This API closes a query passed in the input parameter QueryID. If there is a cursor open for the query, the cursor is closed, freeing the database for other users. This function does not terminate the connection to the database server. Since the connection remains open, no rollback or commit is performed.

5.3 Using Visual Basic with QMF for Windows

Visual Basic is a visual programming language developed by Microsoft that allows you to easily create Windows applications. This chapter shows the steps that need to be taken before using QMF for Windows APIs with Visual Basic, and also shows some examples of programs.

5.3.1 Getting Started

Before you start using the QMF for Windows APIs, there are a few steps required for Visual Basic to be able to access the QMF for Windows APIs. These steps, which must be taken when creating a new project, are as follows:

1. Within Visual Basic, go to the **Project** menu and then to the **References** item. The screen shown in Figure 59 will appear.
2. Add a reference to the QMF for Windows file **qmfwin.tlb** if the QMF for Windows 6.1 Type Library is not found on the list.

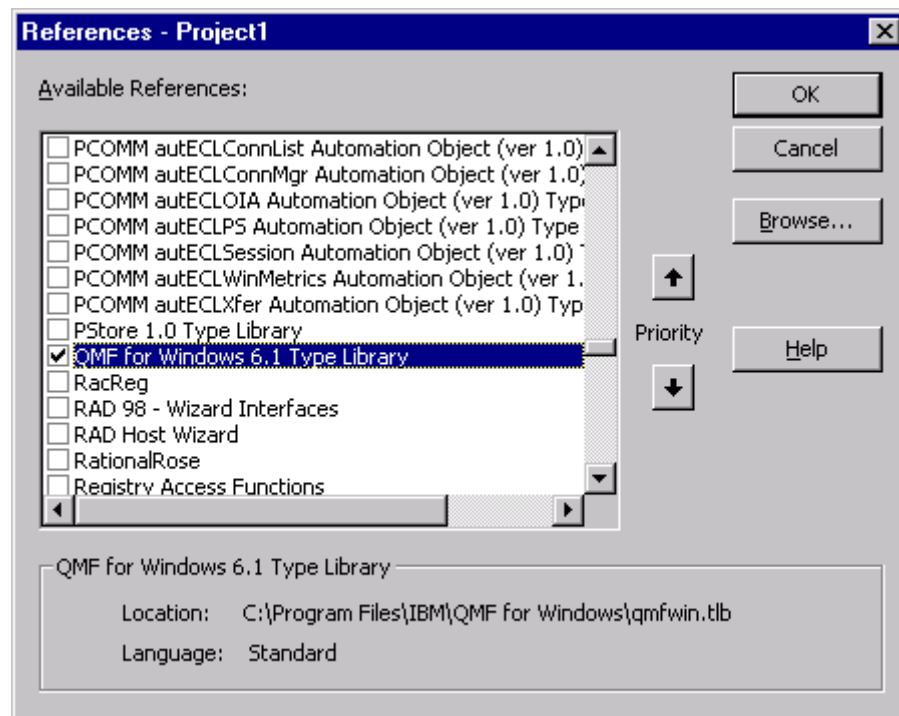


Figure 59. Visual basic configuration

3. Click **OK**.
4. On the source code of your program, use the Dim statement as shown below to create a QMF object:

```
Dim QMFWin As New QMFWin
```

or use the CreateObject statement:

```
Dim QMFWin As Object  
Set QMFWin = CreateObject("QMFWin.Interface")
```

All the APIs are property and methods of the QMFWin object.

5.3.2 Application examples

For a complete understanding and illustration of how Visual Basic programmers use the QMF for Windows APIs, this chapter provides some examples with very basic functionality, like listing the servers, listing the queries, executing a query, and saving a query. Although these examples are very simple, they show how the interface between a Visual Basic application and QMF for Windows works.

Three examples will be used to illustrate the QMF for Windows APIs:

1. Execute a query stored on the server
2. Execute a query stored in a file
3. Execute an SQL statement

All examples shown use these two common procedures:

- **Clear Grid** — Clear all cells of the grid passed on the input parameter Grid. The source code of that procedure is listed below:

```
Public Sub ClearGrid(ByRef Grid As MSFlexGrid)  
  
Grid.Clear  
Grid.Cols = 2  
Grid.Rows = 2  
  
End Sub
```

- **DataIntoGrid** — Fetches the data from a specified query and displays the data into a grid. The query has to be open before calling this procedure. This is done using the FetchNextRow() API. The following example shows the source code for this procedure.


```

Public Sub DataIntoGrid(ByRef Grid As MSFlexGrid, QueryNumber As
Integer)
'This procedure fetches all the data in the QMF Query passed
'on the parameter QueryNumber and display the data on the
'grid passed on the parameter Grid

Dim ColumnHeadings As Variant
Dim i As Integer
Dim myColumnCount As Integer
Dim myRow As Variant
Dim FetchResult As Integer

'Clears the Grid
Call ClearGrid(Grid)

'try to get the number of columns
'of the selected query
myColumnCount = QMFWin.GetColumnCount(QueryNumber)
If myColumnCount <= 0 Then
    MsgBox ("Could not count numbers for columns. " +
QMFWin.GetLastErrorString())
Else
    'if successful set the grdResult with the
    'appropriate numbers of columns
    GrdResult.Cols = myColumnCount
    'try to get the column headers
    If QMFWin.GetColumnHeadings(QueryNumber, ColumnHeadings) <> 0 Then
        MsgBox ("Could not get columns headings. " +
QMFWin.GetLastErrorString())
    Else
        'if successful display the column headings on the grdResult
        Grid.Row = 0
        For i = 0 To (Grid.Cols - 1)
            Grid.Col = i
            Grid.Text = ColumnHeadings(i)
        Next

        Grid.Row = 1

        'try to fetch all the rows from the query
        FetchResult = QMFWin.FetchNextRow(QueryNumber, myRow)
        While FetchResult = 0
            For i = 0 To Grid.Cols - 1
                Grid.Col = i
                Grid.Text = myRow(i)
            Next
            Grid.Rows = Grid.Rows + 1
        End While
    End If
End If
End Sub

```

```

        Grid.Row = Grid.Rows - 1
        FetchResult = QMFWin.FetchNextRow(QueryNumber, myRow)
    Wend

    If FetchResult <> -1 Then
        'if the result of the FetchNextRow API
        'is different than -1, that means that
        'an error occurred
        MsgBox ("Could not fetch next row. " +
    QMFWin.GetLastErrorString())
    Else
        'otherwise, the loop end normally

        'clear the last row
        Grid.Rows = Grid.Rows - 1

    End If
End If
End If
End Sub

```

5.3.3 Example 1 — Execute a query stored on the server

The objective of this program example is to show how to execute a query that is stored at a database server using the QMF for Windows APIs.

In this example, the following objects were added to a form:

- lstServers — ListBox
- lstQueries — ListBox
- txtSQLStatement — TextBox
- grdResult — MSFlexGrid
- lblState — Label

The first thing that this example does is to list all the servers that are configured in the QMF for Windows SDF and show the servers found in a list named `lstServers`. This is done on the event `Form_Load` as listed below:

```

Private Sub Form_Load()
Dim myServerList As Variant
Dim myServer As Variant

'try to get the server list and place it on the ServerList variable
lblState = "Getting Server List..."
If QMFWin.GetServerList(myServerList) <> 0 Then
    'if not successful, display error message

```

```

        lblState = "Could not find server list. " + QMFWin.GetLastErrorString()
        lblState.Refresh
        MsgBox (lblState.Caption)
Else
    'if successful, display the list of server into the lstServers
    lblState = "Displaying Server List"
    lblState.Refresh
    If Not IsEmpty(myServerList) Then
        For Each myServer In myServerList
            lstServers.AddItem (myServer)
        Next
    Else
        lblState = "No server found."
        lblState.Refresh
        MsgBox (lblState.Caption)
    End If
End If
End Sub

```

After that, the user has to select the desired server by double-clicking on the server name. When this happens, it is necessary to initialize the server using the InitializeServer() API. The API will then prompt the user for a userID and password. Once the server is initialized, all existing queries are listed using the GetObjectList() API and displayed in the lstQueries object on the top right of the screen. The source code of that functionality is listed below:

```

Private Sub lstServers_DblClick()
Dim myQueryList As Variant
Dim myQuery As Variant

'try to initialize the server selected on the lstServer
lblState = "Initializing Server..."
lblState.Refresh

If QMFWin.InitializeServer(lstServers.List(lstServers.ListIndex), "", "",
True) <> 0 Then
    'if not successful display error message
    lblState = "Could not initialize server "
    lblState = lblState + lstServers.List(lstServers.ListIndex)
    lblState = lblState + ". "
    lblState = lblState + QMFWin.GetLastErrorString()
    lblState.Refresh
    MsgBox (lblState.Caption)
Else
    'if successful try to list all queries (2048 parameter) in that server
    lblState = "Getting Existing Queries..."
    lblState.Refresh

```

```

If QMFWin.GetQMFOBJECTList("", "", 2048, myQueryList) <> 0 Then
    'if not successful display error message
    lblState = "Could not list queries. " + QMFWin.GetLastErrorString()
    lblState.Refresh
    MsgBox (lblState.Caption)
Else
    'if successful, display all the queries into lstQueries
    lblState = "Displaying Query List"
    lblState.Refresh
    If Not IsEmpty(myQueryList) Then
        For Each myQuery In myQueryList
            lstQueries.AddItem (myQuery)
        Next
    Else
        lblState = "No Queries found."
        lblState.Refresh
        MsgBox (lblState.Caption)
    End If
End If
End Sub

```

The user now has to select one from the query list by double-clicking on the desired query. It is then necessary to initialize the query using the InitializeQuery() API, execute the query using the Open() API, list all rows in that query on the grid using the procedure DataIntoGrid listed in the beginning of 5.3, "Using Visual Basic with QMF for Windows" on page 135, and close the query using the Close() API. Following is the code to do this:

```

Private Sub lstQueries_DblClick()

    'try to initialize the query selected by the user
    lblState = "Initializing Query..."
    lblState.Refresh
    QueryNumber = QMFWin.InitializeQuery(1,
    lstQueries.List(lstQueries.ListIndex))
    If QueryNumber < 0 Then
        'if not successful display error message
        lblState = "Could not initialize query "
        lblState = lblState + lstQueries.List(lstQueries.ListIndex)
        lblState = lblState + ". "
        lblState = lblState + QMFWin.GetLastErrorString()
        lblState.Refresh
        MsgBox (lblState.Caption)
    Else
        'if successful
    End If
End Sub

```

```

'Shows the SQL statement
txtSQLText.Text = QMFWin.GetQueryText (QueryNumber)

'try to open the selected query
lblState = "Executing Query..."
lblState.Refresh
If QMFWin.Open(QueryNumber, 0, False) <> 0 Then
    'if not successful display error message
    lblState = "Could not open the query "
    lblState = lblState + lstQueries.List (lstQueries.ListIndex)
    lblState = lblState + ". "
    lblState = lblState + QMFWin.GetLastErrorString()
    lblState.Refresh
    MsgBox (lblState.Caption)
Else
    lblState = "Displaying Result..."
    lblState.Refresh

    'clear the grdResult
    Call ClearGrid(GrdResult)

    'Display the data in the grid
    Call DataIntoGrid(GrdResult, QueryNumber)

    'try to close the query
    lblState = "Closing the Query..."
    lblState.Refresh
    If QMFWin.Close(QueryNumber) <> 0 Then
        lblState = "Could not close query." + QMFWin.GetLastErrorString()
        lblState.Refresh
        MsgBox (lblState.Caption)
    Else
        lblState = ""
        lblState.Refresh
    End If
End If
End If
End Sub

```

The final result of this example is shown in Figure 60.

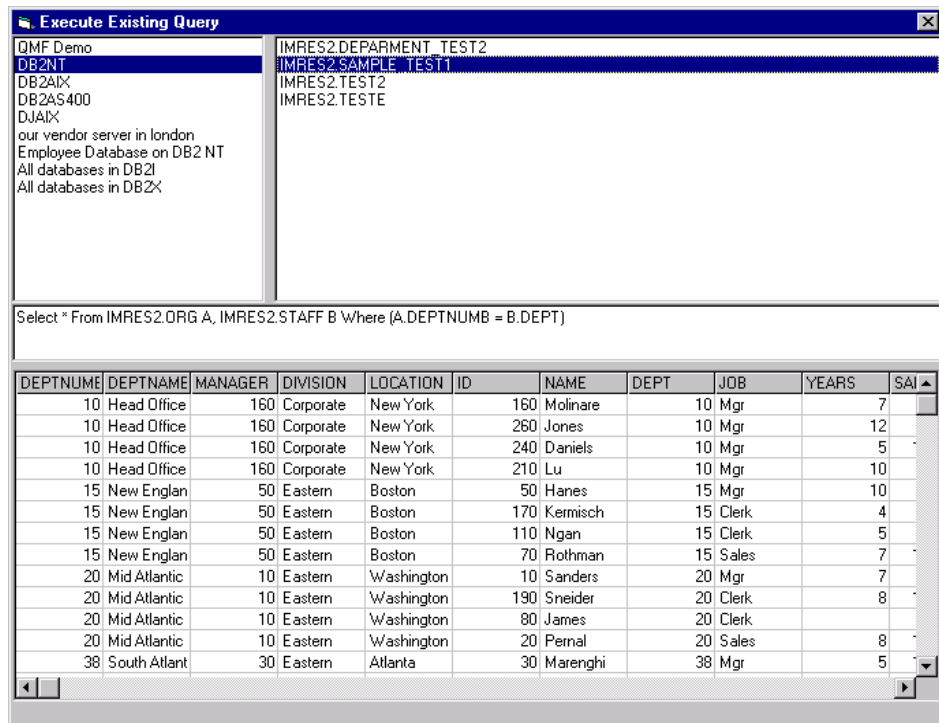


Figure 60. Visual basic — execute query stored on the server

5.3.4 Example 2 — Execute a query stored in a file

When using QMF for Windows, it is possible to save the SQL statement in text files. The objective of this example is to execute a query that was previously saved as this kind of file. Other applications might also save SQL statements as text files, and the following example is able to run these statements as well, if the statements are written in standard SQL.

Note:

This example is not able to execute queries that were saved as Prompted Queries from QMF for Windows (extension *.pq). That is because this kind of saved query does not contain the SQL statement in its standard format. There is no way of executing this kind of query using the QMF APIs. In order to execute them, it is necessary to convert them to SQL statements and save them as Query Files (*.qry).

In this example, the following objects were added to a form:

- lstServers — ListBox
- lstDrives — DriveListBox
- lstDir — DirListBox
- lstFiles — FileListBox
- grdResult — MSFlexGrid
- lblState — Label

The first thing that this example does is to list all the servers that are configured in the QMF for Windows SDF and show them in the list named `lstServers`. This is done on the event `Form_Load`, as listed below:

```
Private Sub Form_Load()  
Dim myServerList As Variant  
Dim myServer As Variant  
  
'try to get the server list and place it  
'on the ServerList variable  
lblState = "Getting Server List..."  
If QMFWin.GetServerList(myServerList) <> 0 Then  
    'if not successful, display error message  
    lblState = "Could not find server list. "  
    lblState = lblState + QMFWin.GetLastErrorString()  
    lblState.Refresh  
    MsgBox (lblState.Caption)  
Else  
    'if successful, display the list of server  
'into the lstServers  
    lblState = "Displaying Server List"  
    lblState.Refresh  
    For Each myServer In myServerList  
        lstServers.AddItem (myServer)  
    Next  
    lblState = ""  
    lblState.Refresh  
End If  
End Sub
```

After that, the user has to select the desired server by double-clicking on the server name. It then is necessary to initialize the server using the `InitializeServer()` API. This API will prompt the user for his userID and password. Once the server is initialized, the driver, directory, and file lists will be enabled for the user to select the file. The source code of that functionality is listed below:

```

Private Sub lstServers_DblClick()
Dim myQueryList As Variant
Dim myQuery As Variant

'Clear the grid
Call ClearGrid(GrdResult)

'try to initialize the server selected on the lstServer
lblState = "Initializing Server..."
lblState.Refresh

If QMFWin.InitializeServer(lstServers.List(lstServers.ListIndex) _
, "", "", True) <> 0 Then
'if not successful display error message
lblState = "Could not initialize server "
lblState = lblState + lstServers.List(lstServers.ListIndex)
lblState = lblState + ". "
lblState = lblState + QMFWin.GetLastErrorString()
lblState.Refresh
MsgBox (lblState.Caption)
Else
'Enables the selection of files after the server is selected
lstDrives.Enabled = True
lstDrives.BackColor = -2147483643
lstDir.Enabled = True
lstDir.BackColor = -2147483643
lstFiles.Enabled = True
lstFiles.BackColor = -2147483643
lblState = ""
lblState.Refresh
End If
End Sub

```

The user now has to select the text file that contains the SQL statement. In this example, the user will only be able to see files with the extension *.qry, which is the extension under which QMF for Windows saves its SQL queries. The following code must be implemented to link the drive, directory and file lists, as shown below:

```

Private Sub lstDrives_Change()
'Changes the path property on the directory list object
lstDir.Path = lstDrives.Drive

End Sub

Private Sub lstDir_Change()
'Changes the path of the File List Objects

```



```
lstFiles.Path = lstDir.Path
```

```
End Sub
```

After the user has found the correct directory, he must double-click on the file name in the list. It is then necessary to initialize the query using the InitializeQuery() API, execute the query using the Open() API, display the result on the grid using the procedure DataIntoGrid listed in the beginning of 5.3, "Using Visual Basic with QMF for Windows" on page 135, and close the query using the Close() API. The code to do that is listed below:

```
Private Sub lstFiles_DblClick()  
  
    'try to initialize the query selected by the user  
    lblState = "Initializing Query..."  
    lblState.Refresh  
  
    'checks if the current directory is the root directory.  
    'That is because the lstDir.Path will not return the "\"  
    'character if the current directory is not the root  
    If Len(lstDir.Path) <> 3 Then  
        QueryNumber = QMFWin.InitializeQuery(2, lstDir.Path + "\" _  
            + lstFiles.List(lstFiles.ListIndex))  
    Else  
        QueryNumber = QMFWin.InitializeQuery(2, lstDir.Path + _  
            lstFiles.List(lstFiles.ListIndex))  
    End If  
  
    If QueryNumber < 0 Then  
        'if not successful display error message  
        lblState = "Could not initialize query "  
        lblState = lblState + lstFiles.List(lstFiles.ListIndex)  
        lblState = lblState + ". "  
        lblState = lblState + QMFWin.GetLastErrorString()  
        lblState.Refresh  
        MsgBox (lblState.Caption)  
    Else  
        'if successful try to open the selected query  
        'without any limits of number of rows  
        lblState = "Executing Query..."  
        lblState.Refresh  
        If QMFWin.Open(QueryNumber, 0, False) <> 0 Then  
            'if not successful display error message  
            lblState = "Could not open the query "  
            lblState = lblState + lstFiles.List(lstFiles.ListIndex)  
            lblState = lblState + ". "  
            lblState = lblState + QMFWin.GetLastErrorString()  
        End If  
    End If  
End Sub
```

```

        lblState.Refresh
        MsgBox (lblState.Caption)
    Else
        lblState = "Displaying Result..."
        lblState.Refresh

        'clear the grdResult
        Call ClearGrid(GrdResult)

        'Display the data in the grid
        Call DataIntoGrid(GrdResult, QueryNumber)

        'try to close the query
        lblState = "Closing the Query..."
        lblState.Refresh
        If QMFWin.Close(QueryNumber) <> 0 Then
            lblState = "Could not close query."
            lblState = lblState + QMFWin.GetLastErrorString()
            lblState.Refresh
            MsgBox (lblState.Caption)
        Else
            lblState = ""
            lblState.Refresh
        End If
    End If
End If
End Sub

```

The final result of this example is shown in Figure 61.

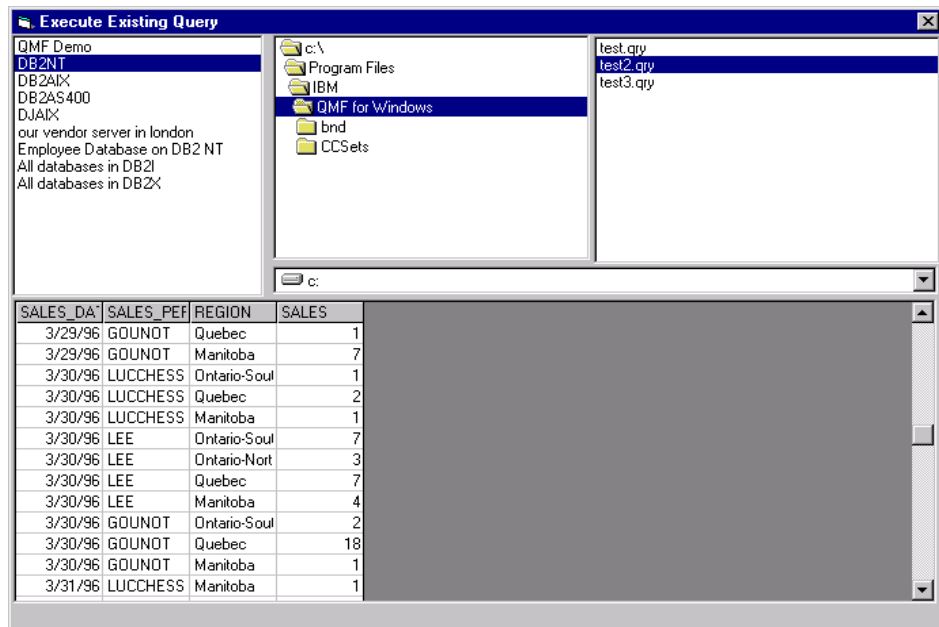


Figure 61. Visual basic — execute query stored on a file

5.3.5 Example 3 — Execute an SQL statement

The objective of this example is to execute SQL directly, which means that an SQL application has either created itself or prompted the user to input it. Always remember that, in an Enterprise Query Environment, all queries are stored in a query repository. It is rare to allow end users to input their own query. This kind of API is more often used in applications for administrative purposes.

In this example, the following objects were added to a form:

- lstServers — ListBox
- txtSQLStatement — RichTextBox
- btClear — CommandButton
- btExecute — CommandButton
- grdResult — MSFlexGrid
- lblState — Label

The first thing that this example does is to list all the servers that are configured in the QMF for Windows SDF and show these servers in a list named `lstServers`. This is done on the event `Form_Load`, as listed below:

```
Private Sub Form_Load()  
Dim myServerList As Variant  
Dim myServer As Variant  
  
'try to get the server list and place it  
'on the ServerList variable  
lblState = "Getting Server List..."  
If QMFWin.GetServerList(myServerList) <> 0 Then  
    'if not successful, display error message  
    lblState = "Could not find server list. "  
    lblState = lblState + QMFWin.GetLastErrorString()  
    lblState.Refresh  
    MsgBox (lblState.Caption)  
Else  
    'if successful, display the list of server  
'into the lstServers  
    lblState = "Displaying Server List"  
    lblState.Refresh  
    If Not IsEmpty(myServerList) Then  
        For Each myServer In myServerList  
            lstServers.AddItem (myServer)  
        Next  
    Else  
        lblState = "No servers found."  
        lblState.Refresh  
        MsgBox ("No servers found.")  
    End If  
End If  
End Sub
```

After that, the user has to select the desired server by double-clicking on the server name. It then is necessary to initialize the server using the `InitializeServer()` API. The API will then prompt the user for his `userID` and password. Once the server is correctly initialized, the SQL text box will be enabled for the user to enter the SQL statement. The source code of that functionality is listed below:

```
Private Sub lstServers_DblClick()  
Dim myQueryList As Variant  
Dim myQuery As Variant
```

```

'try to initialize the server selected on the lstServer
lblState = "Initializing Server..."
lblState.Refresh

If QMFWin.InitializeServer(lstServers.List(lstServers.ListIndex) _
    , "", "", True) <> 0 Then
    'if not successful display error message
    lblState = "Could not initialize server "
    lblState = lblState + lstServers.List(lstServers.ListIndex)
    lblState = lblState + ". "
    lblState = lblState + QMFWin.GetLastErrorString()
    lblState.Refresh
    MsgBox (lblState.Caption)
Else
    btClear.Enabled = True
    btExecute.Enabled = True
    btSave.Enabled = True
    txtSQLStatement.Enabled = True
    txtSQLStatement.BackColor = -2147483643
    txtSQLStatement.SetFocus
    lblState = "Server"
    lblState.Refresh
End If
End Sub

```

Now, the user has to type the SQL statement. In this example, the statement will be "SELECT * FROM EMPLOYEE" where Employee is an example of a table that QMF creates when it is installed. After that the user can execute the SQL by clicking on the Execute button, or clear the entry by clicking on the Clear button. In case the user clicks on the Clear button, the following code must be implemented.

```

Private Sub btClear_Click()

'clear the txtSQLStatement
txtSQLStatement.Text = ""
txtSQLStatement.Refresh

'clear the grdResult
Call ClearGrid(GrdResult)

txtSQLStatement.SetFocus

End Sub

```

Otherwise, if the user clicks on the Execute button, it is necessary to Initialize the query using the InitializeQuery() API, execute the query using the Open() API, display the result on the grid using the procedure DataIntoGrid listed in the beginning of 5.3, "Using Visual Basic with QMF for Windows" on page 135, and close the query using the Close() API. If the statement typed is not a valid statement the Open() API will fail and the application has to handle the error. This is shown in the code below:

```
Private Sub btExecute_Click()
'Execute the SQL Statement written on the txtSQLStatement

If txtSQLStatement.Text <> "" Then
'if there is some SQL statement the
QueryNumber = QMFWin.InitializeQuery(0, txtSQLStatement.Text)
If QueryNumber < 0 Then
'if not successful display error message
lblState = "Could not initialize the query "
lblState = lblState + txtSQLStatement.Text
lblState = lblState + ". "
lblState = lblState + QMFWin.GetLastErrorString()
lblState.Refresh
MsgBox (lblState.Caption)
txtSQLStatement.SetFocus
Else
'if successful try to open the selected query without
'any limits of number of rows
lblState = "Executing Query..."
lblState.Refresh
If QMFWin.Open(QueryNumber, 0, False) <> 0 Then
'if not successful display error message
lblState = "Could not open the query "
lblState = lblState + txtSQLStatement.Text
lblState = lblState + ". "
lblState = lblState + QMFWin.GetLastErrorString()
lblState.Refresh
MsgBox (lblState.Caption)
Else
lblState = "Displaing Result..."
lblState.Refresh

'clear the grdResult
Call ClearGrid(GrdResult)

'Display the data in the grid
Call DataIntoGrid(GrdResult, QueryNumber)

'try to close the query
```

```

        lblState = "Closing the Query..."
        lblState.Refresh
        If QMFWin.Close(QueryNumber) <> 0 Then
            lblState = "Could not close query." _
                + QMFWin.GetLastErrorString()
            lblState.Refresh
            MsgBox (lblState.Caption)
        Else
            lblState = ""
            lblState.Refresh
        End If
    End If
End If
End Sub

```

Another important functionality of this example is the option to save the query on the server. That is done using the SaveQMQuery() API. When the user clicks on the Save button, the application will be prompted for the query owner, name, and comments of the query (if any). If the query is saved successfully, a message will be displayed; otherwise, the error message will appear. The code that does this is listed below:

```

Private Sub btSave_Click()
    Dim Owner As String
    Dim Name As String
    Dim Comment As String

    'if there is any text to save
    If txtSQLStatement.Text <> "" Then

        lblState = "Saving Query..."
        lblState.Refresh

        'prompt the user for Owner of the query
        Owner = InputBox("Please enter the Owner of the Query" _
            , "Owner")

        'prompt the user for Name of the query
        Name = InputBox("Please enter the Name of the Query" _
            , "Name")

        'prompt the user for Comments on the query
        Comment = InputBox("Please enter the Comment of the Query" _
            , "Comment")

        'try to save query on server
        If QMFWin.SaveQMQuery(Owner + "." + Name, txtSQLStatement.Text _
            , Comment, True, True) <> 0 Then

```

```

        'if not successful show error message
        lblState = "Could not save query." _
            + QMFWin.GetLastErrorString()
        lblState.Refresh
        MsgBox (lblState.Caption)
    Else
        'if successful say it to user
        lblState = ""
        lblState.Refresh
        MsgBox ("Query Saved.")
    End If
End If
End Sub

```

There is also a possibility to export the data retrieved from the query into files. This can be done using the API Export(). This API can export the data to text files, HTML files and IXS files. In this example, when the user clicks on the export button, the user is prompted to enter the file name and all the data retrieved from the typed query is saved into that file using the HTML format. The function that does this is listed below:

```

Private Sub btExport_Click()

If txtSQLStatement.Text <> "" Then
    'if there is some SQL statement the
    QueryNumber = QMFWin.InitializeQuery(0, txtSQLStatement.Text)
    If QueryNumber < 0 Then
        'if not successful display error message
        lblState = "Could not initialize the query "
        lblState = lblState + txtSQLStatement.Text
        lblState = lblState + ". "
        lblState = lblState + QMFWin.GetLastErrorString()
        lblState.Refresh
        MsgBox (lblState.Caption)
        txtSQLStatement.SetFocus
    Else
        'if successful try to open the selected query without
        'any limits of number of rows
        lblState = "Executing Query..."
        lblState.Refresh

        'open the query fetching all rows
        If QMFWin.Open(QueryNumber, 0, True) <> 0 Then
            'if not successful display error message
            lblState = "Could not open the query "
            lblState = lblState + txtSQLStatement.Text
            lblState = lblState + ". "

```



```

        lblState = lblState + QMFWin.GetLastErrorString()
        lblState.Refresh
        MsgBox (lblState.Caption)
    Else
        lblState = "Displaing Result..."
        lblState.Refresh

        'export the data to a HTML file
        If QMFWin.Export(QueryNumber, 0, 0, -1, -1, 1, 0, 0, True, _
            InputBox("Enter the file path and name"), 0) = 0 Then
            MsgBox ("Data exported.")
        Else
            MsgBox ("Could not export data. " + QMFWin.GetLastErrorString())
        End If

        'try to close the query
        lblState = "Closing the Query..."
        lblState.Refresh
        If QMFWin.Close(QueryNumber) <> 0 Then
            lblState = "Could not close query." _
                + QMFWin.GetLastErrorString()
            lblState.Refresh
            MsgBox (lblState.Caption)
        Else
            lblState = ""
            lblState.Refresh
        End If
    End If
End If
End Sub

```

The final result of this example is shown in Figure 62.

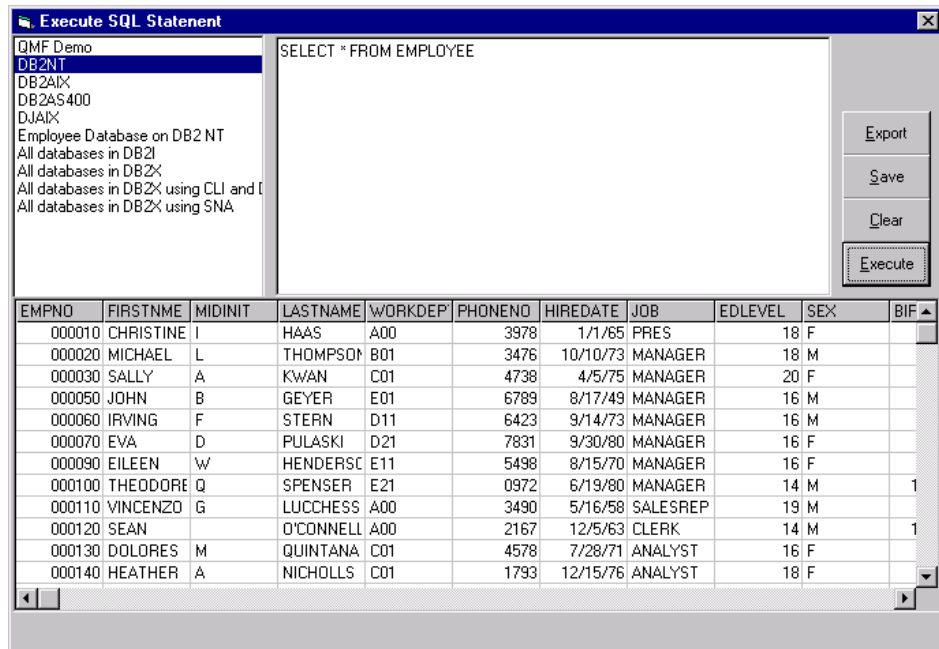


Figure 62. Visual basic — execute SQL statement

5.4 Using Delphi with QMF for Windows

Delphi is a visual programming language developed by Borland/Inprise that allows the easy creation of Windows applications. In this chapter we show the first steps that need to be taken before using QMF for Windows APIs using Delphi. We also show a Delphi program example.

5.4.1 Getting Started

When using Delphi 4.0, the following steps must be done:

1. Go to the **Project** menu and click on **Import Type Library**. The screen shown in Figure 63 will appear.

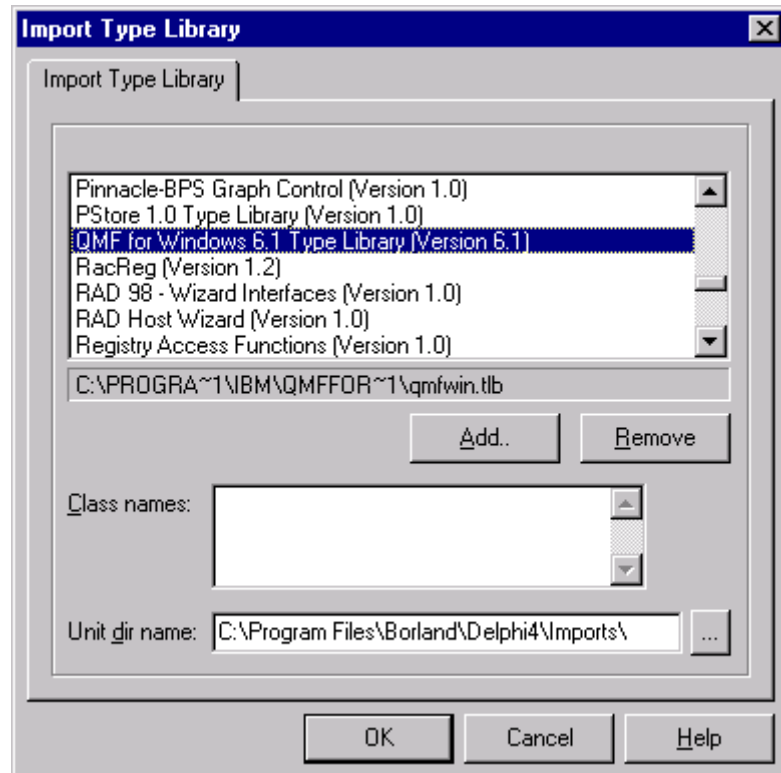


Figure 63. Delphi configuration

2. Add a reference to the QMF for Windows file **qmfwin.tlb** if the QMF for Windows 6.1 Type Library is not found on the list.
3. A new Unit named **QMFWinLibrary_TLB** will be added to the project. Add this unit in the "uses" statement in the desired form. A file called QMFWinLibrary_TLB.pas will be created on the default Imports Delphi directory (ex. C:\Program Files\Borland\Delphi4\Imports). There is no need to create this file every time; this file can be added directly to your project.
4. On the form, go to the private or public declaration and add an object of the QMFWin type, as shown in the example below:

```
uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, QMFWinLibrary_TLB;
```

```

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private

  public
    QMF: QMFWin;
  end;

```

5. Before using the QMF for Windows object it is necessary to initialize it, which can be done in any appropriate event of the application. In the example below, the initialization is in the Form.Create event.

```

procedure TForm1.FormCreate(Sender: TObject);
begin
  QMF:= CoQMFWin.Create;
end;

```

After that, the QMF for Windows APIs are ready to be used. All the APIs are property and methods of that object.

5.4.2 Delphi application example

For a complete understanding and illustration of how Delphi programmers use QMF for Windows APIs, this chapter provides an example of an application with some basic functionalities such as listing the servers, listing the queries, executing a query, and saving a query. Although this example is very simple, it shows how the interface between a Delphi application and QMF for Windows works.

The functionality of this example allows the user to create, edit, delete, and execute queries either on the server or in the file. To do that, the user first has to select a server by double-clicking on it. A window prompting the user for a userID and password will appear. After that, the list of queries stored on that server will be displayed, as shown in Figure 64.

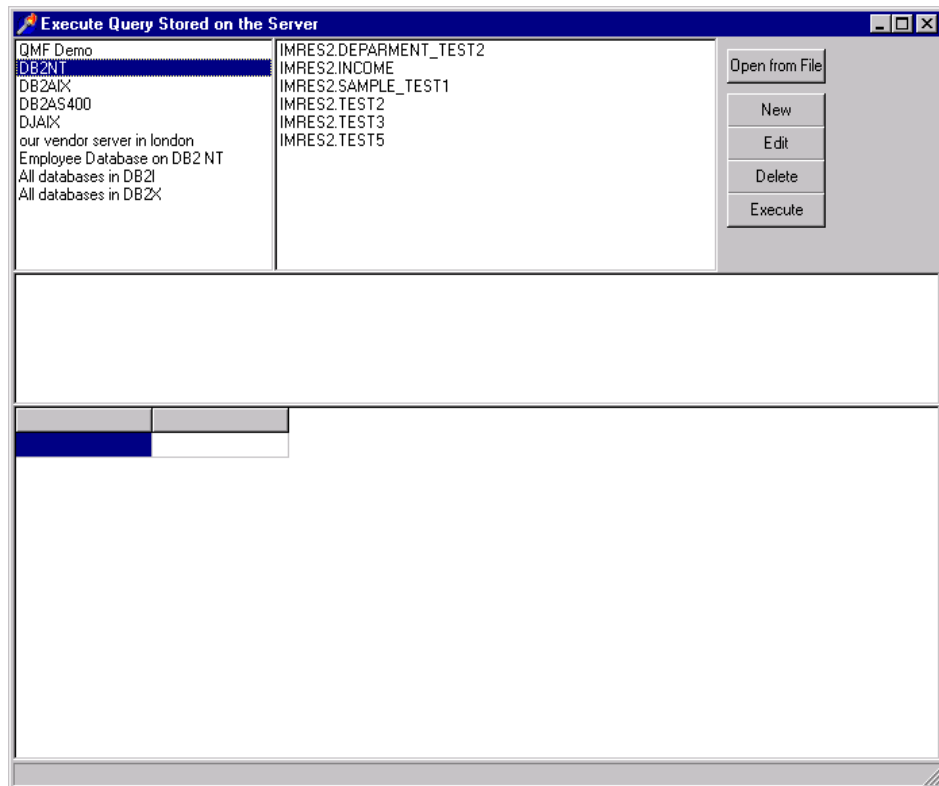


Figure 64. Delphi example, query list

Once the connection to the server has been established, the user can perform several actions with the listed queries.

The first thing the user can do is to open an existing query that is stored in a file. That can be done by clicking on the **Open from File** button. The user will be prompted to select a file and, once the file is selected, the query is executed automatically.

The next thing the user can do is to create a new query. When the **New** button is clicked, the window shown in Figure 65 will appear. The user has to type in the owner of the query, its name, and the SQL statement. After that, there are two possibilities: save the query on the server, or save the query as a file. When saving it on the server, if the owner and name already exist, the new version will try to overwrite the old one, but this will not work if the older version is a prompted query. It will only work if the older query is of the SQL statement type.

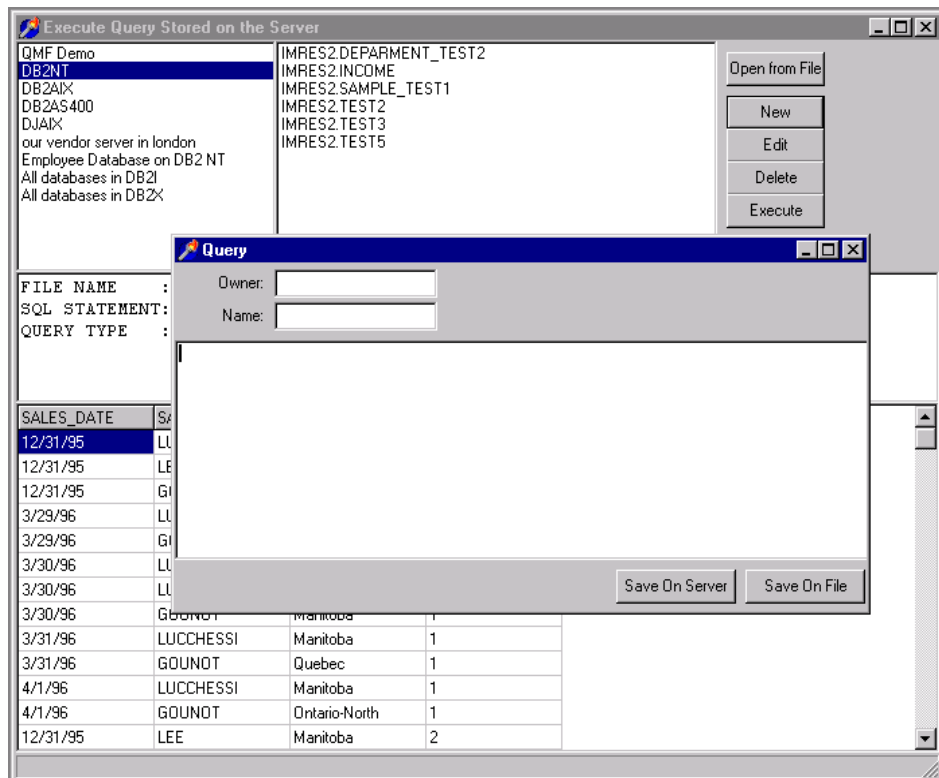


Figure 65. Delphi example, new query

The user can also edit on existing query. In order to do that, the user has to select an existing query by clicking one time over the query name on the list and clicking on the **Edit** button. The same window (shown in Figure 65) will appear, but with the information of the selected query. The user can now modify the SQL statement and save it on the server to overwrite the existing query, or he can modify the owner and name and save it as a new query.

There is also the possibility to delete existing queries. That can be done by selecting a query on the list and clicking on the **Delete** button. Be aware that no confirmation will be prompted before deleting the query.

The last thing the user can do is to execute a query. The user has to select a query from the list and click on the **Execute** button. The same thing happens if the user double-clicks the desired query within the list. Figure 66 shows the final result after a query is executed.

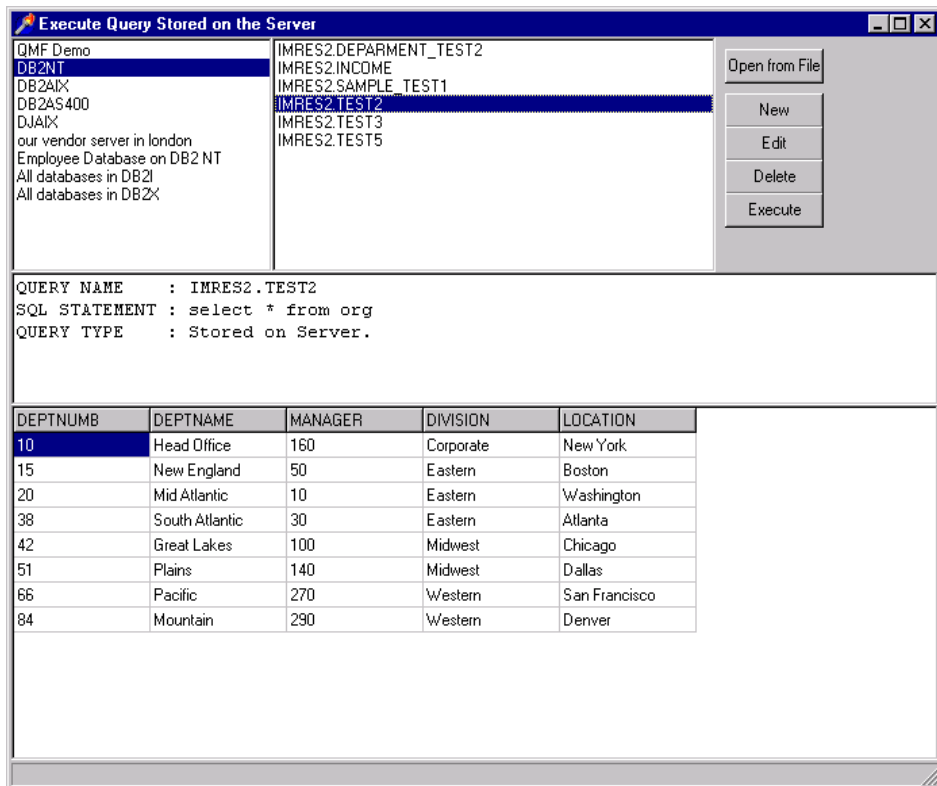


Figure 66. Delphi example, executing query

As you can see, this simple application has only two forms. The first one is called frmMain (Unit1), and the other is called frmQuery (Unit2). Below is the source code of the form frmMain (Unit1), including the uses clauses and the class definition. Pay close attention to the QMFWinLibrary_TLB on the uses clauses. Without that, the Unit will not be able to use the QMF APIs.

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, QMFWinLibrary_TLB, Grids, ExtCtrls, ComCtrls;
```

The class definition of the frmMain is listed below:

```
type
  TfrmMain = class(TForm)
    Panel1: TPanel;
    grdResult: TStringGrid;
    lstServers: TListBox;
    lstQueries: TListBox;
```

```

StatusBar: TStatusBar;
txtLog: TMemo;
btOpenFromFile: TButton;
btNewQuery: TButton;
OpenDialog: TOpenDialog;
btEditQuery: TButton;
btDeleteQuery: TButton;
btExecuteQuery: TButton;
procedure FormCreate(Sender: TObject);
procedure lstServersDbClick(Sender: TObject);
procedure btOpenFromFileClick(Sender: TObject);
procedure btNewQueryClick(Sender: TObject);
procedure btEditQueryClick(Sender: TObject);
procedure btDeleteQueryClick(Sender: TObject);
procedure btExecuteQueryClick(Sender: TObject);
private
public
    QMF: QMFWin;
    QueryNumber: Integer;
    procedure ClearGrid(Grid: TStringGrid);
    procedure DataIntoGrid(Grid: TStringGrid; QueryNumber: Integer);
    procedure RefreshQueryList;
end;

```

The uses clauses here identify that this form will be using the frmQuery.

```
uses Unit2;
```

The procedure ClearGrid is used all over the application. It only clears the TStringGrid passed in the input parameter Grid and resets its number of columns and rows.

```

procedure TfrmMain.ClearGrid(Grid: TStringGrid);
//Clear everything on the grid and
//sets rowcount and colcount to 2
var
    Col, Row: Integer;
begin
    for Col:= 0 to (Grid.ColCount - 1) do
        for Row:= 1 to (Grid.RowCount - 1) do
            Grid.Cells[Col,Row] := '';
    Grid.RowCount:= 2;
    Grid.ColCount:= 2;
    Grid.FixedRows:= 1;
end;

```


This procedure is a generic procedure that receives a string identifying a query (that has to be already open) as input parameter, and places all the data retrieved by that query into the StringGrid passed on the parameter Grid. This function was separated from the rest of the code because it may be useful in other applications that use the QMF for Windows APIs.

```

procedure TfrmMain.DataIntoGrid(Grid: TStringGrid; QueryNumber: Integer);
//This procedure fetches all the data in the QMF Query passed
//on the parameter QueryNumber and display the data on the
//grid passed on the parameter Grid
var
  ColumnHeadings: OleVariant;
  Col, Row: Integer;
  myColumnCount: Integer;
  myRow: OleVariant;
  FetchResult: Integer;
begin
  //Clears the Grid
  ClearGrid(Grid);
  //try to get the number of columns of the selected query
  myColumnCount:= QMF.GetColumnCount(QueryNumber);
  if myColumnCount <= 0 then
    ShowMessage ('Could not count numbers for columns. '
      + QMF.GetLastErrorString())
  else
    //if successful set the grdResult with the
    //appropriate numbers of columns
    begin
      Grid.ColCount:= myColumnCount;
      //try to get the column headers
      if QMF.GetColumnHeadings(QueryNumber, ColumnHeadings) <> 0 then
        ShowMessage ('Could not get columns headings. '
          + QMF.GetLastErrorString())
      else
        //if successful display the column headings on the grdResult
        begin
          for Col:= 0 to (Grid.ColCount - 1) do
            Grid.Cells[Col,0] := ColumnHeadings[Col];
          //try to fetch all the rows from the query
          //and place the data in the grid
          Row:= 0;
          FetchResult:= QMF.FetchNextRow(QueryNumber, myRow);
          while FetchResult = 0 do
            begin
              Row:= Row + 1;
              for Col:= 0 to (Grid.ColCount - 1) do
                Grid.Cells[Col,Row] := myRow[Col];
            end
          end
        end
    end
  end
end

```

```

        Grid.RowCount:= Grid.RowCount + 1;
        FetchResult:= QMF.FetchNextRow(QueryNumber, myRow);
        end;
    Grid.FixedRows:= 1;
    if FetchResult <> -1 then
        //if the result of the FetchNextRow API
        //is different than -1, that means that
        //an error occurred
        ShowMessage ('Could not fetch next row. '
                    + QMF.GetLastErrorString())
    else
        //otherwise, the loop end normally
        //clear the last row
        Grid.RowCount:= Grid.RowCount - 1;
    end;
end;
end;
end;

```

The procedure RefreshQueryList lists all the queries that are on the current server (the server in which the QMF object is currently connected to) and displays these queries in the lstQueries object. This is done using the GetQMFOBJECTList() API. The procedure was separated from the rest of the code because it is called several times from different parts of the program.

```

procedure TfrmMain.RefreshQueryList;
//List all available query in the connected server
var
    myQueryList: OleVariant;
    i: Integer;
begin
    //try to list all queries
    //(2048 parameter) in that server
    StatusBar.SimpleText:= 'Getting Existing Queries...';
    StatusBar.Refresh;
    if QMF.GetQMFOBJECTList('%', '%', 2048, myQueryList) <> 0 then
        //if not successful display error message
        begin
            StatusBar.SimpleText:= 'Could not list queries. '
                                + QMF.GetLastErrorString();
            StatusBar.Refresh;
            ShowMessage (StatusBar.SimpleText);
        end
    else
        //if successful, display all the queries into lstQueries
        begin
            StatusBar.SimpleText:= 'Displaying Query List...';
            StatusBar.Refresh;
        end
    end;
end;

```

```

lstQueries.Clear;
//in case there is no query to be listed
//an exception occurs
try
    for i:= VarArrayLowBound(myQueryList,1)
        to VarArrayHighBound(myQueryList,1) do
        lstQueries.Items.Add(myQueryList[i]);
        StatusBar.SimpleText:=);
        StatusBar.Refresh;
    except
    end;
ClearGrid(grdResult);
end;
end;

```

This procedure is executed once the OnCreate event of the frmMain form is triggered. It list all servers available in the QMF SDF and displays the servers in the lstServers object. The API used to do that is GetServerList().

```

procedure TfrmMain.FormCreate(Sender: TObject);
var
    ServerList: OleVariant;
    i: integer;
begin
    //Clear Grid
    ClearGrid(grdResult);
    //create the QMF Object
    QMF:= CoQMFWin.Create;
    StatusBar.SimpleText:= 'Listing Servers...';
    StatusBar.Refresh;
    //try to list all servers that are configured on
    //QMF for Windows
    if QMF.GetServerList(ServerList) <> 0 then
        //if fails show error message
        begin
            StatusBar.SimpleText:= 'Could not list servers.';
            StatusBar.Refresh;
            ShowMessage(StatusBar.SimpleText);
        end
    else
        //if successful then display all servers in the
        //lstServers object
        try
            for i:= VarArrayLowBound(ServerList,1) to
                VarArrayHighBound(ServerList,1) do
                lstServers.Items.Add(ServerList[i]);
                StatusBar.SimpleText:= '';
            end
        end
    end;
end;

```

```

        StatusBar.Refresh;
    except
    end;
end;

```

This procedure is executed when the DbClick event of the lstServers object is triggered. When the user double-clicks on a specific server name, the application connects to the selected server. That is done using the InitializeServer() API. After that the procedure lists all available queries on the server and then displays them in the lstQueries object by calling the RefreshQueryList procedure listed in the beginning of this chapter.

```

procedure TfrmMain.lstServersDbClick(Sender: TObject);
begin
    lstQueries.Clear;
    ClearGrid(grdResult);
    //if there is any server selected
    if lstServers.ItemIndex <> -1 then
    begin
        StatusBar.SimpleText:= 'Initializing Server...';
        StatusBar.Refresh;
        //try to initialize the server
        if
QMF.InitializeServer(lstServers.Items.Strings[lstServers.ItemIndex]
                    ,'',', True','',null) <> 0 then
            //if not successful show error message
            begin
                StatusBar.SimpleText:= 'Could not initialize server '
                    +
lstServers.Items.Strings[lstServers.ItemIndex]
                    + '. '
                    + QMF.GetLastErrorString();

                StatusBar.Refresh;
                ShowMessage (StatusBar.SimpleText);
                btOpenFromFile.Enabled:= False;
                btNewQuery.Enabled:= False;
                btEditQuery.Enabled:= False;
                btDeleteQuery.Enabled:= False;
                btExecuteQuery.Enabled:= False;
            end
        else
            //if successful, refresh the query list
            //and enable buttons
            begin
                RefreshQueryList;
                btOpenFromFile.Enabled:= True;
                btNewQuery.Enabled:= True;
            end
        end
    end;
end;

```

```

        btEditQuery.Enabled:= True;
        btDeleteQuery.Enabled:= True;
        btExecuteQuery.Enabled:= True;
    end;
end;
end;

```

This procedure is executed when the OnClick event of the button `btOpenFromFile` is triggered. When that happens, a dialog is prompted for the user to select a file containing a query. After the user selects the file, that query is executed. This is done using the `InitializeQuery()` and `Open()` APIs. The procedure also calls the `DataIntoGrid` procedure listed in the beginning of this chapter.

```

procedure TfrmMain.btOpenFromFileClick(Sender: TObject);
//Open an existing query from a file
//and execute the query
begin
    //prompt the user for the file name
    OpenFileDialog.Execute;
    if OpenFileDialog.FileName <> '' then
        //if the user chose a file
        begin
            ClearGrid(grdResult);
            //try to initialize the query selected by the user
            StatusBar.SimpleText:= 'Initializing Query...';
            StatusBar.Refresh;
            QueryNumber:= QMF.InitializeQuery(2, OpenFileDialog.FileName);
            If QueryNumber < 0 then
                //if not successful display error message
                begin
                    StatusBar.SimpleText:= 'Could not initialize query. '
                        + QMF.GetLastErrorString();

                    StatusBar.Refresh;
                    ShowMessage (StatusBar.SimpleText);
                end
            else
                //if successful try to open the selected query
                //without any limits of number of rows
                begin
                    StatusBar.SimpleText:= 'Executing Query...';
                    StatusBar.Refresh;
                    if QMF.Open(QueryNumber, 0, False) <> 0 then
                        //if not successful display error message
                        begin
                            StatusBar.SimpleText:= 'Could not open the query. '
                                + QMF.GetLastErrorString();
                        end
                    else
                        DataIntoGrid(QueryNumber);
                end
            end;
        end;
    end;
end;

```

```

        StatusBar.Refresh;
        ShowMessage (StatusBar.SimpleText);
    end
else
    begin
        //if successful
        StatusBar.SimpleText:= 'Displaying Result...';
        StatusBar.Refresh;
        //Show to user what is being displayed
        txtLog.Lines.Clear;
        txtLog.Lines.Add('FILE NAME: ' + OpenFileDialog.FileName);
        txtLog.Lines.Add('SQL STATEMENT: '
            + QMF.GetQueryText (QueryNumber));
        txtLog.Lines.Add('QUERY TYPE: Stored on File. ');
        //clear the grdResult
        ClearGrid(GrdResult);
        //Display the data in the grid
        DataIntoGrid(GrdResult, QueryNumber);
        //try to close the query
        StatusBar.SimpleText:= 'Closing the Query...';
        StatusBar.Refresh;
        if QMF.Close(QueryNumber) <> 0 then
            begin
                StatusBar.SimpleText:= 'Could not close query.'
                    + QMF.GetLastErrorString();

                StatusBar.Refresh;
                ShowMessage (StatusBar.SimpleText);
            end
        else
            begin
                StatusBar.SimpleText:= '';
                StatusBar.Refresh;
            end;
        end;
    end;
end;
end;

```

The following procedure is executed when the OnClik event of the button btNewQuery is triggered. It clears all fields and opens the frmQuery form.

```

procedure TfrmMain.btNewQueryClick(Sender: TObject);
//Shows the form frmQuery and clear
//all it's fields
begin
    frmQuery.txtOwner.Text:= '';
    frmQuery.txtName.Text:= '';

```

```

frmQuery.txtSQLStatement.Lines.Clear;
frmQuery.Show;
end;

```

The next procedure is executed when the OnClick event of the button btEditQuery is triggered. It first checks the type of the query to be edited. If the query is of the prompted query type, it cannot be edited. This type of query can only be edited using the QMF interface. If the query is of the type plain SQL, the frmQuery form will be displayed showing the query's informations. To retrieve the query type the GetQMFObjectInfo() API is used. To get the SQL statement from a query the GetQueryText() API is used.

```

procedure TfrmMain.btEditQueryClick(Sender: TObject);
//Open the form frmQuery with and display
//the selected query information on it
var
    DotPosition: integer;
    QueryName: String;
    QueryInfo: OleVariant;
begin
    //if there is any query selected
    if lstQueries.ItemIndex <> -1 then
        begin
            //copy the name of the selected query
            //using the format owner.name
            QueryName:= lstQueries.Items.Strings[lstQueries.ItemIndex];
            StatusBar.SimpleText:= 'Getting query type...';
            StatusBar.Refresh;
            //get the query type, PROMPTED or SQL
            if QMF.GetQMFObjectInfo(QueryName,3,2,QueryInfo) <> 0 then
                //if not successful show error message
                begin
                    StatusBar.SimpleText:= 'Could not get query type. '
                        + QMF.GetLastErrorString();
                    StatusBar.Refresh;
                    ShowMessage (StatusBar.SimpleText);
                end
            else
                //if successful verify the query type
                begin
                    if QueryInfo = 'PROMPTED' then
                        //if the query is a prompted query it can not be edit
                        begin
                            StatusBar.SimpleText:= 'This query is a prompted query and '
                                + 'can not be edit here.';
                            StatusBar.Refresh;
                            ShowMessage (StatusBar.SimpleText);
                        end
                    else
                        //if the query is a plain SQL query it can be edit
                        begin
                            frmQuery.txtSQLStatement.Lines.Clear;
                            frmQuery.txtSQLStatement.Lines.Text:= QueryInfo;
                            frmQuery.Show;
                        end
                    end
                end
        end
    end
end;

```

```

        end
    else
        begin
            //if the query is a SQL query then
            //find the '.' character in the string
            DotPosition:= pos('.',QueryName);
            //copy the owner part
            frmQuery.txtOwner.Text:= copy(QueryName,0,DotPosition-1);
            //copy the name part
            frmQuery.txtName.Text:= copy(QueryName,DotPosition+1,500);
            //list the SQL statement of the query
            frmQuery.txtSQLStatement.Lines.Clear;
            frmQuery.txtSQLStatement.Lines.Add
                (QMF.GetQMFQueryText(QueryName));

            //Shows the form
            frmQuery.Show;
        end;
    end;
end;
end;

```

The following procedure will be executed when the OnClick event of the button `btDeleteQuery` is triggered. It simply deletes the selected query, which is done using the `DeleteQMFObject()` API.

```

procedure TfrmMain.btDeleteQueryClick(Sender: TObject);
//delete the selected query from the server
var
    QueryName: String;
begin
    //if there is any query selected
    if lstQueries.ItemIndex <> -1 then
        begin
            //copy the owner.name of the selected query
            QueryName:= lstQueries.Items.Strings[lstQueries.ItemIndex];
            //try to delete the query
            if QMF.DeleteQMFObject(QueryName) <> 0 then
                //if not successful show error message
                begin
                    StatusBar.SimpleText:= 'Could not delete query ' + QueryName;
                    StatusBar.Refresh;
                    ShowMessage (StatusBar.SimpleText);
                end
            else
                //if successful refresh the query list
                begin
                    ShowMessage('Query Deleted. ');
                end
            end;
        end;
    end;
end;

```



```

        RefreshQueryList;
    end;
end;
end;

```

The next procedure is executed when the OnClick event of the button btExecuteQuery is triggered. It executes the selected query using the Open() API and displays the result on the grid using the DataIntoGrid procedure that is listed in the beginning of this chapter.

```

procedure TfrmMain.btExecuteQueryClick(Sender: TObject);
//execute the selected query and display
//the retrieved data in the grid
begin
    //if there is any query selected
    if lstQueries.ItemIndex <> -1 then
    begin
        ClearGrid(grdResult);
        StatusBar.SimpleText:= 'Initializing Query...';
        StatusBar.Refresh;
        //try to initialize the query selected by the user
        QueryNumber:= QMF.InitializeQuery(1,
            lstQueries.Items.Strings[lstQueries.ItemIndex]);
        if QueryNumber < 0 then
            //if not successful display error message
            begin
                StatusBar.SimpleText:= 'Could not initialize query '
                    +
                    lstQueries.Items.Strings[lstQueries.ItemIndex]
                    + '. '
                    + QMF.GetLastErrorString();
                StatusBar.Refresh;
                ShowMessage (StatusBar.SimpleText);
            end
        else
            //if successful try to open the selected query
            //without any limits of number of rows
            begin
                StatusBar.SimpleText:= 'Executing Query...';
                StatusBar.Refresh;
                if QMF.Open(QueryNumber, 0, False) <> 0 then
                    //if not successful display error message
                    begin
                        StatusBar.SimpleText:= 'Could not open the query '
                            + lstQueries.Items.Strings
                                [lstQueries.ItemIndex]
                            + '. '
                    end
                end
            end
        end
    end
end;

```



```
StdCtrls;
```

The code listed below shows the class definition for the form frmQuery.

```
type
TfrmQuery = class(TForm)
    SaveDialog: TSaveDialog;
    txtSQLStatement: TMemo;
    btSaveOnServer: TButton;
    btSaveOnFile: TButton;
    lblOwner: TLabel;
    lblName: TLabel;
    txtOwner: TEdit;
    txtName: TEdit;
    procedure btSaveOnServerClick(Sender: TObject);
    procedure btSaveOnFileClick(Sender: TObject);
private
    {Private declarations}
public
    {Public declarations}
end;
```

Note that this form uses some resources from the form frmMain. That requires the uses of the Unit1.

```
uses Unit1;
```

The next procedure is executed when the OnClick event of the button btSaveOnServer is triggered. When that occurs, the procedure will save the query on the server. A problem may rise when the owner and name already exist. QMF has different types of queries, a prompted query and an SQL query. If the owner and name of the query being saved exists and the existing query is a prompted query, an error will occur. Otherwise, if the existing object is of the type SQL query, then the existing query will be replaced. This is done using the SaveQMFQuery() API.

If the owner and name already exists, the query will be replaced.

```
procedure TfrmQuery.btSaveOnServerClick(Sender: TObject);
//save the query on the server
begin
    //if there is a owner and a name
    if (txtOwner.Text <> '') and (txtName.Text <> '') then
    begin
        //try to save the query on the server with replace option
        if frmMain.QMF.SaveQMFQuery(txtOwner.Text + '.' + txtName.Text
            ,txtSQLStatement.Text
            ,'',True,True) <> 0 then
```

```

        //if not successful show error message
        ShowMessage('Could not save query. '
            + frmMain.QMF.GetLastErrorString())
    else
        begin
            //if successful display message and
            //refresh the list of queries
            frmMain.RefreshQueryList;
            Hide;
            ShowMessage('Query Saved. ');
        end;
    end;
end;

```

This procedure is executed when the OnClick event of the button btSaveOnFile is triggered. When executed, the user will be prompted to select a file name from to save the query to. This procedure does not use any of the QMF for Windows APIs, but uses the SaveToFile method of the TMemo class instead to save the text to a file.

```

procedure TfrmQuery.btSaveOnFileClick(Sender: TObject);
//save query on file
begin
    //prompt the user for a file name
    SaveDialog.FileName:= '';
    SaveDialog.Execute;
    //if some file was selected
    if SaveDialog.FileName <> '' then
        begin
            //save the query into the file
            txtSQLStatement.Lines.SaveToFile(SaveDialog.FileName);
            Hide;
            ShowMessage('Query saved on file ' + SaveDialog.FileName);
        end;
    end;
end;

```

5.5 Using C++ with QMF for Windows

The following sections provide some short hints on using C++ for application development with QMF for Windows.

5.5.1 Getting started

If you are using Microsoft Visual C++ and MFC, create a wrapper class for the QMF for Windows API object from the QMF for Windows type library, qmfwin.tlb. Then use the CreateDispatch() function:

```
COleException e;  
IQMFWin QMFWin;  
QMFWin.CreateDispatch("QMFWin.Interface", &e);
```

5.5.2 C++ specifics

When using C++ to access the QMF for Windows API, the programmer has to keep some basic considerations in mind. C++ is a language that does not allocate memory automatically. That means that all the memory allocation has to be done by the application. This is especially important when using the QMF APIs because most of these API return values in the form of Variant. So the C++ application has to handle and manipulate these Variant results.

You must properly initialize the Variant variable before calling functions that return data into this variable. Visual Basic and Delphi do this automatically, but Visual C++ programmers must call VariantInit().

5.6 General programming considerations

Following are some general considerations to be aware of when creating applications using the QMF for Windows API.

5.6.1 Two phase commit

A two phase commit takes place when a single transaction has to occur on different databases. A well known example for a two phase commit application is the money transfer from one bank to another. When the money is deducted from the originating account, it needs to be guaranteed that the target account will be credited. This requires the application to have two database connections open at the same time in order to have both transactions under a single control. Two phase commit is in charge of data consistency between the database servers involved within this unit of work.

Many DBMSs have a two phase commit protocol that makes this process independent from the application, meaning that the application sees the two phase commit as a regular commit and the DBMS makes sure that the transaction is done correctly. However, to use that protocol, it is necessary to have a connection to all databases involved on the transaction opened at the same time.

Since each QMF for Windows instance within an application can only be connected to one server at a time, it is not possible to use the two phase commit protocol provided by the database directly. There are some other ways of implementing to phase commit in your application.

The first would be to use the DB2 DataJoiner and link all DBMSs on this process to it. The application would then connect to the a single database and the DB2 DataJoiner would communicate to the several DBMS using the two phase commit protocol and the transaction would be completed.

The second option for implementing two phase commit would be controlling the two phase commit inside your application. It is possible to create two different QMF instances in your application and connect each instance to a different DBMS. However, these two instances are not able to communicate to each other without implementing some additional code in the application. For example, it is possible to have an application that copies a query from one server to another. The following code listed below shows how it can be done.

```
var
  QMF1: QMFWin;
  QMF2: QMFWin;
  QueryNumber: Integer;
  Comment: OleVariant;
begin
  QMF1:= CoQMFWin.Create;
  QMF2:= CoQMFWin.Create;
  if (QMF1.InitializeServer('DB2NT','XXXX','XXXX',False,'',false) = 0)
  and (if QMF2.InitializeServer('DB2AIX','YYYY','YYYY',False,'',false) = 0)
  then
    if QMF1.GetQMFObjectInfo('IMRES2.TEST2',0,2,Comment) = 0 then
      QMF2.SaveQMFQuery('db2inst1.test2'
        ,QMF1.GetQMFQueryText('IMRES2.TEST2')
        ,Comment
        ,True
        ,True);
end;
```

To implement the two phase commit inside your application a very good knowledge of the two phase commit process as well as the DBMSs involved is necessary.

5.6.2 Editing prompted queries

There are two ways of storing a query: the first way is to store the query on the server, while the second way is to store the query in a file.

QMF for Windows allows users to work with prompted queries. This type of query is very different from the SQL queries, and they are stored using a different structure. Hence, the way to manipulate this type of query in your application is also different.

If the prompted query is stored on the server, the QMF APIs will manage the difference between the prompted query and the SQL query, so that for the application they look exactly the same. That means that your application can execute, delete, and view all properties from a prompted query just as from an SQL query. However, it is not possible to edit or change a prompted query. That can only be done using the QMF for Windows user interface. Using the QMF for Windows environment, the prompted query can be converted to an SQL query, and the application can then modify this query.

When prompted queries are stored in files, the QMF API will not be able to work with them. In order for the application to access these queries, they have to be converted to SQL queries from within the QMF for Windows environment.

5.6.3 Other QMF APIs

Many QMF APIs use the Variant variables as output parameters. Be aware that due to problems in Microsoft Excel 7.0 and Microsoft Access 7.0 (and possibly other 32-bit products that use Visual Basic for Applications), string data in Variant variables received from this APIs may not be translated from Unicode (used by OLE) to ANSI (used by VBA). When this occurs, only the first character of the string is displayed. To remedy this problem, set the variable equal to an empty string before you call the QMF for Windows function that uses the variable.

Chapter 6. User's guide

This chapter is dedicated to the end user who needs to become familiar with the QMF for Windows product. It explains how to install, configure, and use most of the QMF for Windows functions.

6.1 Product Installation and configuration

To install QMF for Windows you have to execute the product installation file. The following describes the product installation assuming that you either have the product distribution media available or can access the installation files copied on a network drive. For more information on the advanced installation, including having the pre-configured option set by the database administrator, see 3.8.1, "Advanced installation" on page 65.

If using the product CD-ROM for installation, inserting it into the systems drive will automatically start the installation process. Using a network drive will require you to start the installation by double-clicking the SETUP.EXE file. The installation itself is done just as with any other Windows product. In the beginning you will be able to choose the kind of installation you want, as shown in Figure 67.

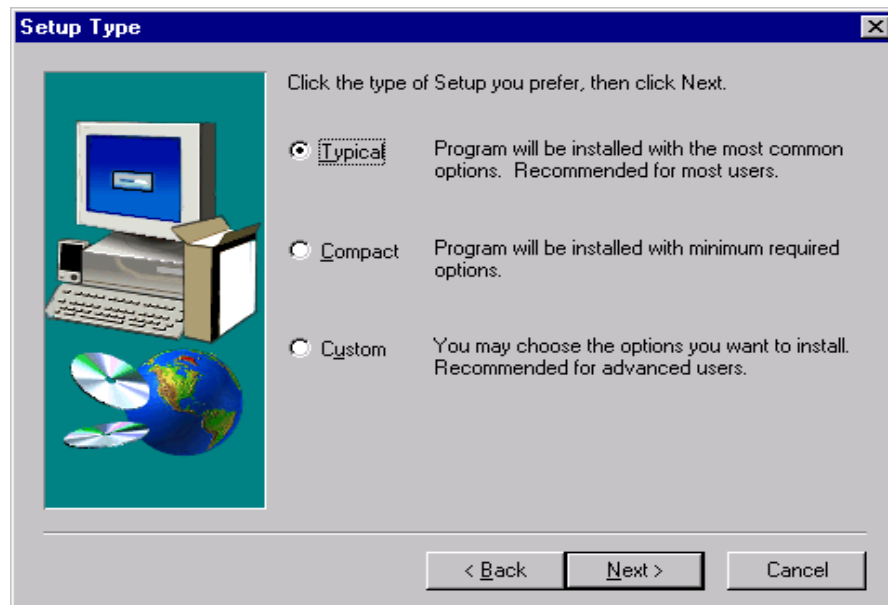


Figure 67. Installation options

The **Typical** installation will install the most common components, the **Compact** installation will install only the necessary components, and the **Custom** installation allows the components installed to be chosen. If this last option is selected, the window shown in Figure 68 will appear.

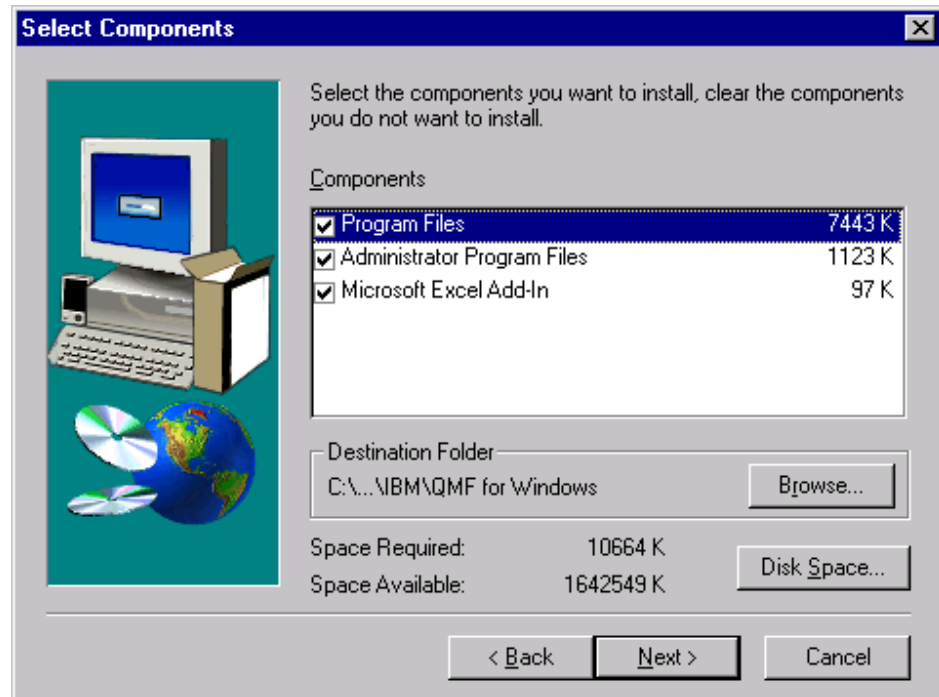


Figure 68. Custom installation

In case the Custom Installation is selected, the available components to be chosen are:

- The Program Files, which are the necessary files for the QMF for Windows environment
- The Administrator Program Files, which are the files for the administrator environment
- The Microsoft Excel Add-In; this will create a shortcut button on your Excel to execute QMF for Windows.

In the same window it is also possible to change the default directory for the installation by clicking on the **Browse...** button, and the next window will give you the opportunity to change the installation directory.

After successful installation, it is necessary to configure QMF for Windows to be able to access the database servers. The **server definition file (SDF)** contains all of the technical information needed by QMF for Windows to access the database servers.

For QMF for Windows, the term database server is used differently than the term used by database administrators. If one DBMS controls multiple databases (or subsystems), each of these is called a database server when using QMF for Windows.

There are many functions in QMF that must have an active connection to a database server. Every time a connection to a server is required, a dialog as shown in Figure 69 will appear. It is important to remember which server you are currently connected to.

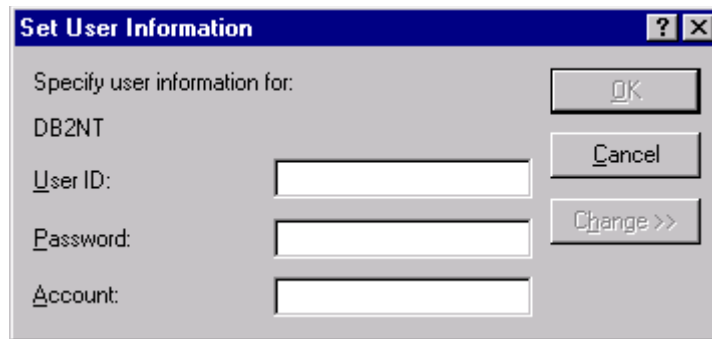


Figure 69. Connection window

QMF does not restrict the number of database servers you can have access to. Please refer to Chapter 3, “Getting started” on page 25 for more information on how to configure a connection to a database server, as this typically is not an end user task, but should be provided by the database administrator.

There are two ways you can use server definition files:

- Each user can be allowed to have his or her own SDF
- A single SDF can be shared by multiple users over a file-sharing network.

Having a local SDF on the end user system requires the user to either create the file himself, which is not recommended, or has to download the pre-configured file from a shared network drive and copy it to the appropriate directory in the local system.

This second approach has the advantage that it centralizes the administration of the SDF; the DBA or System Programmer, using the QMF for Windows Administrator, edits a particular server definition file, that only needs to be created and maintained centrally as a single file, and the users need only to point to that file when they run QMF for Windows.

If you are using your own local file, you have to use the QMF for Windows Administrator to configure the SDF file (for more information on how to configure the access to the servers, see 3.8, “Installing QMF for Windows” on page 65). If the server definition file is already defined, you only have to configure the local QMF to use that file. From the main window in QMF for Windows, specify the SDF using the **Options...** item from the **View** menu to open the window shown in Figure 70.

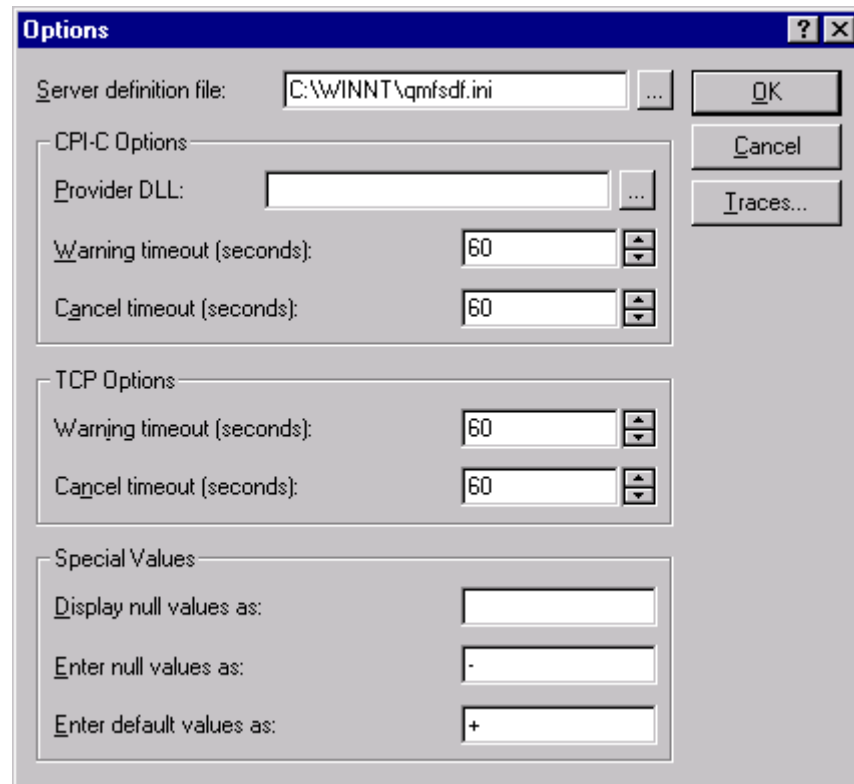


Figure 70. Setting up the server definition file

To change the path to the SDF, type the file path and name in the entry field or click on the button on the right of the field to select it. No other modifications have to be done in that window.

6.2 Basic concepts

QMF has four types of objects:

- Queries
- Forms
- Procedures
- Tables

Each of these objects has a different functionality and characteristics, but they are all linked together in some way — a query needs a table, a query plus a form is required to generate a report, and so on.

All these objects usually reside within a Database Management System (DBMS). In short, DBMSs are systems that control everything that happens in the databases — from user permissions to data consistency. Some examples of DBMSs are DB2, Oracle, Sybase, Informix, MS SQL, and many others.

A single DBMS can control one or more databases, meaning that you can have one DBMS installed in a computer and have several databases in it. Figure 71 shows the structure of it. For example, a company may have different databases for each department using the same **DBMS**. For QMF for Windows, each database is called a **database server**.

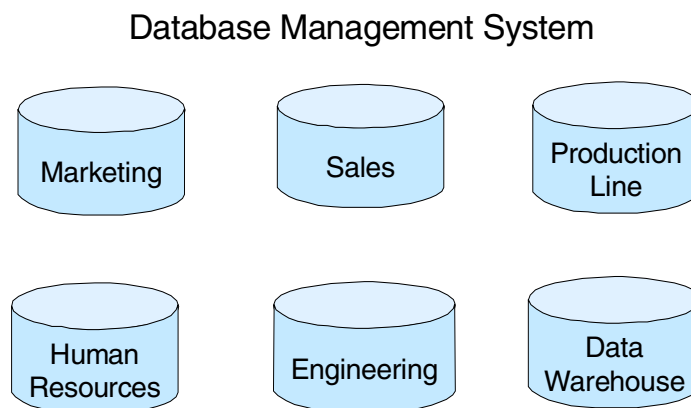


Figure 71. Database management systems

Inside each database you have data, and that data is organized inside tables. Each group of data is in a separate **table**, as shown in Figure 72. For example, customer data will be in one table, while product data will be in another table.

Each group of data has its own characteristics. In other words, the data that will be stored for customers is different from the data that will be stored for products. For example some data for customers might include name, address, and telephone; while products typically have data such as product number, weight, price, and description. Each of these characteristics will be a different **column** in the table.

One table stores the data of all members of the group, or, to stay with the example, all customer data has to be in the same table. Each customer data entry will be a **row** in this table.

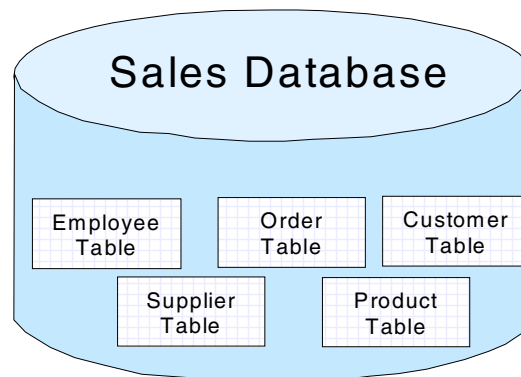


Figure 72. Tables in a database

Once the table is created, it may be necessary to retrieve or manipulate the data in it. To do this, a **Query** is used. Queries are objects that contain the information necessary to retrieve or modify a specific set of data from a database. This information is in a format called Structured Query Language (SQL), which is a standardized language used to tell the DBMS what data should be retrieved or modified. When a query is executed, the SQL statement is sent to the DBMS, which processes it and returns the data resulting from that SQL or modifies the defined data. A query is created to retrieve data from one or more tables. A query can retrieve all data in the table or just a subset of that data. You can think of the result of a query as a different view of data in the table (or tables).

After you create a query and receive the result, you may want to modify the result format. In QMF, forms provide a way of specifying different views to the data retrieved by a query. Formatting, such as where each column will be displayed, the width of each column, breaks, groups, and summaries, may be specified. The result of a query, combined with a form, is a report.

The most important concept to keep in mind is that these objects are linked together. A query is always linked to a table, and a form is always linked to a query, as shown in Figure 73.

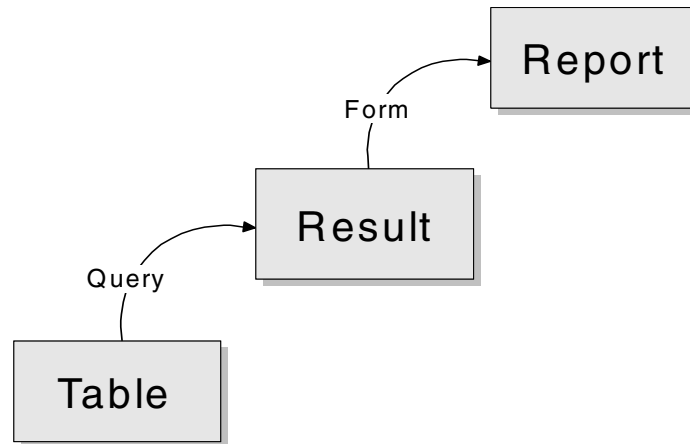


Figure 73. Object links

6.3 Accessing existing objects

As mentioned before, QMF for Windows allows its objects (Tables, Queries, Forms, and Procedures) to be stored either within a database server's QMF tables, or within files that are located on LAN accessible network drives. The following shows the access to existing QMF objects using both of these methods.

6.3.1 Objects stored at a server

To access objects stored at a database server, use the process listed below:

1. Click on the **Open From Server** button or use the **File** and **Open From Server...** menu. A window as the one shown in Figure 74 will be displayed. There you have two options:
 - Select the object directly, if you know the owner and name
 - Select the object from a list

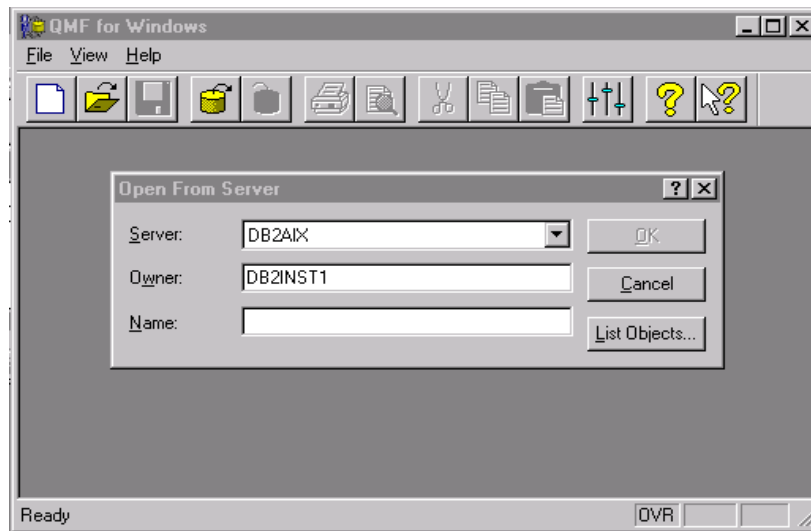


Figure 74. Open object from server

2. To select the object directly, select a database server, type in the owner, and the name of the object, and click on the **OK** button.
3. To select the object from a list click on the **List Objects...** button. The list of objects looks like the one in Figure 75.

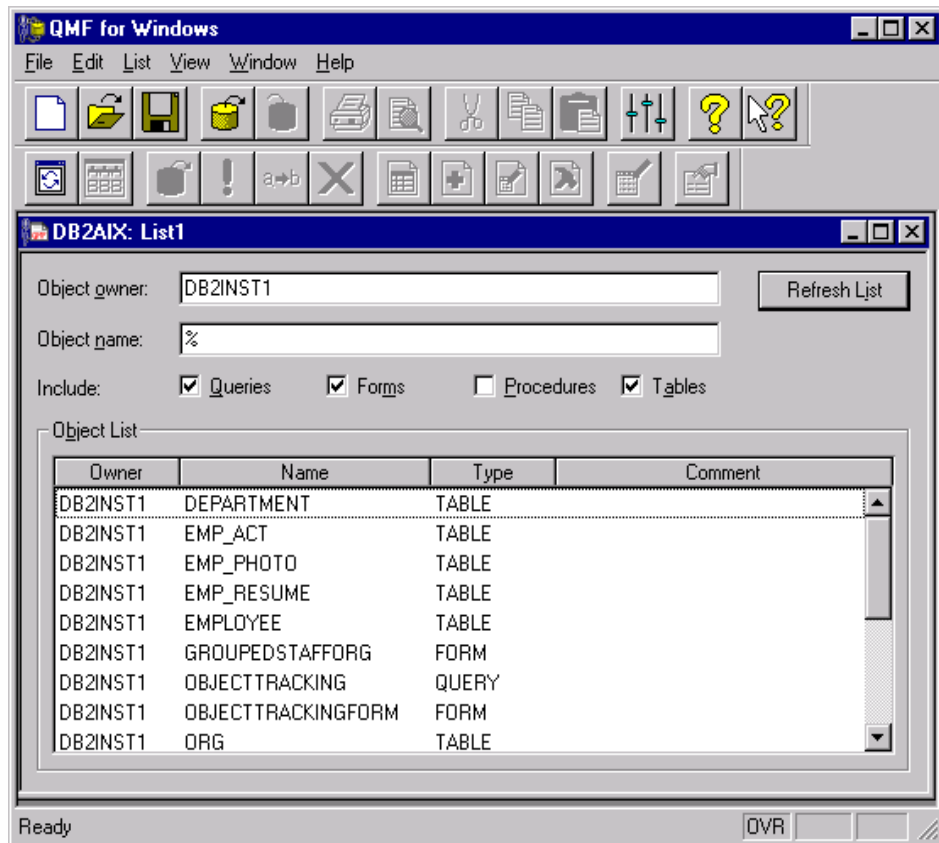


Figure 75. List of objects

4. In that list, you may wish to adjust the filter of the objects you want to see. The first filter is the owner. Type the owner, and only objects from that owner will appear. The second one is the object name. In both cases it is possible to use the character '%' as a wildcard. The last filter is the object type. Select the object type you want to see displayed in the list by clicking on the type name.
5. After any modification in the filters you have to click on the **Refresh List** button or the **List** and the **Refresh List** menus for the filter to take effect.
6. Find the object you want to access in the list and just double-click in it or use the **List** and **Display** menus and the object will be displayed.

An alternative way to reduce the contents of the list shown in Figure 75 is to select the unwanted object and remove it from the list by clicking the right mouse button and selecting **Remove from List**. After doing that, the object will not be displayed until the **Refresh Button** is clicked again. But be careful **not** to select the **Delete** option, as this would delete the object entry in the database.

Opening a form shows some specifics:

- As a form is always linked to a query, opening a form will directly display the result data if the associated query has been opened before. If not, the form will be displayed without any data.
- However, the best way of accessing forms is to access a query and then use the **Display Report** button or use the **Query** and **Display Report...** menus. Then choose the existing form either from the server or from a file and it will be opened.
- Yet another option would be to open the form and then select the query you want it to be linked by using the menu **Form** and **Select Query for Data...** menus.

In order to customize the list presented, it can be saved to a file by clicking the **Save** button on the toolbar or using the **File** and **Save As...** menus.

Note:

There is no way of seeing if a query is an SQL query or a prompted query in the list directly. However, when you access a query, QMF will automatically identify the sub-type of the query and display the correct window for you. To see the sub-type of the query, click on the query with the right mouse button and go to **Properties**, or use the **List** and **Property** menus.

6.3.2 Objects stored in a file

QMF objects can also be stored within files. To access these objects, follow the process below:

1. Use the menus **File** and **Open**. A window as shown in Figure 76 will be displayed.
2. You can select the file type you want to open, and only the files that correspond to that type will be displayed.
3. Then select the directory in which the file is stored.

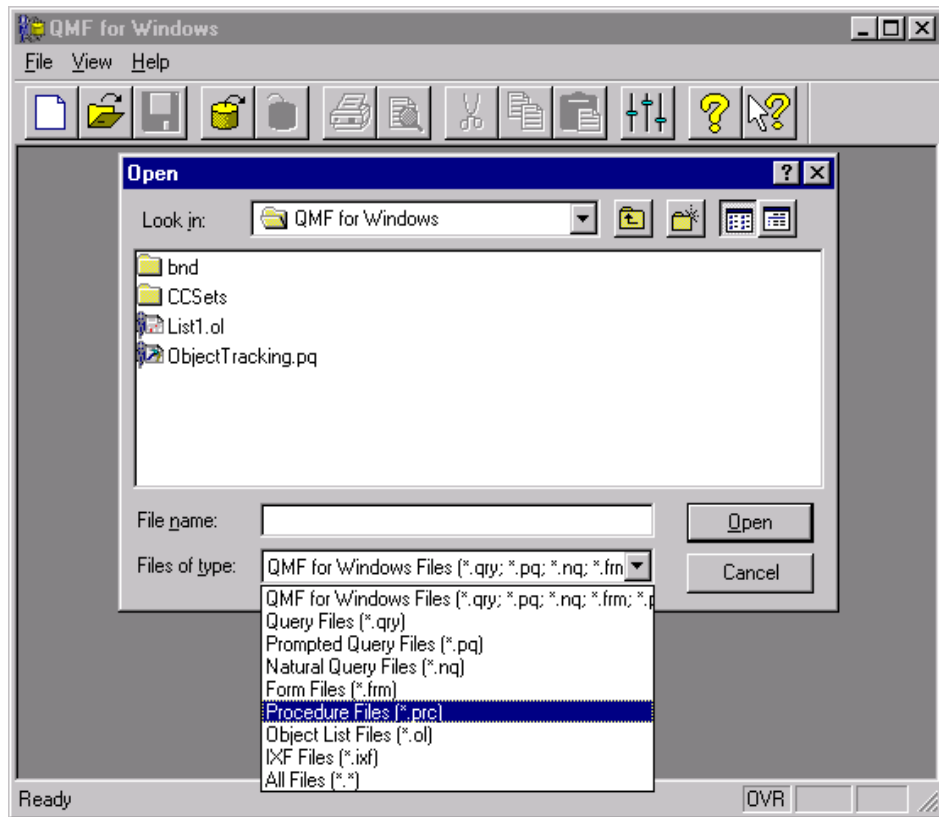


Figure 76. Open object from file

4. Select the file from the list
5. Click on the **Open** button

QMF will automatically open the file and display the object.

6.4 Working with objects

Different objects have different functionality. Therefore, after having opened an object, different operations may be performed. In this chapter, only the most important functions will be discussed.

6.4.1 Tables

When accessing an existing table, QMF for Windows creates a new SQL query that retrieves all the data in the table and executes it. After that, you can see the name of the server accessed and a sequential number for the executed query in the title of the window. From that moment on, it is possible to use this new query as any other query. You can see the SQL statement by clicking on the **View SQL** button, or use the **View** and **SQL** menus and modify it if you want to. This is an easy way of creating a new SQL query.

Like every new SQL query you create, when you try to close the window, QMF will prompt if you want to save the query or not.

6.4.2 Queries

As said before, there are two types of queries, the SQL query and the prompted query. Depending on the type, the window that will be displayed when you open the query will be different. However, the functionality and the possible actions for the query may be the same.

6.4.2.1 Run a Query

The most important function of a query is to retrieve the data from the database. That is done by executing the query. To do this, you have to click on the **Run Query** button, use the **Query** and **Run** menu, or use the shortcut **Ctrl + R**. The result of the query will be displayed in a grid, as shown in Figure 77.

DB2NT: PromptedQuery1: IMRES2.PROMPTED_QUERY

FIRSTNAME	LASTNAME	DEPTNAME	SALARY	BONUS	COMM	TOTAL_INCOME
JAMES	JEFFERSON	ADMINISTRATION SYSTEMS	22180.00	400.00	1774.00	24354.00
SYBIL	JOHNSON	ADMINISTRATION SYSTEMS	17250.00	300.00	1380.00	18930.00
DANIEL	SMITH	ADMINISTRATION SYSTEMS	19180.00	400.00	1534.00	21114.00
SALVATORE	MARINO	ADMINISTRATION SYSTEMS	26760.00	600.00	2301.00	31661.00
SALLY	KWAN	INFORMATION CENTER	36250.00	800.00	3060.00	42110.00
HEATHER	NICHOLLS	INFORMATION CENTER	26420.00	600.00	2274.00	31294.00
DOLORES	QUINTANA	INFORMATION CENTER	23800.00	500.00	1904.00	26204.00
IRVING	STERN	MANUFACTURING SYSTEMS	32250.00	500.00	2580.00	35330.00
JENNIFER	LUTZ	MANUFACTURING SYSTEMS	29840.00	600.00	2387.00	32827.00
WILLIAM	JONES	MANUFACTURING SYSTEMS	18270.00	400.00	1462.00	20132.00
DAVID	BROWN	MANUFACTURING SYSTEMS	27740.00	600.00	2217.00	30557.00
JAMES	WALKER	MANUFACTURING SYSTEMS	20450.00	400.00	1636.00	22486.00
MARILYN	SCOUTTEN	MANUFACTURING SYSTEMS	21340.00	500.00	1707.00	23547.00
MASATOSHI	YOSHIMURA	MANUFACTURING SYSTEMS	24680.00	500.00	1974.00	27154.00
ELIZABETH	PIANKA	MANUFACTURING SYSTEMS	22250.00	400.00	1780.00	24430.00
BRUCE	ADAMSON	MANUFACTURING SYSTEMS	25280.00	500.00	2022.00	27802.00
EILEEN	HENDERSON	OPERATIONS	29750.00	600.00	2380.00	32730.00
MAUDE	SETRIGHT	OPERATIONS	15900.00	300.00	1272.00	17472.00
PHILIP	SMITH	OPERATIONS	17750.00	400.00	1420.00	19570.00
ETHEL	SCHNEIDER	OPERATIONS	26250.00	500.00	2100.00	28850.00
MICHAEL	THOMPSON	PLANNING	41250.00	800.00	3300.00	45350.00
RAMLAL	MEHTA	SOFTWARE SUPPORT	19950.00	400.00	1596.00	21946.00
JASON	GOUNOT	SOFTWARE SUPPORT	23840.00	500.00	1907.00	26247.00
WING	LEE	SOFTWARE SUPPORT	25370.00	500.00	2030.00	27900.00
CHRISTINE	HAAS	SPIFFY COMPUTER SERVICE DIV.	52750.00	1000.00	4220.00	57970.00

Row 1 of 28

Figure 77. Query result

6.4.2.2 Cancel Query

Some queries may take a while to execute, due to a large amount of data in the tables, network traffic, or the number of simultaneous requests to the DBMS. If you want to cancel the request, click on the **Cancel Query** button or use the **Query** and **Cancel** menu.

6.4.2.3 View SQL, View Prompted Query, and View Result

After the query is executed, the SQL statement or the Prompted Query window will no longer be visible. To switch from one window to the other you have to use the **View SQL**, **View Prompted**, and the **View Results** button or their related menus. Only one view will be available at a time.

View SQL allows you to view the SQL statement that was issued to the DBMS. If you are working with prompted queries, you can see the SQL generated, but you cannot change it.

View Prompted will only work for prompted queries. When working with SQL queries, you will not be able to see it in a prompted query window. Therefore the **View Prompted** button and menu will always be disabled when an SQL query is the active window. The **View Prompted** allows you to see the prompted query window and change it if necessary.

Finally the **View Results** brings back the grid with the result of the query. This option will only be available after the query has been executed at least once. There is a great difference between run a query and view the result. When you run a query, QMF will send the SQL statement to the DBMS, it will process the SQL statement and send back the data. Depending on the query, that process can take several minutes and consume a lot of resources at the database server as well as generating traffic on the network. When you run a query for the first time, QMF submits the SQL statement to the DBMS and stores the result locally. If you use **View Results**, QMF will display the last retrieved data that is stored locally in your computer without using the network or the DBMS. Hence it is faster and the resources are free for others users.

6.4.2.4 Find

On the result grid window, it is possible to find a particular value in the grid. To do this, select to result window and click on the **Find** button or use the **Edit** and **Find** menu. A window as shown in Figure 78 will appear. Type in the text you want to find and click on the **Find Next** button.

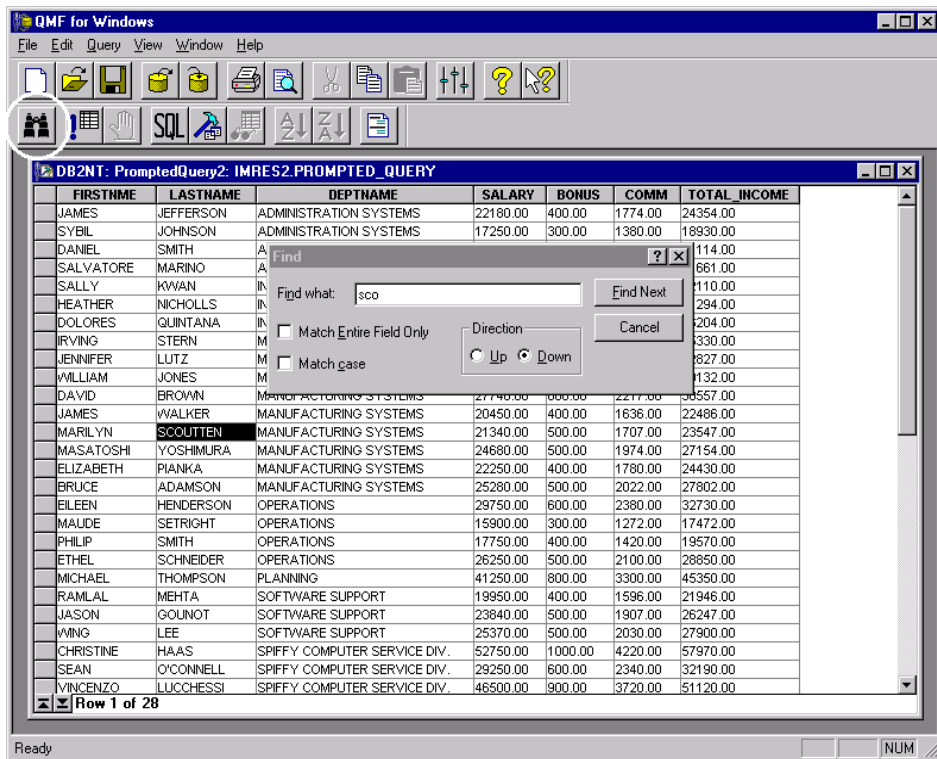


Figure 78. Find

6.4.2.5 Sort (order by)

In the result window it is possible to sort the grid by one (or more) columns. To do this, select the columns you want to sort by clicking on the header of the column. Then either click on the **Sort Ascending** or **Sort Descending** button, or else use the **Edit** and **Sort Ascending** or **Sort Descending** menu. The result will then be sorted by the column you selected.

Be aware that this sort is not permanent. If you run the query again, the result may not be sorted by the column you selected. To sort the result permanently by one column, change the SQL statement using the ORDER BY clause in the SQL query or the **Sort Condition** in the prompted query.

6.4.2.6 Modify data

Depending on the query, it is possible to modify the data in the database directory in the grid. Only queries that access data from not more than one table (those that do not use joins or sub-queries), and that do not use the group function, can edit the data directly from within the grid. You also need to have the proper privilege being set in the resource limit group (see 6.8, “Checking your resource limits” on page 243).

If you have the required permission and the query allows you to edit data, on the result grid, double-click on the cell you want to change, type in the new value and press the Enter key. A warning message will be displayed to ensure that you really want to change that data. To continue with the change click on the **OK** button or, to cancel, click on the **Cancel** button. If you decide to change the data, a message will be displayed regarding the successful operation.

6.4.2.7 Print

At any time you can print the query or its result by clicking on the **Print** button or using the **File** and **Print** menus.

6.4.2.8 Export

After you have executed a query, it is possible to export the data to files using different formats. To do so, go to the **File** and **Export Data...** menu. The window shown in Figure 79 will appear. Be aware that this option is only enabled when the query has already been run successfully at least once.

In this window you have to type in the file name and select the file type.

Available file types are:

- Text/DEL File (*.txt), which is a delimited ASCII file
- HTML File (*.html)
- IXF File (*.ixf), which is the Independent Exchange Format used by DB2 UDB
- CSV File (*.csv)

You can also define some other options by clicking on the **Option...** button where you can specify if all of the data or just the selected data is to be exported, and also define if the column headers are to be exported together with the data. After everything is defined, click on the **Save** button and the data will be exported.

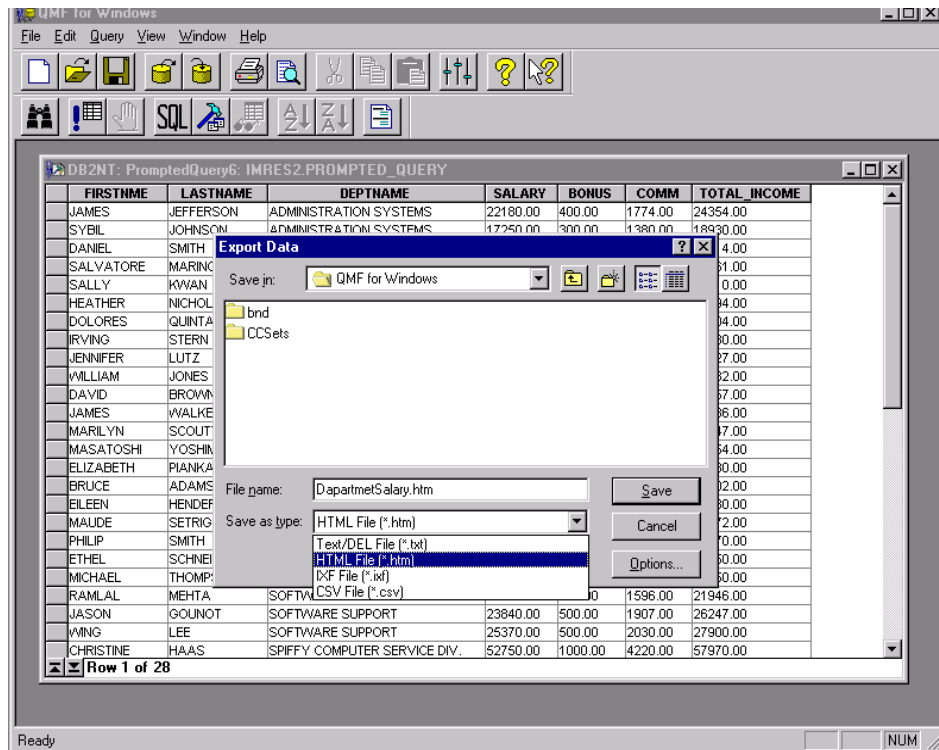


Figure 79. Export data

6.4.3 Forms and Reports

This section discusses the most important actions that can be performed using a QMF Form or Report. For instructions on creating or modifying an existing form, please refer to 6.5.3, “Create new form and report” on page 211.

6.4.3.1 Check for errors

After you create or modify a form, you can check for possible errors in your modifications. Refer to 6.5.3, “Create new form and report” on page 211 for more information on how to create or modify forms.

6.4.3.2 Print reports

To print a report, you have to click on the **Print Report** button or use the **File** and **Print Report...** menus. The form will be printed.

Be aware that the form is going to be printed the way you see it on the window. That means, if the report is displaying no data, it will be printed with no data. Also take care regarding the width of the form. If it does not fit on one page, another page will be used.

6.4.3.3 Convert forms to HTML

When using forms, you can convert them to HTML. This is done by clicking on the **Convert to HTML** button or using the **Form** and **Convert to HTML** menus. This will open a window as shown in Figure 80.

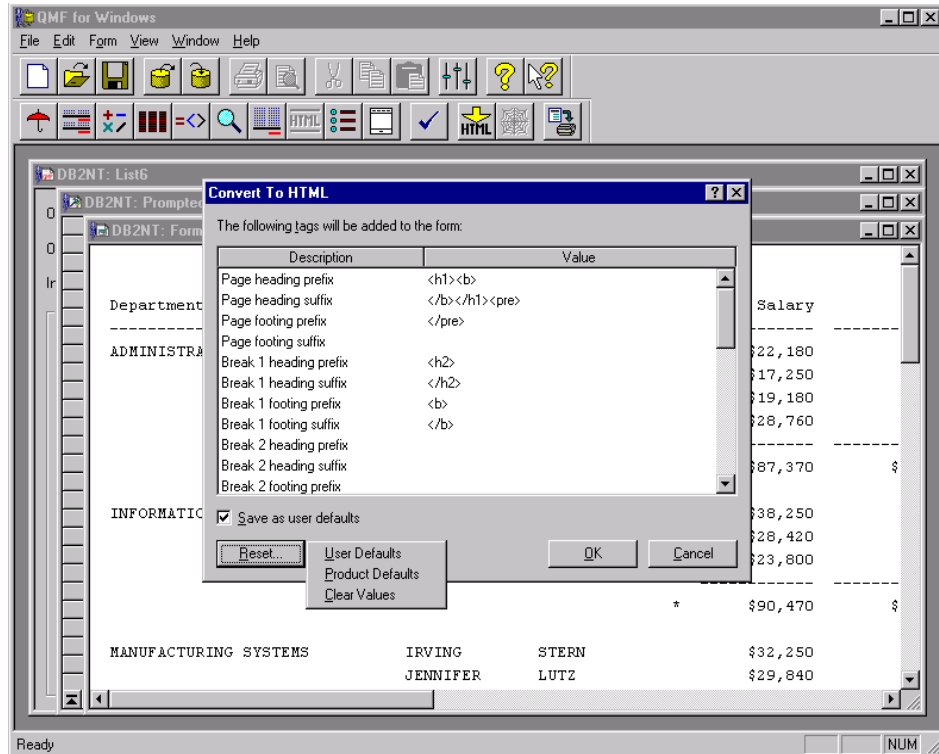


Figure 80. Convert form to HTML

In this window you can specify the HTML tags you want before and after each part of the form. If you do not know the HTML tags, you can use the default tags by clicking on the button **Reset...** and select the **Products Default**. For a better formatting, complete the default by typing the `<PRE>` tag in the end of the Page Heading Suffix and type the `</PRE>` tag in the Page Footing Prefix. That way the form is going to be displayed in the browser exactly like in QMF. Do not forget to select the **Save as user default** before clicking on the **OK** button. That will save the changes you made on the HTML tags.

After that, a new form will be created with some HTML tags in it, as shown in Figure 81. At this point, it is still possible to use all the functions for a normal form, such as saving it at a server or checking for errors.

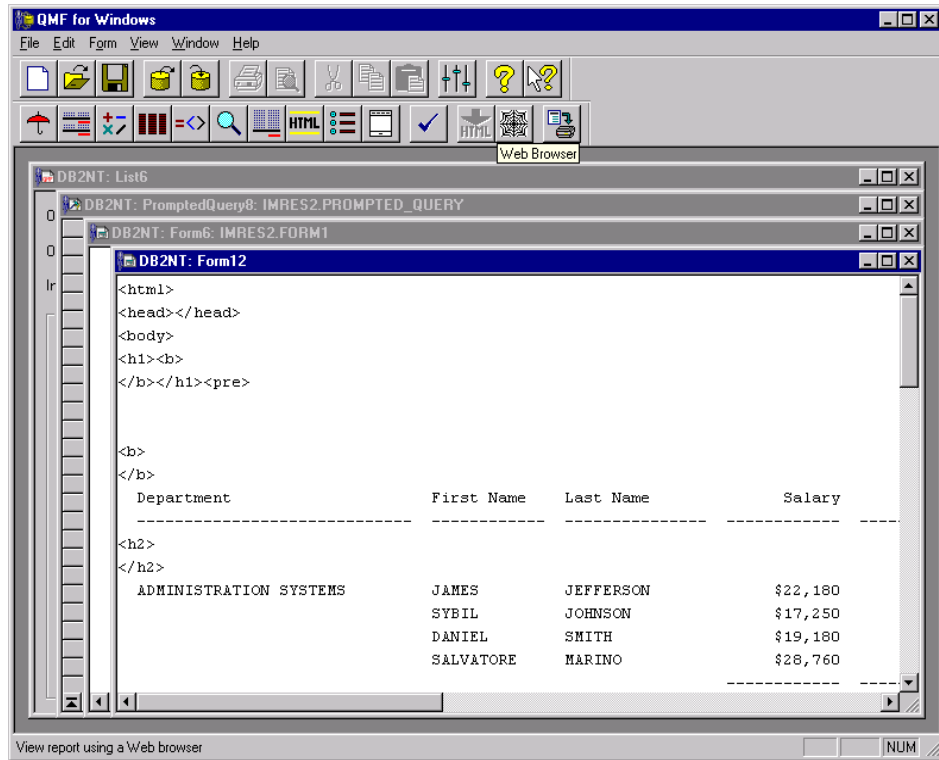


Figure 81. HTML form

However, a new function is available for this new form, which is to preview the result in the default Web browser configured for the local system. To do this, you have to click on the **Web Browser** button or use the **Form** and **View in Web Browser** menus. Your default Web browser will be automatically launched, and the form will be displayed in it, as shown in Figure 82.

Department	First Name	Last Name	Salary	Bonus	Commission	Total Income
ADMINISTRATION SYSTEMS	JAMES	JEFFERSON	\$22,180	\$400	\$1,774	\$24,354
	SYBIL	JOHNSON	\$17,250	\$300	\$1,380	\$18,930
	DANIEL	SMITH	\$19,180	\$400	\$1,534	\$21,114
	SALVATORE	MARINO	\$28,760	\$600	\$2,301	\$31,661
			\$87,370	\$1,700	\$6,989	\$96,059
INFORMATION CENTER	SALLY	KWAN	\$38,250	\$800	\$3,060	\$42,110
	HEATHER	NICHOLLS	\$28,420	\$600	\$2,274	\$31,294
	DOLORES	QUINTANA	\$23,800	\$500	\$1,904	\$26,204
			\$90,470	\$1,900	\$7,238	\$99,608
MANUFACTURING SYSTEMS	IRVING	STERN	\$32,250	\$500	\$2,580	\$35,330
	JENNIFER	LUTZ	\$29,840	\$600	\$2,387	\$32,827
	WILLIAM	JONES	\$18,270	\$400	\$1,462	\$20,132
	DAVID	BROWN	\$27,740	\$600	\$2,217	\$30,557

Figure 82. HTML form result

6.4.3.4 Export report

It is also possible to export the form to a text file. To do this, go to the **File** and **Export Report** menu. A window appears prompting for the file path and name. Select it and click on the **Save** button to export the form.

6.4.4 Procedures

Procedures allow you to do the following operations.

6.4.4.1 Run

To run a procedure, simply click on the **Run Procedure** button, use the **Procedure** and **Run** menus or use the shortcut **Ctrl + R**. The execution of the procedure will begin immediately.

6.4.4.2 Find

You can find any text string inside the procedure by using the **Find** button or the **Edit** and **Find** menus. A window will appear for you to enter the text string to be found. Then click on the **Find Next** button.

6.4.4.3 Print

It is always possible to print the procedure by clicking on the **Print** button or using the **File** and **Print** menus.

6.5 Create new objects

QMF for Windows allows you to create your own objects: Query, Form, Procedure, and Table. Each type of object has a different function, and is created in a different way. This will be explained in the following sections.

Although each type of object has its specific characteristics, all have some things in common. They can all be stored either on a server or in a file, and they are all visible in the Object List. However, the most important aspect they have in common is a database server. Whenever you create a new object, a database server is associated with that object.

When an object is created, it is automatically associated with the last used database server, but it can be changed using the menu of the active object (such as Query or Procedure) and then clicking on the item **Set Server...** A window similar to the one in Figure 83 showing the available servers will appear. You can then select the desired server and click on the **OK** button, and the object will be linked to the database server selected. Note that the title of the object window shows the server the object is related to. If you change the server, the title will be changed too.

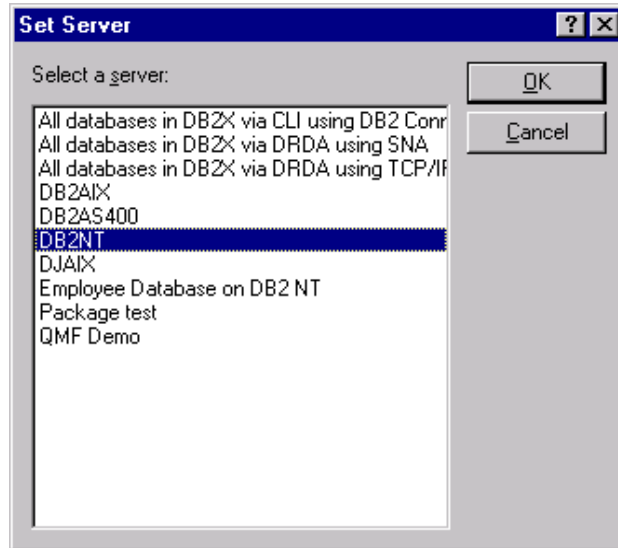


Figure 83. Set server window

6.5.1 Create new tables

Most of the time there will be no need to create your own tables. That is because in the majority of the cases, the tables will have already been created by a DBA, and the data in those tables will be ready for use. However, if you have the need and the appropriate privilege in the database, you can create your own tables.

There are two ways of creating tables using QMF:

1. The first way is to create an SQL query containing the data definition language (DDL) statements to create a table and execute it. This action is not recommend for beginners. Use it only if you have a very good knowledge of SQL statements and database management systems.
2. The second way is to create a new table out of the result of a query. After the data was retrieved when executing a query, save the result into a new table. To do this, follow the steps below:
 - a. Open and execute the query that will retrieve the data you want to save. For more information on how to execute a query, refer to 6.3, “Accessing existing objects” on page 183.

- b. After the query has been executed, go to the **File** and **Save Data...** menus. A window as shown in Figure 84 will be displayed. Type the name and the comments for the new table. In case the table already exists, it will not be replaced. If the data you are trying to save does not have the same layout as the existing table, an error will occur. Otherwise, two actions can be taken: The first possibility is to replace the existing table with the new table, and the second possibility is to replace the existing data with the new data. That option can be selected in the bottom of the windows shown in Figure 84.

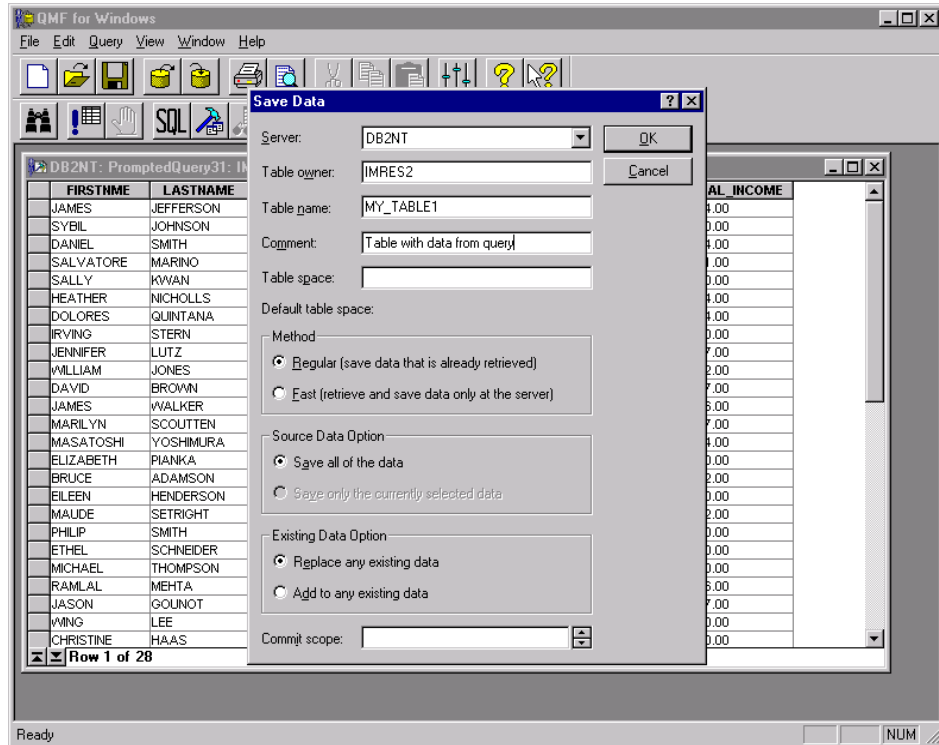


Figure 84. Save data into new table

6.5.2 Create new queries

QMF for Windows has two types of queries, SQL query and prompted query. An SQL query is a query where you have to type the SQL statement which means that you have to know how to write SQL statements. The other option is the prompted query. In that case, QMF for Windows will display a window where you can, by pointing and clicking, create your query without knowing SQL. These two types of queries have the same result; the only difference is the way they are built.

6.5.2.1 SQL queries

SQL queries are queries where the user has to type the SQL statement. Therefore, it is necessary to have a good working knowledge of SQL. (If you do not know SQL, then it is probably better to create prompted queries. Refer to 6.5.2.2, “Prompted queries” on page 203 for instructions on how to create prompted queries.) To create a new SQL query, follow the steps below:

1. Go to the **File** menu, then to the **New** menu, and finally to the **SQL Query** menu as shown in Figure 85, or click on the **New SQL Query** button on the toolbar.

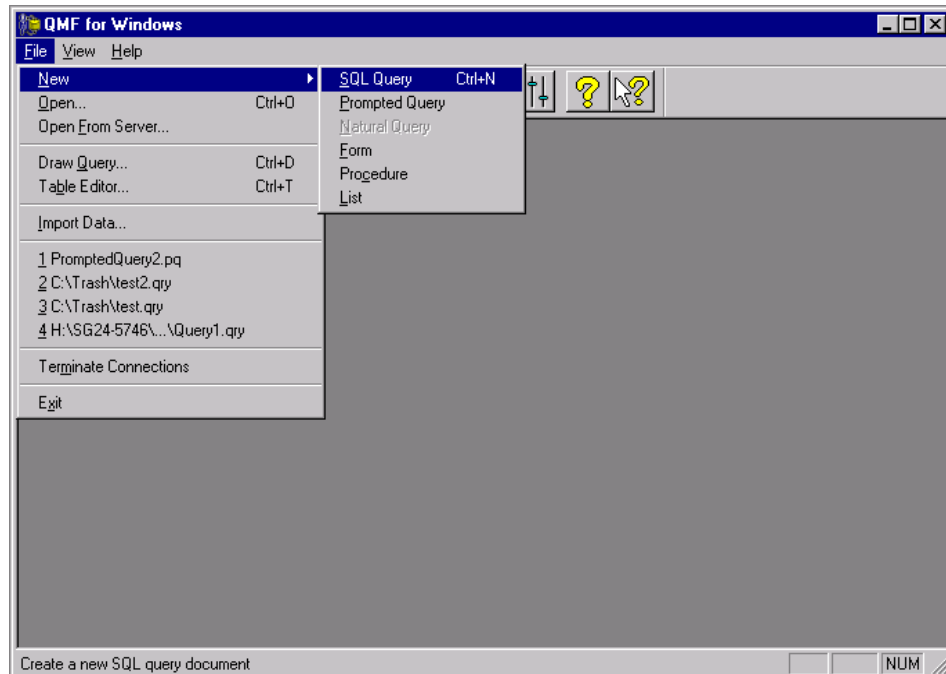


Figure 85. New SQL query menu

2. A window similar to the one in Figure 86 will be displayed. Pay close attention to the title of that window, because it shows the server name and the query name in the format SERVER: QUERY.

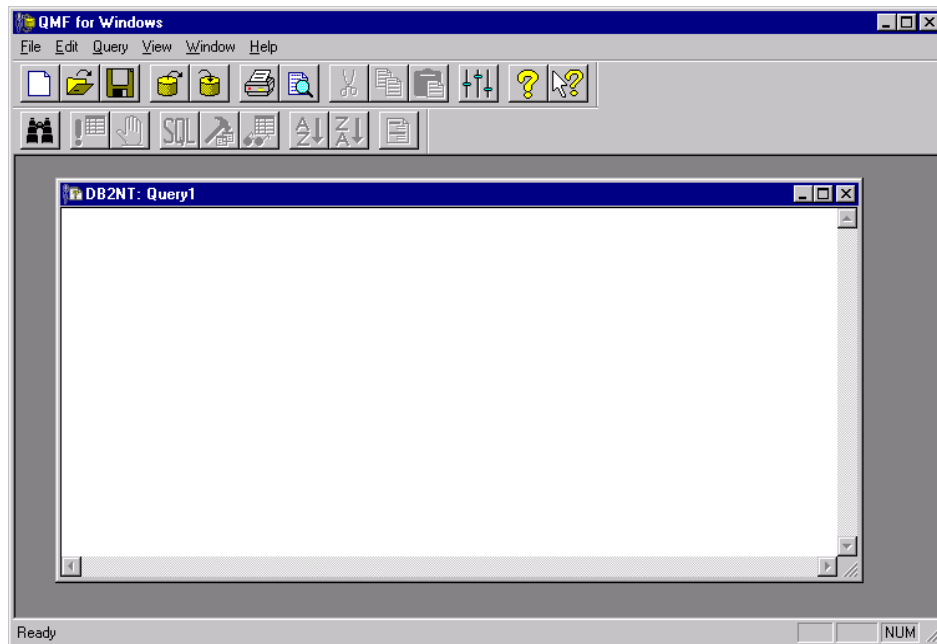


Figure 86. New query window

3. If the server shown is not the server you want to access, click on the **Query** menu and select the **Set Server...** menu to switch to the desired server.
4. In the SQL Query window, type the SQL statement.
5. Before saving the query you can execute it to test your SQL statement and to check if the result is exactly what you want. You can run the query using the **Run Query** button, using the **Query** and **Run** menus, or using the short cut **Ctrl + R**. To go back and change the SQL statement, you have to click on the **View SQL** button. Repeat this process until the query returns the desired result.
6. As a last step, when the query is correct, you may wish to save it if it will be used repetitively. There are two possibilities for saving the query. You can save it on a server or in a file. To save it on a server you can use the **File** and **Save at Server...** menus or just click on the **Save at Server** button and the window shown in Figure 87 will be displayed. In that window, you have to enter the owner, query name, comment, select the appropriate checkbox if the query is to be shared with other users and click on the **OK** button. To save the query in a file, it is necessary to click on the **Save** button, or use the **File** and **Save** or **Save As...** menus.

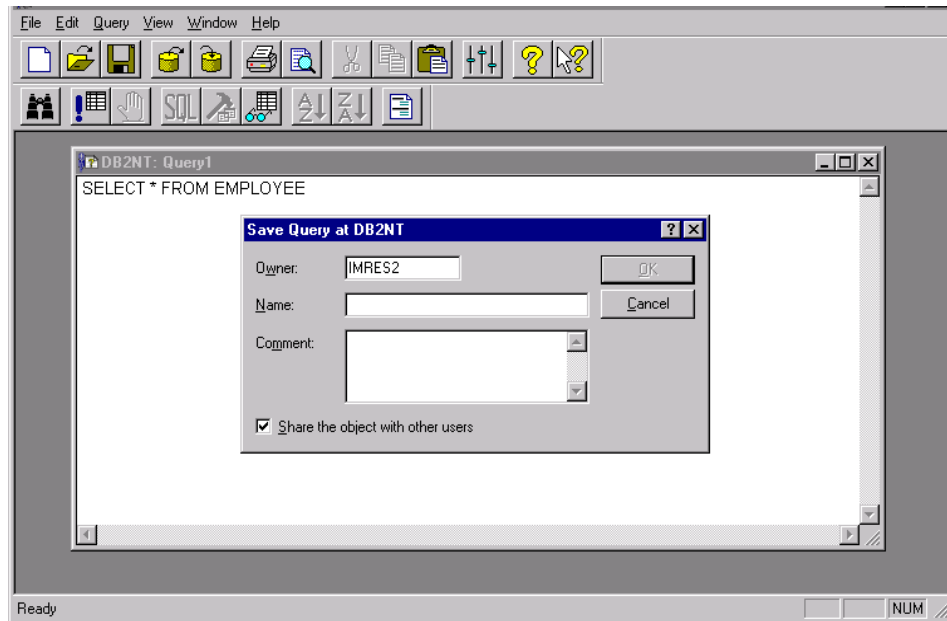


Figure 87. Save new SQL query at server

There is also another way of creating SQL queries. In the list of objects, select a table. Some additional options in the toolbar and menu will become available. The options are **Draw Select**, **Draw Insert** and **Draw Update**. When using one of these options a new SQL query will be created with an SQL statement predefined to use the selected table. Some modifications in these SQL statements may be necessary.

One powerful feature of QMF is that it allows you to create queries using variables. That means that whenever you execute a query QMF will prompt you to input the values for these variables and execute the query with the values you entered. To do this, use the '&' character as a prefix to a string in the query and it will become a variable. For example:

```
SELECT *
FROM OWNER.TABLE
WHERE COLUMN = &VAR
```

In this case, the &VAR is called a *substitution variable*. Whenever this query is executed, a window such as shown in Figure 88 will prompt for the value of that variable. Be aware that whatever text you type in that window will be copied into the query. So, if you are comparing the value with a column of the string data type, you also have to include quotations marks.

For more information on variables, please refer to Appendix A, “Working with variables” on page 277.

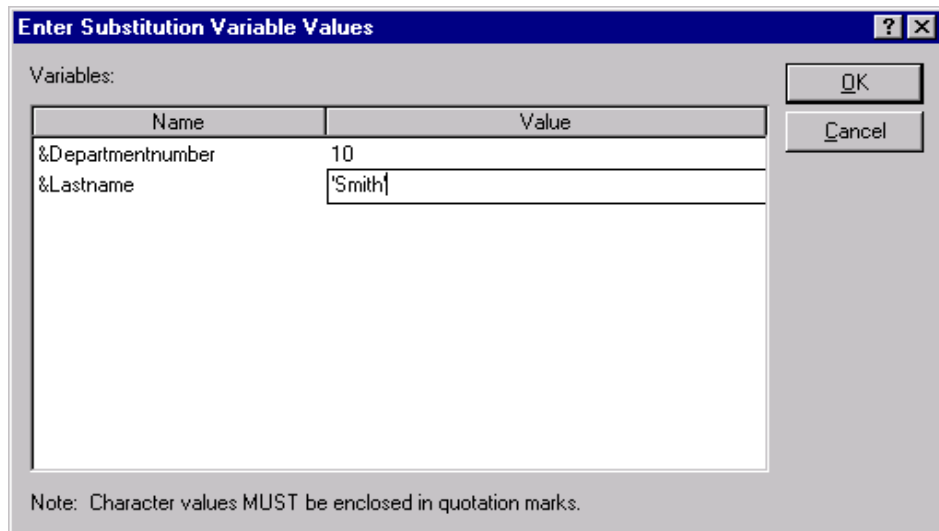


Figure 88. Substitution variable

6.5.2.2 Prompted queries

A prompted query is an easy way of creating your queries. Using prompted queries, the user does not have to know how to write SQL statements. Still, it is important to have some knowledge of SQL, and also some knowledge of your database design. To create a prompted query, follow the steps below:

1. Go to the **File** menu, select **New** and **Prompted Query**. After this, a window similar to the one on Figure 89 will be displayed.

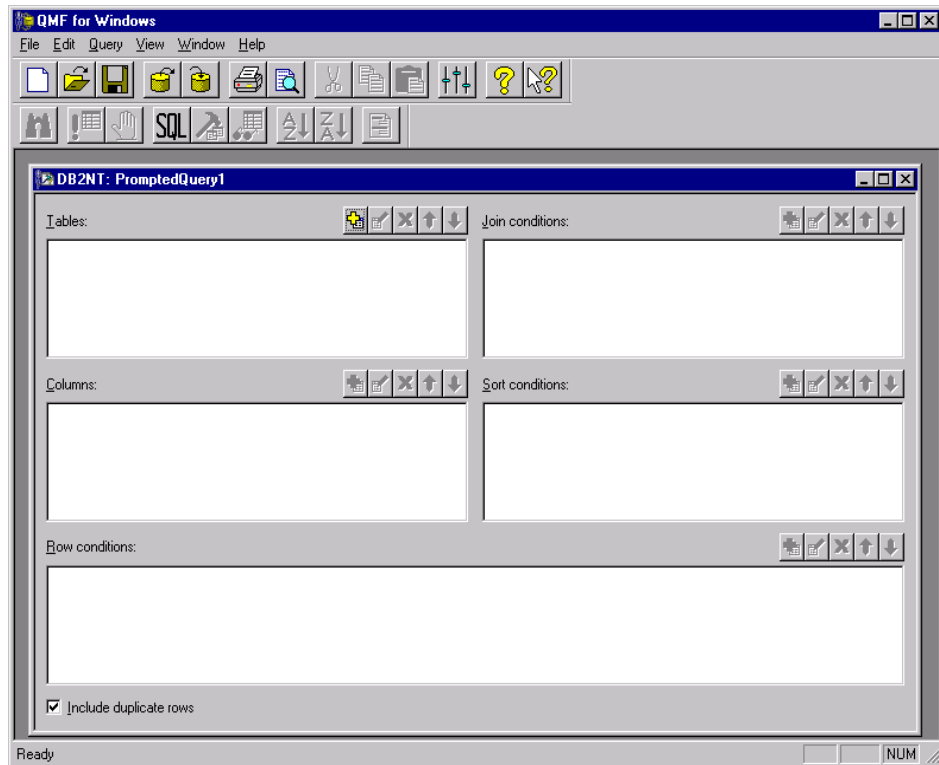


Figure 89. New prompted query window

2. Pay attention to the title of this window as it indicates the database server accessed by that query and the query name. If the server is not the one you wish, go to **Query** menu and select **Set Server...** to change it.
3. This window is divided into several sections. The first one you have to fill out is the table section, which is in the top left corner. Within this section you need to select the tables being using in the new query. To do this, click on the **Add Table** button. A dialog will be displayed to either enter the table owner and name or click on the **Add From List...** button to select it from a list, as shown in Figure 90. To add a table from the list, double-click on the table name, or select the table and click the **Add** button.

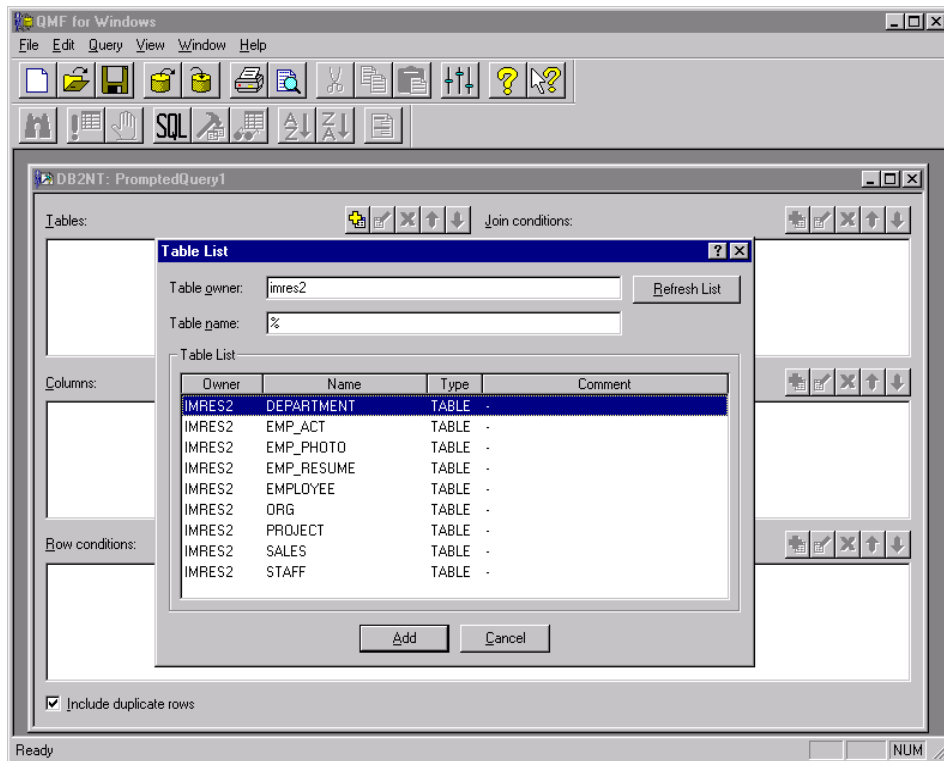


Figure 90. Select tables to a new prompted query,

4. If you select more than one table, another window will be displayed for you to enter the join condition to select those columns used to link the tables to each other. This window is shown in Figure 91. This join condition is displayed in the second section on the top right of the window. There is always the possibility to change, add, or delete join conditions at any time by using the button at the top of this section. QMF for Windows remembers the most recently used join condition for any set of tables to save time in the creation of future join conditions using the same set of tables. The selection of the join conditions is extremely important for the result of the query. Therefore, you have to know the structure of the tables inside your database. Be aware that the join condition must be accurate; otherwise, the result of the query may not be what you expected.

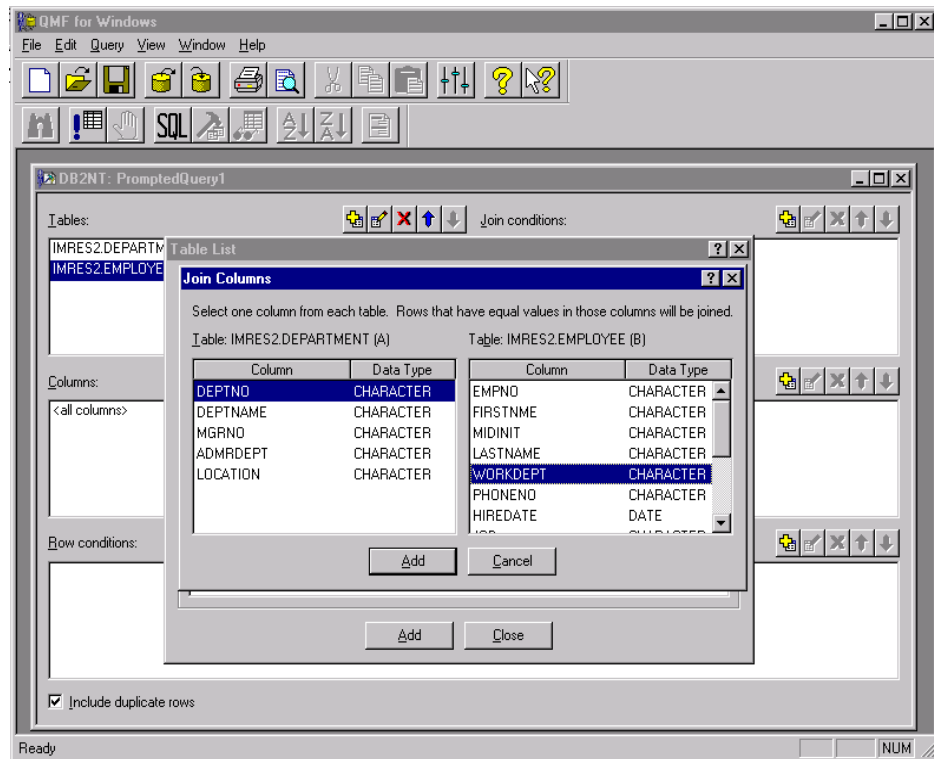


Figure 91. Adding join condition to a new prompted query

5. After the tables are selected and the join condition is set, the next section to fill out is the Column section. There, you have to select which columns will be retrieved and in which order. There is also the possibility to do some calculations on the columns. To do this, it is necessary to click on the **Add Column** button, and a window similar to the one in Figure 92 will be displayed. You can select the column by double-clicking on the column name, or selecting it and then clicking on the **Add** button. In this window, it is also possible to add an *SQL expression* for a new column, like adding two columns (as shown in Figure 92), dividing a column by 100 to get the percentage, and more complex formulas such as concatenating strings, or any other formulas that may be necessary. In case your expression is written incorrectly, QMF for Windows will not add the column, and will display an error message.

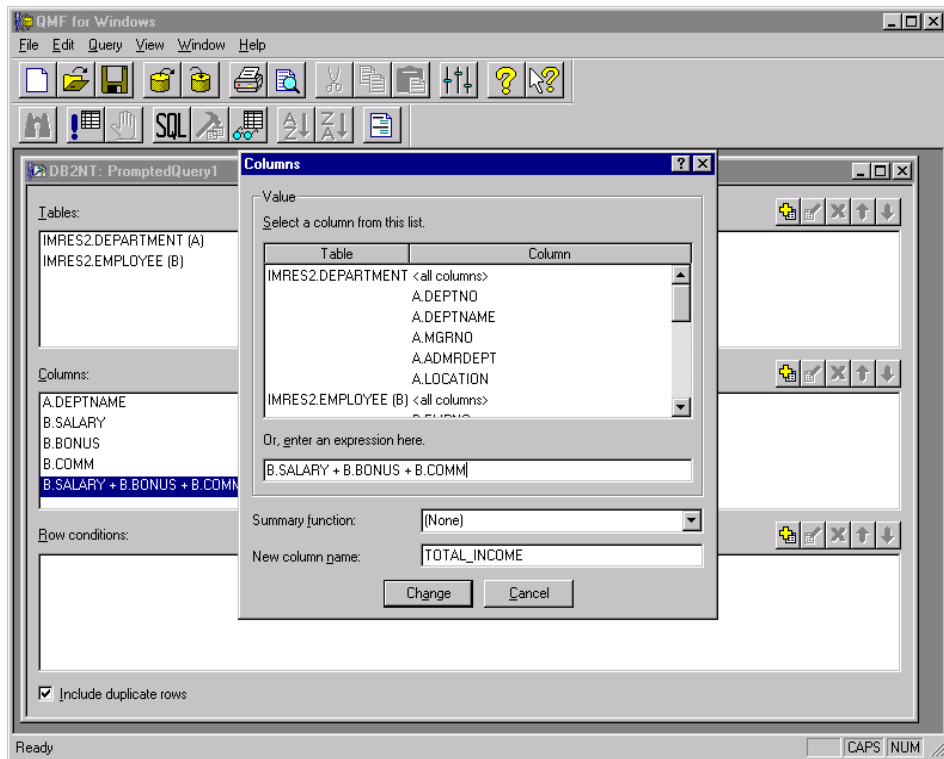


Figure 92. Adding columns to a new prompted query

6. The next section is the Sort section. In this section you can select the column by which the query result will be sorted. You can also select if the sort is to be ascending or descending. To add a sort condition, click on the **Add Sort Condition** button, and a window similar to the one in Figure 93 will appear. Then select the column for the sort condition, select the direction of the sort (ascending or descending), and click on the **Add** button.

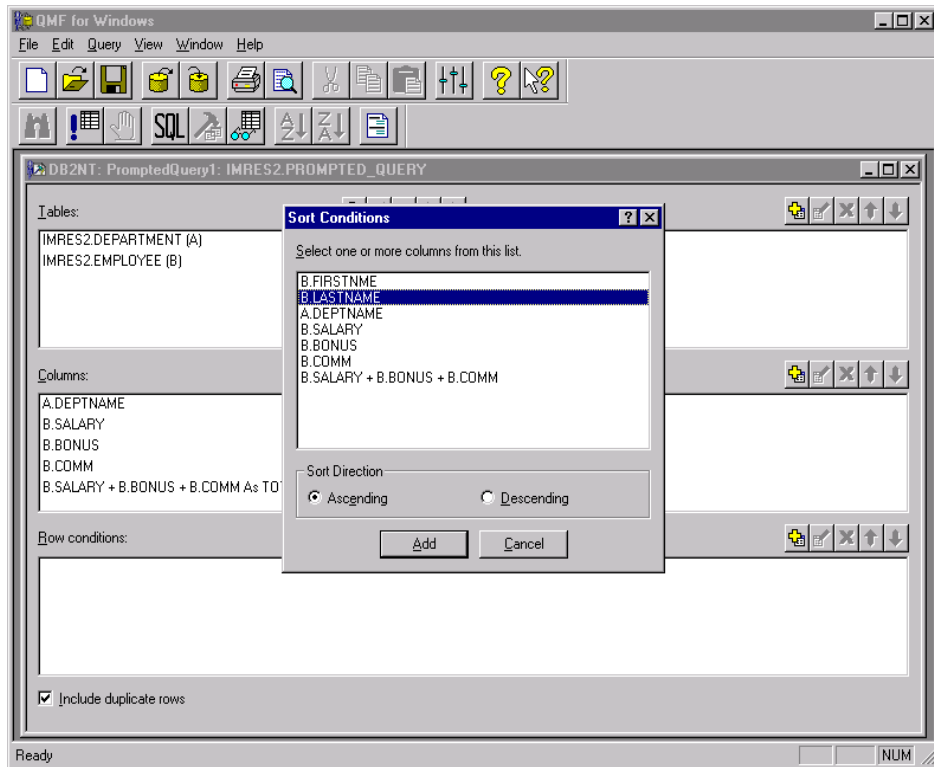


Figure 93. Add sort condition to a new prompted query

7. The last section is the Row Conditions section. It allows you to select a subset of the data in the selected tables. In other words, you can use a condition to retrieve only the rows that correspond to it. For example, if you selected a sales table and want only the sales numbers from a certain date, you can enter this condition here. To add the Row Conditions, click on the **Add Row Conditions** button and a window similar to the one in Figure 94 will appear. In this window, it is possible to select a column, an operator, and a value (as shown) where the column is the HIREDATE, the operator is “Less than or equal to” and the value is “01/01/1980”. This condition means that the data shown in the result will only apply to employees that have been hired before or at the first of January, 1980. Click on the **Add** button to add the condition to the query. This process can be repeated several times until all conditions are added.

When adding more than one condition, pay close attention to the option at the top left corner of that window that says “Connector”. This will link the different conditions to one another by the AND clause or the OR clause. For example, if a second condition is added to the prior example saying that only employees who are male are to be selected, and the two conditions are linked by the AND clause, it would mean that only employees who correspond to *both* conditions will be in the result. If these conditions are linked by the OR clause, it would mean that employees who correspond to *either* of the conditions would be shown in the result.

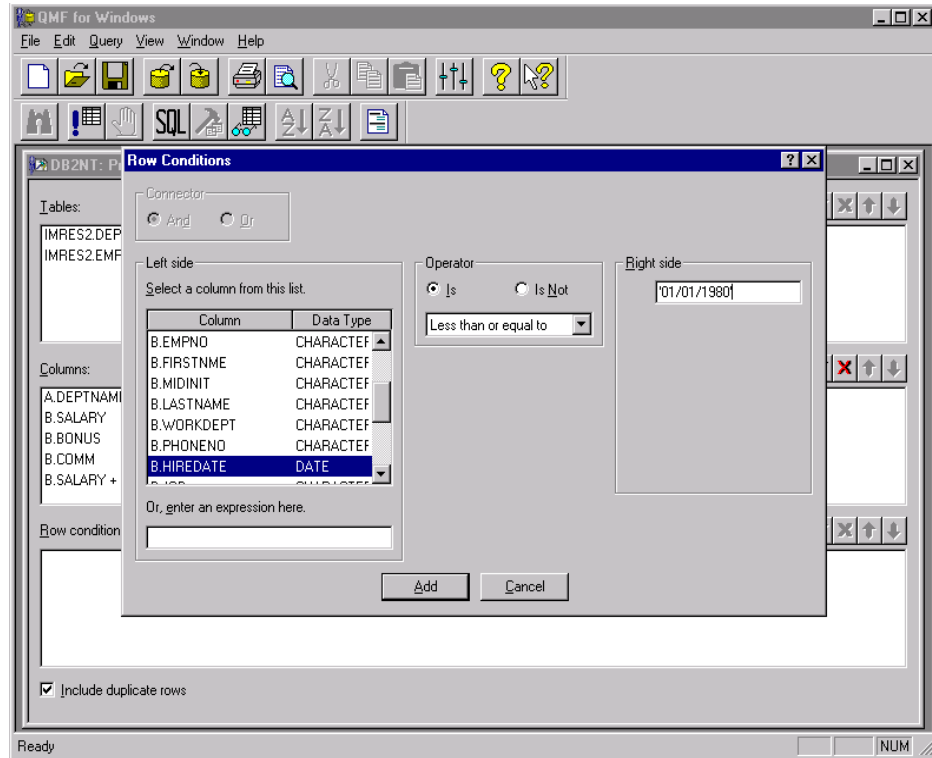


Figure 94. Add row condition to a new prompted query

- The prompted query is now ready and can be run by clicking the **Run Query** button on the toolbar, using the **Query** and **Run** menu, or using the shortcut **CTRL +R**. Doing this, the result of the query will be displayed as shown in Figure 95, and you can verify if your query is exactly the way you wanted it to be. If you want to change the query later, click on the **View Prompted** button on the toolbar and the Prompted Query window will be displayed again. Then go to the section you want to modify and click on the **Change** button for the selected section.

FIRSTNAME	LASTNAME	DEPTNAME	SALARY	BONUS	COMM	TOTAL_INCOME
BRUCE	ADAMSON	MANUFACTU	25280.00	500.00	2022.00	27802.00
DAVID	BROWN	MANUFACTU	27740.00	600.00	2217.00	30557.00
JOHN	GEYER	SUPPORT SE	40175.00	800.00	3214.00	44189.00
JASON	GOUNOT	SOFTWARE	23840.00	500.00	1907.00	26247.00
CHRISTINE	HAAS	SPIFFY COM	52750.00	1000.00	4220.00	57970.00
EILEEN	HENDERSON	OPERATIONS	29750.00	600.00	2380.00	32730.00
JAMES	JEFFERSON	ADMINISTRA	22180.00	400.00	1774.00	24354.00
SYBIL	JOHNSON	ADMINISTRA	17250.00	300.00	1380.00	18930.00
WILLIAM	JONES	MANUFACTU	18270.00	400.00	1462.00	20132.00
SALLY	KVYAN	INFORMATI	38250.00	800.00	3060.00	42110.00
WMING	LEE	SOFTWARE	25370.00	500.00	2030.00	27900.00
VINCENZO	LUCCHESSI	SPIFFY COM	46500.00	900.00	3720.00	51120.00
JENNIFER	LUTZ	MANUFACTU	29840.00	600.00	2387.00	32827.00
SALVATORE	MARINO	ADMINISTRA	28760.00	600.00	2301.00	31661.00
RAMLAL	MEHTA	SOFTWARE	19950.00	400.00	1596.00	21946.00
HEATHER	NICHOLLS	INFORMATI	28420.00	600.00	2274.00	31294.00
SEAN	O'CONNELL	SPIFFY COM	29250.00	600.00	2340.00	32190.00
ELIZABETH	PIANKA	MANUFACTU	22250.00	400.00	1780.00	24430.00
DOLORES	QUINTANA	INFORMATI	23800.00	500.00	1904.00	26204.00
ETHEL	SCHNEIDER	OPERATIONS	26250.00	500.00	2100.00	28850.00
MARILYN	SCOUTTEN	MANUFACTU	21340.00	500.00	1707.00	23547.00
MAUDE	SETRIGHT	OPERATIONS	15900.00	300.00	1272.00	17472.00
DANIEL	SMITH	ADMINISTRA	19180.00	400.00	1534.00	21114.00
PHILIP	SMITH	OPERATIONS	17750.00	400.00	1420.00	19570.00
IRVING	STERN	MANUFACTU	32250.00	500.00	2580.00	35330.00
MICHAEL	THOMPSON	PLANNING	41250.00	800.00	3300.00	45350.00
JAMES	WALKER	MANUFACTU	20450.00	400.00	1636.00	22486.00
MASATOSHI	YOSHIMURA	MANUFACTU	24680.00	500.00	1974.00	27154.00

Figure 95. Displaying result of the new prompted query

- If you want to see the SQL statement that the prompted query is going to submit to the server, you can do that by clicking on the **View SQL** button. Notice that the SQL is read-only, meaning you cannot change the SQL statement directly. To do this, it is necessary to convert your prompted query to an SQL query using the **Query** and **Convert to SQL** menus. Doing this will create a new SQL query, and the original prompted query will be maintained, but they are not linked to each other.

Note:

This is different than the way QMF works on the host platform.

So, if you change the prompted query the SQL query will not be changed automatically and vice versa. Sometimes it is necessary to convert prompted queries to SQL queries for other reasons.

- The final step is to save the query. This can be done in two ways: save the query at a server, or save it in a file. To save the query at a server, you can either click on the **Save At Server** button or use the **File** and the **Save At Server...** menu. A window as shown in Figure 96 will appear. You then have to type the owner, name, and comments of that query as well as selecting if that query can be shared with others users. Click on the **OK** button and the query will be saved. To save a query in a file, you have to use the **Save** button or the **File** and **Save As...** menu.

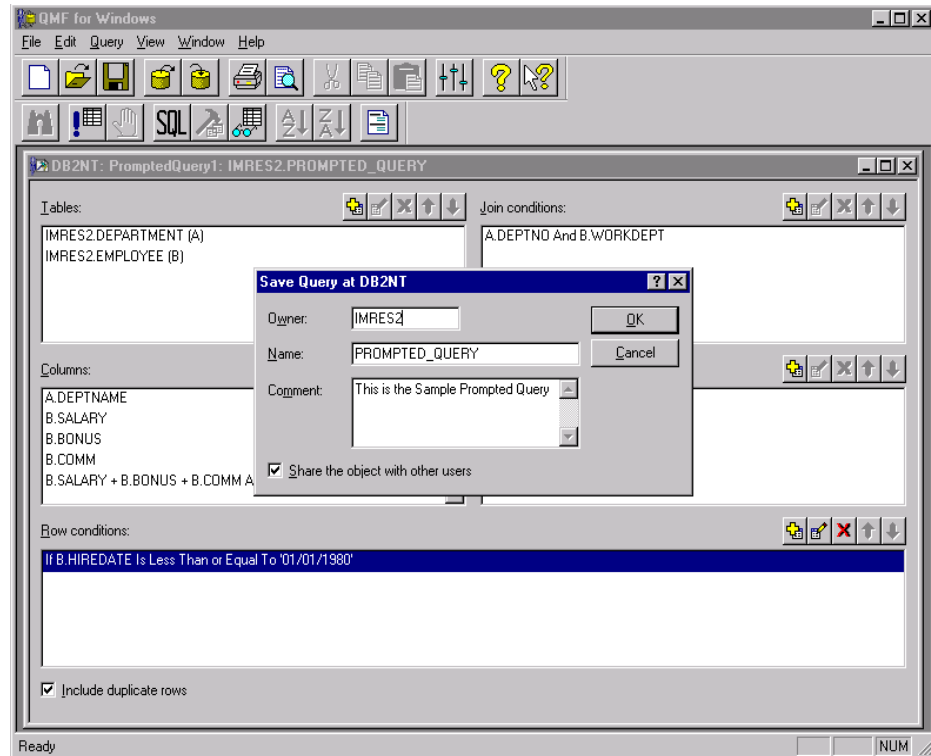


Figure 96. Saving new prompted query

6.5.3 Create new form and report

Forms provide the possibility to specify the way the data retrieved from a Query will be organized. You can define where each column will be displayed, the width of each column, breaks, groups, summaries, and many other things. After the data is formatted the way you want it to be, it can be printed or exported to HTML or to a text file.

The most important concept that should be kept in mind is that a form is always linked to a query, and that those two objects together define the report. **There is no QMF object called report**, as the report is the result of a query displayed according to a linked form. The columns in the form must correspond precisely to the columns of the query. You can have two queries that use the same layout and use the same form to display the data, but queries with different layouts cannot use the same form.

There are two main ways of creating a form. You can create an empty form and create all fields from the beginning, or you can use the default form using a query from the starting point.

6.5.3.1 Empty form

The first thing you have to know when creating an empty form is the structure of your query — this means the name and data type of all columns from the query and the order in which they are selected. In the following example, we will create a form for the query called PROMPTED_QUERY that was created in 6.5.2.2, “Prompted queries” on page 203. The columns of that query are specified in the Table 5.

Table 5. Columns on query PROMPTED_QUERY

Column	Data Type
FIRSTNME	STRING(12)
LASTNAME	STRING(15)
DEPTNAME	STRING(29)
SALARY	NUMERIC (9,2)
BONUS	NUMERIC (9,2)
COMM	NUMERIC (9,2)
TOTAL_INCOME	NUMERIC (9,2)

To create a new form, follow the steps below:

1. Go to the **File, New and Form** menu. A window similar to the one in Figure 97 will appear.

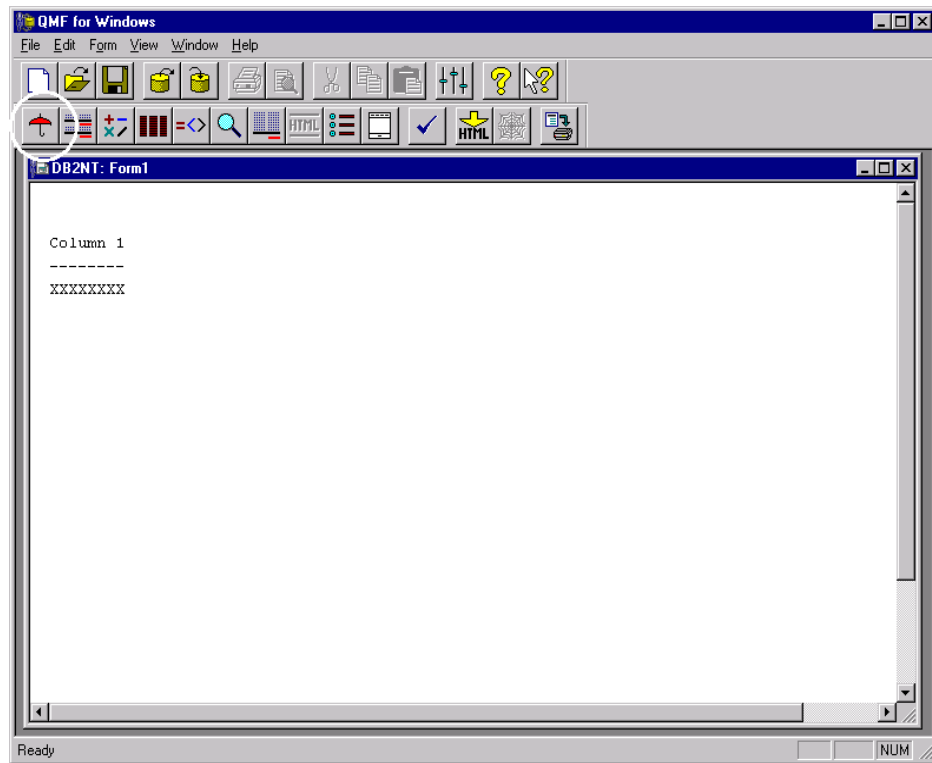


Figure 97. New blank form

2. To create new columns for the form, click on the **Main** button or use the **Form** and **Main...** menu. The window shown in Figure 98 will be displayed.

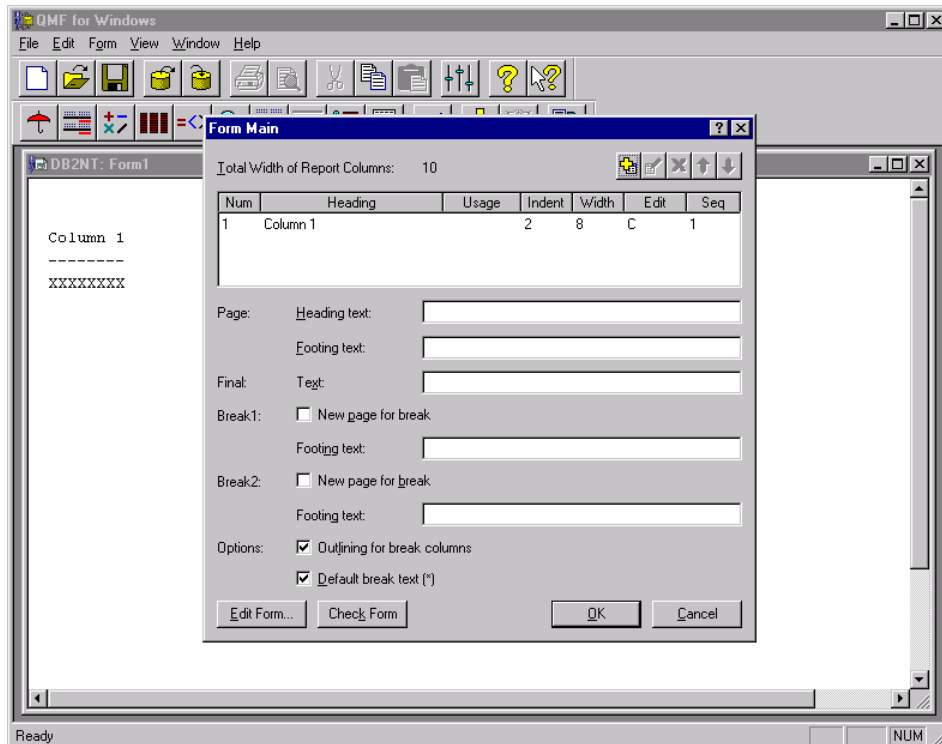


Figure 98. New blank form main window

3. In that window, click on the existing column called **Column 1** and change its name to the one you would like to appear as the column header.
4. Now it is necessary to add the other columns. To add a column, click on the **Add Column** button and a window as shown in Figure 99 will be displayed. In that window, type a name for the column and click on the **Add** button. The column will be added and the window will be cleared for you to enter the next column. Repeat this process until all required columns are added. On the last column, instead of clicking on the **Add** button, click on the **OK** button and the window will be closed.
5. After all columns are added, it is necessary to modify the width of these columns. That can be done by simply clicking on the width column on the **Main** window for each column, and modifying its value. Remember that the width also should match the width of the columns in the query.
6. The space between the columns is 2, by default. But you can change it by changing the **Indent** value for each column.

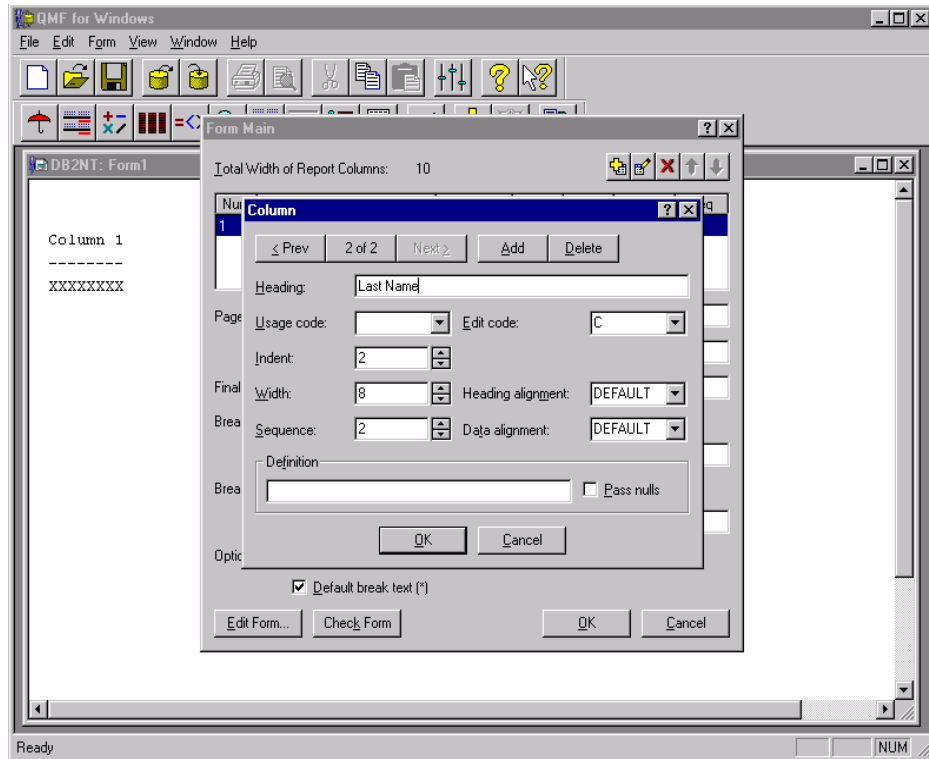


Figure 99. Adding columns to a new blank form

7. Another adjustment that can be made to the columns is the **Edit Code**. This code indicates the data type of the column in the form. Again, this data type has to match the data type in the query. There are several Edit Codes for several data types. Table 6 shows the available Edit Codes. There are some data types in this table that will be rarely used. To modify the Edit Code, just click on the **Edit** column and choose the Edit Code from the list. Click on the **OK** button and the changes will be applied to your form.

Table 6. Edit Codes

Data Type	Edit Code	Format	Notes
Character	C	Display character data.	No special formatting
	CW	Display character data with wrapping based on column width.	No special formatting unless the value cannot fit within the width of the column. In that case, text will be wrapped within the column width to subsequent lines.
	CT	Display character data with wrapping based on column value.	No special formatting unless the value cannot fit within the width of the column. In that case, text will be wrapped based on the column value to subsequent lines. That is, text will be wrapped at the end of a line when a blank is found. If the wrapped text is too long to fit in the column and does not contain a blank, the text is wrapped within the column width.
	CDx	Display character data with wrapping based on the specified delimiter.	The column is always wrapped based on the delimiter specified. The delimiter can be any single character, including a blank. The delimiter character does not appear in the report.
	X	Format data as a series of hexadecimal characters.	
	XW	Format data as a series of hexadecimal characters.	Columns are wrapped according to the rules specified for the CW edit code.
	B	Format data as a series of 0's and 1's.	
	BW	Format data as a series of 0's and 1's.	Columns are wrapped according to the rules specified for the CW edit code.
Date	TDYx	YYYY/MM/DD	Year first.
	TDMx	MM/DD/YYYY	Month first.
	TDDx	DD/MM/YYYY	Day first.
	TDYAx	YY/MM/DD	Two-digit year first.
	TDMAx	MM/DD/YY	Month first and two-digit year.
	TDDAx	DD/MM/YY	Day first and two-digit year.
	TDL	Formats date data according to the format on the Windows Control Panel.	

Data Type	Edit Code	Format	Notes
Graphic	G	Display graphic data.	No special formatting
	GW	Display graphic data with wrapping based on column width.	No special formatting unless the value cannot fit within the width of the column. In that case, text will be wrapped within the column width to subsequent lines.
Numeric	E	Displays numbers in scientific notation.	Up to 17 significant digits are shown, even if the width of the column can accommodate more.
	Dnn	Displays numbers in decimal notation.	Display leading zeros? NO Display negative sign? YES Display thousands separators? YES Display currency symbol? YES Display percent sign? NO
	Inn	Displays numbers in decimal notation.	Display leading zeros? YES Display negative sign? YES Display thousands separators? NO Display currency symbol? NO Display percent sign? NO
	Jnn	Displays numbers in decimal notation.	Display leading zeros? YES Display negative sign? NO Display thousands separators? NO Display currency symbol? NO Display percent sign? NO
	Knn	Displays numbers in decimal notation.	Display leading zeros? NO Display negative sign? YES Display thousands separators? YES Display currency symbol? NO Display percent sign? NO
	Lnn	Displays numbers in decimal notation.	Display leading zeros? NO Display negative sign? YES Display thousands separators? NO Display currency symbol? NO Display percent sign? NO
	Pnn	Displays numbers in decimal notation.	Display leading zeros? NO Display negative sign? YES Display thousands separators? YES Display currency symbol? NO Display percent sign? YES

Data Type	Edit Code	Format	Notes
Time	TTSx	HHxMMxSS	24-hour clock, including seconds.
	TTCx	HHxMMxSS	12-hour clock, including seconds.
	TTAx	HHxMM	Abbreviated 24-hour clock (no seconds).
	TTAN	HHMM	Abbreviated 24-hour clock (no seconds) without any delimiter between time values.
	TTUx	HHxMM PMHHxMM AM	USA format.
	TTL	Formats time data according to the format on the Windows Control Panel.	
Timestamp	TSI	yyyy-mm-dd-hh.mm.ss.nnn nnn	yyyy is the four-digit year mm is the two-digit month dd is the two-digit day hh is the two-digit hour mm is the two-digit minute ss is the two-digit second nnnnnn is the six-digit microsecond
User Defined	VSSN	xxx-xx-xxxx	Social security number format.
	VEL	xxx) xxx-xxxx	Telephone number format.
	VZIP	xxxxx-xxxx	Zip code format.

8. If you want to display a title on top of the report, type the text in the **Heading Text** field. It is also possible to display some text at the bottom of the page by entering it in the **Footing Text** field. For beginning users, we do not recommend changing any other parameter on this window.
9. The form is now ready to be used if every column in the form matches the columns in the query. However, to use the form in the future, it is necessary to save it. You can save the form on the server or in a file. To save the form on the server, click on the **Save At Server** button or use the **File** and **Save At Server...** menu. The window shown in Figure 100 will appear. Fill out the owner, name and comment fields as well as selecting if the form is to be shared with other users, and click on the **OK** button. To save the form in a file, click on the **Save** button or use the **File** and **Save As...** menu.

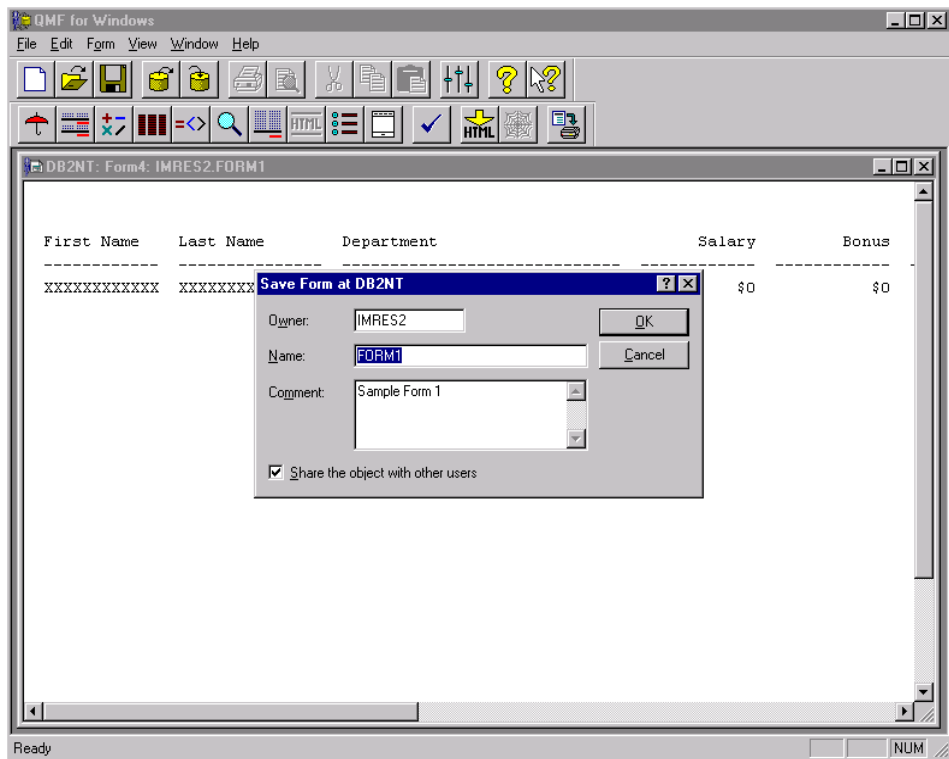


Figure 100. Saving new blank form

10. To use the form, you first have to open run the query. For more information on how to access existing objects, refer to 6.3, “Accessing existing objects” on page 183.
11. After the query has been executed, go to the **Query and Display Report...** menu. A window as the one shown in Figure 101 will be displayed, prompting you to select a form. To use the form you just saved, either select the option **From Database** or **From File**, depending on how the form was saved. In case it was saved on the server, select the database, owner, and name of the form (or select it from the list by clicking on the **List Form...** button). If the form was saved in a file, select the file name and click on the **OK** button. The form will be displayed with the result data of the query in it.

6.5.3.2 Default form

There is also a way of creating a form directly from a query, which is the most common use of creating a form. This means that QMF will create the form using the query columns, data types, width, names, and so on. This way, it is not necessary to go through all the steps as when creating a new empty form. To create a default form, follow the steps below:

1. Open the query you wish to create the form for and run it.
2. Go to the **Query** and **Display Report...** menu, and a window such as the one in Figure 101 will appear.

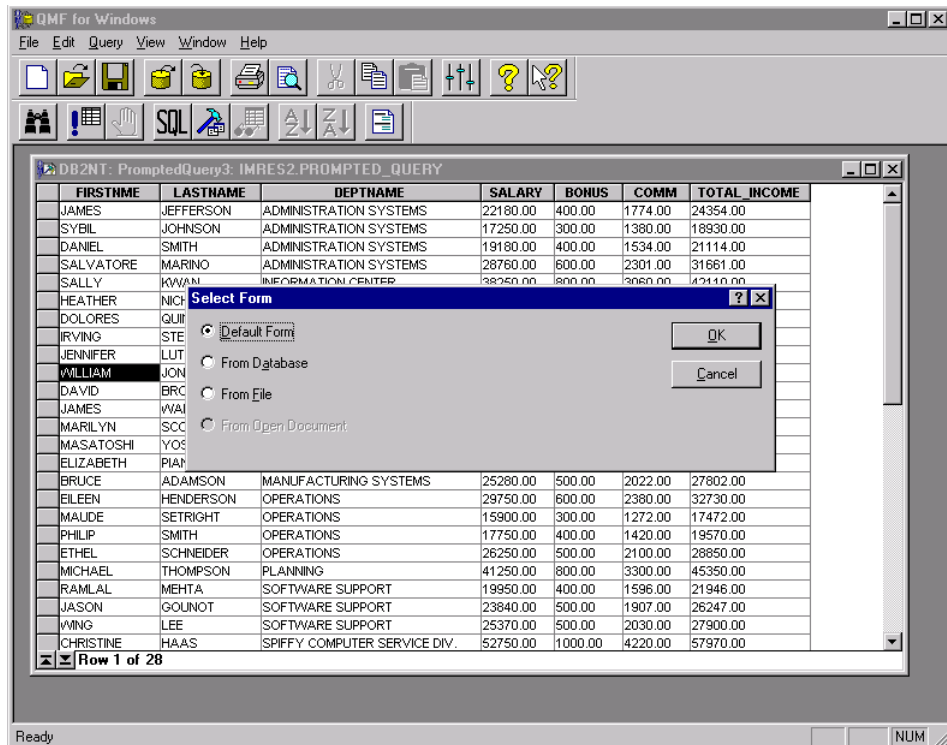


Figure 101. Ways of creating new form

3. Select the **Default Form** option and click the **OK** button. A new form will be automatically created with the same structure as the query result. Also, the query data will be shown in this form.

4. You can now save the form on the server or on a file. To save the form on the server, click on the **Save At Server** button or use the **File** and **Save At Server...** menu. The window shown in Figure 100 will appear. Fill out the owner, name, and comment fields, as well as selecting if the form is to be shared with others users. To save the form in a file, Click on the **Save** button or use the **File** and **Save As...** menu. Then click the **OK** button.

If the form is saved, instead of creating another default form every time you access the query, you can use this form by selecting the option **From Database** or **From File**.

6.5.3.3 Changing the Form

Many reports require additional formatting, totals, grouping, and so on. Until now, only a minimal number of forms options have been shown, but there are more options possible when using forms. In this chapter, only the most common things will be shown.

A very important feature of the forms are the **Usage Codes**. These are functions that can be applied to each column to provide more information within the form. To access this, it is necessary to click on the **Main** button or use the **Form** and **Main...** menu. A complete reference on these usage codes is available in Table 7. We will only discuss the most common functions.

Table 7. Usage codes

Usage Code	Description	Notes
ACROSS	Enables you to produce a report with horizontal control breaks.	<p>In an ACROSS report:</p> <p>1) The number and titles of the columns in the report are dependent on the values in the ACROSS column. There is one set of report columns for each value in the ACROSS column and the heading for each is the value of the column. The set of report columns includes a column for each one that uses an aggregation usage code (for example, SUM, AVERAGE, COUNT). NOTE: You can only have one ACROSS column in a report.</p> <p>2) The number of rows and the title of each row in the report are dependent on the values in the GROUP column(s). There is one row for each value in the GROUP column(s) and the title of each row is the value of the column(s). NOTE: The CSUM, PCT, CPCT, TPCT, and TCPCT usage codes are only partially supported when generating reports that also use the ACROSS usage code.</p>

Usage Code	Description	Notes
AVERAGE	Average of the values in the column.	This usage code is only valid for numeric data. This calculated value appears as a total in the report. The calculated value is formatted with the edit code of the column.
BREAKn	Provide a control break level (where n represents a number between 1 and 6).	For example, BREAK1 specifies a control column for a level-1 break and BREAK2 specifies a control column for a level-2 break. Any change in the value of the column causes a section break in the report. Subtotals are displayed for columns whose usage is one of the aggregation usages. Also, the text specified in the appropriate Form Break component is displayed. Your query should use an ORDER BY clause that matches your BREAK columns.
BREAKnX	Same as BREAKn, except the control column is omitted from the report.	Same as BREAKn.
CALCid	The evaluation of the calculation expression in the Form Calculations component whose ID equals "id".	This calculated value appears as a total in the report and applies only to the last row of data. The calculated value is formatted with the edit code of the column. If the column value is used in the calculation, only the last row of data is evaluated.
COUNT	Count of the non-null values in the column.	This calculated value appears as a total in the report. The calculated value is formatted with the edit code K.
CPCT	Cumulative percentage each value of the column is of the current total.	This calculated value replaces each detail line value and also appears as a total in the report. The calculated value is formatted with the edit code of the column. NOTE: The CPCT usage code is only partially supported when generating reports that also use the ACROSS usage code.
CSUM	Cumulative sum of the values in the column.	This calculated value replaces each detail line value and also appears as a total in the report. The calculated value is formatted with the edit code of the column. NOTE: The CSUM usage code is only partially supported when generating reports that also use the ACROSS usage code.
FIRST	First value in the column.	This calculated value appears as a total in the report. The calculated value is formatted with the edit code of the column.

Usage Code	Description	Notes
GROUP	Displays only one line of summary data for each set of values in the column.	More than one column can have usage GROUP. If so, a change in value in any column starts a new group. All other columns with no usage code are omitted from the report.
LAST	Last value in the column	This calculated value appears as a total in the report. The calculated value is formatted with the edit code of the column.
MAXIMUM	Maximum value in the column.	This calculated value appears as a total in the report. The calculated value is formatted with the edit code of the column.
MINIMUM	Minimum value in the column.	This calculated value appears as a total in the report. The calculated value is formatted with the edit code of the column.
OMIT	Excludes the column from the report	The column and its values are not included in the tabular report. The values in the column can still appear in the report (for example, in a break footing) by use of form variables (such as &n, where n represents the column number).
PCT	Percentage each value of the column is of the current total.	This calculated value replaces each detail line value and also appears as a total in the report. The calculated value is formatted with the edit code of the column. NOTE: The PCT usage code is only partially supported when generating reports that also use the ACROSS usage code.
STDEV	Standard deviation of the values in the column.	This usage code is only valid for numeric data. This calculated value appears as a total in the report. The calculated value is formatted with the edit code of the column.
SUM	Sum of the values in the column.	This usage code is only valid for numeric data. This calculated value appears as a total in the report. The calculated value is formatted with the edit code of the column.
TPCT	Percentage each value of the column is of the final total.	This calculated value replaces each detail line value and also appears as a total in the report. The calculated value is formatted with the edit code of the column. NOTE: The TPCT usage code is only partially supported when generating reports that also use the ACROSS usage code.
TCPCT	Cumulative percentage each value of the column is of the final total.	This calculated value replaces each detail line value and also appears as a total in the report. The calculated value is formatted with the edit code of the column. NOTE: The TCPCT usage code is only partially supported when generating reports that also use the ACROSS usage code.

The first important function is the SUM. This function allows you to view the total of a specific column. It is clear that this function only works for columns that are numeric. In our example, using the form created in 6.5.3.1, “Empty form” on page 212, we can use this function on the Salary, Bonus, Commission, and Total Income columns by simply clicking on the **Usage** column and selecting the SUM function from the list as shown in Figure 102. After this, a new row will be displayed at the end of the form showing the resulting sum of the columns.

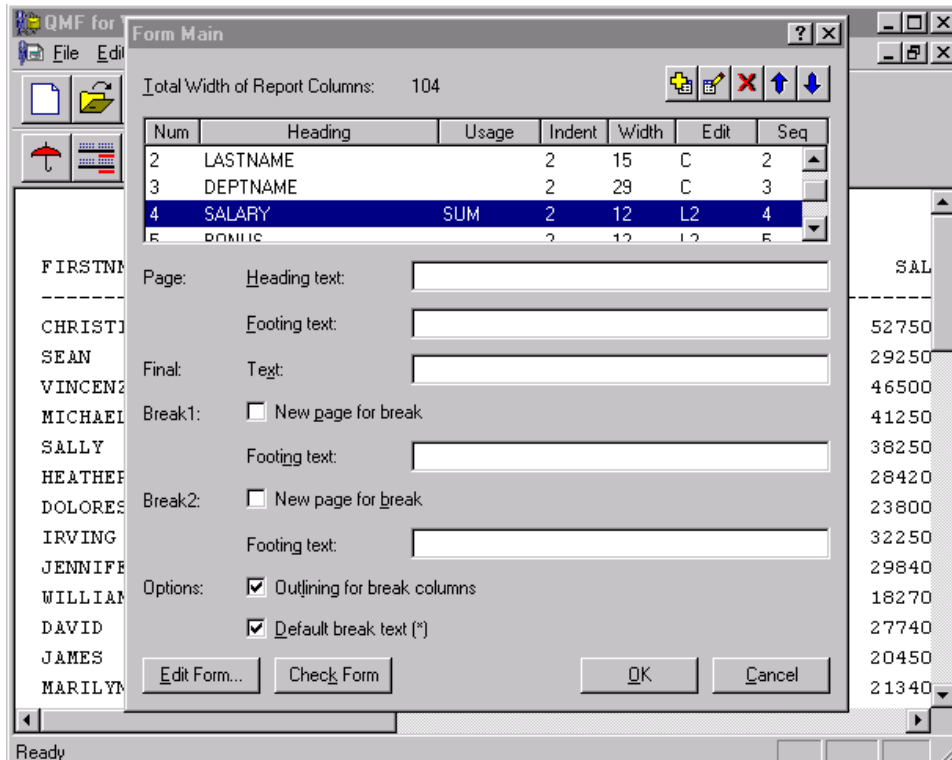


Figure 102. Using the sum function

In the same way we used the SUM function, we could have used many other functions such as AVERAGE, COUNT, MAXIMUM, and so on. The resulting value would be different, but the layout of the form would be the same.

Another important usage code is called BREAK. This function creates a new break every time the value of a specific column changes. **Therefore, if you want to see a break by a specific column, the query must be ordered by that column.** In our example, we will create a break by Department, so the query must be ordered by Department. To do this, click on the **Usage** and

select the BREAK function on the column Department. The result is shown in Figure 103.

First Name	Last Name	Department	Salary	Bonus
JAMES	JEFFERSON	ADMINISTRATION SYSTEMS	\$22,180	\$400
SYBIL	JOHNSON		\$17,250	\$300
DANIEL	SMITH		\$19,180	\$400
SALVATORE	MARINO		\$28,760	\$600
			* \$87,370	\$1,700
SALLY	KWAN	INFORMATION CENTER	\$38,250	\$800
HEATHER	NICHOLLS		\$28,420	\$600
DOLORES	QUINTANA		\$23,800	\$500
			* \$90,470	\$1,900
IRVING	STERN	MANUFACTURING SYSTEMS	\$32,250	\$500
JENNIFER	LUTZ		\$29,840	\$600
WILLIAM	JONES		\$18,270	\$400
DAVID	BROWN		\$27,740	\$600
JAMES	MAYFED		\$20,450	\$400

Figure 103. Result of group and sum functions

As you can see, every time the value in the column Department changes, a break is generated. Also the sum of the Salary, Bonus, Commission, and Total Income are displayed per Department as well as a total sum at the end of the form. There are other functions, such as the GROUP function, that do similar things.

There is still one more thing that could improve our example report. That is to move the Department column to the first position. This can be done in the **Main** window using the **Seq** column as shown in Figure 104. This number represents the sequence in which the column will appear in the form. To change it, click on the number and either type the number you want or use the scroll buttons to increase or decrease the number. You have to reorder the number yourself; otherwise, you will have two columns with the sequence number, and QMF will use the column with lower number in the column **Num** to appear first.

Note:

There is a great difference between the **Num** column and the **Seq** column. The **Seq** column represents the order in which the columns will appear in the form, while the **Num** column identifies the order in which the data is mapped from the query. If you change the **Num** by clicking in the arrows on the top right corner of the **Main** screen, you will be changing the position of the columns. This means that the order of the columns on the form will no longer be corresponding to the order of the columns in the query, and that may create major problems such as not displaying the data or displaying the data incorrectly. Only use the arrows to change the position of the columns if you are sure that the columns in the query have also changed.

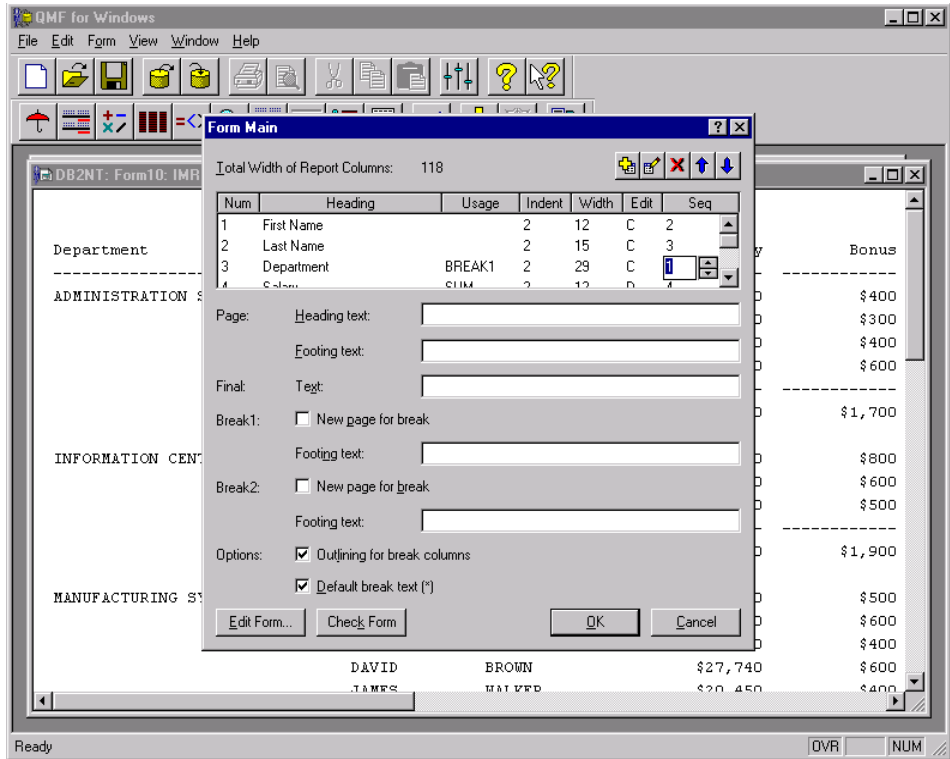


Figure 104. Modify column order in a form

The result of this change is shown in Figure 105 below. The Department Column is now shown in the first position of the form.

Department	First Name	Last Name	Salary	Bonus

ADMINISTRATION SYSTEMS	JAMES	JEFFERSON	\$22,180	\$400
	SYBIL	JOHNSON	\$17,250	\$300
	DANIEL	SMITH	\$19,180	\$400
	SALVATORE	MARINO	\$28,760	\$600
		*	\$87,370	\$1,700

INFORMATION CENTER	SALLY	KWAN	\$38,250	\$800
	HEATHER	NICHOLLS	\$28,420	\$600
	DOLORES	QUINTANA	\$23,800	\$500
		*	\$90,470	\$1,900

MANUFACTURING SYSTEMS	IRVING	STERN	\$32,250	\$500
	JENNIFER	LUTZ	\$29,840	\$600
	WILLIAM	JONES	\$18,270	\$400
	DAVID	BROWN	\$27,740	\$600
	JAMES	MAI VEP	\$20,450	\$400

Figure 105. Final result of the modified form

To make sure that there is no problem with your form, you can click on the **Check** button or use the **Form** and **Check** menu and QMF will check the entire form and show a message with the result.

6.5.4 Create new procedures

Procedures are a set of commands that will be executed one after the other. For example, it is possible to create a procedure that runs a specific query, opens a form using that query, exports that form to an HTML file, and copies it to a specified directory. A procedure contains all the commands to perform the desired actions, and when the procedure is executed, the commands will be executed in the sequence specified within the procedure.

Procedures are useful for many things; for example, scheduling a procedure to do its work overnight, especially for jobs that take a long time to be completed. It is possible create a procedure that, overnight, executes several queries and forms, export them to text files, and imports them into your favorite spreadsheet application, such as Excel or Lotus 123. When you arrive the next morning, this work (which could take several hours) has already been done for you.

To create a procedure, follow the steps below:

1. Go to the **File**, **New** and **Procedure** menu. A window similar to the one in Figure 106 will appear.

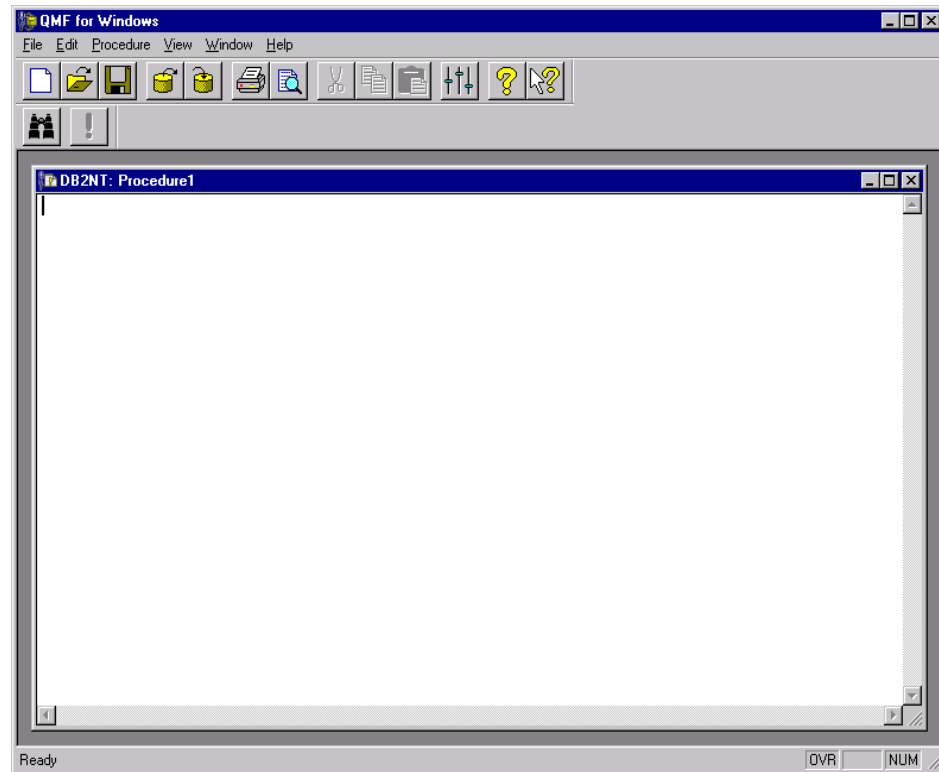


Figure 106. New procedure

2. In this window, you must now type your procedure. To do this, you need to know the commands available for procedures. A list of these commands can be found in Table 8. For a complete reference on their possible syntax, refer to the on-line help. Type the commands you want with the appropriate syntax in the window. The commands must be one per line, and they will be executed one after the other. The second command will not start until the first command has completed, and so on.

Table 8. Procedure commands

Command	Description
BOTTOM	BOTTOM scrolls to the last row of a query result set. This command is equivalent to FORWARD MAX
CONNECT	CONNECT will establish a connection to a database server. Subsequent procedure commands will run at the specified server. The running procedure's server will also be changed to the specified server. No immediate action is taken upon any other current objects within the procedure. However, subsequent commands which affect those objects may result in additional processing.
CONVERT	CONVERT will convert a prompted query to a new SQL query. The original query (whether a named object in the database or a temporary object) is unaffected by this operation.
DISPLAY	DISPLAY will display an object in temporary storage or an existing object that was saved in the database. DISPLAY for an object in temporary storage will act only upon the current object; there is no way to DISPLAY an object from temporary storage that is not the current object.
DRAW	DRAW creates a basic query for a table based on the description of the table in the database.
ERASE	ERASE removes a query, form, procedure, or table from the database.
EXPORT	EXPORT copies objects from the database or temporary storage to a file.
FORWARD	FORWARD scrolls forward in a query result set. The only acceptable parameter for this command is MAX, making it equivalent to BOTTOM.
IMPORT	IMPORT copies data from a file into temporary storage or the database.
PRINT	PRINT prints a copy of an object in temporary storage or from the database.

Command	Description
RUN	RUN executes procedures or queries from temporary storage or the database.
SAVE	SAVE stores the contents of an object in temporary storage into the database.
SEND TO	SEND TO exports reports and data from the database or temporary storage and sends them to the specified target or application in your Send To folder.
SET GLOBAL	SET GLOBAL sets the values of existing global variables or creates new variables and values. Any new global variables created exist for the entire QMF for Windows session (unless manually deleted).
SHOW	SHOW displays objects from temporary storage and is similar to DISPLAY. SHOW QUERY, SHOW FORM, and SHOW PROC activate the window containing the current query, form, or procedure, respectively. SHOW REPORT is a synonym for SHOW FORM. SHOW GLOBALS opens the Global Variables dialog box.
WINDOWS	The WINDOWS command activates the target Windows application, document, or URL.

3. Before saving the procedure, you should test it. To run a procedure, click on the **Run Procedure** button from the toolbar (shown in Figure 107) or use the **Procedure** and **Run** menu. The execution should start immediately after that. In case there is any error in the syntax of any command in the procedure, an error message will be displayed. If no error occurs, the commands will be executed. If necessary, change the procedure until it performs its intended operation without errors.
4. After testing the procedure, you may wish to save it. It is possible to save a procedure at a server or in a file. To save a procedure at the server, click on the **Save At Server** button or use the **File** and **Save At Server...** menu. A window similar to the one in Figure 107 will appear. Then type the owner, name, and comments of the procedure, select if the procedure is to be shared with other users, and click on the **OK** button. To save the procedure in a file, use the **Save** button or use the **File** and **Save As...** menu.

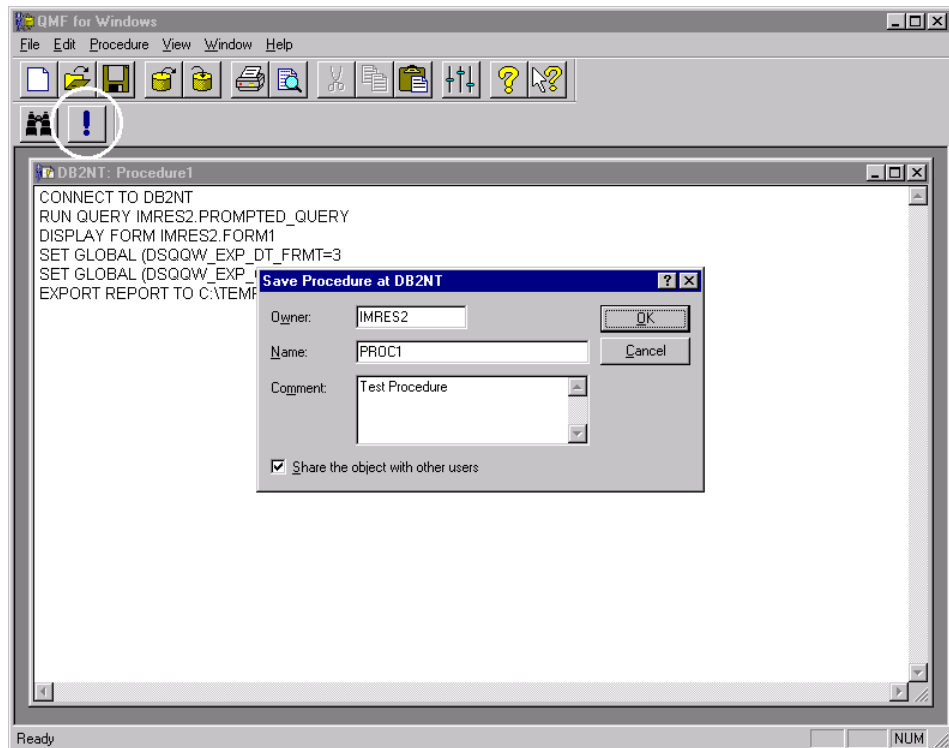


Figure 107. Saving new procedure

Procedures can vary tremendously from one to another. The one shown above is only a very simple example of what a procedure can do.

6.6 Using Data Snap-Ins for QMF for Windows

The QMF for Windows Snap-Ins are small packages that, when installed, link data from another software application to QMF for Windows. In other words, it is possible to use the software interface together with QMF for Windows. For example, it is possible to use the Microsoft Excel spreadsheet application, and from that software, execute some of the functions of QMF for Windows.

These Snap-Ins do not come with the product installation. You have to install them separately, and they are available from the Rocket Software Web page. The versions of the Snap-Ins, as of the writing of this book, are also available on the CD-ROM that comes with the book, but we strongly recommend checking the Rocket Software Web page to look for the latest available versions. The URL to do so is:

<http://www.rocketsoftware.com/QMF/>

Go to the “Companion Products” section to access the downloadable code. There is one Snap-In for each different product.

6.6.1 Lotus 123

After you have installed the Snap-In for Lotus 123, it is possible to run a QMF query, and the data that is retrieved will be placed inside the spreadsheet. To do this, follow the process below:

1. Open Lotus 123 and create a spreadsheet.
2. Go to the **Edit** and **Data Snap-In for QMF** menu.
3. A window will appear, prompting you to choose the database server from the list and requiring the user ID and password to be typed in. Fill in this information and click on the **OK** button. Note that the database servers in the list are the same database servers defined in the QMF server definition file.
4. After the login is done, a window as shown in Figure 108 will appear. In that window, you can select the owner and the name of the query directly if you know it, or you can use the wildcard '%' to view a list of available queries. After you do that, click on the **OK** button.

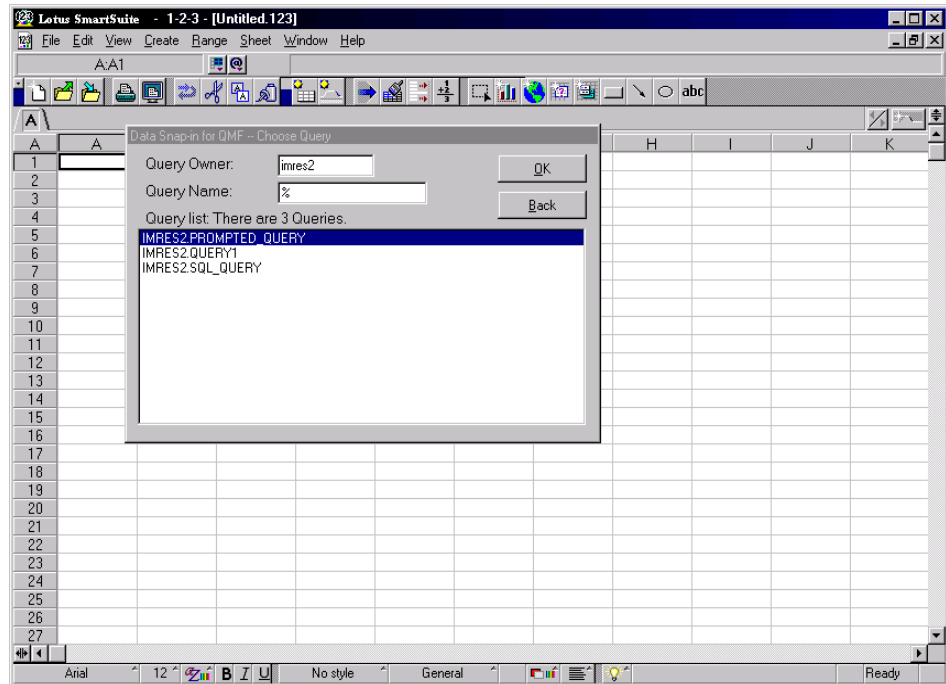


Figure 108. Lotus 123 Snap-In

5. A window as shown in Figure 109 will appear. In that window you can select several ways of importing the data. The most common one is the **Import Data** option that places the result of the data in the spreadsheet, but there are other options, like the **Group** option and the **Cross Tab Report**. Whenever you change this option, the window will also change, prompting for specific information for the selected way of importing the data.
6. After the configuration of the options on how the data is going to be imported, it is necessary to select the starting cell on the spreadsheet in which the data will be imported. Click on the cell you want and then click on the field **Place Result At**. The cell identification will appear automatically.

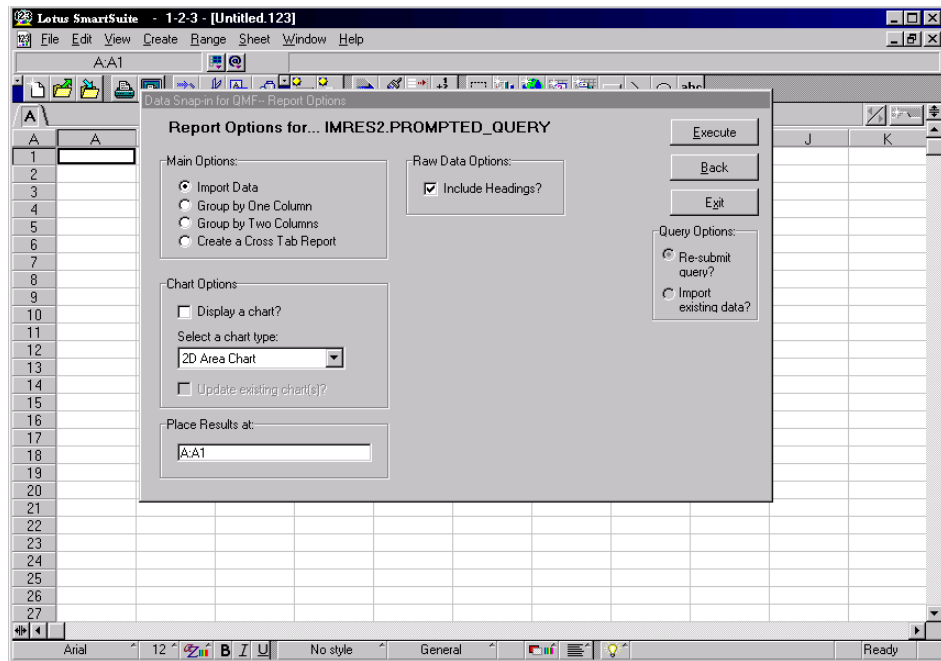


Figure 109. Formatting data using Lotus 123 Snap-In

7. The last step is to click on the **Execute** button, and the data will be imported into your spreadsheet as shown in Figure 110.

A	B	C	D	E	F	G	H
FIRSTNAME	LASTNAME	DEPTNAME	SALARY	BONUS	COMM	TOTAL_INCOME	
2	JAMES	JEFFERSON	ADMINISTRATION SYSTEMS	22180	400	1774	24354
3	SYBIL	JOHNSON	ADMINISTRATION SYSTEMS	17250	300	1380	18930
4	DANIEL	SMITH	ADMINISTRATION SYSTEMS	19180	400	1534	21114
5	SALVATORE	MARINO	ADMINISTRATION SYSTEMS	28760	600	2301	31661
6	SALLY	KWAN	INFORMATION CENTER	38250	800	3060	42110
7	HEATHER	NICHOLLS	INFORMATION CENTER	28420	600	2274	31294
8	DOLORES	QUINTANA	INFORMATION CENTER	23800	500	1904	26204
9	IRVING	STERN	MANUFACTURING SYSTEMS	32250	500	2580	35330
10	JENNIFER	LUTZ	MANUFACTURING SYSTEMS	29840	600	2387	32827
11	WILLIAM	JONES	MANUFACTURING SYSTEMS	18270	400	1462	20132
12	DAVID	BROWN	MANUFACTURING SYSTEMS	27740	600	2217	30557
13	JAMES	WALKER	MANUFACTURING SYSTEMS	20450	400	1636	22486
14	MARILYN	SCOUTTEN	MANUFACTURING SYSTEMS	21340	500	1707	23547
15	MASATOSHI	YOSHIMURA	MANUFACTURING SYSTEMS	24680	500	1974	27154
16	ELIZABETH	PIANKA	MANUFACTURING SYSTEMS	22250	400	1780	24430
17	BRUCE	ADAMSON	MANUFACTURING SYSTEMS	25280	500	2022	27802
18	EILEEN	HENDERSON	OPERATIONS	29750	600	2380	32730
19	MAUDE	SETRIGHT	OPERATIONS	15900	300	1272	17472
20	PHILIP	SMITH	OPERATIONS	17750	400	1420	19570
21	ETHEL	SCHNEIDER	OPERATIONS	26250	500	2100	28850
22	MICHAEL	THOMPSON	PLANNING	41250	800	3300	45350
23	RAMLAL	MEHTA	SOFTWARE SUPPORT	19950	400	1596	21946
24	JASON	GOUNOT	SOFTWARE SUPPORT	23840	500	1907	26247
25	WING	LEE	SOFTWARE SUPPORT	25370	500	2030	27900
26	CHRISTINE	HAAS	SPIFFY COMPUTER SERVICE DIV.	52750	1000	4220	57970
27	SEAN	O'CONNELL	SPIFFY COMPUTER SERVICE DIV.	29250	600	2340	32190

Figure 110. Result of Lotus 123 Snap-In

At this point, you can use this data, as it is a regular spreadsheet. Keep in mind that any modification in the data in the spreadsheet will not affect the data in the database server and vice versa.

6.6.2 Microsoft Excel

After you have installed the Snap-In for Microsoft Excel, it is possible to run a QMF query, and the data that it retrieves will be placed inside the spreadsheet. To do this, follow the process below:

1. Open Microsoft Excel and create a new spreadsheet.
2. There will be a button on the toolbar called **Data Snap-In for QMF**. If this button is not visible, go to the **Tools** and **Add-Ins...** menus and select the **Data Snap-In for QMF**.
3. Click on that button and a window prompting to select the database server from the list will appear, followed by a window to type in userID and password. Fill in this information and click the **OK** button. Note that the database servers in the list are the same database servers defined in the QMF server definition file.

4. After the login is done, a window as shown in Figure 111 will appear. In that window, you can select the owner and the name of the query directly if you know it, or you can use the wildcard '%' to view a list of available queries. Then click the **OK** button.

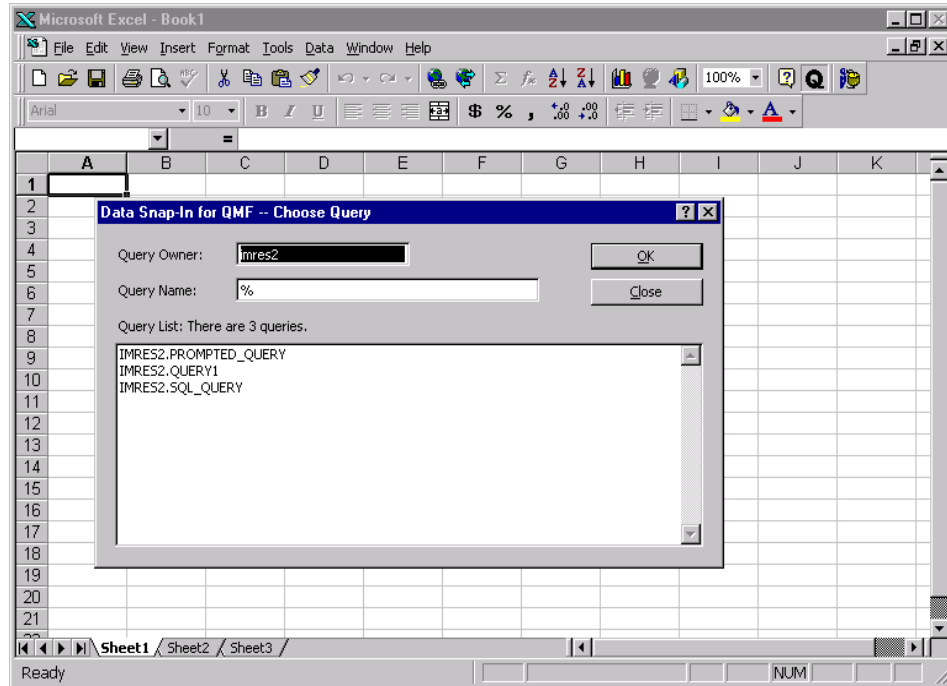


Figure 111. Excel Snap-In

5. A window as shown in Figure 112 will appear, where you can select several ways of importing the data. The most common one is the **Import Data** option where it simply place the result of the data in the spreadsheet but there are other options like the **Group** option, and also **graphics**. Whenever you change this option, this window will also change, prompting for specific information for the selected way to importing the data.

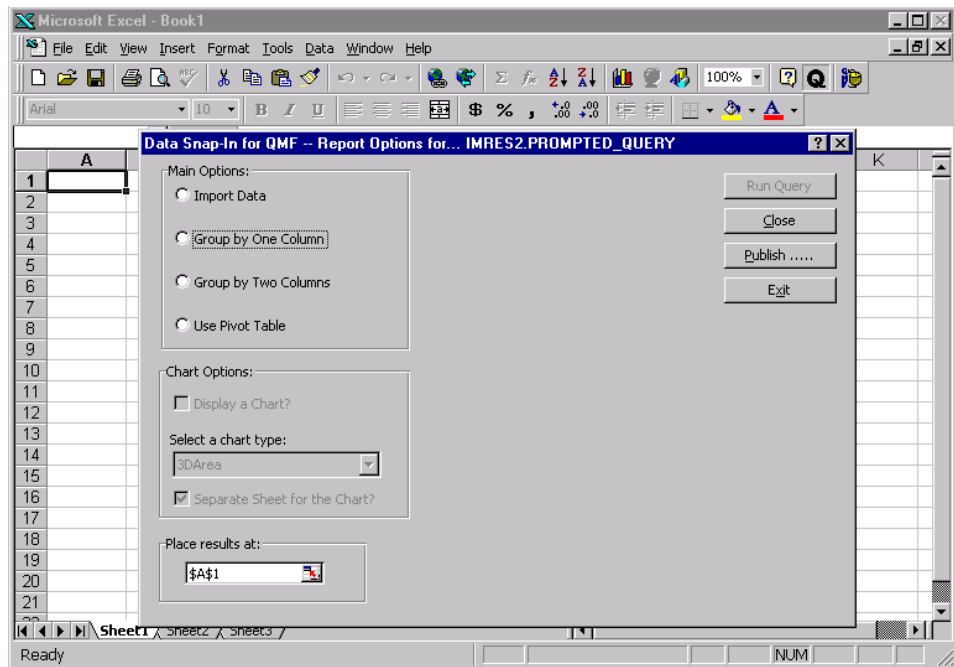


Figure 112. Formatting data using Excel Snap-In

6. Finally, click the **Run Query** button and the data will be imported to the spreadsheet, as shown in Figure 113. The windows will not be closed automatically, so you have to click on the **Close** button yourself.

	A	B	C	D	E	F	G
1	FIRSTNAME	LASTNAME	DEPTNAME	SALARY	BONUS	COMM	TOTAL_INCOME
2	JAMES	JEFFERSON	ADMINISTRATION SYSTEMS	22180	400	1774	24354
3	SYBIL	JOHNSON	ADMINISTRATION SYSTEMS	17250	300	1380	18930
4	DANIEL	SMITH	ADMINISTRATION SYSTEMS	19180	400	1534	21114
5	SALVATORE	MARINO	ADMINISTRATION SYSTEMS	28760	600	2301	31661
6	SALLY	KWAN	INFORMATION CENTER	38250	800	3060	42110
7	HEATHER	NICHOLLS	INFORMATION CENTER	28420	600	2274	31294
8	DOLORES	QUINTANA	INFORMATION CENTER	23800	500	1904	26204
9	IRVING	STERN	MANUFACTURING SYSTEMS	32250	500	2580	35330
10	JENNIFER	LUTZ	MANUFACTURING SYSTEMS	29840	600	2387	32827
11	WILLIAM	JONES	MANUFACTURING SYSTEMS	18270	400	1462	20132
12	DAVID	BROWN	MANUFACTURING SYSTEMS	27740	600	2217	30557
13	JAMES	WALKER	MANUFACTURING SYSTEMS	20450	400	1636	22486
14	MARILYN	SCOUTTEN	MANUFACTURING SYSTEMS	21340	500	1707	23547
15	MASATOSHI	YOSHIMURA	MANUFACTURING SYSTEMS	24680	500	1974	27154
16	ELIZABETH	PIANKA	MANUFACTURING SYSTEMS	22250	400	1780	24430
17	BRUCE	ADAMSON	MANUFACTURING SYSTEMS	25280	500	2022	27802
18	EILEEN	HENDERSON	OPERATIONS	29750	600	2380	32730
19	MAUDE	SETRIGHT	OPERATIONS	15900	300	1272	17472
20	PHILIP	SMITH	OPERATIONS	17750	400	1420	19570
21	ETHEL	SCHNEIDER	OPERATIONS	26250	500	2100	28850

Figure 113. Result of Excel Snap-In

At this point, you can use this data, as it is a regular spreadsheet. Keep in mind that any modification in the data in the spreadsheet will not affect the data in the database server, and vice versa.

6.6.3 Microsoft Access

After you have installed the Snap-In for Microsoft Access, it is possible to run a QMF query, and the data that is retrieved will be placed inside the tables in a Microsoft Access database. To do this, follow these steps:

1. Open Microsoft Access and create a new database, or open an existing one.
2. There will be a new button in the toolbar called **Data Snap-In for QMF**. If this button is not visible, go to the **Tools, Add-Ins and Add-In Manager** menu and select the **Data Snap-In for QMF**.
3. Click on that button. A window will appear prompting you to choose a server from the list, followed by a window to type in the userID and password. Fill in the information and click on the **QMF** button. Note that the servers in the list are the same servers defined in the QMF server definition file.

4. After the login, a window as shown in Figure 114 will appear. In that window, you can select the owner and the name of the query directly if you know it, or you can use the wildcard '%' to view a list of available queries and then click the **OK** button. In this window, there is also an opportunity to save the query itself to MS Access. This means that a new query will be created inside MS Access with the same SQL statement as the one in QMF. That can be done by clicking on the **Save** button instead.

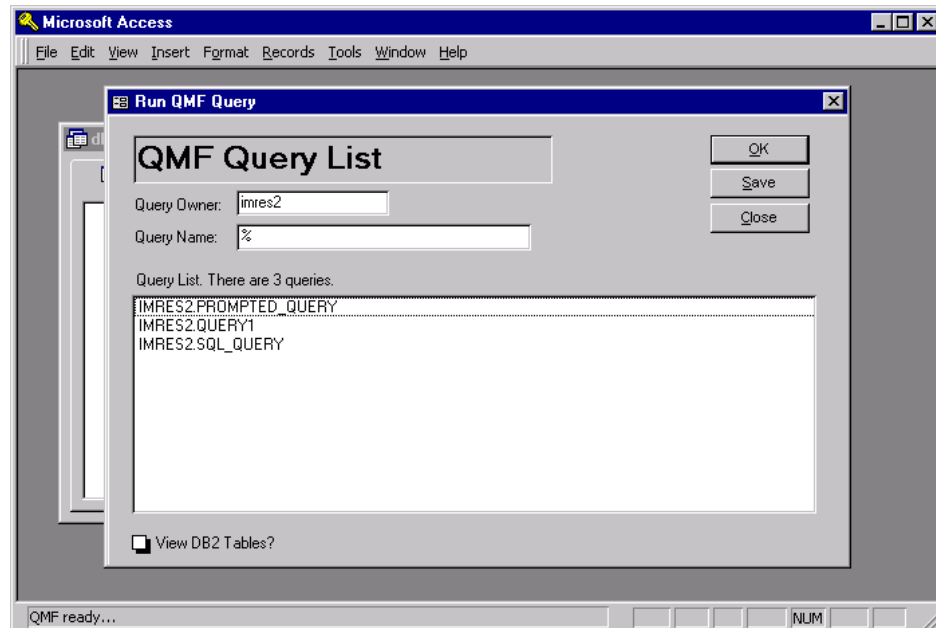


Figure 114. Access Snap-In

5. A window as shown in Figure 115 will appear. In that window, you can either select an existing table where you want to place the data, or, if you do not specify the table, a new table will be created. If selecting an existing table and the result of the QMF query and the selected table do not match, an error will occur.

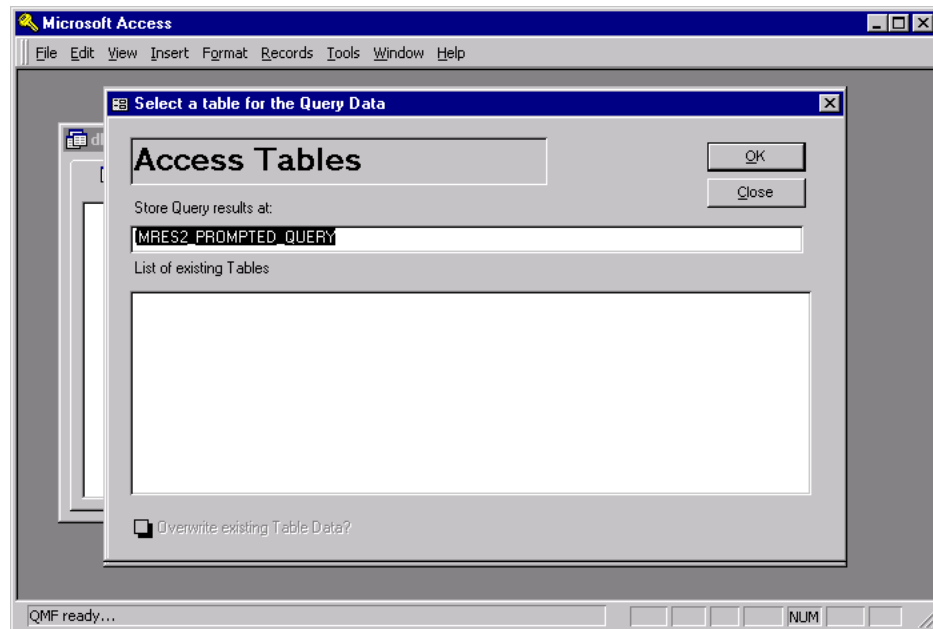


Figure 115. Selecting a table to place the data

6. The next step is to select a Microsoft Access Report List as shown in Figure 116.

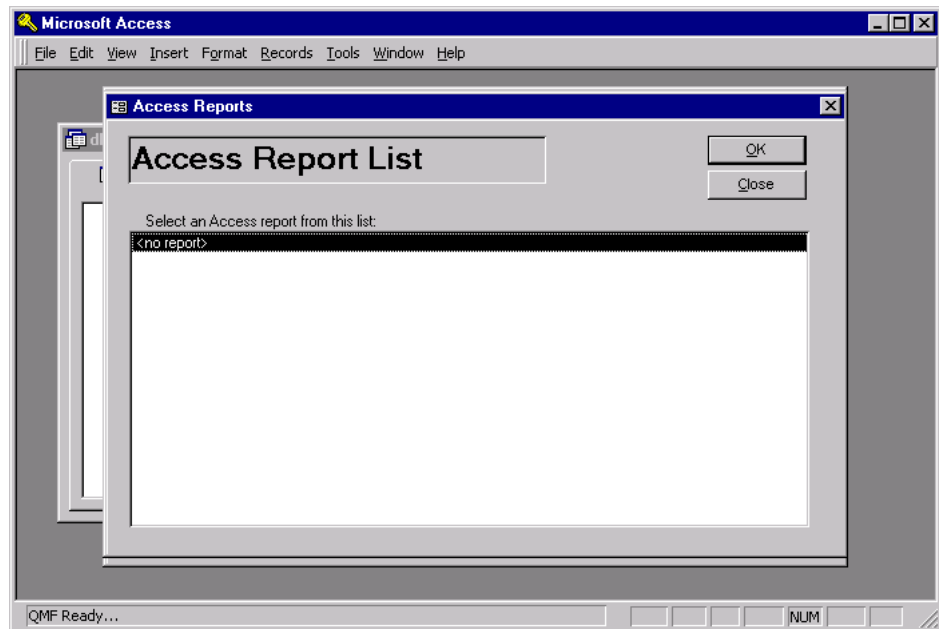


Figure 116. Select and report list

7. The last step necessary is to click the **OK** button, and the data will be imported. In the case of the example shown, a new table is created and the data is imported into it as shown in Figure 117.

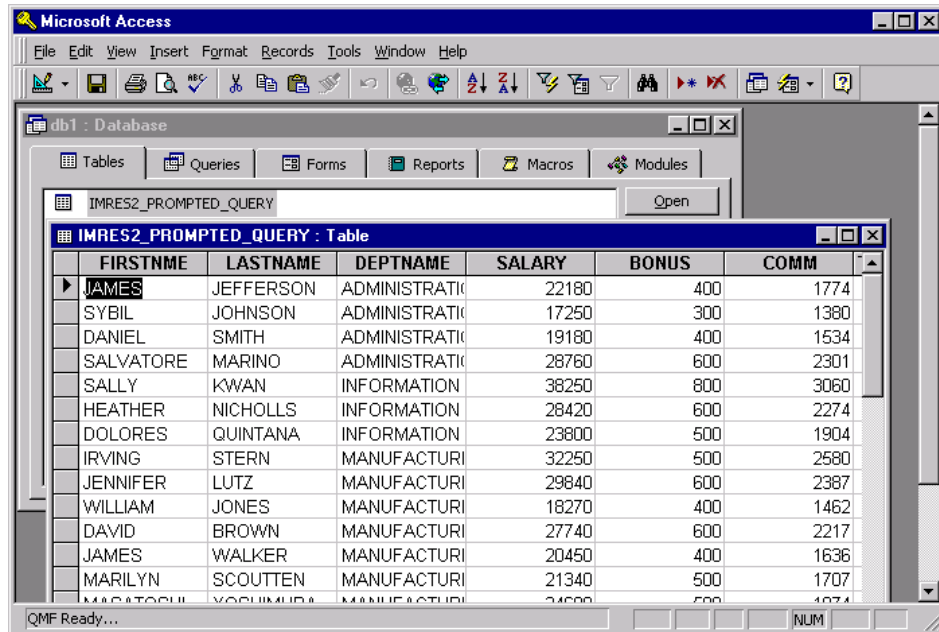


Figure 117. Result of access Snap-In

At this point, you can use this data, as it is a regular Access table. Keep in mind that any modification in the data in that table will not affect the data in the database server and vice versa.

6.7 Converting dynamic SQL to static SQL

Before you decide to use static SQL, you should consider whether using static SQL or dynamic SQL is the best technique for your application. The DBA group in your enterprise is usually responsible for this decision.

If the DBA authorizes you to convert dynamic SQL to static SQL, follow the procedure described in 4.4.1, “Convert dynamic SQL to static SQL” on page 115.

For most DB2 users, static SQL — embedded in a host language program and bound before the program runs — provides a straightforward, efficient path to DB2 data. You can use static SQL when you know the exact SQL statement before your application needs to execute it.

Dynamic SQL prepares and executes the SQL statements within a program while the program is running. It is used when the user creates the SQL statements dynamically, for example, using the QMF for Windows interface. DB2 prepares and executes those statements as dynamic SQL statements.

6.7.1 Dynamic versus static SQL

To access DB2 data, an SQL statement requires an access path. Two big factors in the performance of an SQL statement are the amount of time that DB2 uses to determine the access path at run time, and whether the access path is efficient. DB2 determines the access path for a statement at either of these times:

- When you bind the plan or package that contains the SQL statement
- When the SQL statement is executed.

The time at which DB2 determines the access path depends on these factors:

- Whether the statement is executed *statically* or *dynamically*
- Whether the statement contains input host variables

For dynamic SQL statements, DB2 determines the access path at run time when the statement is prepared. This can make the performance worse than that of static SQL statements. However, if you execute the same SQL statement very often, you can use the dynamic statement cache to decrease the number of times that those dynamic statements must be prepared. Ask your DBA about the dynamic cache option in DB2.

6.8 Checking your resource limits

One major function of QMF for Windows is governing. This means that QMF for Windows can limit certain groups of users to access just a subset of QMF functions, or it can limit the access by time to prevent incorrect use of the data or for balancing the resource consumption at the database server.

It is important to keep in mind that QMF for Windows limits will not overwrite the database server limits. In other words, you may have permission to update a table on the QMF for Windows level, but you may not have the permission to do this on the database level. In this case, you will not be allowed to update that table. You must have permission both in QMF and the database in order to execute a command. Therefore, you may see specific permission available to you in the QMF for Windows resource limits, but still may not be able to execute the command, due to the database limits.

These limits are defined for each database server. Thus, you may have the right to execute one function on one server, but you do not have it on another server. These restrictions are usually defined by a the DBA, and there are some environments where there are different DBAs for different databases. Also, the databases may contain different types of data, and the restrictions applied may also differ.

Although you cannot change the resource limits, you can check them at any time. To do this, first you have to be connected to the database server by accessing one object in that database, such as the list of objects or a query. Then go to the **View** and **Resource Limits...** menu.

A window as shown in Figure 118 will appear, asking for you to choose if you want the most current resource limits. The reason for this is that the moment QMF connects to a database server, it retrieves the resource limits and stores them on your local computer. While you are working, the DBA may have changed these limits, and you will not see them until the next connection. Therefore, when you try to see your resource limits, QMF will ask you if you want to see the most current version of these limits stored on the server, or if you want to see the ones stored on your local computer.

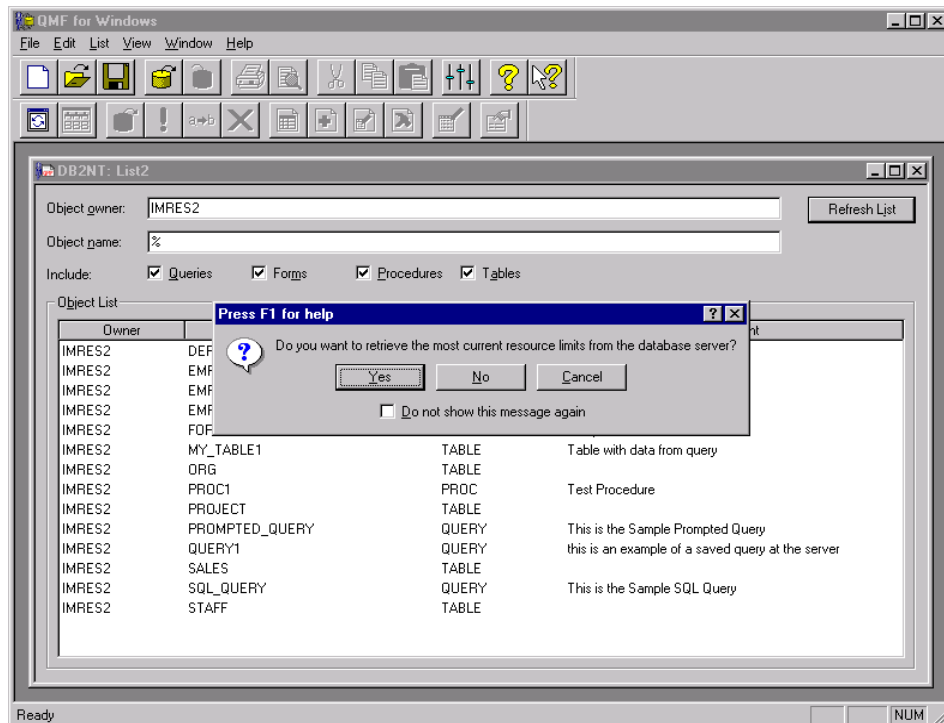


Figure 118. Retrieve the most current resource limits

After that, a window as shown in Figure 119 will be displayed showing the resource limits. In this window you can see (but not change) the limits that are applied to you. As you can see, these limits set the timeout of the connection, the maximum number of connections, the maximum number of rows to fetch, the SQL verbs you can use in your SQL statements, and many other limits. To check what each of these resource limits mean, refer to 4.2.2, “Creating schedules” on page 94. In case you need to change these limits for any reason, talk to the DBA responsible for setting your resource limits.

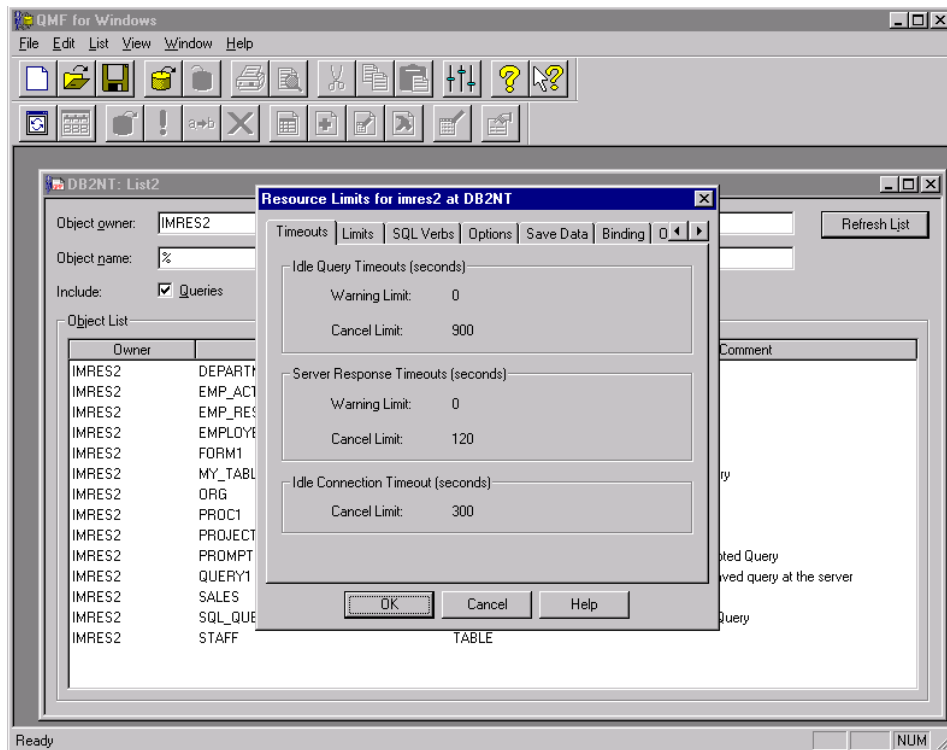


Figure 119. Resource limits

6.9 Security

QMF for Windows Administrator is the administrative component of QMF for Windows, and using this module should be strictly an administrator task. There should not be any need for an end user to run QMF for Windows Administrator. QMF for Windows Administrator is used to maintain resource limits groups. To use QMF for Windows Administrator to maintain resource limits groups, you must have the authorization to execute the QMF for Windows Administrator package. This prevents unauthorized users from changing the limits that are established by the administrator.

6.9.1 Change password capability

Users can change their host and workstation passwords from within QMF for Windows as shown in Figure 119.

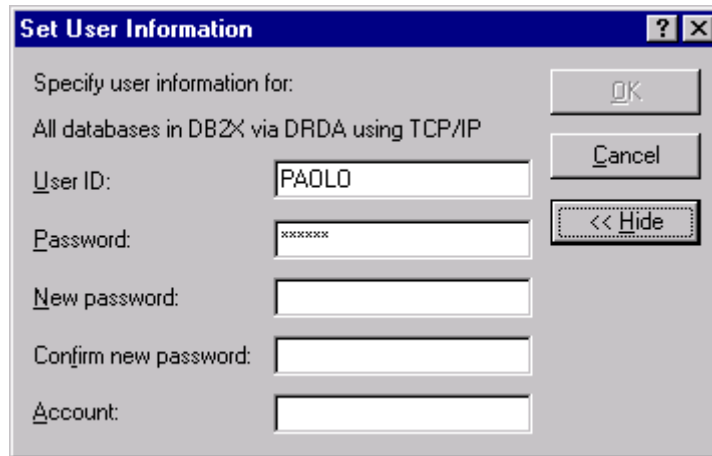


Figure 120. Change password

6.9.2 Lists

The database administrator might want to restrict the visibility of these objects to a certain number of users. To do this, QMF for Windows allows the creation of predefined lists that the users will see by default when working with the product. Lists are also useful to simplify the day's work of the users by providing them with a tailored set of QMF for Windows objects by default.

If a user starts working with predefined queries, and so on, the usual way would be to use the **File -> Open from Server** menu. This would open a window that then needs to be modified if the user does not need to see all objects from all users. Creating a predefined list provides an alternative to simplify the work by using the **File -> Open** menu to see a list of tailored objects.

The following steps need to be performed to create and save a predefined list.

1. On the **File** menu of QMF for Windows, click **New -> List** to open the windows shown in Figure 121. Make sure to select the correct server to create the list from. If the window does not show the required server (like the 'DB2AIX' in the sample shown), go to the List -> Set Server menu to set the active server accordingly.
2. You may now specify the owner of the objects that need to be in the list, the object directly by name, and the type of objects to be included in the list.

3. Click the **Refresh List** button to create the list. This list can then be modified by removing certain objects individually from this list.
4. Save the newly created list using the **File -> Save As** menu. This will create a file in the default installation directory that can then be opened using the **File -> Open** menu.

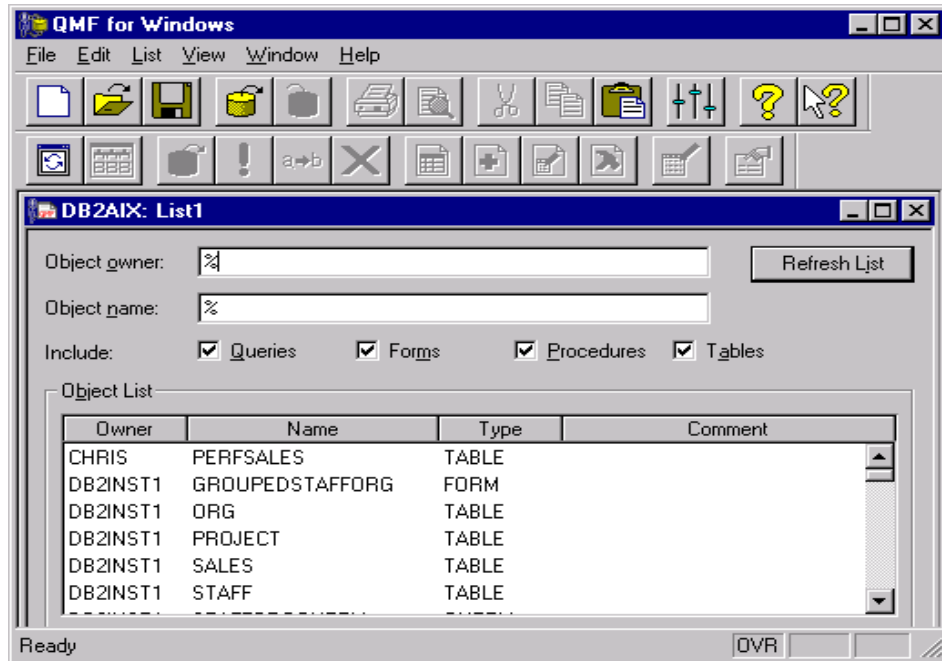


Figure 121. Lists

Within the List window, several options are available using the icons in the Toolbar or the right mouse button. These options are:

- **Display object** to view the selected object. This function is available for Queries, Forms, Procedures, and Tables.
- **Run object** to execute the selected object. This is only available for Queries and Procedures.
- **Draw object** to create a query based on a selected table. The type of query may be either a SELECT query, an SQL UPDATE query, an SQL INSERT query, or a prompted query. This options only works for tables.
- **Edit object** is available only for tables and will open the Table Editor for this table.

- **Properties** is available for all four types of objects, and displays the properties of the selected object, including comments, attributes, and historical usage information.

6.10 Customizing the interface

QMF for Windows allow you to customize the interface to adapt it to the way it suits you best. That is possible by selecting the toolbar in the window you want to modify.

QMF for Windows automatically detects what type of window is currently active and switches the toolbar for you. So, if for example, the Query window is the active window, the Query toolbar will be visible, while if a Procedure is the active window, the Procedure toolbar will be visible. However, if you do not want a specific toolbar to appear, click on the toolbar with the right mouse button and deselect the toolbar you do not want to appear.

It is also possible to customize a toolbar by modifying the buttons that appear in it. That can be done by clicking on the toolbar you wish to change with the right button of the mouse and select the **Customize...** option. A window as show in Figure 122 will appear. On the right side of that window you can see the current buttons that are displayed in the toolbar, while on the left side the available buttons are shown. To add a new button to the toolbar, select the button you want to include on the left list and click on the **Add** button. To remove a button, select the button you want to remove in the list at the right and click the **Remove** button.

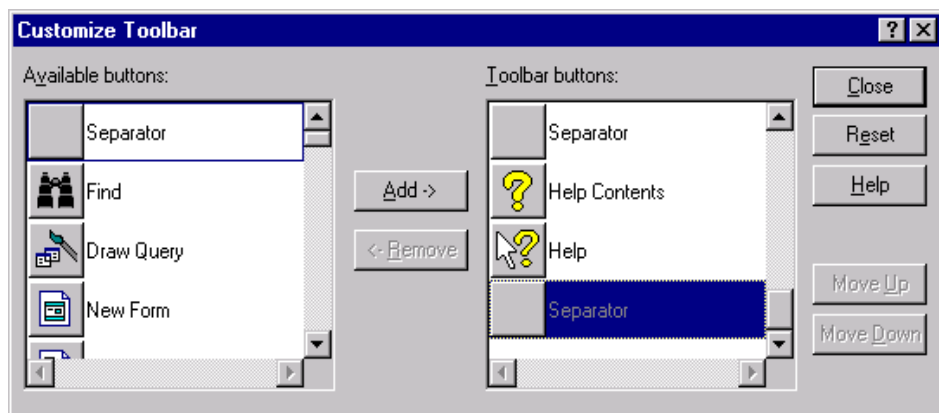


Figure 122. Customize toolbar

The QMF toolbars are dockable, which means it is possible to move the toolbars around and place them at the different sides of the window, as shown in Figure 123. It is also possible, instead of docking the toolbar on one side of the window, to leave it floating as a separate window. To do this, you have to drag the toolbar with the right hand mouse button and drop it at the desired place.

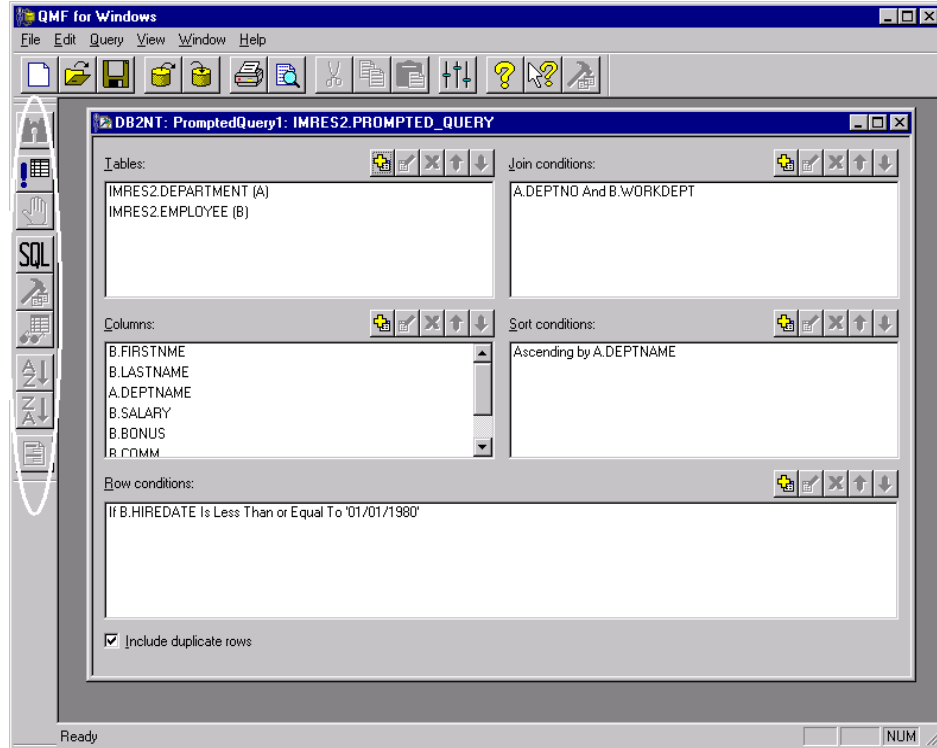


Figure 123. Move the toolbar

6.11 Migrating from OS/2 Query Manager

Longtime users of Query Manager/2 have relied upon its native connectivity to DB2 and its robust performance in demanding environments. Many have used it to accumulate valuable assets in the form of libraries of Query Manager/2 reports. But at the same time, in addition to not being Year 2000 compliant, OS/2 Query Manager does not allow the use of all the new functions introduced into DB2 UDB, and therefore leaves users unable to extend the use of their reports to more applications across the enterprise.

Rocket Software provides a service offering to migrate Query Manager/2 objects to QMF for Windows. Once the conversions are finished, Query Manager/2 objects become native to the QMF for Windows environment, allowing you take full advantage of all the features and benefits of QMF for Windows, including:

- Across-the-board support for all DB2 database platforms, from DB2 Personal Edition to the multi-node OS/390 sysplex — without database gateways, middleware, or ODBC drivers.
- Native DRDA and DB2 support, including all DB2-specific SQL, static SQL, and stored procedures.
- Rapid integration with Windows "suite" applications, such as spreadsheets and analysis tools.
- A robust API that power users and Windows developers can use to quickly build completely customized applications.
- Converts Query Manager/2 objects into corresponding QMF for Windows objects, including:
 - Queries
 - Prompted queries
 - Forms
 - Procedures
- Rapid processing
- Converted objects are saved to DB2 database of choice.

Converted objects may be used in evaluation or fully licensed versions of QMF for Windows.

The Rocket Migration Utility for Query Manager/2 will not convert Query Manager/2 menus or panels. Once the Query Manager/2 objects are converted to QMF for Windows objects, they are stored on any DB2 database and accessed from the database by QMF for Windows. Alternatively, individual users may elect to store copies of their own QMF for Windows objects on their PCs.

The QMF for Windows procedure environment is not identical to the Query Manager/2 environment. Therefore, Query Manager/2 customers are likely to find that some of their Query Manager/2 procedures — especially those containing REXX code — do not work after conversion in the QMF for Windows environment. Some objects may require editing by hand, and others will need to be completely rewritten or even discarded, for example because of the fact that QMF for Windows does not support REXX procedures.

Chapter 7. Web considerations

The Web is changing every aspect of our lives, but no area is undergoing as rapid and significant a change as the way businesses operate. As businesses incorporate Internet technology into their core business processes, they start to dynamically alter the way to do business. Today, companies large and small are using the Web to communicate with their partners, to connect with their back-end data-systems, and to transact commerce.

This new Web + IT paradigm merges the standards, simplicity and connectivity of the Internet with the core processes that are the foundation of business. The new killer applications are interactive and transaction intensive; they let people do business in more meaningful ways from anywhere at anytime. This forces many customers to change operations to support a 24-hour day, 7-day work week.

An e-business company is a company that has changed their way to do business to support it to be done using the Web infrastructure. To manage transitions smoothly, you need to remember two important ideas:

- Start simple, but plan to grow fast.
- Build on what you have.

In short, e-business isn't about re-inventing your business. It's about altering your current business processes to improve operating efficiencies which in turn will strengthen the value you provide to your customers — value that cannot be generated by any other means, and value that will give you a serious advantage over your competition.

So how does your company become an e-business? How can becoming an e-business help you maximize the value of your information technology investment? How can it help you reduce your costs and grow your revenue? There are four important areas or stages in this process. We think of these four stages collectively as the e-business cycle. They are:

- Transforming core business processes
- Building flexible, expandable e-business applications
- Running a scalable, available, safe environment
- Leveraging knowledge and information you've gained through e-business systems

There isn't a set order or hierarchy to this cycle. Successful e-businesses start at different points, and you can too. But first you must identify which of your core business processes are most suitable for, or most in need of, conversion to e-business.

One way to get started would be to use as many existing technology and products available within a company, and start with using the Web as a transport or access layer for the result presentation.

7.1 Web presence basics

The following section gives a short introduction on how QMF for Windows allows you to expand the accessibility of queries and reports generated to be accessible through the Internet or intranet.

7.1.1 How does it work?

While QMF for Windows already allows broad access to queries and reports by making QMF objects already residing in a DB2 host system available for end users with Windows based systems, this accessibility is expanded even more by making the results visible using Web Browser software. Using a Web browser allows the data to be visible from any computer running any operating system, as long as a Web browser for this environment is available.

The term *intranet* will be used for talking about Web technology that has been implemented to be used from within a company's network, whereas the term *Internet* will indicate accessibility to this information from outside the company. QMF for Windows provides an easy way to get started delivering data through the Web.

A typical environment might look like the one shown in Figure 124.

Be aware that any kind of web implementation has to be planned carefully as the resource requirements are different to a standard environment. Therefore, the final web based application might end with a lower limit of concurrent users allowed in order to achieve acceptable performance.

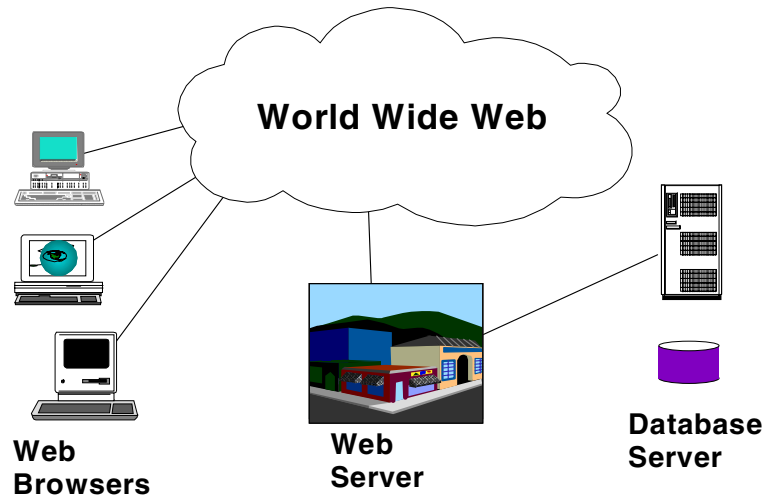


Figure 124. Web environment

7.2 Static reports

The easy way to use Web Technology to make query results and information more broadly accessible is to simply publish query results to a Web server. Using scheduling mechanisms, these reports can be updated on a frequent basis.

But, compared to dynamic reports (discussed in 7.3, “Dynamic reports” on page 262), static reports will not provide “up-to-the-minute” information.

Generating static reports is best done using a procedure like the following:

1. Execute the query.
2. Generate the report using a QMF form.
3. Export the report to an HTML file with a predefined name in the Web server’s HTTP directory.

Using this approach, the Web server does not need to have QMF for Windows installed at this server. QMF may be installed anywhere within the network, because only the resulting HTML document needs to be transferred to the Web server.

The following sections cover some specifics for the use of QMF for Windows HTML based reports and the way to schedule this entire process.

7.2.1 Convert a standard QMF Form to HTML

Let's consider the following example scenario:

1. The user will go to an HTML document that lists all available departments within a company.
2. From that screen, he can select a specific department to get detailed information on this department.

The first action to take is to create the query that lists the departments and shows them as an HTML document, including links to the specific reports. Figure 125 shows a query example.

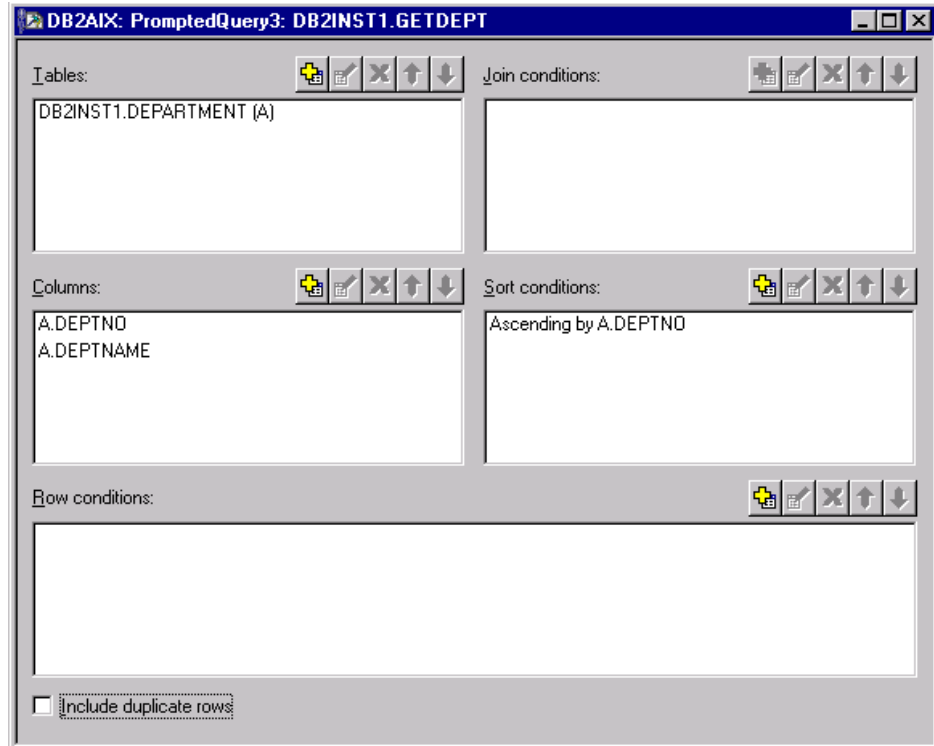


Figure 125. Convert form to HTML

The next step is to create the form to display the result in HTML format. This can be done by using the **Convert to HTML** button. This example has been created without specifying any parameters in the screen that shows up after clicking this button. Clicking on the **HTML** button allows you to further specify the HTML form by selecting **Edit Form...**. First select the Main.. selection to specify the report header, as shown in Figure 126.

Form Main

Total Width of Report Columns: 39

Num	Heading	Usage	Indent	Width	Edit	Seq
1	DEPTNO		2	6	C	1
2	DEPTNAME		2	29	C	2

Page: Heading text:

Footing text:

Final: Text:

Break1: New page for break

Footing text:

Break2: New page for break

Footing text:

Options: Outlining for break columns

Default break text (*)

Figure 126. Form main

At this point, it still possible to use all the functions for a normal form, such as saving it at a server, checking for errors, and so on. See Figure 127.

Form Main

Total Width of Report Columns: 67

Num	Heading	Usage	Indent	Width	Edit	Seq
1	DEPT	BREAK1	2	6	L	1
2	DEPTNAME		2	14	C	2
3	NAME		2	9	C	3
4	ID		2	6	L	4

Page: Heading text: <H1>Personal Cost</H1>

Footing text:

Final: Text:
<HR>

Break1: New page for break

Footing text: </TABLE>

Break2: New page for break

Footing text:

Options: Outlining for break columns
 Default break text (*)

Edit Form... Check Form OK Cancel

Figure 127. Form main window

After defining the main header, go to the **HTML -> Edit Form.. -> Details** menu to specify the appearance of the body of the document. The example shown will list the query result in table format and define an HTML anchor for each department. Clicking this anchor will link the user to the detailed report for each department.

In the example shown in Figure 128, the **Detailed Heading Text** section specifies a horizontal line as a separator between the heading and the body, and also specifies the body to be formatted as a table. In the Detailed Block Text part of the screen, the definitions are made to create the links and descriptions for the final document.

Form Details

1 of 1

 Enable: YES

Heading

Include column headings with detail headings

Detail Heading Text

Line	Alignment	Text
1	LEFT	 <hr><table>
2	LEFT	
3	LEFT	

Block

New page for detail block
 Blank lines after block: 0

Repeat detail heading
 Put tabular data at line: NONE

Keep block on page

Detail Block Text

Line	Alignment	Text
1	LEFT	<tr><td>&1</td><td>&2</td></tr>
2	LEFT	
3	LEFT	

Figure 128. Form detail

Click the **OK** button to apply the modifications.

Other queries have been defined to select the detailed information for each department, and the HTML form has been created, as explained in 6.4.3.3, “Convert forms to HTML” on page 194.

7.2.2 Report preview feature

A new function available for this form is to preview the result in the default Web browser configured for the local system. To do this, you have to click on the **Web Browser** button or use the **Form** and **View in Web Browser** menus. Your default Web browser will be launched automatically, and the form will be displayed in it, as shown in Figure 129.

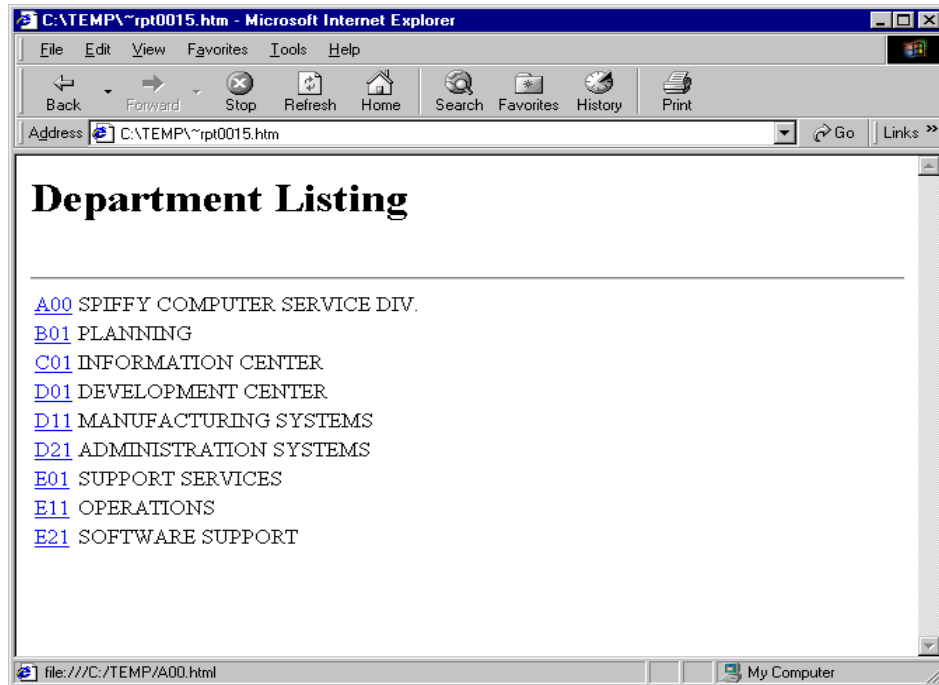


Figure 129. HTML form preview

7.2.3 Scheduling

The last step necessary in the small example is to automate the process of executing the queries and saving the HTML reports to the Web server's HTML directory. Figure 130 shows the procedure to do this.

```
DB2AIX: Procedure1: DB2INST1.RUNHTML
run query db2inst1.getdept (f=db2inst1.webdepartment
export report to d:\www\html\listdept.html
run query db2inst1.detaila00 (f=htmldeptdetail
export report to d:\www\html\la00.html
run query db2inst1.detailb01 (f=htmldeptdetail
export report to d:\www\html\lb01.html
run query db2inst1.detailc01 (f=htmldeptdetail
export report to d:\www\html\lc01.html
run query db2inst1.detaild11 (f=htmldeptdetail
export report to d:\www\html\ld11.html
run query db2inst1.detaild21 (f=htmldeptdetail
export report to d:\www\html\ld21.html
run query db2inst1.detaile11 (f=htmldeptdetail
export report to d:\www\html\le11.html
run query db2inst1.detaile21 (f=htmldeptdetail
export report to d:\www\html\le21.html
```

Figure 130. Publishing procedure

In the example shown, the Web server default HTML directory is located on the D drive under \WWW\HTML, but if the Web server is not the same system that QMF for Windows runs on, this path as well points to a shared LAN drive of the Web server system.

Using the Windows NT scheduler, this procedure can be scheduled to be executed once a day to update the information within the report on a daily basis.

For additional possibilities on how to implement a Web presence without having to code a new application (such as the one described in 7.3, “Dynamic reports” on page 262), you can look at Rocket Software’s Web page for the “Web Warehouse”. This can be found at the URL:

http://www.rocketsoftware.com/QMF/html/qmf_web_warehouse.asp

The Web warehouse shown there builds a complete application without writing a single line of code in a programming language, using only the basic functionality of QMF for Windows.

7.3 Dynamic reports

This section on dynamic reports explains another method of running QMF for Windows queries and forms directly from the Web Browser's interface. Using this method, the query will be initiated at the time the user clicks on a certain link within a Web page, rather than accessing a previously prepared Web page.

The benefit here is due to the fact that the result displayed is the result of the query executed at this time, and therefore guarantees up-to-date results. The trade-off using this approach as a general (or single) implementation is that it uses more resources — on the database server as well as on the Web server — and therefore may be performance-sensitive. Using this access method is shown in Figure 131.

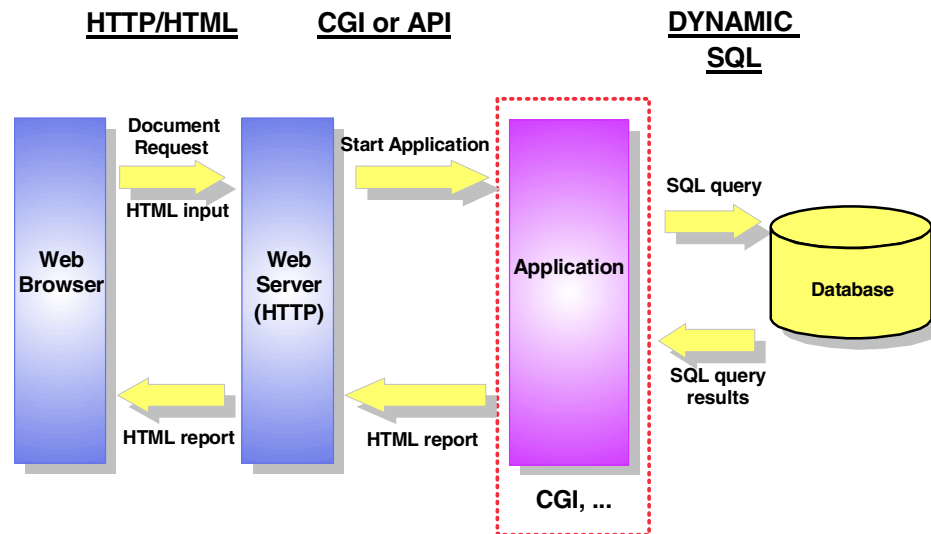


Figure 131. Dynamic reports

The sequence for this type of approach is as follows:

1. The end user clicks on a link in an HTML document shown on the Web browser. This document may also first prompt for some variables that will then be passed to the application.
2. The Web server receives the request for a document and starts the application that has been linked with this document.
3. The application retrieves the variables for the SQL query and issues the query to a database server.

4. The application receives the result from the query and has to present the result in an HTML format and send it back to the Web server.
5. The Web server sends the HTML document, containing the query result, back to the end user.

Today, there are many ways available to write applications that are “Web enabled”. Starting with the Common Gateway Interface (CGI), this list is expanded through the use of Web Server APIs that improve performance compared to CGI, Java applets, Active Server Pages (ASPs), and so on. The following section will describe an application example written using CGI, but all other application development approaches will work as well.

7.3.1 CGI

A generic method of generating dynamic reports is to write an application that uses CGI. The CGI applications are a way of using a programming or scripting language on a server, to respond to requests from a Web client by executing a file that returns HTML built “on-the-fly”. In other words, a CGI script is called from a Web client, the script is executed by the Web server, and the script returns HTML to the Web client as the output of its execution.

This HTML can be anything that the CGI application generates. To create dynamic reports, this CGI could generate a report for a given input query, meaning that the user would select a query on his Web browser, and the result would also be displayed on his Web browser.

One of the great advantages of creating CGI applications, as well as other Web technologies, is that there is no need to install any application, system, or middleware on the client. Only the Web browser and the network middleware are required. This can be very helpful in large corporations once it reduces the work of installing and administrating each computer.

To make your CGI application access QMF for Windows, it is necessary to use the QMF APIs. See Chapter 5, “Developer’s guide” on page 125 for more details on how to use the APIs, or see Appendix B, “QMF for Windows APIs” on page 293 for the complete API reference. In this chapter we provide a small CGI application example to illustrate how these applications work.

Using the Web browser, the user has to indicate the Internet address of the CGI application. In our case, the address was:

```
http://balboa.almaden.ibm.com/cgi-bin/cgisample.exe
```

where `cgisample.exe` is the CGI executable file. Once it is done, the screen shown in Figure 132 will appear.

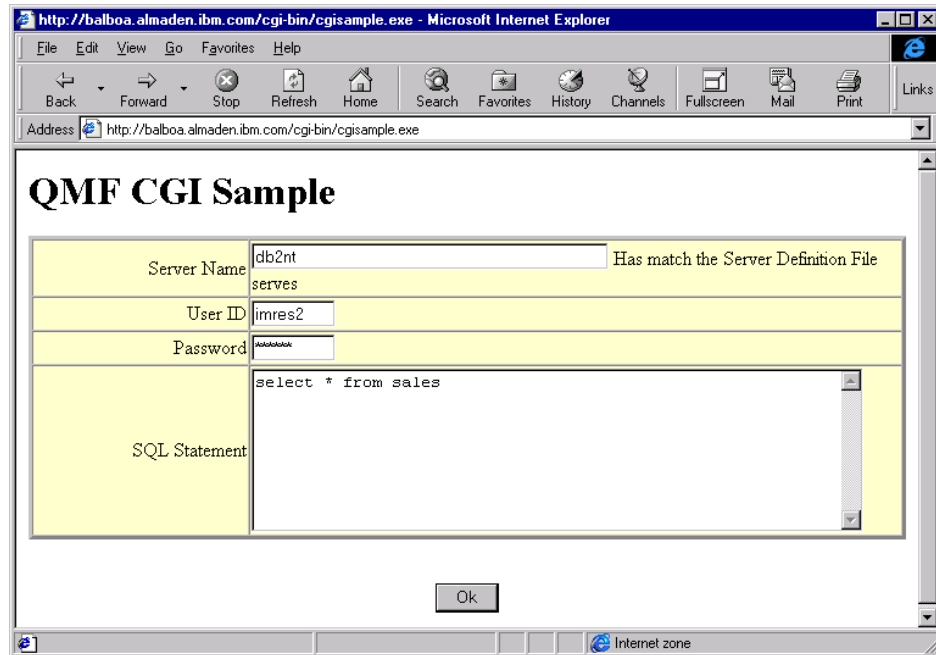


Figure 132. CGI example — first screen

The user then has to input some information. The first input required is the server name, which must match exactly with the name defined on QMF for Windows. These names are defined on the server definition file by the QMF administrator. The next inputs required are the user ID and password. Remember that in some systems such as AIX or OS/390, it may be case-sensitive. The last input is the SQL statement of the query. Only queries using the SELECT verb are allowed in this example. If the SQL statement is not properly written, an error will occur.

After all the input parameters have been correctly filled in, the user must click on the **OK** button. All the input parameters will be sent through the Web to the CGI application. The CGI application will then use the QMF APIs to connect to the server, execute the query, and return the result, as shown in Figure 133.

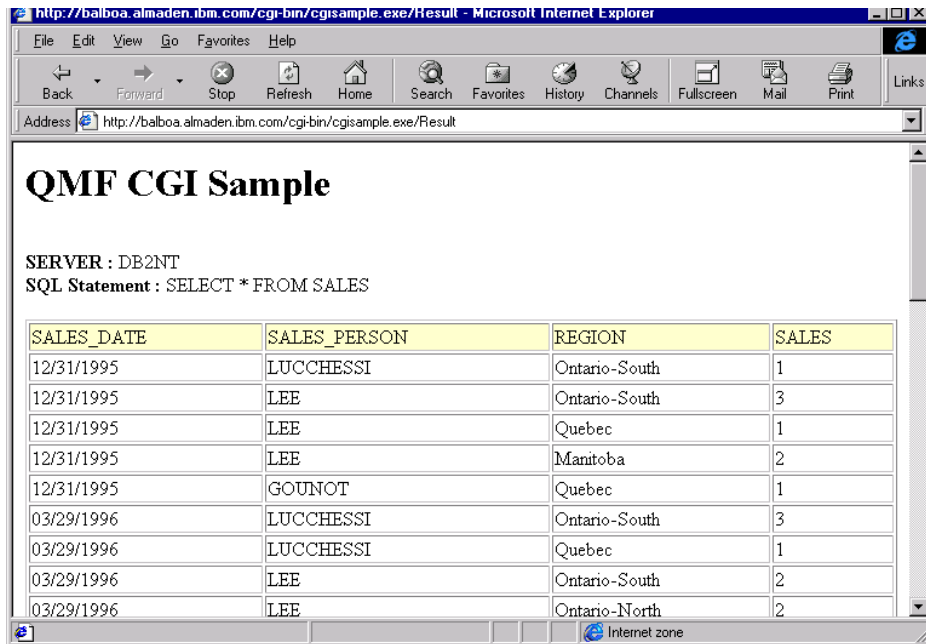


Figure 133. CGI example — second screen

The user can repeat the operation again by clicking on the **Back** button of the browser, inputting the parameters, and submitting again.

The source code below shows how this CGI application was implemented. Note that the `QMFWinLibrary_TLB` is included on the `uses` clause of the unit. Without it, the CGI application will not be able to access the QMF APIs.

```
uses
  Windows, Messages, SysUtils, Classes, HTTPApp,
  ExtCtrls, QMFWinLibrary_TLB, ComObj, ole2;
```

The class definition is listed below. Two procedures were used in this example:

```
type
  TWebMod = class(TWebModule)
    procedure WebModactConnectAction(Sender: TObject; Request: TWebRequest;
      Response: TWebResponse; var Handled: Boolean);
    procedure WebModactResultAction(Sender: TObject; Request: TWebRequest;
      Response: TWebResponse; var Handled: Boolean);
  private
  public
  end;
```

```
var
  WebMod: TWebMod;
```

The first procedure simply creates an HTML answer for the first request of the user, showing the screen as shown in Figure 132. The source code of this procedure is listed below:

```
procedure TWebMod.WebModactConnectAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  //simply respond to the user an HTML with input boxes for
  //the server name, user ID, password and SQL statement
  Response.Content := '<HTML>'
    + '<BODY>'
    + '<H1>'
    + '<H1>QMF CGI Sample</H1>'
    + '<FORM METHOD="POST" ACTION="http://'
    + Request.Host
    + Request.ScriptName + '/Result">'
    + '<CENTER><TABLE BORDER=3 CELLSPACING=0 COLS=1'
    + '          'WIDTH="36%" BGCOLOR="#FFFFCC" >'
    + '<TR>'
    + '  <TD ALIGN=RIGHT WIDTH="25%">'
    + '    Server Name'
    + '  </TD>'
    + '  <TD ALIGN=LEFT>'
    + '    <INPUT TYPE="TEXT" NAME="edServerName" SIZE="40">'
    + '    Has match the Server Definition File serves'
    + '  </TD>'
    + '</TR>'
    + '<TR>'
    + '  <TD ALIGN=RIGHT WIDTH="25%">'
    + '    User ID'
    + '  </TD>'
    + '  <TD ALIGN=LEFT>'
    + '    <INPUT TYPE="TEXT" NAME="edUserID" SIZE="8">'
    + '  </TD>'
    + '</TR>'
    + '<TR>'
    + '  <TD ALIGN=RIGHT>'
    + '    Password'
    + '  </TD>'
    + '  <TD ALIGN=LEFT>'
    + '    <INPUT TYPE="PASSWORD" NAME="edPassword" SIZE="8">'
    + '  </TD>'
    + '</TR>'
```

```

+ '<TR>'
+ '  <TD ALIGN=RIGHT>'
+ '    SQL Statement'
+ '  </TD>'
+ '  <TD ALIGN=LEFT>'
+ '    <TEXTAREA NAME="edSQL" COLS=60 ROWS="8" WRAP="SOFT">'
+ '    </TEXTAREA>'
+ '  </TD>'
+ '</TR>'
+ '</TABLE></CENTER>'
+ '<CENTER>'
+ '<BR><INPUT TYPE="SUBMIT" VALUE="  Ok  "></TD>'
+ '</CENTER>'
+ '</FORM>'
+ '</BODY>'
+ '</HTML>';
end;

```

The second procedure receives the input parameters and connects to the server, executes the query, and creates an HTML response with the result of the query. The source code of this procedure is listed below:

```

procedure TWebMod.WebModactResultAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
var
  QMF: QMFWin;
  Col: Integer;
  myColumnCount: Integer;
  FetchResult: Integer;
  QueryNumber: Integer;
  ColumnHeadings: OleVariant;
  myRow: OleVariant;
  UserID: String;
  Password: String;
  ServerName: String;
  SQLStatement: String;
begin
  //Receives the ServerName, UserID, Password and SQLStatement
  //and copy them to a local variable
  UserID:= Trim(Request.ContentFields.Values['edUserID']);
  Password:= Trim(Request.ContentFields.Values['edPassword']);
  ServerName:= Trim(Request.ContentFields.Values['edServerName']);
  SQLStatement:= Trim(Request.ContentFields.Values['edSQL']);
  if (UserID = '')
  or (Password = '')
  or (ServerName = '')
  or (SQLStatement = '') then

```

```

//if one of the fields are empty then display error message
Response.Content:= 'Field empty.'
else
begin
//Initialize the Ole - Required when creating CGIs
OleInitialize(nil);
//Initialize QMF Object
QMF:= CoQMFWin.Create;
//try to initialize server
if QMF.InitializeServer(ServerName,UserID,
                        Password, False, '', True) <> 0 then
//if not successful then display error message
Response.Content:= 'Could not initialize server. '
                  + QMF.GetLastErrorString()
else
begin
//try to initialize the query selected by the user
QueryNumber:= QMF.InitializeQuery(0,SQLStatement);
if QueryNumber < 0 then
//if not successful display error message
Response.Content:= 'Could not initialize query '
                  + QMF.GetLastErrorString()
else
//if successful try to open the selected query
//without any limits of number of rows
begin
if QMF.Open(QueryNumber, 0, False) <> 0 then
//if not successful display error message
Response.Content:= 'Could not open the query '
                  + QMF.GetLastErrorString()
else
begin
//try to get the number of columns of the selected query
myColumnCount:= QMF.GetColumnCount(QueryNumber);
if myColumnCount <= 0 then
Response.Content:= 'Could not count numbers for columns. '
                  + QMF.GetLastErrorString()
else
//if successful set the HTML with the
//appropriate numbers of columns
begin
//try to get the column headers
if QMF.GetColumnHeadings(QueryNumber,
                          ColumnHeadings) <> 0 then
Response.Content:= 'Could not get columns headings. '
                  + QMF.GetLastErrorString()
else

```

```

//if successful display the column headings
//creating the HTML for the response
begin
Response.Content:= '<HTML>'
    + '<BODY>'
    + '<h1>QMF CGI Sample</h1>'
    + '<BR><b>SERVER: </b>' + ServerName
    + '<BR><b>SQL Statement: </b>' + SQLStatement
    + '<BR><center><table BORDER COLS='
    + IntToStr(myColumnCount)
    + ' WIDTH="100%" >'
    + '<BR>'
    + '<tr BGCOLOR="#FFFFCC">';
//fill the HTML with the column name
for Col:= 0 to (myColumnCount - 1) do
    Response.Content:= Response.Content
        + '<td>'
        + ColumnHeadings[Col]
        + '</td>';
Response.Content:= Response.Content + '</tr>';
//try to fetch all the rows from the query
//and place the data on the HTML
FetchResult:= QMF.FetchNextRow(QueryNumber, myRow);
while FetchResult = 0 do
    begin
    for Col:= 0 to (myColumnCount - 1) do
        Response.Content:= Response.Content
            + '<td>'
            + VarToStr(myRow[Col])
            + '</td>';

        Response.Content:= Response.Content + '</tr>';
        FetchResult:= QMF.FetchNextRow(QueryNumber, myRow);
    end;

//close HTML
Response.Content:= Response.Content
    + ' </table></center>'
    + '</BODY>'
    + '</HTML>';

if FetchResult <> -1 then
    //if the result of the FetchNextRow API
    //is different than -1, that means that
    //an error occurred
    Response.Content:= 'Could not fetch next row. '
        + QMF.GetLastErrorString();

```

Chapter 8. Summary

Throughout this book, we have described the benefits of using the QMF family of integrated tools to build an Enterprise Query Environment. This environment will answer the main needs of an enterprise when having widely distributed database systems installed. QMF for Windows, which has been used mainly to demonstrate QMF, is representative of all other QMF tools available on different platforms. The entire environment not only helps the database administrator with the daily work of performance monitoring and checking for resource consumption, but it also simplifies the end users work.

For the information system employees, the main benefits are:

- Easy to install database front end across platforms
- Allows for object tracking
- Allows for resource planning
- Ability to cancel long running queries
- Central object repository
- Allows for Web publishing
- Batch mode
- Can create new queries using a Windows GUI and share these with existing host users
- No porting necessary to execute existing host QMF queries from Windows

For the end user, the main benefits are:

- Ready-to-run applications
- One tool for all databases
- Can share new queries and formats with other users
- Tightly integrated with standard applications

Figure 134 illustrates the benefits of the Enterprise Query Environment.

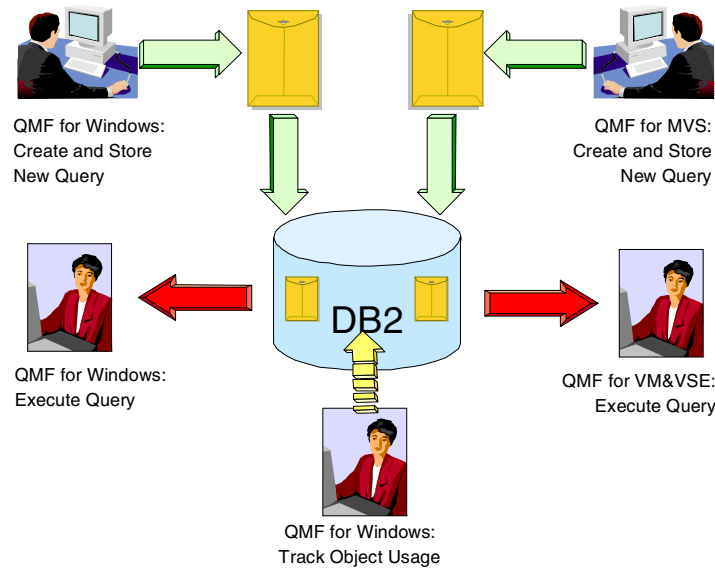


Figure 134. An Enterprise Query Environment

The Enterprise Query Environment allows for easy coexistence of Queries, Forms, Reports, and Procedures on all DB2 platforms. Wherever a new object is created, storing it at the appropriate database server makes this object accessible from all other platforms where QMF is installed.

8.1 Future directions

The main development issues of QMF for Windows up to now have been to make the functionality of the QMF host product available for users with Windows based systems.

There are, however, some functions not fully exploited with the current version of the product. For example, some of the DB2 database features, like the capability to store large objects (LOBS) inside the database tables, are not yet supported. Also, as the QMF for Windows product reaches a new type of end user, its interface will change in future releases. The "grid" — that is, the way the query result is presented to the user after executing a query without specifying a form to be used — will allow for better direct formatting. This will give the end user an easy way to reformat the result presentation without having to define a separate form to do this.

Another enhancement will also be seen, with the support of REXX logic to be available within the QMF procedures. This will allow procedures to be more flexible, and will enhance the functionality of procedures to incorporate logic, rather than being “linear” only.

To furthermore expand the platforms that QMF is available on, the product will be written in Java, thus allowing the product to be installed and executed on any operating system that provides a Java Virtual Machine (JVM). This way, QMF will also be available to be used from AIX based operating systems, thus allowing both Windows and AIX users to access DB2 on any platform using the same GUI.

8.2 QMF Personal Portal

The QMF for Windows product GUI may actually offer too many sophisticated possibilities for the end user who only needs to run existing queries using predefined formats. To make life easier for this type of user, Rocket Software provides a free download of an alternative front end called Rocket Personal Portal. This product will run on any workstation where QMF for Windows is installed. It provides a simple user interface for launching *centrally shared* QMF queries and reports, and sending the results to spreadsheets, desktop databases and browsers. The biggest advantage for the user is the way Personal Portal visualizes the information. It uses a tree-structure to do this, so the user can expand the objects for each server, while at the same time being able to see all other available servers, as shown in Figure 135.

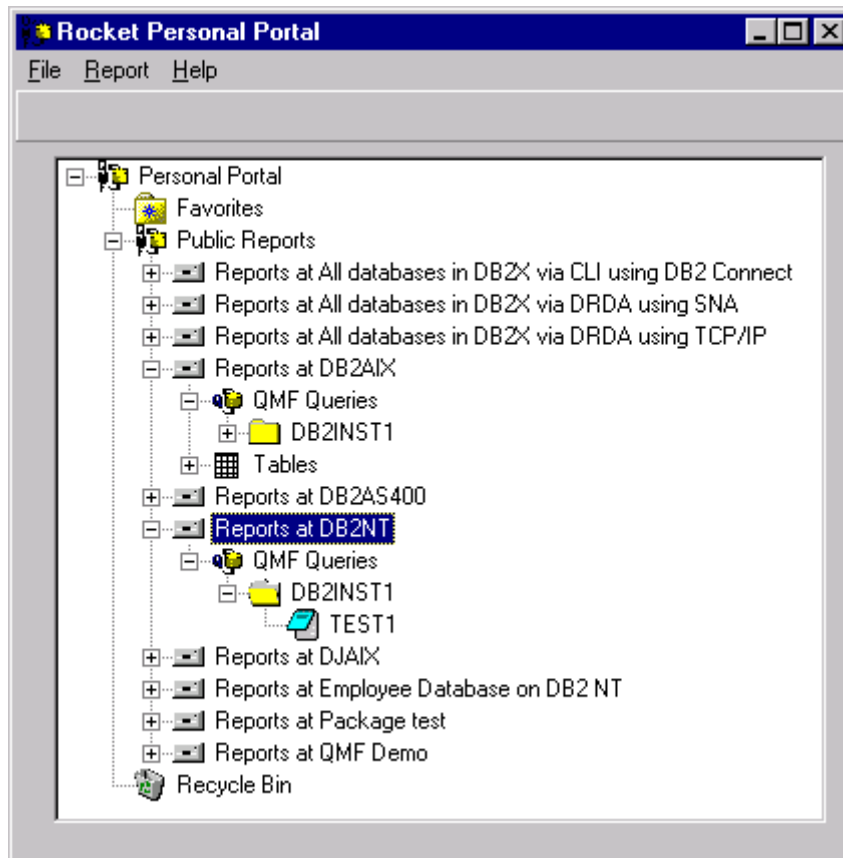


Figure 135. Rocket personal portal

Once the user has selected a specific query it, can be executed in by either double-clicking on its name or selecting the **Report -> Run** menu. The query result will then be displayed in the way specified in the Report -> Properties menu, by default pointing to the Notepad application. Within these properties, the user is also able to select the form used to format the query result. Other available applications predefined by Personal Portal are:

- Web Browser (.HTML format)
- Lotus 123 (.123 format)
- MS Excel (.XLS format)
- Lotus Wordpro (.LWP format)
- MS Word (.DOC format)
- MS Access (.MDB format)

In addition, the user can specify if the report is to be written in a temporary file or written permanently to the disk drive. Another option is to select a certain QMF Procedure to be executed before running the query. Figure 136 shows the Properties screen where all these definitions can be made, either temporarily or permanently, by selecting the **Save as Favorite** button.

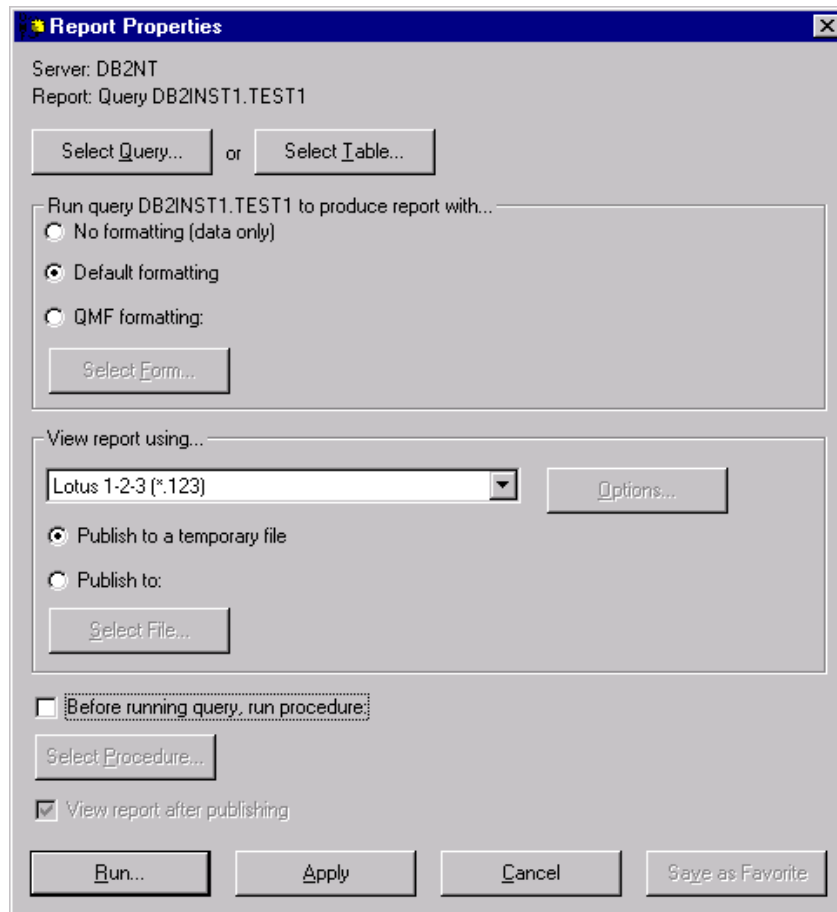


Figure 136. Properties screen

The end user can even select favorite queries and report settings for the Favorites folder, to allow easy access to the most frequently used objects. Personal Portal allows all the objects shown in the main window to be copied to the Favorites folder, as shown in Figure 137.

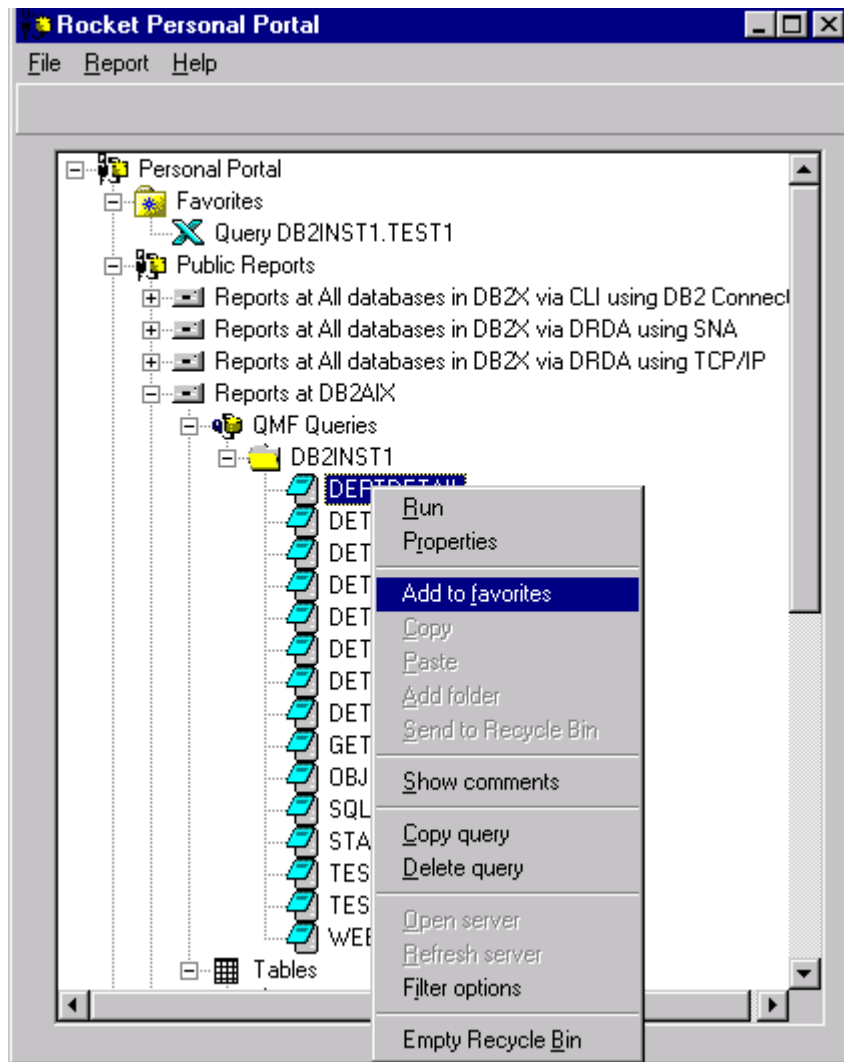


Figure 137. Copy to favorites

As the Rocket Personal Portal is downloadable for free from the Rocket Software Web site, please check for the latest version of this application.

Appendix A. Working with variables

Variables, as their name implies, are a part of a computer program code that can be modified while that program is running. In this way, the same program can be used for several actions. Let's take, for example, a very simple program that only sums two numbers. The program might look like this:

```
2 + 2 = X
```

In that case, the program result will always be 4, and that program will only be useful if we want to sum 2 and 2. If we want to have a generic program that sums any two numbers, we have to use variables. A program using variables might look like this:

```
A is an Integer  
B is an Integer  
A + B = X
```

In this case, A and B are variables; they can be any integer numbers, and so can the result, X. Now the same program can be used to sum any two numbers.

QMF for Windows has two kind of variables, *Substitution Variables* and *Global Variables*. The substitution variables are used in QMF objects for substituting variables to strings at run time. In this way, you can substitute a part of a SQL statement and make it more generic. Substitution variables are only active while the object (Query, Procedure, Form) is running. Hence, just one object can access it, and after the execution, the variable no longer exists. On the other hand, global variables are active while QMF is active, which means that variables will have the same value until the instance of QMF is finished, and can be accessed by all QMF objects. Different instances of QMF will not be able to see each other's global variables.

There is a way of keeping global variables from one instance to another. To do this, you need to modify an option in your Windows registry, and QMF will automatically save all global variables created in the Windows registry. You do this by going to the Windows registry editor in the directory **HKEY_CURRENT_USER\SOFTWARE\IBM\RDBI\Options**, creating a new entry called SaveGlobal as a DWORD value, and setting this value to 1, as shown in Figure 138.

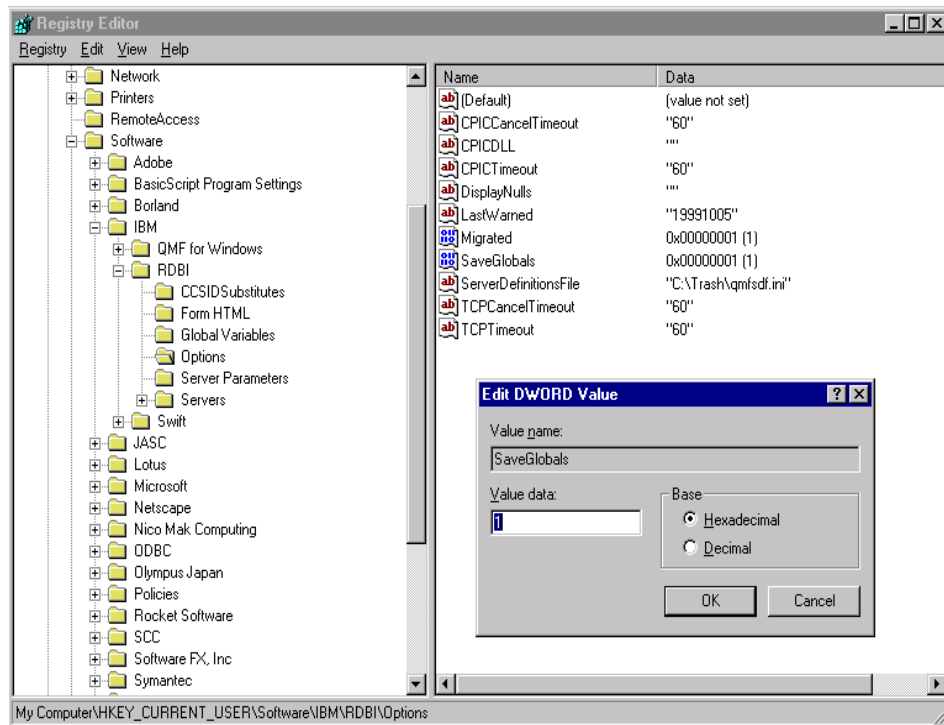


Figure 138. Windows registry

From that moment on, all the global variables that you create will be stored in **HKEY_CURRENT_USER\SOFTWARE\IBM\RDBI\Global Variables** in the Windows registry, and will be kept from instance to instance.

The structure of variables, then, are as shown in Figure 139. Substitution variables stay active only during the execution of the object. Global variables stay active while the QMF instance is active and, after the proper settings, the global variables can be permanent in the Windows registry.

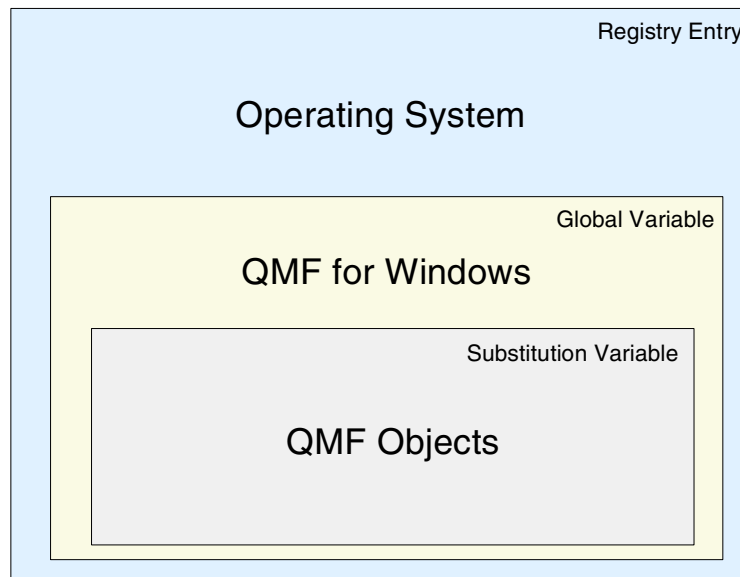


Figure 139. Variable structure

A.1 Substitution variables

Substitution variables are very useful for creating generic objects. Imagine that you need a query which lists all the information from a customer who is in debt. If you have to create one query for each customer who is in debt, you will probably need to create many queries, and they will not be flexible. When you use variables, you can create one single query that, before running, prompts you for the name or identification of the customer you want to see. Such a query may look like this:

```
SELECT *
FROM CUSTOMER
WHERE DEBIT = 'YES'
AND CUSTOMER_NUMBER = &CUSTOMER_NUMBER
```

The '&' character indicates that the string following is a substitution variable. When you run that query, a prompt will appear for you to input the customer number, as shown in Figure 140. After typing in the value, click on the **OK** button and the query will be executed with the value that you typed. So, while using the same query, you can retrieve all information from all customers that are in debt without the need to create several different queries, one for each customer. In QMF for Windows, you can have as many variables as you want in one query, or use the same variable in different places of the query.

Note:
QMF for OS/390, VM, and VSE have a total limit of 10 variables.

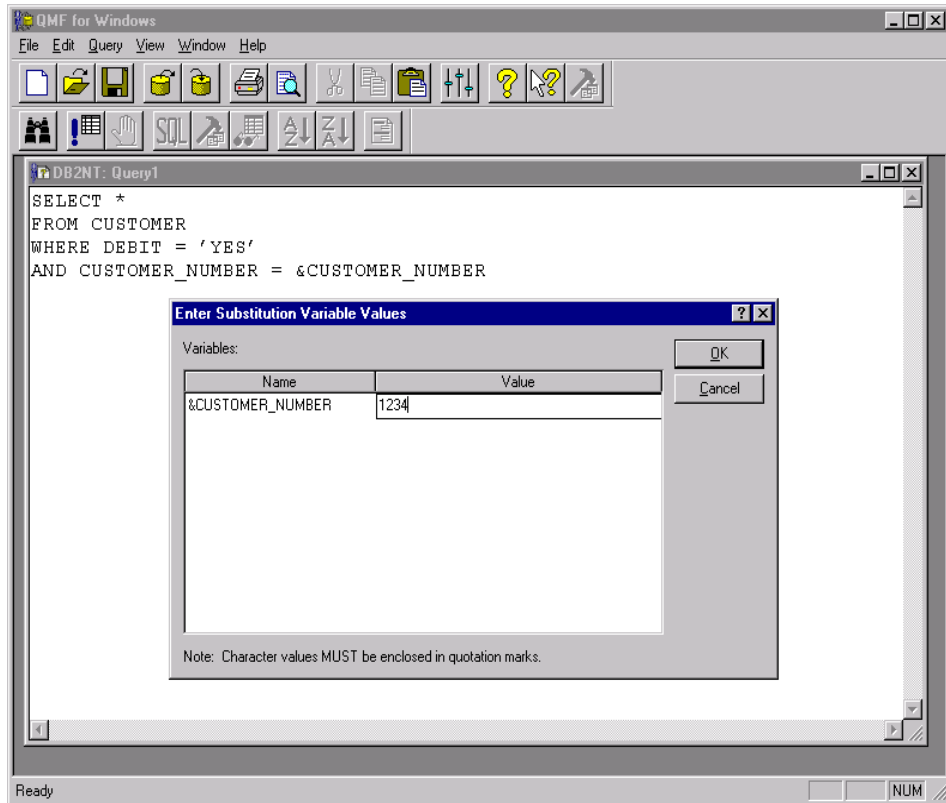


Figure 140. Substitution variable

When using the substitution variables, you need to remember that they will be substituted as typed; that means, if you are using the variable to substitute a value to compare with a string column, it has to be delimited by quotation marks, just as in the SQL statement.

Substitution variables can also be used in procedures with the same result. However, if your procedure has to run overnight, you cannot use substitution variables because the process will be stopped each time a variable is found and it will prompt the user to enter the value, thus interrupting the procedure. The solution to this problem is user defined global variables.

A.2 Global variables

Global variables are variables that stay active as long as the QMF instance is active. Also, it is possible to configure QMF to save your global variables from one instance to another as explained earlier. There are two types of global variables, the user defined global variables and the pre-loaded global variables.

A.2.1 User defined global variables

User defined global variables are global variables that the user defines for the execution of queries or procedures. In that case, if in the query or procedure, a variable that the user already defined in the global variable is used, no window will be displayed for the user to enter the value. The value defined for that global variable will be used instead.

To create your own global variables, follow the process below:

1. Go to the **View** and **Global Variables...** menus. A window as shown in Figure 141 will appear.

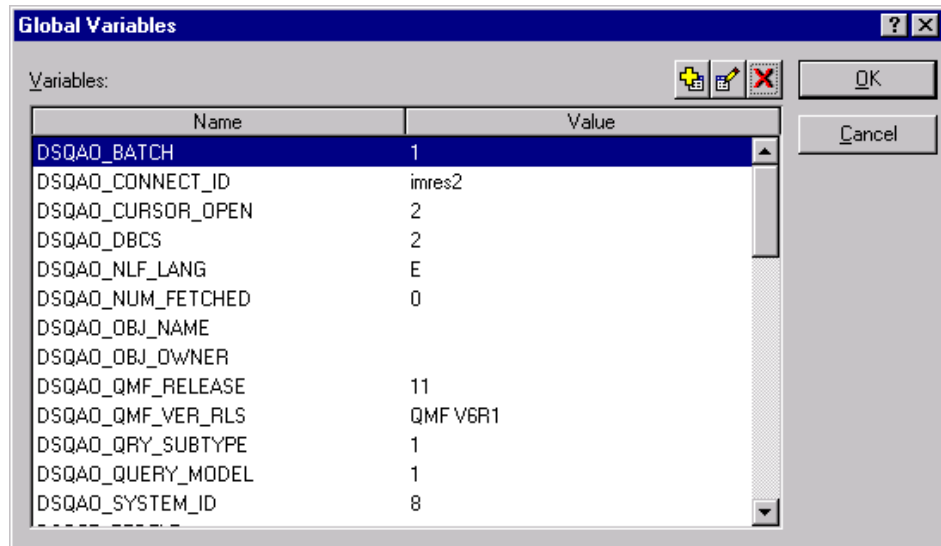


Figure 141. Global variables

2. Click on the **Add** button, and a window as shown in Figure 142 will appear.

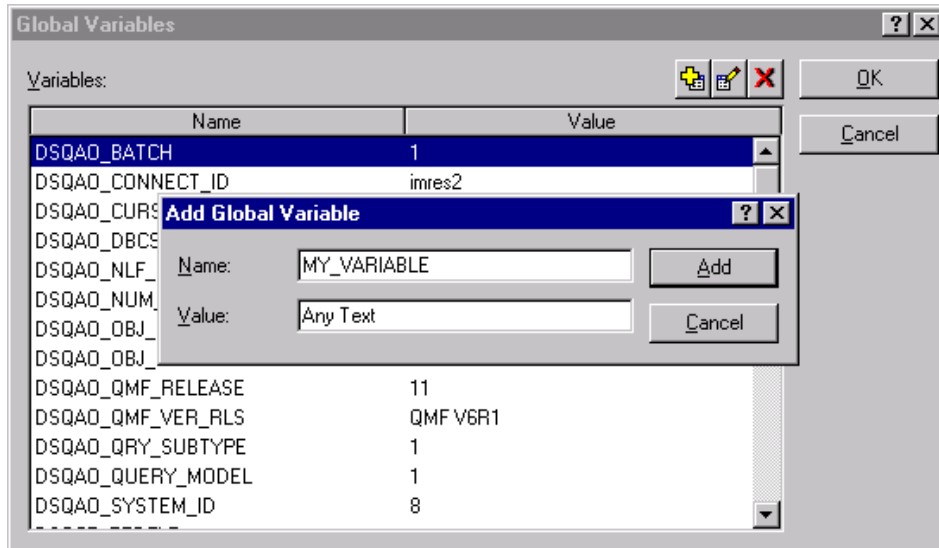


Figure 142. Adding global variables

3. In that window, type the global variable name and value. The name of the variable may be a maximum of 17 characters long, and the value cannot be longer than 55 characters. Also, variables may not begin with 'DSQ'.
4. After that, click on the **Add** button and the variable will be created as shown in Figure 143.

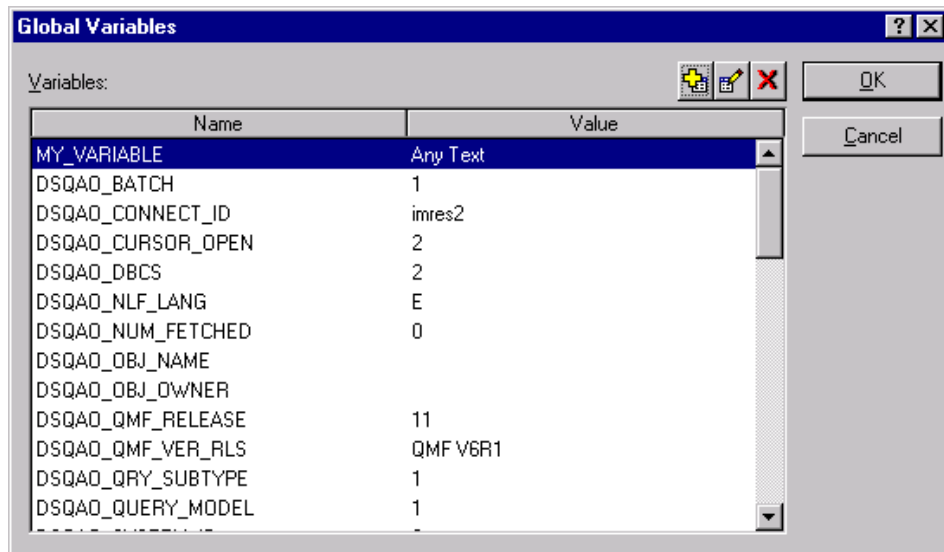


Figure 143. New global variable created

To edit the value or change the name of a global variable use the **Edit** button and to delete a variable use the **Delete** button.

A.2.2 Pre-loaded global variables

QMF for windows has a set of global variables that are pre-loaded with the product. You cannot change or delete any of these variables, or change the values in most of these variables.

QMF for Windows provides many variables for use in your queries, forms, and procedures. All of the global variables defined for host QMF are recognized by QMF for Windows, however, many are not applicable to the Windows environment. Only those listed in the following tables are used and accessible by QMF for Windows. Any references in queries, forms, or procedures to QMF host global variables that are not supported by QMF for Windows are ignored. See the host QMF reference for a complete listing of all host QMF global variables. Global variables prefixed by DSQQW are QMF for Windows global variables only; all other DSQ-prefixed global variables are supported in both environments.

The naming convention for global variables is DSQcc_XXXXXXXX, where DSQcc can be one of the following category identifiers, as shown in Table 9.

Table 9. Global variables naming convention

Identifier	Description
DSQAO	State information
DSQCP	Information about the Table Editor
DSQDC	Control of how QMF for Windows displays information
DSQEC	Control of how QMF for Windows executes commands and procedures
DSQQW	Controls specific to the QMF for Windows environment

A.2.2.1 State information variables — DSQAO

Global variables whose names begin with DSQAO provide state information about QMF for Windows. None of these variables can be modified by the SET GLOBAL command. Table 10 shows these variables.

Table 10. DSQAO global variables

Name	Length	Description
DSQAO_BATCH	1	Batch or interactive mode. Value can be '1' for an interactive session or '2' for a batch session. See the /Batch command line parameter.
DSQAO_CURSOR_OPEN	1	The status of the current query object's database cursor. Value can be '1' if the cursor is open or '2' if the cursor is closed.
DSQAO_DBCS	1	DBCS support status. Value can be '1' if DBCS support is present or '2' if DBCS support is not present.
DSQAO_NLF_LANG	1	National language of session. Value is 'E' for the English language.
DSQAO_NUM_FETCHED	10	The number of rows fetched by the current query object.
DSQAO_OBJ_NAME	18	The name of the current query, form, or procedure object. If there is no current object, the value is blank.
DSQAO_OBJ_OWNER	8	The owner of the current query, form, or procedure object. If there is no current object, the value is blank.

Name	Length	Description
DSQAO_QMF_RELEASE	2	Numeric release number of QMF for Windows. For QMF for Windows Version 6.1, this is '11'
DSQAO_QMF_VER_RLS	10	Version and release of QMF for Windows. For QMF for Windows Version 6.1, this is 'QMF V6R1'
DSQAO_QUERY_MODEL	1	Model of the current query object. Value can be '1' for relational.
DSQAO_QRY_SUBTYPE	1	Subtype of the current query object. Value can be '1' for SQL queries or '3' for prompted queries.
DSQAO_SYSTEM_ID	1	Current operating system. Values can be: '6' for Windows 3.x, '7' for Windows 95 or Windows 98, or '8' for Windows NT.

A.2.2.2 Table editor variables — DSQCP

Global variables whose names begin with DSQCP control the operation of the Table Editor. All of these variables can be modified by the SET GLOBAL command. Table 11 shows these variables.

Table 11. DSQCP global variables

Name	Length	Description
DSQCP_TEDFLT	1	Defines the reserved character used in the Table Editor to indicate a default value for a column. The default value is '+'. This variable can also be set on the Options dialog box.
DSQCP_TENULL	1	Defines the reserved character used in the Table Editor to indicate a null value for a column. The default value is '-'. This variable can also be set on the Options dialog box.

A.2.2.3 Display information variables — DSQCP

Global variables whose names begin with DSQDC control how QMF for Windows displays information. All of these variables can be modified by the SET GLOBAL command. Table 12 shows these variables.

Table 12. DSQCP global variables

Name	Length	Description
DSQDC_CURRENCY	18	Defines the custom currency symbol to use when the DC edit code is specified.
DSQDC_DISPLAY_RPT	1	Whether or not to display a report after a RUN QUERY command in a procedure. Value can be '0' to not display a report or '1' to automatically display a report with the default form. The default value is '0'.
DSQDC_LIST_ORDER	2	Sets the default sort order for objects in a List window. Value for the first character can be '1' (default order), '2' (sorted by object owner), '3' (sorted by object name), or '4' (sorted by object type). Value for the second character can be 'A' (sorted in ascending order) or 'D' (sorted in descending order). The default value is '1A'.

A.2.2.4 Command and procedures variables — DSQEC

Global variables whose names begin with DSQEC control how QMF for Windows executes commands and procedures. All of these variables can be modified by the SET GLOBAL command. Figure 13 shows these variables.

Table 13. DSQEC global variables

Name	Length	Description
DSQEC_FORM_LANG	1	Defines the default NLF language, in which a form will be saved or exported. Value can be '0' for the presiding NLF language or '1' for English. The default value is '1'.
DSQEC_NLFCMD_LANG	1	Defines the expected NLF language for commands in procedures. Value can be '0' for the presiding NLF language or '1' for English. The default value is '0'.

Name	Length	Description
DSQEC_RESET_RPT	1	Determines whether or not to prompt the user when an incomplete data object will affect performance. Value can be '0' (complete the data object without prompting), '1' (prompt the user whether or not to complete the data object), '2' (reset the data object without prompting)
DSQEC_SHARE	1	Specifies the default value for whether or not to share a saved object with other users. Value can be '0' (do not share the object) or '1' (share the object).

A.2.2.5 Windows environment variables — DSQQW

Global variables whose names begin with DSQQW are specific to the QMF for Windows environment. All of these variables can be modified by the SET GLOBAL command. Table 14 shows these variables.

Table 14. DSQQW global variables

Name	Length	Description
DSQQW_CONNECTIONS	1	Controls the use of database server connections while running a procedure. Value can be '0' to minimize the number of connections or '1' to allow a new connection for each RUN QUERY command. Specifying a value of '0' can force QMF for Windows to reset or complete a data object before continuing execution of a procedure. The default value is '0'.
DSQQW_EXP_DT_FRMT	1	The format to use when exporting data with the EXPORT DATA command in a procedure. Value can be '0' for text, '2' for HTML, '3' for CSV, or '4' for IXF. The default value is '0'.
DSQQW_EXP_OUT_MDE	1	The IXF variation to use when exporting data to an IXF file. Value can be '0' for System/370 character-mode IXF or '1' for PC/IXF. The default value is '1'.

Name	Length	Description
DSQQW_FST_SV_DATA	1	Controls the use of "fast mode" when saving data with the SAVE DATA command in a procedure. Value can be '0' to not use fast mode or '1' to use fast mode. The default value is '0'.
DSQQW_HTML_REFTEXT	55	The text that appears in a report when the &REF variable is used. The default value is 'Back To'.
DSQQW_QUERY_LANG	1	Specifies the subtype of query created when a DISPLAY QUERY command is executed but no query object exists. Value can be '0' for SQL or '1' for prompted. The default value is '0'.
DSQQW_RPT_COPIES	10	Specifies the number of copies to print when printing a report with the PRINT REPORT command in a procedure. The default value is '1'.
DSQQW_RPT_FONT	55	Specifies the font face name to use when printing a report with the PRINT REPORT command in a procedure. The default value is 'Courier New'.
DSQQW_RPT_FONT_BD	1	Specifies the font bold attribute to use when printing a report with the PRINT REPORT command in a procedure. The default value is '0'.
DSQQW_RPT_FONT_IT	1	Specifies the font italic attribute to use when printing a report with the PRINT REPORT command in a procedure. The default value is '0'.
DSQQW_RPT_FONT_SZ	2	Specifies the font point size to use when printing a report with the PRINT REPORT command in a procedure. The default value is '10'.

Name	Length	Description
DSQQW_RPT_LEN_TYP	1	Specifies the type of page length when printing a report with the PRINT REPORT command in a procedure. Value can be '0' to fit the length to the printed page, '1' to specify an explicit number of lines, or '2' to specify a continuous report with no page breaks. The default value is '0'.
DSQQW_RPT_NUM_CHR	10	Specifies the number of characters to fit across a printed page when printing a report with the PRINT REPORT command in a procedure. This has an effect only when DSQQW_RPT_WID_TYP is '1'. The default value is '80'.
DSQQW_RPT_NUM_LNS	10	Specifies the number of lines to fit down a printed page when printing a report with the PRINT REPORT command in a procedure. This has an effect only when DSQQW_RPT_LEN_TYP is '1'. The default value is '60'.
DSQQW_RPT_ORIENT	1	The page orientation to use when printing a report with the PRINT REPORT command in a procedure. Value can be '0' for portrait or '1' for landscape. The default value is '0'.
DSQQW_RPT_USE_PS	1	Specifies what page formatting options (page length, page width, and so on) to use when printing a report with the PRINT REPORT command in a procedure. Value can be '0' to use the values specified on the PRINT REPORT command or in global variables, or '1' to use the values specified in the form document's page setup. The default value is '1'.
DSQQW_RPT_WID_TYP	1	Specifies the type of page width when printing a report with the PRINT REPORT command in a procedure. Value can be '0' to fit the width to the printed page, or '1' to specify an explicit number of characters. The default value is '0'.

Name	Length	Description
DSQQW_SHOW_QUERY	1	Specifies which view of a query to display when a SHOW QUERY command is issued from a procedure. Value can be '0' for SQL or prompted view or '1' for results view. The default value is '0'.
DSQQW_STRIP_SPACE	1	Specifies whether or not to remove trailing spaces from the lines of queries and procedures retrieved from a database server. Value can be '0' to retain trailing spaces or '1' to remove trailing spaces. The default value is '1'.
DSQQW_SV_DATA_C_S	10	The number of rows to insert before committing the unit of work when saving data with a SAVE DATA command in a procedure. Value can be '0' for all of the rows or an explicit number of rows. The default value is '0'.
DSQQW_UEDIT_DLL	55	The name of the DLL implementing the user edit routines to make available when working with forms. The default value is 'rsuedit.dll'.

A.3 Form variables

In addition to the previously mentioned substitution variables and global variables, QMF for Windows provides another form of variables, called form variables.

Form variables are codes you can insert into text fields to produce information on the report itself. For example, you can insert a date variable to produce the current date whenever the report is printed. Different form variables are available, depending on the part of the form you are editing.

The following form variables are available:

- **&ROW** - Displays the number of the current row of data.
- **&DATE** - Displays the current date.
- **&TIME** - Displays the current time.
- **&PAGE** - Displays the current page number.

- **&COUNT** - Displays the number of rows retrieved or printed since the last break at the same level.
- **&CALCid** - Identifies a Form Calculation expression to use, where "id" is the ID number of the expression
- **&n** - Displays the value of a column, where "n" is the column number.
- **&an** - Displays the aggregation of a column, where "n" is the column number, and "a" is one the following aggregation variables: AVG, COUNT, CPCT, CSUM, FIRST, LAST, MAX, MIN, PCT, STDEV, SUM, TCPCT, or TPCT. The aggregation is based on the rows retrieved or printed since the last break at the same level.
- **Global Variables** – Displays the value of the global variable.
- **HTML Variables** – Displays the value of the HTML variable.

Table 15 shows the types of variables that can be used in each part of a form:

Table 15. Form variables

	&ROW	&DATE	&TIME	&PAGE	&COUNT	CALCid	&n	&an
Page Heading	X	X	X	X			X	
Page Footing	X	X	X	X			X	
Break Heading	X	X	X	X			X	
Break Footing	X	X	X	X	X	X	X	X
Calc Expression	X	X	X	X	X		X	X
Column Definition	X	X	X				X	
Condition	X	X	X				X	
Detail Heading	X	X	X	X			X	
Detail Block	X	X	X	X	X	X	X	X
Final Text	X	X	X	X	X	X	X	X

Global variables and HTML variables can be used throughout the form.

Appendix B. QMF for Windows APIs

This appendix lists all of the QMF for Windows Application Programming Interfaces (APIs) that the developer has available to create new applications.

B.1 AddDecimalHostVariable

short AddDecimalHostVariable(long QueryID, short Type, short Precision, short Scale, const VARIANT& Value)

Description

This function applies the data in Value to a variable in the static SQL statement initialized with QueryID. You call this function for each variable in the statement. QMF for Windows makes no attempt to match values to variables, so you must call this function in the same order as the variables in the SQL statement.

Parameters

Table 16 shows the parameters for this API.

Table 16. AddDecimalHostVariable parameters

Name	Description
QueryID	The ID of the query as returned from InitializeStaticQuery().
Type	The SQL data type of the value to be passed to the database server. This value influences the conversion of Value from a VARIANT data type to the value actually passed. The only valid value for AddDecimalHostVariable() is 484 (RSDT_DECIMAL).
Precision	The precision of the decimal value.
Scale	The scale of the decimal value.
Value	The data value to substitute in the statement. To specify a null value, the type of the variant should be set to VT_EMPTY.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString() or GetLastErrorType() to get additional error information.

B.2 AddHostVariable()

short AddHostVariable(long QueryID, short Type, const VARIANT& Value)

Description

This function applies the data in Value to a variable in the static SQL statement initialized with QueryID. You must call this function for each variable in the statement. QMF for Windows makes no attempt to match values to variables, so you must call this function in the same order as the variables in the SQL statement.

Parameters

Table 17 shows the parameters for this API.

Table 17. AddHostVariable parameters

Name	Description
QueryID	The ID of the query as returned from InitializeStaticQuery().
Type	The SQL data type of the value to be passed to the database server. This value influences the conversion of Value from a VARIANT data type to the value actually passed.
Value	The data value to substitute in the statement. To specify a null value, the type of the variant should be set to VT_EMPTY.

Table 18 shows the valid values for parameter Type.

Table 18. Valid values for the parameter type

Value	Meaning
384	(RSDT_DATE) Date
388	(RSDT_TIME) Time
392	(RSDT_TIMESTAMP) Time stamp
448	(RSDT_VARCHAR) Variable length character string
452	(RSDT_CHAR) Character string
464	(RSDT_VARGRAPHIC) Variable length graphic
468	(RSDT_GRAPHIC) Graphic
480	(RSDT_FLOAT) Floating point number
496	(RSDT_INTEGER) 4-byte integer
500	(RSDT_SMALLINT) 2-byte integer

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()` to get additional error information.

B.3 BindDecimalHostVariable()

short BindDecimalHostVariable(BSTR CollectionName, BSTR PackageName, short SectionNumber, short Number, BSTR Name, short DataType, short Precision, short Scale)

Description

This function binds a variable in the specified section. Include the text `:H` in the SQL text as a placeholder for a host variable. For each decimal host variable in the SQL text, you must call `BindDecimalHostVariable()` to specify information about the variable.

Parameters

Table 19 shows the parameters for this API.

Table 19. BindDecimalHostVariable parameters

Name	Description
CollectionName	The collection ID of the package you want to bind.
PackageName	The name of the package you want to bind.
SectionNumber	The section number of the statement within the collection and package you want to bind.
Number	The identifier for the variable you want to bind. The first variable in the SQL statement is variable 0, etc.
Name	Used by the database server for diagnostic purposes. This value is not validated nor required by QMF for Windows.
DataType	The SQL data type of the variable. The only valid value for <code>BindDecimalHostVariable()</code> is 484 (<code>RSDT_DECIMAL</code>).
Precision	The precision of the decimal variable.
Scale	The scale of the decimal variable.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

B.4 BindHostVariable()

short BindHostVariable(BSTR CollectionName, BSTR PackageName, short SectionNumber, short Number, BSTR Name, short DataType, short Length)

Description

This function binds a variable in the specified section. Include the text :H in the SQL text as a placeholder for a host variable. For each host variable in the SQL text, you must call BindHostVariable() to specify information about the variable.

Parameters

Table 20 shows the parameters for this API.

Table 20. BindHostVariable parameters

Name	Description
CollectionName	The collection ID of the package you want to bind.
PackageName	The name of the package you want to bind.
SectionNumber	The section number of the statement within the collection and package you want to bind.
Number	The identifier for the variable you want to bind. The first variable in the SQL statement is variable 0, etc.
Name	Used by the database server for diagnostic purposes. This value is not validated nor required by QMF for Windows.
DataType	The SQL data type of the variable.
Length	The length of the variable.

Table 21 shows the valid values for parameter DataType.

Table 21. Valid values for the parameter DataType

Value	Meaning
384	(RSDT_DATE)Date
388	(RSDT_TIME)Time
392	(RSDT_TIMESTAMP)Time stamp
448	(RSDT_VARCHAR)Variable length character string
452	(RSDT_CHAR)Character string
464	(RSDT_VARGRAPHIC)Variable length graphic

Value	Meaning
468	(RSDT_GRAPHIC)Graphic
480	(RSDT_FLOAT)Floating point number
484	(RSDT_DECIMAL)Decimal
496	(RSDT_INTEGER)4-byte integer
500	(RSDT_SMALLINT)2-byte integer

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

B.5 BindSection()

short BindSection(BSTR CollectionName, BSTR PackageName, short SectionNumber, BSTR SQLText)

Description

This function sets the SQL text to be used in the specified section number of the collection and package during binding.

Parameters

Table 22 shows the parameters for this API.

Table 22. *BindSection* Parameters

Name	Description
CollectionName	The collection ID of the package you want to bind.
PackageName	The name of the package you want to bind.
SectionNumber	The section number of the statement within the collection and package you want to bind.
SQLText	The SQL text for the statement you want to bind.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

B.6 CancelBind()

short CancelBind(BSTR CollectionName, BSTR PackageName)

Description

This function cancels a previously initialized bind operation. All information regarding the named package is released.

Parameters

Table 23 shows the parameters for this API.

Table 23. CancelBind Parameters

Name	Description
CollectionName	The collection name used in the previous call to StartBind().
PackageName	The package name used in the previous call to StartBind().

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastSQLCode(), GetLastSQLError(), or GetLastSQLState() to get additional error information.

B.7 ChangePassword()

short ChangePassword(BSTR NewPassword)

Description

This function changes the password for the user ID previously specified on the InitializeServer() call.

Note

Not all types of database servers support changing passwords. If the server specified on the InitializeServer() call does not support changing passwords, an error is returned, and the password is not changed.

Parameters

Table 24 shows the parameters for this API.

Table 24. *ChangePassword* parameters

Name	Description
NewPassword	The new password.

Return value

Zero if successful, non-zero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

B.8 ClearList()

short ClearList(short Type)

Description

This function re-initializes the internal list specified by the Type parameter.

Parameters

Table 25 shows the parameters for this API.

Table 25. *ClearList* parameters

Name	Description
Type	Either the value <code>RSL_SERVER</code> or <code>RSL_QUERY</code>

Return value

Zero if successful, `RS_ERROR_OUTOFRANGE` if unsuccessful.

Related topics

`Open()`

B.9 Close()

short Close(long QueryID)

Description

This function closes a query and invalidates QueryID. If there is a cursor open for the query, the cursor is closed, freeing the database for other users. This function does not terminate the connection to the database server. Since the connection remains open, no rollback or commit is performed.

Note

The name of this function conflicts with the Microsoft Access 2.0 keyword Execute. If you are using MS Access 2.0, place square brackets [] around the function name.

Parameters

Table 26 shows the parameters for this API.

Table 26. Close parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastErrorSQLCode(), GetLastErrorSQLError(), or GetLastErrorSQLState() to get additional error information.

Related topics

Execute(), Open()

B.10 Commit()

short Commit()

Description

This function commits any changes you made in the current unit of work, ends the current unit of work, closes any open cursors, and invalidates all outstanding query IDs.

Note

The name of this function conflicts with the Microsoft Access 2.0 keyword Execute. If you are using MS Access 2.0, place square brackets [] around the function name.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

Related topics

`Rollback()`

B.11 CompleteQuery()

`short CompleteQuery(long QueryID)`

Description

This function fetches all rows of a result set and stores them internally in QMF for Windows. If there is a cursor open for the query, the cursor is closed, freeing the database for other users. You use `FetchNextRow()` or `FetchNextRows()` to retrieve the rows. Call `Close()` when you are done with this query.

Parameters

Table 27 shows the parameters for this API.

Table 27. Completequery parameters

Name	Description
QueryID	The ID of the query as returned from <code>InitializeQuery()</code> .

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

B.12 CopyToClipboard()

`short CopyToClipboard(long QueryID, long FirstRow, long FirstCol, long LastRow, long LastCol, BOOL IncludeColHeadings, [VARIANT DateTimeFormat])`

Description

This function copies the specified range of rows and columns to the Clipboard. If you have not retrieved row data for all of the rows that you want to copy to the Clipboard, you call `CompleteQuery()` prior to calling this

function. An error message will be returned if you attempt to copy rows that have not been retrieved from the database.

Parameters

Table 28 shows the parameters for this API.

Table 28. *CopyToClipboard* parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().
FirstRow	The first row you want to include in the copy.
FirstCol	The first column you want to include in the copy.
LastRow	The last row you want to include in the copy, or -1 if all rows are included.
LastCol	The last column that you want to include in the copy, or -1 if all columns are included.
IncludeColHeadings	Use nonzero to include the column headings in the first row and zero to not include them
DateTimeFormat	Optionally, the format to use for date and time values. Valid values are 0 (ISO format), 1 (USA format), 2 (EUR format), 3 (JIS format), or 4 (Windows Control Panel format). The default value is 4.

Note

The value of a first row in a result set is 0, and the value of the last row is one less than the total number of rows. The value of the first column in a result set is 0, and the value of the last column is one less than the total number of columns.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString() or GetLastErrorType() to get additional error information. If the result set is empty or no rows have been retrieved from the database, nonzero is returned unless FirstRow = 0 and LastRow = -1. In this case, zero is returned and an empty string is copied to the Clipboard.

Related topics

Export()

B.13 DeleteQMFObject()

short DeleteQMFObject(BSTR OwnerAndName)

Description

This function deletes a QMF object (query, form, procedure, or table).

Parameters

Table 29 shows the parameters for this API.

Table 29. DeleteQMFObject parameters

Name	Description
OwnerAndName	A string containing the owner and name, separated by a period, of the object that you want to delete. For example, John.Query2

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastErrorSQLCode(), GetLastErrorSQLError(), or GetLastErrorSQLState() to get additional error information.

B.14 EndBind()

short EndBind(BSTR CollectionName, BSTR PackageName)

Description

This function completes the bind process for a static SQL package. Calling this function causes QMF for Windows to send the complete information for the current package to the database for processing.

Parameters

Table 30 shows the parameters for this API.

Table 30. EndBind parameters

Name	Description
CollectionName	The collection name used in the previous call to StartBind().
PackageName	The package name used in the previous call to StartBind().

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

B.15 Execute()

short `Execute(long QueryID)`

Description

This function executes an SQL statement that uses an SQL verb other than `SELECT`. Use `Execute()` when the statement does not return any results. For statements that do return results, use `ExecuteEx()`. For statements using the `SELECT` verb, use `Open()` instead of `Execute()` or `ExecuteEx()`. To determine the verb used by a query, call `GetQueryVerb()`.

Note

The name of this function conflicts with the Microsoft Access 2.0 keyword `Execute`. If you are using MS Access 2.0, place square brackets [] around the function name.

Parameters

Table 31 shows the parameters for this API.

Table 31. Execute parametes

Name	Description
QueryID	The ID of the query as returned from <code>InitializeQuery()</code> .

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

B.16 ExecuteEx()

short `ExecuteEx(long QueryID, VARIANT* Result)`

Description

This function executes an SQL statement that uses an SQL verb other than `SELECT`. Use `ExecuteEx()` when the statement returns results, for example,

with a SELECT INTO statement. For statements that do not return any results, use Execute(). For statements using the SELECT verb, use Open() instead of Execute() or ExecuteEx(). To determine the verb used by a query, call GetQueryVerb().

Parameters

Table 32 shows the parameters for this API.

Table 32. ExecuteEx parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().
Result	A pointer to a VARIANT in which the result will be stored. The result is an array (variant type VT_ARRAY VT_VARIANT) containing one value for each column in the result. Each value is specified in either its native data type or the closest variant data type. The supported return types are: string (variant type VT_BSTR), float (variant type VT_R4), double (variant type VT_R8), short (variant type VT_I2), long (variant type VT_I4), and binary (variant type VT_UI1 VT_ARRAY). You must properly initialize the VARIANT before calling this function. Visual Basic does this automatically. Visual C++ programmers must call VariantInit().

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastSQLCode(), GetLastSQLError(), or GetLastSQLState() to get additional error information.

B.17 ExecuteStoredProcedure()

```
short ExecuteStoredProcedureEx(long QueryID, [VARIANT vaCommitOK],
[VARIANT vaMaxResultSets], [VARIANT vaColumnNames], [VARIANT
vaColumnLabels], [VARIANT vaColumnComments])
```

Description

This function executes an SQL statement that uses the SQL verb CALL, to run a stored procedure at the database server. Use ExecuteStoredProcedure() when the stored procedure does not return any results (instead of or in addition to result sets). For stored procedures that do return results, use ExecuteStoredProcedureEx().

To initialize a stored procedure for execution with ExecuteStoredProcedure(), first call InitializeQuery() specifying an SQL statement that uses the CALL

statement. The stored procedure name must be specified as a literal in the CALL statement. Any parameters specified in the CALL statement (constant or otherwise) are ignored. Instead, use AddHostVariable() to specify the input and output host variables.

If the stored procedure returns result sets, call GetStoredProcedureResultSets() to retrieve the query IDs for the result sets.

Parameters

Table 33 shows the parameters for this API.

Table 33. ExecuteStoredProcedure parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery(). The SQL text for the query should specify a CALL statement.
vaCommitOK	An optional boolean value specifying whether the stored procedure can commit the unit of work or if this operation should be restricted. The default value is True.
vaMaxResultSets	An optional numeric value specifying the maximum number of result sets that the stored procedure should be allowed to return. Specify zero if you do not want the stored procedure to return any result sets or if the database server does not support returning result sets from stored procedures over DRDA.
vaColumnNames	An optional boolean value specifying whether or not the database should return column names for the columns in each returned result set.
vaColumnLabels	An optional boolean value specifying whether or not the database should return column labels for the columns in each returned result set.
vaColumnComments	An optional boolean value specifying whether or not the database should return column comments for the columns in each returned result set.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastErrorSQLCode(), GetLastErrorSQLError(), or GetLastErrorSQLState() to get additional error information.

B.18 ExecuteStoredProcedureEx()

short ExecuteStoredProcedure(long QueryID, VARIANT* Result, [VARIANT vaCommitOK], [VARIANT vaMaxResultSets], [VARIANT vaColumnNames], [VARIANT vaColumnLabels], [VARIANT vaColumnComments])

Description

This function executes an SQL statement that uses the SQL verb CALL, to run a stored procedure at the database server. Use ExecuteStoredProcedureEx() when the stored procedure returns results (instead of or in addition to result sets). For stored procedures that do not return results, use ExecuteStoredProcedure().

To initialize a stored procedure for execution with ExecuteStoredProcedure(), first call InitializeQuery() specifying an SQL statement that uses the CALL statement. The stored procedure name must be specified as a literal in the CALL statement. Any parameters specified in the CALL statement (constant or otherwise) are ignored. Instead, use AddHostVariable() to specify the input and output host variables.

If the stored procedure returns result sets, call GetStoredProcedureResultSets() to retrieve the query IDs for the result sets.

Parameters

Table 34 shows the parameters for this API.

Table 34. ExecuteStoredProcedureEx parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery(). The SQL text for the query should specify a CALL statement.
Result	A pointer to a VARIANT in which the result will be stored. The result is an array (variant type VT_ARRAY VT_VARIANT) containing one value for each column in the result. Each value is specified in either its native data type or the closest variant data type. The supported return types are: string (variant type VT_BSTR), float (variant type VT_R4), double (variant type VT_R8), short (variant type VT_I2), long (variant type VT_I4), and binary (variant type VT_UI1 VT_ARRAY). You must properly initialize the VARIANT before calling this function. Visual Basic does this automatically. Visual C++ programmers must call VariantInit().

Name	Description
vaCommitOK	An optional boolean value specifying whether the stored procedure can commit the unit of work or if this operation should be restricted. The default value is True.
vaMaxResultSets	An optional numeric value specifying the maximum number of result sets that the stored procedure should be allowed to return. Specify zero if you do not want the stored procedure to return any result sets or if the database server does not support returning result sets from stored procedures over DRDA.
vaColumnNames	An optional boolean value specifying whether or not the database should return column names for the columns in each returned result set.
vaColumnLabels	An optional boolean value specifying whether or not the database should return column labels for the columns in each returned result set.
vaColumnComments	An optional boolean value specifying whether or not the database should return column comments for the columns in each returned result set.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

B.18.1 Export()

short Export(long QueryID, long FirstRow, long FirstCol, long LastRow, long LastCol, short Format, short StringDelimiter, short ColumnDelimiter, BOOL IncludeColHeadings, BSTR FileName, [VARIANT DateTimeFormat])

Description

This function exports the specified range of rows and columns using the specified options to the specified file. You call `CompleteQuery()` prior to calling this function if you have not retrieved row data for all of the rows you want to export. An error message is returned if you attempt to export rows that have not been retrieved from the database.

Note

The name of this function conflicts with the Microsoft Access 2.0 keyword Execute. If you are using MS Access 2.0, place square brackets [] around the function name.

Parameters

Table 35 shows the parameters for this API.

Table 35. Export parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().
FirstRow	The first row you want to include in the export.
FirstCol	The first column you want to include in the export.
LastRow	The last row you want to include in the export, or -1 if all rows are included.
LastCol	The last column that you want to include in the export, or -1 if all columns are included.
Format	Specifies the output format.
StringDelimiter	Specifies the string delimiter. This parameter is ignored if Format is RSEF_HTML
ColumnDelimiter	Specifies the column delimiter. This parameter is ignored if Format is RSEF_HTML
IncludeColHeadings	Use nonzero to include the column headings in the first row and zero to not include them.
FileName	A string containing the name of the file to which you want to write the export.
DateTimeFormat	Optionally, the format to use for date and time values. Valid values are 0 (ISO format), 1 (USA format), 2 (EUR format), 3 (JIS format), or 4 (Windows Control Panel format). The default value is 4.

Note

The value of a first row in a result set is 0, and the value of the last row is one less than the total number of rows. The value of the first column in a result set is 0, and the value of the last column is one less than the total number of columns.

Table 36 shows the valid values for parameter Format.

Table 36. Valid values for the parameter format

Value	Meaning
0 (RSEF_TEXT)	The output file will be written in plain text format.
1 (RSEF_HTML)	The output file will be written in HTML (Hyper Text Markup Language) format, and the data will be organized in an HTML table.
2 (RSEF_CSV)	The output file will be written in CSV (comma separated values) format.
3 (RSEF_PCIXF)	The output file will be written in PC/IXF format.
4 (RSEF_S370IXF)	The output file will be written in System/370 IXF format.

Table 37 shows the valid values for parameter StringDelimiter.

Table 37. Valid values for the parameter StringDelimiter

Value	Meaning
0 (RSSD_NONE)	No string delimiter is used.
1 (RSSD_SINGLEQUOTE)	Strings are delimited by a single quote character (').
2 (RSSD_DOUBLEQUOTE)	Strings are delimited by a double quote character (").

Table 38 shows the valid values for parameter ColumnDelimiter.

Table 38. Valid values for the parameter ColumnDelimiter

Value	Meaning
0 (RSCD_SPACE)	Columns are delimited by a space character ().
1 (RSCD_TAB)	Columns are delimited by a tab character (\t).
2 (RSCD_COMMA)	Columns are delimited by a comma character (,).

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()` to get additional error information. If the result set is empty or no rows have been retrieved from the database, nonzero is returned unless `FirstRow = 0`, and `LastRow = -1`. In this case, zero is returned and an empty file is written.

Related topics

`CopyToClipboard()`

B.19 ExportForm()

`short ExportForm(BSTR OwnerAndName, BSTR FileName)`

Description

This function exports the specified QMF form to the specified file.

Parameters

Table 39 shows the parameters for this API.

Table 39. ExportForm parameters

Name	Description
OwnerAndName	A string containing the owner and name, separated by a period, of the form that you want to export. For example, John.Query2
FileName	A string containing the name of the file to which you want to write the exported form.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

Related topics

`PrintReport()`

B.20 ExportReport()

`short ExportReport(long QueryID, short SourceType, BSTR Source, BSTR OutputFileName, short PageLength, short PageWidth, BOOL`

IncludeDateTime, BOOL IncludePageNumbers, [VARIANT Format],
[VARIANT UseFormPageSetup])

Description

This function creates a report for the specified query and writes it to a file. You specify the formatting and layout for the report in a QMF form. The output file is an ASCII text file with each line separated by a pair of carriage return and line feed characters, and each page separated by a form feed character. It is best to view the output file using a fixed-pitch font.

Parameters

Table 40 shows the parameters for this API.

Table 40. *ExportReport* parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().
SourceType	specifies the source of the form.
Source	The name (either a filename or Owner.Name) of the form you want to use.
OutputFileName	The name of the file to which you want to write the report.
PageLength	The number of lines on each page of the report. A PageLength of -1 specifies continuous output (no page breaks unless the report is wider than PageWidth).
PageWidth	The number of characters on each line of the report. A PageWidth of -1 specifies continuous output (lines are made just long enough to contain the full width of the report).
IncludeDateTime	Nonzero specifies that the date and time is included at the bottom of each page. Zero specifies that the date and time is not included.
IncludePageNumbers	Nonzero specifies that page numbers are included at the bottom of each page. Zero specifies that page numbers are not included.
Format	Optionally, specifies the format of the exported report. If zero, the format is plain text, specifying that the output should be exactly that produced by the form (text or HTML, depending on the type of form). If nonzero, the format is HTML, specifying that the output should be HTML. With non-HTML forms, the output is converted to HTML by adding HTML tags at the beginning and end of the output. The default value is zero.

Name	Description
UseFormPageSetup	Optionally, if nonzero specifies that the PageLength, PageWidth, IncludeDateTime, and IncludePageNumbers parameters should be ignored, and values for them should instead be taken from the values saved with the specified form. The default value is zero.

Table 41 shows the valid values for parameter SourceType.

Table 41. Valid values for the parameter SourceType

Value	Meaning
0 (RSF_DEFAULT)	Use the default form. FormName should be an empty string.
1 (RSF_DATABASE)	Use a form from the database. Specify the form owner and name (Owner.Name) in the FormName parameter. To use a form located on a different database server, first use ExportForm() to export the form to a file and then specify a SourceType of RSF_FILE.
2 (RSF_FILE)	Use the form contained in a file. Specify the filename in the FormName parameter.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastSQLCode(), GetLastSQLError(), or GetLastSQLState() to get additional error information.

Related topics

ExportForm()

B.21 FastSaveData()

short FastSaveData(long QueryID, BOOL Replace, BSTR TableName, BSTR TableSpaceName, [VARIANT Comment])

Description

This function runs the query specified by QueryID and saves the results of that query in the specified table in the specified table space. The query is run and the data is saved directly into the specified table at the database server. You can use this function to save rows that have not been retrieved from the database. If the specified table already exists, the new data must have the same number and types of columns as the existing table.

Note

Not all database servers are able to handle this type of process, so check with your system administrator if you receive errors when saving data with this method.

This function operates in a separate unit of work than other API functions, and its results are automatically committed. Calling Commit() or Rollback() will have no effect on changes made by this function.

Parameters

Table 42 shows the parameters for this API.

Table 42. *FastSaveData* parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery()
Replace	Use nonzero if you want the specified data to replace any existing data in the table. Use zero if you want the specified data to be appended to any existing data in the table.
TableName	The name of the table in which you want to store the data. If the table does not exist, QMF for Windows will create it.
TableSpaceName	The name of the table space in which the table exists or will be created. If TableSpaceName is NULL or an empty string, QMF for Windows will use the default table space. If you have configured QMF for Windows to always use the default table space, this parameter is ignored. See RSR_SDDIFFERENTTS in the description for GetResourceLimit().
Comment	Optionally, a string that specifies a comment for the table in which the data is saved.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastErrorSQLCode(), GetLastErrorSQLError(), or GetLastErrorSQLState() to get additional error information.

B.22 FetchNextRow()

short FetchNextRow(long QueryID, VARIANT* Row)

Description

This function fetches the next row of data from the database.

Parameters

Table 43 shows the parameters for this API.

Table 43. *FetchNextRow* parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().
Row	A pointer to a VARIANT in which the result will be stored. The result is an array (variant type VT_ARRAY VT_VARIANT) containing one value for each column in the row. Call GetColumnCount() to determine the number of values in the array. Each value is specified in either its native data type or the closest variant data type. The supported return types are: string (variant type VT_BSTR), float (variant type VT_R4), double (variant type VT_R8), short (variant type VT_I2), long (variant type VT_I4), and binary (variant type VT_UI1 VT_ARRAY). When the end of the result set has been reached (there are no more rows to fetch) or if the result set is empty, the result is empty (variant type VT_EMPTY) instead of an array. You must properly initialize the VARIANT before calling this function. Visual Basic does this automatically. Visual C++ programmers must call VariantInit().

Note

Due to a bug in Microsoft Excel 7.0 and Microsoft Access 7.0 (and possibly other 32-bit Microsoft products that use Visual Basic for Applications), string data in Variant variables received from QMF for Windows may not be translated from Unicode (used by OLE) to ANSI (used by VBA). When this occurs, only the first character of the string is displayed. To remedy this problem, set the variable equal to an empty string before you call the QMF for Windows function that uses the variable.

Return value

Zero if successful, nonzero if unsuccessful. When the end of the result set is reached, the return value is -1. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastErrorSQLCode(), GetLastErrorSQLError(), or GetLastErrorSQLState() to get additional error information.

Related topics

FetchNextRows()

B.23 FetchNextRowEx()

short FetchNextRowEx(long QueryID)

Description

This function fetches the next row of data from the database. You can use this function in environments that do not support VARIANT arrays, such as Microsoft Access 2.0. Use this function in conjunction with GetColumnValue() to retrieve the data in each column for the current row.

Parameters

Table 44 shows the parameters for this API.

Table 44. FetchNextRowEx parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().

Return value

Zero if successful, nonzero if unsuccessful. When the end of the result set is reached, the return value is -1. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastErrorSQLCode(), GetLastErrorSQLError(), or GetLastErrorSQLState() to get additional error information.

Related topics

FetchNextRowsEx()

B.24 FetchNextRows()

short FetchNextRows(long QueryID, VARIANT* Rows, long* NumRows)

Description

This function fetches the next NumRows rows of data from the database.

Parameters

Table 45 shows the parameters for this API.

Table 45. FetchNextRows parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().

Name	Description
Rows	<p>A pointer to a VARIANT in which the result will be stored. The result is a two dimensional array (variant type VT_ARRAY VT_VARIANT) containing one value for each column in each row. Call GetColumnCount() to determine the number of columns in the array. The dimensions of the array are [NumRows][ColumnCount], even if the number of unfetched rows in the result set is less than NumRows (in this case, the array will contain extra, unused entries). Each value is specified in either its native data type or the closest variant data type. The supported return types are: string (variant type VT_BSTR), float (variant type VT_R4), double (variant type VT_R8), short (variant type VT_I2), long (variant type VT_I4), and binary (variant type VT_UI1 VT_ARRAY).</p> <p>When the end of the result set has been reached (there are no more rows to fetch) or if the result set is empty, the result is empty (variant type VT_EMPTY) instead of an array.</p> <p>You must properly initialize the VARIANT before calling this function. Visual Basic does this automatically. Visual C++ programmers must call VariantInit().</p>
NumRows	<p>A pointer to a long containing the number of rows to fetch. If the number of unfetched rows in the result set is less than NumRows, NumRows is reset to the actual number of rows contained in the result.</p>

Note

Due to a bug in Microsoft Excel 7.0 and Microsoft Access 7.0 (and possibly other 32-bit Microsoft products that use Visual Basic for Applications), string data in Variant variables received from QMF for Windows may not be translated from Unicode (used by OLE) to ANSI (used by VBA). When this occurs, only the first character of the string is displayed. To remedy this problem, set the variable equal to an empty string before you call the QMF for Windows function that uses the variable.

Return value

Zero if successful, nonzero if unsuccessful. When the end of the result set is reached, the return value is -1. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastSQLCode(), GetLastSQLError(), or GetLastSQLState() to get additional error information.

Related topics

FetchNextRow()

B.25 FetchNextRowsEx()

short FetchNextRowsEx(long QueryID, long* NumRows)

Description

This function fetches the next NumRows rows of data from the database. You can use this function in environments that do not support VARIANT arrays, such as Microsoft Access 2.0. Use this function in conjunction with GetColumnValueEx() to retrieve the data in each column for a given row.

Parameters

Table 46 shows the parameters for this API.

Table 46. FetchNextRowsEx parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().
NumRows	A pointer to a long containing the number of rows to fetch. If the number of un fetched rows in the result set is less than NumRows, NumRows is reset to the actual number of rows contained in the result.

Return value

Zero if successful, nonzero if unsuccessful. When the end of the result set is reached, the return value is -1. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastErrorSQLCode(), GetLastErrorSQLError(), or GetLastErrorSQLState() to get additional error information.

Related topics

FetchNextRowEx()

B.26 FlushQMFCache()

void FlushQMFCache()

Description

This function tells QMF for Windows to flush its cache of QMF information, discarding its contents. The next time QMF for Windows needs QMF information it obtains it from the database. Normally, QMF Windows caches QMF information obtained from the database to reduce database traffic and improve performance. You call this function prior to calling GetQMFObjectInfo(), GetQMFFQueryText(), or GetQMFObjectList() to ensure that the information returned by these functions is up to date.

Return value

None.

B.27 GetColumnCount()

long GetColumnCount(long QueryID)

Description

This function returns the number of columns in the result set.

Parameters

Table 47 shows the parameters for this API.

Table 47. *GetColumnCount* parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().

Return value

The number of columns in each row if successful. If unsuccessful, 0 or -1. If the return value is 0 or -1, you can call GetLastErrorString(), GetLastErrorType(), GetLastSQLCode(), GetLastSQLError(), or GetLastSQLState() to get additional error information.

B.28 GetColumnDataValue()

short GetColumnDataValue(long QueryID, long Index)

Description

This function returns the data value for the column specified in Index for the current row of data. After calling this function, the Value property can be interrogated for the returned value. You use this function with FetchNextRowEx() to access the data in a single row of data.

Parameters

Table 48 shows the parameters for this API.

Table 48. *GetColumnDataValue* parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().
Index	The zero based index of the row of data to be retrieved.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

B.29 GetColumnHeader()

BSTR GetColumnHeader(long QueryID, long Index, short* Result)

Description

This function returns the column header (column name) associated with the index `Index`.

Parameters

Table 49 shows the parameters for this API.

Table 49. GetColumnHeader parameters

Name	Description
QueryID	The ID of the query as returned from <code>InitializeQuery()</code> .
Index	The zero based index of the column header to be retrieved.
Result	Zero if successful, nonzero if unsuccessful. If <code>Result</code> is nonzero, you can call <code>GetLastErrorString()</code> , <code>GetLastErrorType()</code> , <code>GetLastSQLCode()</code> , <code>GetLastSQLError()</code> , or <code>GetLastSQLState()</code> to get additional error information.

Note

Column headings are not available for static SQL statements. For query IDs returned from `InitializeStaticQuery()`, `GetColumnHeader()` returns a string of the form `Coln` where `n` is the column number.

Return value

The string returned represents the column name as specified in the `Index` parameter.

B.30 GetColumnHeaderEx()

short GetColumnHeaderEx(long QueryID, long Index)

Description

This function retrieves the column header (column name) associated with the index Index. After calling this function, the Value property can be interrogated for the returned value.

Parameters

Table 50 shows the parameters for this API.

Table 50. *GetColumnHeaderEx* parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().
Index	The zero based index of the column header to be retrieved.

Note

Column headings are not available for static SQL statements. For query IDs returned from InitializeStaticQuery(), GetColumnHeaderEx() will return a string of the form Coln where n is the column number.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is zero, query the Value property for the string representing the column name. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastErrorSQLCode(), GetLastErrorSQLError(), or GetLastErrorSQLState() to get additional error information.

B.31 GetColumnHeadings()

short GetColumnHeadings(long QueryID, VARIANT* Headings)

Description

This function returns the column headings (also referred to as column names).

Parameters

Table 51 shows the parameters for this API.

Table 51. *GetColumnHeadings* parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().

Name	Description
Headings	A pointer to a VARIANT in which the result is stored. The result is an array of strings (variant type VT_ARRAY VT_BSTR) containing one string for each column heading. You must properly initialize the VARIANT before calling this function. Visual Basic does this automatically. Visual C++ programmers must call VariantInit().

Note

Due to a bug in Microsoft Excel 7.0 and Microsoft Access 7.0 (and possibly other 32-bit Microsoft products that use Visual Basic for Applications), string data in Variant variables received from QMF for Windows may not be translated from Unicode (used by OLE) to ANSI (used by VBA). When this occurs, only the first character of the string is displayed. To remedy this problem, set the variable equal to an empty string before you call the QMF for Windows function that uses the variable.

Note

Column headings are not available for static SQL statements. For query IDs returned from InitializeStaticQuery(), GetColumnHeadings() will return the strings Col1, Col2, etc.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastSQLCode(), GetLastSQLError(), or GetLastSQLState() to get additional error information.

B.32 GetColumnValue()

short GetColumnValue(long QueryID, long Index, VARIANT* Value)

Description

This function returns the data value for the column specified in Index for the current row of data. You use this function with FetchNextRowEx() to access the data in a single row of data.

Parameters

Table 52 shows the parameters for this API.

Table 52. *GetColumnValue* parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().
Index	The zero based index of the row of data to be retrieved.
Value	A pointer to a VARIANT in which you want to store the results. The result is a data value based on the variant type. You must properly initialize the VARIANT before calling this function. Visual Basic does this automatically. Visual C++ programmers must call VariantInit().

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastErrorCode(), GetLastErrorError(), or GetLastErrorState() to get additional error information.

B.33 GetColumnValueEx()

short GetColumnValueEx(long QueryID, long RowIndex, long ColIndex, VARIANT* Value)

Description

This function returns the data value for the column specified in ColIndex for the row of data specified in RowIndex. You use this function with FetchNextRowsEx() to access the data in a single row of data

Parameters

Table 53 shows the parameters for this API.

Table 53. *GetColumnValueEx* parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().
RowIndex	The zero based index of the row to be retrieved.
ColIndex	The zero based index of the column to be retrieved.

Name	Description
Value	A pointer to a VARIANT in which you want to store the result. You can query the resulting variant to find out the data type for further processing. You must properly initialize the VARIANT before calling this function. Visual Basic does this automatically. Visual C++ programmers must call VariantInit().

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastErrorSQLCode(), GetLastErrorSQLError(), or GetLastErrorSQLState() to get additional error information.

B.34 GetDefaultServerName()

BSTR GetDefaultServerName()

Description

This function returns a string containing the default server name.

Return value

A string that specifies the default server name.

B.35 GetGlobalVariable()

BSTR GetGlobalVariable(BSTR Name)

Description

This function retrieves the value of the specified global variable.

Parameters

Table 54 shows the parameters for this API.

Table 54. GetGlobalVariable parameters

Name	Description
Name	A string that contains the name of the variable you want to set.

Return value

A string containing the global variable value, or NULL if the variable has no value or an error occurs.

B.36 GetHostVariableNames()

short GetHostVariableNames(long QueryID, VARIANT* Names)

Description

This function returns an array of the names of all host variables referenced in the specified query. The query must be a static query referencing host variables (either stored with the QMF query or created by AddHostVariable()).

Parameters

Table 55 shows the parameters for this API.

Table 55. GetHostVariableNames parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().
Names	A pointer to a VARIANT in which you want to store the result array.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString() to get additional error information.

B.37 GetHostVariableTypeNames()

short GetHostVariableTypeNames(long QueryID, VARIANT* TypeNames)

Description

This function returns an array of the strings describing the data types of all host variables referenced in the specified query. The query must be a static query referencing host variables (either stored with the QMF query or created by AddHostVariable()).

Parameters

Table 56 shows the parameters for this API.

Table 56. GetHostVariableTypeNames parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().
TypeNames	A pointer to a VARIANT in which you want to store the result array.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` to get additional error information.

B.38 GetHostVariableTypes()

`short GetHostVariableTypes(long QueryID, VARIANT* Types)`

Description

This function returns an array of the data types of all host variables referenced in the specified query. The query must be a static query referencing host variables (either stored with the QMF query or created by `AddHostVariable()`). See `AddHostVariable()` for a list of the data types that can be returned.

Parameters

Table 57 shows the parameters for this API.

Table 57. GetHostVariableTypes parameters

Name	Description
QueryID	The ID of the query as returned from <code>InitializeQuery()</code> .
Types	A pointer to a <code>VARIANT</code> in which you want to store the result array.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` to get additional error information.

B.39 GetLastErrorString()

`BSTR GetLastErrorString()`

Description

This function returns a string containing information about the most recent error. If you call this function after a function that executed successfully (with no errors), then this function returns information about the last error that occurred during a prior function call. To avoid confusion, always call this function immediately after calling a function that returned an error.

Return value

A string containing error information. If no errors occurred since you created the QMF for Windows API object, `NULL` is returned.

Related topics

GetLastErrorType(), GetLastErrorSQLCode(), GetLastErrorSQLError(),
GetLastErrorSQLState()

B.40 GetLastErrorType()

short GetLastErrorType()

Description

This function returns the type of the most recent error. If you call this function after a function that executed successfully (with no errors), then this function returns information about the last error that occurred during a prior function call. To avoid confusion, always call this function immediately after calling a function that returned an error.

Return value

The number returned indicates the type of error. Table 58 shows the error types number and their meaning.

Table 58. GetLastErrorType - error types

Value	Meaning
0 (RS_ERROR_NONE)	No errors have occurred since the QMF for Windows API object was created.
1 (RS_ERROR_SQL)	A SQL error occurred. If the error occurred during a call to a function that takes QueryID as an argument, call Close() to close that query. No rollback is performed. You can continue to use the QMF for Windows API object, although you may encounter additional errors.
2 (RS_ERROR_USER_CANCEL)	A user cancelled the operation, usually by clicking Cancel on the busy window. This causes QMF for Windows to perform an implicit rollback (invalidating all outstanding query IDs) and destroy the connection to the database. You must call InitializeServer() or ReinitializeServer() if you wish to continue.

Value	Meaning
3 (RS_ERROR_FATAL_GOV)	A fatal governor error occurred. One possibility is that the QMF for Windows API timed out because the maximum allowable idle time was exceeded. This causes QMF for Windows to perform an implicit rollback (invalidating all outstanding query IDs) and destroy the connection to the database. You must call InitializeServer() or ReinitializeServer() if you wish to continue.
4 (RS_ERROR_NONFATAL_GOV)	A non-fatal governor error occurred. Either the maximum allowable number of rows to fetch was exceeded, or the SQL verb is not allowed. If the error occurred during a call to a function that takes QueryID as an argument, call Close() to close that query. No rollback is performed and the connection to the database is unaffected, so you may continue to use the QMF for Windows API object.
5 (RS_ERROR_OTHER)	A general error occurred. No rollback is performed. You can continue to use the QMF for Windows API object, although you may encounter additional errors.

Related topics

GetLastErrorString(), GetLastSQLCode(), GetLastSQLError(),
GetLastSQLState()

B.41 GetLastSQLCode()

long GetLastSQLCode()

Description

This function returns the SQL code for the most recent error. If you call this function after a function that executed successfully (with no errors), then this function returns information about the last error that occurred during a prior function call. To avoid confusion, always call this function immediately after calling a function that returned an error.

Return value

The SQL code for the most recent error. If no errors occurred since you created the QMF for Windows API object, or the most recent error was not an SQL error, zero is returned.

Related topics

GetLastErrorString(), GetLastErrorType(), GetLastError(),
GetLastSQLState()

B.42 GetLastError()

VARIANT GetLastError()

Description

This function returns detailed SQL error information for the most recent error. If you call this function after a function that executed successfully (with no errors), then this function returns information about the last error that occurred during a prior function call. To avoid confusion, always call this function immediately after calling a function that returned an error.

Return value

An array (variant type VT_ARRAY | VT_VARIANT) containing error information. If no errors occurred since you created the QMF for Windows API object, or the most recent error was not an SQL error, empty (variant type VT_EMPTY) is returned. The array format is shown in Table 59.

Table 59. Format of the array returned by the GetLastError API

Element	Type	Contents
0	long (VT_I4)	Code
1	string (VT_BSTR)	State
2	string (VT_BSTR)	ErrProc
3	string (VT_BSTR)	RDBName
4	long (VT_I4)	ErrD1
5	long (VT_I4)	ErrD2
6	long (VT_I4)	ErrD3
7	long (VT_I4)	ErrD4
8	long (VT_I4)	ErrD5
9	long (VT_I4)	ErrD6
10	string (VT_BSTR)	Warn0
11	string (VT_BSTR)	Warn1
12	string (VT_BSTR)	Warn2

Element	Type	Contents
13	string (VT_BSTR)	Warn3
14	string (VT_BSTR)	Warn4
15	string (VT_BSTR)	Warn5
16	string (VT_BSTR)	Warn6
17	string (VT_BSTR)	Warn7
18	string (VT_BSTR)	Warn8
19	string (VT_BSTR)	Warn9
20	string (VT_BSTR)	WarnA
21	string (VT_BSTR)	MessageTokens

Related topics

GetLastErrorString(), GetLastErrorType(), GetLastErrorCode(),
GetLastSQLState()

B.43 GetLastErrorState()

BSTR GetLastErrorState()

Description

This function returns the SQL state for the most recent error. If you call this function after a function that executed successfully (with no errors), then this function returns information about the last error that occurred during a prior function call. To avoid confusion, always call this function immediately after calling a function that returned an error.

Return value

A string containing the SQL code for the most recent error. If no errors occurred since you created the QMF for Windows API object, or the most recent error was not an SQL error, NULL is returned.

Related topics

GetLastErrorString(), GetLastErrorType(), GetLastErrorCode(),
GetLastSQLError()

B.44 GetOption()

short GetOption(short Option, VARIANT* Value)

Description

Gets the specified option value in QMF for Windows.

Parameters

Table 60 shows the parameters for this API.

Table 60. *GetOption* parameters

Name	Description
Option	Specifies which option to retrieve
Value	A pointer to a VARIANT in which the result is stored. You must properly initialize the VARIANT before calling this function. Visual Basic does this automatically. Visual C++ programmers must call VariantInit().

Table 61 shows the valid values for parameter Option.

Table 61. *Valid values for the parameter option.*

Value	Meaning
0 (RSO_SERVER_DEFINITION_FILE)	Server definition file name.
1 (RSO_CPIC_DLL)	CPI-C Provider DLL file name.
2 (RSO_CPIC_TIMEOUT_WARNING)	CPI-C warning timeout (in seconds). This limit is not used for the QMF for Windows API.
3 (RSO_CPIC_TIMEOUT_CANCEL)	CPI-C cancel timeout (in seconds).
4 (RSO_TCP_TIMEOUT_WARNING)	TCP warning timeout (in seconds). This limit is not used for the QMF for Windows API.
5 (RSO_TCP_TIMEOUT_CANCEL)	TCP cancel timeout (in seconds).
6 (RSO_DISPLAY_NULLS_STRING)	The string to use to display null values.
7 (RSO_ENTER_NULLS_STRING)	The string to use to enter null values.
8 (RSO_ENTER_DEFAULTS_STRING)	The string to use to enter default values.
9 (RSO_TRACE_FILE_1)	Trace file 1 name.
10 (RSO_TRACE_FILE_2)	Trace file 2 name.
11 (RSO_TCP_TRACE_LEVEL)	TCP trace level.

Value	Meaning
12 (RSO_CPIC_TRACE_LEVEL)	CPI-C trace level.
13 (RSO_DDM_TRACE_LEVEL)	DDM trace level.

Note

Due to a bug in Microsoft Excel 7.0 and Microsoft Access 7.0 (and possibly other 32-bit Microsoft products that use Visual Basic for Applications), string data in Variant variables received from QMF for Windows may not be translated from Unicode (used by OLE) to ANSI (used by VBA). When this occurs, only the first character of the string is displayed. To remedy this problem, set the variable equal to an empty string before you call the QMF for Windows function that uses the variable.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString() or GetLastErrorType() to get additional error information.

Related topics

SetOption()

B.45 GetOptionEx()

short GetOptionEx(short Option)

Description

Gets the specified option value in QMF for Windows. When the option value is returned, you must query the Option property for the data.

Parameters

Table 62 shows the parameters for this API.

Table 62. GetOptionEx parameters

Name	Description
Option	The option values are the same as those for the GetOption() call.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()` to get additional error information.

Related topics

`GetOption()`, `SetOption()`

B.46 GetProcText()

BSTR `GetProcText(long ProcID)`

Description

This function returns the text that is executed for the specified procedure, after variable substitution. You should use `SetProcVariable()` to set the value of any variables used in the procedure before calling this function.

Parameters

Table 63 shows the parameters for this API.

Table 63. GetProcText parameters

Name	Description
ProcID	The ID of the procedure as returned from <code>InitializeProc()</code> .

Return value

If successful, a string containing the procedure text is returned. If unsuccessful, NULL is returned. If the return value is NULL, you can call `GetLastErrorString()` or `GetLastErrorType()` to get additional error information.

B.47 GetProcVariables()

short `GetProcVariables(long ProcID, VARIANT* Variables)`

Description

This function returns an array of the names of all of the variables in the procedure's text. You must assign values to these variables by calling `SetProcVariable()` prior to running the procedure using `RunProc()`.

Parameters

Table 64 shows the parameters for this API.

Table 64. *GetProcVariables* parameters

Name	Description
ProcID	The ID of the procedure as returned from InitializeProc().
Variables	A pointer to a VARIANT in which the result is stored. The result is an array of strings (variant type VT_ARRAY VT_BSTR), with each string containing the name of one variable. If there are no variables in the procedure, the result is empty (variant type VT_EMPTY). You must properly initialize the VARIANT before calling this function. Visual Basic does this automatically. Visual C++ programmers must call VariantInit().

Note

Due to a bug in Microsoft Excel 7.0 and Microsoft Access 7.0 (and possibly other 32-bit Microsoft products that use Visual Basic for Applications), string data in Variant variables received from QMF for Windows may not be translated from Unicode (used by OLE) to ANSI (used by VBA). When this occurs, only the first character of the string is displayed. To remedy this problem, set the variable equal to an empty string before you call the QMF for Windows function that uses the variable.

Return value

Zero if successful, nonzero if unsuccessful. If there are no variables in the procedure, the return value is RS_ERROR_NO_DATA (-1). If the return value is nonzero, you can call GetLastErrorString() or GetLastErrorType() to get additional error information.

B.48 GetQMFOBJECTINFO()

short GetQMFOBJECTINFO(BSTR OwnerAndName, short Type, short Time, VARIANT* Value)

Description

This function returns information about a QMF object (either a form or a query). The information returned is specified by the Type and Time parameters.

Parameters

Table 65 shows the parameters for this API.

Table 65. *GetQMFObjectInfo* parameters

Name	Description
OwnerAndName	A string containing the owner and name, separated by a period, of the object for which you want to retrieve information. For example, John.Query2
Type	Specifies the type of information to get
Time	Specifies first used, last used, or last modified
Value	A pointer to a VARIANT in which the result is stored. For RSI_TIMESUSED, RSI_TIMESRUN, RSI_TIMESCANCELLED, and RSI_LEVEL, the result is a short (variant type VT_I2). For RSI_RESTRICTED the result is a boolean (variant type VT_BOOL). For all others, the result is a string (variant type VT_BSTR). You must properly initialize the VARIANT before calling this function. Visual Basic does this automatically. Visual C++ programmers must call VariantInit().

Note

Due to a bug in Microsoft Excel 7.0 and Microsoft Access 7.0 (and possibly other 32-bit Microsoft products that use Visual Basic for Applications), string data in Variant variables received from QMF for Windows may not be translated from Unicode (used by OLE) to ANSI (used by VBA). When this occurs, only the first character of the string is displayed. To remedy this problem, set the variable equal to an empty string before you call the QMF for Windows function that uses the variable.

Table 66 shows the valid values for parameter Type.

Table 66. *Valid values for the parameter type*

Value	Meaning
0 (RSI_COMMENT)	Comment
1 (RSI_LEVEL)	Level
2 (RSI_TYPE)	Type
3 (RSI_SUBTYPE)	Sub type
4 (RSI_RESTRICTED)	Restricted
5 (RSI_MODEL)	Model

Value	Meaning
6 (RSI_TIMESUSED)	Number of times used.
7 (RSI_TIMESRUN)	Number of times run.
8 (RSI_TIMESANCELED)	Number of times canceled.
9 (RSI_DATE)	Date first used, last used, or last modified.
10 (RSI_TIME)	Time first used, last used, or last modified.
11 (RSI_USERID)	User ID first used, last used, or last modified.
12 (RSI_SQLID)	SQL ID first used, last used, or last modified.
13 (RSI_ENVIRONMENT)	Environment first used, last used, or last modified.
14 (RSI_MODE)	Mode first used, last used, or last modified.
15 (RSI_COMMAND)	Command first used, last used, or last modified.

Table 67 shows the valid values for parameter Time.

Table 67. Valid values for the parameter Time

Value	Meaning
0 (RST_FIRSTUSED)	First used.
1 (RST_LASTUSED)	Last used.
2 (RST_LASTMODIFIED)	Last modified.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

B.49 GetQMFOBJECTINFOEX()

short GetQMFOBJECTINFOEX(BSTR OwnerAndName, short Type, short Time)

Description

This function returns information about a QMF object. The information returned is specified by the Type and Time parameters. After calling this function, the QMFOBJECTINFO property can be interrogated for the returned value.

Parameters

Table 68 shows the parameters for this API.

Table 68. *GetQMFOBJECTINFOEX* parameters

Name	Description
OwnerAndName	A string containing the owner and name, separated by a period, of the object for which you want to retrieve information. For example, John.Query2
Type	Specifies the type of information to get
Time	Specifies first used, last used, or last modified

Table 69 shows the valid values for parameter Type.

Table 69. *Valid values for the parameter Type*

Value	Meaning
0 (RSI_COMMENT)	Comment.
1 (RSI_LEVEL)	Level.
2 (RSI_TYPE)	Type.
3 (RSI_SUBTYPE)	Sub type.
4 (RSI_RESTRICTED)	Restricted.
5 (RSI_MODEL)	Model
6 (RSI_TIMESUSED)	Number of times used.
7 (RSI_TIMESRUN)	Number of times run.
8 (RSI_TIMESANCELED)	Number of times canceled.
9 (RSI_DATE)	Date first used, last used, or last modified.
10 (RSI_TIME)	Time first used, last used, or last modified.
11 (RSI_USERID)	User ID first used, last used, or last modified.
12 (RSI_SQLID)	SQL ID first used, last used, or last modified.
13 (RSI_ENVIRONMENT)	Environment first used, last used, or last modified.
14 (RSI_MODE)	Mode first used, last used, or last modified.
15 (RSI_COMMAND)	Command first used, last used, or last modified.

Table 70 shows the valid values for parameter Time.

Table 70. Valid values for the parameter Time

Value	Meaning
0 (RST_FIRSTUSED)	First used.
1 (RST_LASTUSED)	Last used.
2 (RST_LASTMODIFIED)	Last modified.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

B.50 GetQMFOBJECTLIST()

short GetQMFOBJECTLIST(BSTR Owner, BSTR Name, short Type, VARIANT* List)

Description

This function returns an array of the names of all QMF objects matching the patterns specified in the Owner and Name parameters.

Parameters

Table 71 shows the parameters for this API.

Table 71. GetQMFOBJECTLIST parameters

Name	Description
Owner	A string containing the owner of the objects you want to include in the returned list. To include all objects regardless of the owner this parameter was to be left blank or contain the "%" character.
Name	A string containing the name of the objects you want to include in the returned list. To include all objects regardless of the name this parameter was to be left blank or contain the "%" character.
Type	Specifies the types of QMF objects you want to include in the list. These values can be added together to specify multiple object types

Name	Description
List	A pointer to a VARIANT in which the result are stored. The result is an array of strings (variant type VT_ARRAY VT_BSTR), each of the format Owner.Name. If no matching QMF queries are found, the result is empty (variant type VT_EMPTY). You must properly initialize the VARIANT before calling this function. Visual Basic does this automatically. Visual C++ programmers must call VariantInit().

Note

Due to a bug in Microsoft Excel 7.0 and Microsoft Access 7.0 (and possibly other 32-bit Microsoft products that use Visual Basic for Applications), string data in Variant variables received from QMF for Windows may not be translated from Unicode (used by OLE) to ANSI (used by VBA). When this occurs, only the first character of the string is displayed. To remedy this problem, set the variable equal to an empty string before you call the QMF for Windows function that uses the variable.

Table 72 shows the valid values for parameter Type.

Table 72. Valid values for the parameter Type

Value	Meaning
2048 (RSQ_MASK_QUERIES)	Include QMF queries in the list.
1024 (RSQ_MASK_FORMS)	Include QMF forms in the list.
512 (RSQ_MASK_PROCS)	Include QMF procedures in the list.
256 (RSQ_MASK_TABLES)	Include tables in the list.

Return value

Zero if successful, nonzero if unsuccessful. If no matching QMF objects are found, the return value is zero. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastErrorSQLCode(), GetLastErrorSQLError(), or GetLastErrorSQLState() to get additional error information.

B.51 GetQMFOBJECTListEx()

short GetQMFOBJECTListEx(BSTR Owner, BSTR Name, short Type, short Index)

Description

This function returns the name of the QMF object matching the patterns specified in the Owner and Name parameters referenced by the Index parameter. After calling this function, the Value property can be interrogated for the returned value.

Parameters

Table 73 shows the parameters for this API.

Table 73. *GetQMFOBJECTListEx* parameters

Name	Description
Owner	A string containing the owner of the objects you want to include in the returned list.
Name	A string containing the name of the objects you want to include in the returned list.
Type	Specifies the types of QMF objects you want to include in the list. These values can be added together to specify multiple object types
Index	The index of the list of QMF objects that match the pattern.

Table 74 shows the valid values for parameter Type.

Table 74. *Valid values for the parameter Type*

Value	Meaning
2048 (RSQ_MASK_QUERIES)	Include QMF queries in the list.
1024 (RSQ_MASK_FORMS)	Include QMF forms in the list.
512 (RSQ_MASK_PROCS)	Include QMF procedures in the list.
256 (RSQ_MASK_TABLES)	Include tables in the list.

Return value

Zero if successful, nonzero if unsuccessful. If no matching QMF objects are found, the return value is RS_ERROR_OUTOFRANGE. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastSQLCode(), GetLastSQLError(), or GetLastSQLState() to get additional error information.

B.52 GetQMFPProcText()

BSTR GetQMFPProcText(BSTR OwnerAndName)

Description

This function retrieves the text stored in the specified procedure.

Parameters

Table 75 shows the parameters for this API.

Table 75. *GetQMFProcText* parameters

Name	Description
OwnerAndName	A string containing the owner and name, separated by a period, of the procedure you want to retrieve. For example, John.Proc2

Return value

A string containing the text for the procedure that was retrieved, or NULL if the procedure could not be retrieved. If the return value is NULL, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

B.53 `GetQMFQueryText()`

`BSTR GetQMFQueryText(BSTR OwnerAndName)`

Description

This function retrieves the SQL text stored in the specified query.

Parameters

Table 76 shows the parameters for this API.

Table 76. *GetQMFQueryText* parameters

Name	Description
OwnerAndName	A string containing the owner and name, separated by a period, of the query you want to retrieve. For example, John.Query2

Return value

A string containing the text for the query that was retrieved, or NULL if the query could not be retrieved. If the return value is NULL, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

B.54 `GetQueryText()`

`BSTR GetQueryText(long QueryID)`

Description

This function returns the SQL text that is executed for the specified query, after variable substitution. You should use `SetVariable()` to set the value of any variables used in the query before calling this function.

Parameters

Table 77 shows the parameters for this API.

Table 77. *GetQueryText* parameters

Name	Description
QueryID	The ID of the query as returned from <code>InitializeQuery()</code> .

Note

The query text is not available for static SQL statements. For query IDs returned from `InitializeStaticQuery()`, `GetQueryText()` returns an empty string.

Return value

If successful, a string containing the SQL text is returned. If unsuccessful, NULL is returned. If the return value is NULL, you can call `GetLastErrorString()` or `GetLastErrorType()` to get additional error information.

B.55 `GetQueryVerb()`

BSTR `GetQueryVerb(long QueryID)`

Description

This function returns a string containing the SQL verb you used in the query.

Parameters

Table 78 shows the parameters for this API.

Table 78. *GetQueryVerb* parameters

Name	Description
QueryID	The ID of the query as returned from <code>InitializeQuery()</code> .

Note

The query verb is not available for static SQL statements. For query IDs returned from `InitializeStaticQuery()`, `GetQueryVerb()` returns an empty string.

Return value

If the query is successful, a string containing the SQL verb is returned. If the query is unsuccessful, NULL is returned. If the return value is NULL, you can call `GetLastErrorString()` or `GetLastErrorType()` to get additional error information.

B.56 GetResourceLimit()

`short GetResourceLimit(short Resource, long* Value)`

Description

This function gets the requested resource limit. You must call `InitializeServer()` prior to calling this function, since resource limits are handled on a per-server basis.

Parameters

Table 79 shows the parameters for this API.

Table 79. GetResourceLimit parameters

Name	Description
Resource	The valid values for this parameter are shown in Table 80
Value	A pointer to a long in which the result is stored. The result is the value of the requested resource limit. For boolean values, the result is non-zero for true, zero for false. For <code>RSR_SAVE_DATA_TABLE_SPACE_NAME</code> , <code>RSR_DEF_COLLECTION</code> , and <code>RSR_ACCOUNT_STRING</code> , -1 is returned and the <code>ResourceLimit</code> property can be interrogated for the returned string value.

Table 80 shows the valid values for parameter `Resource`.

Table 80. Valid values for the parameter Resource

Value	Meaning
0 (<code>RSR_IDLE_CONNECTION_TIMEOUT</code>)	Idle connection timeout (in seconds).

Value	Meaning
1 (RSR_IDLE_QUERY_TIMEOUT_CANCEL)	Idle query timeout (in seconds).
2 (RSR_IDLE_QUERY_TIMEOUT_WARNING)	Idle query timeout (in seconds). This is the warning limit; it is not enforced for the QMF for Windows API.
3 (RSR_SERVER_RESPONSE_TIMEOUT_CANCEL)	Server timeout (in seconds).
4(RSR_SERVER_RESPONSE_TIMEOUT_WARNING)	Server timeout (in seconds). This is the warning limit; it is not enforced for the QMF for Windows API.
5 (RSR_MAX_ROWS_TO_FETCH_CANCEL)	Maximum number of rows to fetch.
6 (RSR_MAX_ROWS_TO_FETCH_WARNING)	Maximum number of rows to fetch. This is the warning limit; it is not enforced for the QMF for Windows API.
7 (RSR_MAX_BYTES_TO_FETCH_CANCEL)	Maximum number of bytes to fetch.
8 (RSR_MAX_BYTES_TO_FETCH_WARNING)	Maximum number of bytes to fetch. This is the warning limit; it is not enforced for the QMF for Windows API.
9 (RSR_MAX_CONNECTIONS)	Maximum number of connections allowed to the database server.
10 (RSR_ALLOW_SERVER_ACCESS_UI)	Is access allowed to the database server from the QMF for Windows user interface?

Value	Meaning
11 (RSR_ALLOW_SERVER_ACCESS_API)	Is access allowed to the database server from the QMF for Windows API?
12 (RSR_FETCH_ALL_ROWS)	Automatically fetch all rows?
13 (RSR_CONFIRM_UPDATES)	Confirm database server updates? This option has no effect on the QMF for Windows API; database updates are not confirmed for the QMF for Windows API.
14 (RSR_SUMMARY_TRACKING)	Is summary object tracking enabled?
15 (RSR_DETAILED_TRACKING)	Is detailed object tracking enabled?
16 (RSR_SQL_TRACKING)	Is SQL text tracking enabled?
17 (RSR_ADHOC_TRACKING)	Is ad hoc object tracking enabled?
18 (RSR_ALLOW_ACQUIRE)	Is the SQL verb ACQUIRE allowed?
19 (RSR_ALLOW_ALTER)	Is the SQL verb ALTER allowed?
20 (RSR_ALLOW_COMMENT)	Is the SQL verb COMMENT allowed?
21 (RSR_ALLOW_CREATE)	Is the SQL verb CREATE allowed?
22 (RSR_ALLOW_DELETE)	Is the SQL verb DELETE allowed?
23 (RSR_ALLOW_DROP)	Is the SQL verb DROP allowed?
24 (RSR_ALLOW_EXPLAIN)	Is the SQL verb EXPLAIN allowed?

Value	Meaning
25 (RSR_ALLOW_GRANT)	Is the SQL verb GRANT allowed?
26 (RSR_ALLOW_INSERT)	Is the SQL verb INSERT allowed?
27 (RSR_ALLOW_LABEL)	Is the SQL verb LABEL allowed?
28 (RSR_ALLOW_LOCK)	Is the SQL verb LOCK allowed?
29 (RSR_ALLOW_REVOKE)	Is the SQL verb REVOKE allowed?
30 (RSR_ALLOW_SELECT)	Is the SQL verb SELECT allowed?
31 (RSR_ALLOW_SET)	Is the SQL verb SET allowed?
32 (RSR_ALLOW_SIGNAL)	Is the SQL verb SIGNAL allowed?
33 (RSR_ALLOW_UPDATE)	Is the SQL verb UPDATE allowed?
34 (RSR_ALLOW_CALL)	Is the SQL verb CALL allowed?
35 (RSR_ALLOW_SAVE_DATA)	Is Save Data command allowed?
36 (RSR_SAVE_DATA_TABLE_SPACE_NAME)	The default table space name for the Save Data command.
37 (RSR_SAVE_DATA_TABLE_SPACE_NAME_OVERRIDE)	Can the default table space name for the Save Data command be overridden by the user?
38 (RSR_ALLOW_BIND_PACKAGE)	Allow binding of packages?
39 (RSR_DEF_COLLECTION)	The default collection name for binding packages.

Value	Meaning
40 (RSR_DEF_COLLECTION_OVERRIDE)	Can the default collection name for binding packages be overridden by the user?
41 (RSR_DEF_ISOLATION_LEVEL)	The default isolation level for binding packages.
42 (RSR_DEF_ISOLATION_LEVEL_OVERRIDE)	Can the default isolation level for binding packages be overridden by the user?
43 (RSR_ALLOW_TABLE_EDIT)	Allow use of the table editor?
44 (RSR_ALLOW_EXPORT)	Is exporting of data allowed?
45 (RSR_ALLOW_SAVED_QUERIES_ONLY)	Is the user allowed to run only saved queries?
46 (RSR_ALLOW_DROP_PACKAGE)	Allow dropping of packages?
47 (RSR_QUERY_ISOLATION_LEVEL)	The isolation level to use when running queries.
48 (RSR_ACCOUNT_STRING)	The string specifying account information to pass when connecting to the database server.
49 (RSR_ACCOUNT_OVERRIDE)	Can the string specifying account information to pass when connecting to the database server be overridden by the user?

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()` to get additional error information.

B.57 GetResourceLimitEx()

`short GetResourceLimitEx(short Resource)`

Description

This function gets the requested resource limit. You must call `InitializeServer()` prior to calling this function, since resource limits are handled on a per-server basis. After a call to this function, query the `ResourceLimit` property for the result.

Parameters

Table 81 shows the parameters for this API.

Table 81. GetResourceLimitEx parameters

Name	Description
Resource	The resource values are the same as those for the <code>GetResourceLimit()</code> call. This values are shown in Table 80
Value	A pointer to a long in which the result is stored. The result is the value of the requested resource limit. For boolean values, the result is non-zero for true, zero for false. For <code>RSR_SAVE_DATA_TABLE_SPACE_NAME</code> , <code>RSR_DEF_COLLECTION</code> , and <code>RSR_ACCOUNT_STRING</code> , -1 is returned and the <code>ResourceLimit</code> property can be interrogated for the returned string value.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()` to get additional error information.

B.58 GetRowCount()

`long GetRowCount(long QueryID)`

Description

This function returns the number of rows currently in QMF for Windows's internal buffer. This may be greater than the number of rows retrieved with

FetchNextRow() or FetchNextRows(), since QMF for Windows buffers data received from the database.

This function returns the number of rows already retrieved from the database. If you want to retrieve the total number of rows in the result set, you can:

- Call CompleteQuery() and fetch all the rows using FetchNextRow() or FetchNextRows().
- Specify FetchAllRows = TRUE when you call Open().

Parameters

Table 82 shows the parameters for this API.

Table 82. GetRowCount parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().

Return value

The number of rows if successful (0 if no rows have been retrieved), or -1 if unsuccessful. If -1, you can call GetLastErrorString() or GetLastErrorType() to get additional error information.

B.59 GetServerList()

short GetServerList(VARIANT* List)

Description

This function returns an array containing the names of the database servers defined in QMF for Windows's Server Definition File (SDF). You must define a database server in the SDF file if you want to access it using the QMF for Windows API.

Parameters

Table 83 shows the parameters for this API.

Table 83. *GetServerList* parameters

Name	Description
List	A pointer to a VARIANT in which the result is stored. The result is an array of strings (variant type VT_ARRAY VT_BSTR), with each string containing the name of one database server. If you have not defined any database servers, the result is empty (variant type VT_EMPTY). You must properly initialize the VARIANT before calling this function. Visual Basic does this automatically. Visual C++ programmers must call VariantInit().

Note

Due to a bug in Microsoft Excel 7.0 and Microsoft Access 7.0 (and possibly other 32-bit Microsoft products that use Visual Basic for Applications), string data in Variant variables received from QMF for Windows may not be translated from Unicode (used by OLE) to ANSI (used by VBA). When this occurs, only the first character of the string is displayed. To remedy this problem, set the variable equal to an empty string before you call the QMF for Windows function that uses the variable.

Return value

Zero if successful, nonzero if unsuccessful. If you have not defined any database servers, the return value is zero. If the return value is nonzero, you can call GetLastErrorString() or GetLastErrorType() to get additional error information.

B.60 GetServerListEx()

short GetServerListEx(short Index)

Description

This function retrieves the name of the server referenced by the Index parameter. After calling this function, the Value property can be interrogated for the returned value.

Parameters

Table 84 shows the parameters for this API.

Table 84. *GetServerListEx* parameters

Name	Description
Index	An index into the list of servers.

Return value

Zero if successful, `RS_ERROR_OUTOFRANGE` when the index is greater than the number of servers available, nonzero if unsuccessful. If you have not defined any database servers, the return value is `RS_ERROR_OUTOFRANGE`. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()` to get additional error information.

B.61 `GetStoredProcedureResultSets()`

`short GetStoredProcedureResultSets(long QueryID, VARIANT* ResultSets)`

Description

This function retrieves the query IDs for the result sets returned by the stored procedure executed with the original `QueryID`. Each query ID returned can be used with `FetchNextRow()` or `FetchNextRows()` to retrieve the result set rows, and `Close()` when the end of each result set is reached.

Parameters

Table 85 shows the parameters for this API.

Table 85. *GetStoredProceduresResultSets* parameters

Name	Description
QueryID	The ID of the original query as returned from <code>InitializeQuery()</code> .
ResultSets	A pointer to a <code>VARIANT</code> in which the query IDs for the result sets are stored. The result is an array of long integers (variant type <code>VT_ARRAY VT_I4</code>), with each integer being the query ID for the corresponding result sets. If the stored procedure did not return any result sets, the result is empty (variant type <code>VT_EMPTY</code>). You must properly initialize the <code>VARIANT</code> before calling this function. Visual Basic does this automatically. Visual C++ programmers must call <code>VariantInit()</code> .

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()` to get additional error information.

B.62 GetVariables()

short `GetVariables(long QueryID, VARIANT* Variables)`

Description

This function returns an array of the names of the all of the variables in the query's SQL text. You must assign values to these variables by calling `SetVariable()` prior to running the query using either `Open()` or `Execute()`.

Parameters

Table 86 shows the parameters for this API.

Table 86. *GetVariables* parameters

Name	Description
QueryID	The ID of the query as returned from <code>InitializeQuery()</code> .
Variables	A pointer to a VARIANT in which the result is stored. The result is an array of strings (variant type <code>VT_ARRAY VT_BSTR</code>), with each string containing the name of one variable. If there are no variables in the SQL statement, the result is empty (variant type <code>VT_EMPTY</code>). You must properly initialize the VARIANT before calling this function. Visual Basic does this automatically. Visual C++ programmers must call <code>VariantInit()</code> .

Note

Due to a bug in Microsoft Excel 7.0 and Microsoft Access 7.0 (and possibly other 32-bit Microsoft products that use Visual Basic for Applications), string data in Variant variables received from QMF for Windows may not be translated from Unicode (used by OLE) to ANSI (used by VBA). When this occurs, only the first character of the string is displayed. To remedy this problem, set the variable equal to an empty string before you call the QMF for Windows function that uses the variable.

Return value

Zero if successful, nonzero if unsuccessful. If there are no variables in the SQL statement, the return value is `RS_ERROR_NO_DATA` (-1). If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()` to get additional error information.

B.63 GetVariablesEx()

short GetVariablesEx(long QueryID, short Index)

Description

This function returns the name of the variable in the query's SQL text referenced by the Index parameter. After calling this function, the Value property can be interrogated for the returned value. You must assign values to this variable (and all others in the SQL text) by calling SetVariable() prior to running the query using either Open() or Execute().

Parameters

Table 87 shows the parameters for this API.

Table 87. GetVariablesEx parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().
Index	An index into the internal list of variables. Query the Value property for the string that corresponds with the index passed in. If there are no variables in the SQL statement, the function will return RS_ERROR_NO_DATA.

Return value

Zero if successful, nonzero if unsuccessful. If there are no variables in the SQL statement, the return value is RS_ERROR_NO_DATA (-1). If the return value is nonzero, you can call GetLastErrorString() or GetLastErrorType() to get additional error information.

B.64 InitializeProc()

long InitializeProc(short SourceType, BSTR Source)

Description

This function sets the text that you want to use in a procedure. You can pass the text as a parameter to this function, read it from a text file, or obtain it from an existing procedure.

Parameters

Table 88 shows the parameters for this API.

Table 88. InitializeProc parameters

Name	Description
SourceType	Specifies the source for the procedure text.
Source	A string containing the text, the owner and name (Owner.Name) of the procedure, or the name of a file containing the procedure text.

Table 89 shows the valid values for parameter SourceType.

Table 89. Valid values for the parameter SourceType

Value	Meaning
0 (RSS_STRING)	The text is contained in the Source parameter.
2 (RSS_FILE)	The text is contained in the text file whose name is specified by the Source parameter.
3 (RSS_QMFPROC)	The text is contained in the procedure whose owner and name are specified by the Source parameter

Return value

If successful, the ID of the procedure (ProcID). If unsuccessful, -1. You must use this value in all interface calls that require the ProcID parameter.

B.65 InitializeQuery()

long InitializeQuery(short SourceType, BSTR Source)

Description

This function sets the SQL text that you want to use in a query. You can pass the SQL text as a parameter to this function, read it from a text file, or obtain it from an existing query. Call Close() when you are finished with the query.

Parameters

Table 90 shows the parameters for this API.

Table 90. InitializeQuery parameters

Name	Description
SourceType	Specifies the source for the SQL statement text

Name	Description
Source	A string containing the SQL text, the owner and name (Owner.Name) of the query, or the name of a file containing SQL text.

Table 91 shows the valid values for parameter SourceType.

Table 91. Valid values for the parameter sourcetype

Value	Meaning
0 (RSS_STRING)	The SQL text is contained in the Source parameter.
1 (RSS_QMFQUERY)	The SQL text is contained in the query whose owner and name are specified by the Source parameter.
2 (RSS_FILE)	The SQL text is contained in the text file whose name is specified by the Source parameter.

Return value

If successful, the ID of the query. If unsuccessful, -1. You must use this value in all interface calls that require the QueryID parameter.

B.66 InitializeServer()

short InitializeServer(BSTR ServerName, BSTR UserID, BSTR Password, BOOL ForceDialog, [VARIANT Account], [VARIANT SuppressDialog])

Description

This function initializes a connection to a database server. You must call this function prior to calling any other function in the QMF for Windows API. You can call this function multiple times. However, if you call this function and do not end by calling Commit() or Rollback() an implicit rollback results.

Parameters

Table 92 shows the parameters for this API.

Table 92. InitializeServer parameters

Name	Description
ServerName	A string containing the name of the database server that you want to use. This name must match one of the names defined in QMF for Windows's Server Definition File. Call GetServerList() to retrieve a list of valid servers.

Name	Description
UserID	A string containing the User ID you want to use. If UserID is NULL or an empty string, QMF for Windows will attempt to use the UserID from the most recent query, if available. Otherwise, QMF for Windows will display the User Information Dialog box to obtain a User ID and password.
Password	A string containing the password for the specified user ID. If Password is NULL or an empty string, QMF for Windows will try to use a memorized password if available (requires Windows For Workgroups). If no password is available, QMF for Windows will display the User Information dialog box to obtain a password.
ForceDialog	Nonzero indicates that QMF for Windows will display the User Information dialog box regardless of whether a UserID and Password were specified. This gives the user a chance to change the information before it is used. Zero indicates that QMF for Windows should display the User Information dialog box only when necessary.
Account	Optionally, a string specifying accounting information to pass to the server when connecting. The server may use this information in a job accounting system.
SuppressDialog	Nonzero indicates that QMF for Windows will never display the User Information dialog box, even if a user ID and password have not been specified. In this case, an error is returned indicating that no user ID and password were specified. This option is useful when executing in an environment where no user is present to respond to the User Information dialog box, for example, on a web server.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

Related topics

`SetParent()`

B.67 InitializeStaticQuery()

`long InitializeStaticQuery(BSTR CollectionName, BSTR PackageName, BSTR ConsistencyToken, short SectionNumber)`

Description

This function specifies the section of a package that you want to run as a static query.

Parameters

Table 93 shows the parameters for this API.

Table 93. *InitializeStaticQuery* parameters

Name	Description
CollectionName	The name of a previously-bound collection.
PackageName	The name of a previously-bound package.
ConsistencyToken	The token used by the above named collection and package.
SectionNumber	The section number of the statement within the collection and package you want to run.

Return value

If successful, the ID of the query. If unsuccessful, -1. You must use this value in all interface calls that require the QueryID parameter.

B.68 IsStatic()

BOOL IsStatic(long QueryID)

Description

This function determines whether or not the specified query ID refers to a static query or a dynamic query.

Parameters

Table 94 shows the parameters for this API.

Table 94. *IsStatic* parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery() or InitializeStaticQuery().

Return value

Returns nonzero if successful and QueryID refers to a static query, zero otherwise.

B.69 Open()

short Open(long QueryID, long RowLimit, BOOL FetchAllRows)

Description

Use this function to run a query that uses the SELECT verb, by opening a cursor in the database for the query. Use FetchNextRow() or FetchNextRows() to retrieve the data for the query, and call Close() when you are done. If QMF for Windows is configured to automatically fetch all rows (see RSR_AUTOFETCHALLROWS in the description for GetResourceLimit()) or the FetchAllRows parameter is nonzero, QMF for Windows fetches all rows of the result set into its internal buffer before returning from this call.

Note

The name of this function conflicts with the Microsoft Access 2.0 keyword Execute. If you are using MS Access 2.0, place square brackets [] around the function name.

Note

Use this function only in statements that contain the SQL verb SELECT. For statements containing any other verb, for example, SET, call Execute() instead. To determine the verb used by a query, call GetQueryVerb().

Parameters

Table 95 shows the parameters for this API.

Table 95. Open parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().
RowLimit	A number indicating the maximum number of rows to retrieve from the database. Zero indicates that no limit is enforced except for the row limit established by the QMF for Windows Administrator program.
FetchAllRows	A boolean value that indicates whether or not all rows in the result set are automatically fetched into QMF for Windows's internal buffer. If nonzero, all rows are automatically fetched, closing the cursor and freeing the database for use by other users. This is the same as calling CompleteQuery().

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

B.70 Prepare()

short `Prepare(long QueryID)`

Description

This function prepares the query specified by `QueryID`. The statement is examined by the database server, checking for object existence, required authorizations, etc. If the query is a `SELECT` statement, information about the columns returned by the statement are available after completing `Prepare()`.

Parameters

Table 96 shows the parameters for this API.

Table 96. Prepare parameters

Name	Description
QueryID	The ID of the query as returned from <code>InitializeQuery()</code> .

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

Related topics

`Execute()`, `Open()`

B.71 PrintReport()

short `PrintReport(long QueryID, short SourceType, BSTR Source, BSTR OutputFileName, short PageLength, short PageWidth, BOOL IncludeDateTime, BOOL IncludePageNumbers, [VARIANT Format], [VARIANT UseFormPageSetup])`

Description

`PrintReport()` is a synonym for the `ExportReport()` function.

B.72 ReinitializeServer()

short ReinitializeServer()

Description

This function reinitializes the connection to a database server. Normally, you only need to call this function if one of the other QMF for Windows API functions returns an error. Calling this function results in an implicit rollback, which closes any open cursors and invalidates all outstanding query IDs.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastErrorSQLCode(), GetLastErrorSQLError(), or GetLastErrorSQLState() to get additional error information.

B.73 Rollback()

short Rollback()

Description

This function cancels any changes made in the current unit of work, ends the current unit of work, closes any open cursors, and invalidates all outstanding query IDs.

Note

The name of this function conflicts with the Microsoft Access 2.0 keyword Execute. If you are using MS Access 2.0, place square brackets [] around the function name.

Note

The rollback only affects SQL changes that were run by calling Open() or Execute(). Rollback does not affect changes made by other QMF for Windows API functions, such as FastSaveData(), SaveData(), or DeleteQMFObject().

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastErrorSQLCode(), GetLastErrorSQLError(), or GetLastErrorSQLState() to get additional error information.

Related topics

Commit()

B.74 RunProc()

short RunProc(long ProcID)

Description

This function runs the specified procedure. The procedure runs to completion or until an error occurs. You cannot access any of the results of the procedure (for example, data from a query that was run) through this programming interface. However, any files exported or data saved by the procedure are available after the run.

Parameters

Table 97 shows the parameters for this API.

Table 97. RunProc parameters

Name	Description
ProcID	The ID of the procedure as returned from InitializeProc().

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString(), GetLastErrorType(), GetLastSQLCode(), GetLastSQLError(), or GetLastSQLState() to get additional error information.

B.75 SaveData()

short SaveData(long QueryID, long FirstRow, long FirstCol, long LastRow, long LastCol, BOOL Replace, BSTR TableName, BSTR TableSpaceName, BSTR ServerName, BSTR UserID, BSTR Password, BOOL ForceDialog, [VARIANT Account], [VARIANT Comment], [VARIANT CommitScope])

Description

This function saves the specified range of rows and columns into the specified table in the specified table space. You must call CompleteQuery() prior to calling this function if you have not retrieved row data for all of the rows you want to save in the table. If you try to save rows that have not been retrieved from the database, the save will fail. If the table already exists, the new data must have the same number and types of columns as the existing table.

This function operates in a separate unit of work than other API functions and its results are automatically committed. Calling Commit() or Rollback() will have no effect on changes you make using this function.

Parameters

Table 98 shows the parameters for this API.

Table 98. SaveData parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().
FirstRow	The first row you want to include in the save. The value of a first row in a result set is 0.
FirstCol	The first column you want to include in the save. The value of the first column in a result set is 0.
LastRow	The last row you want to include in the save, or 1 if all rows are included. The value of the last row in a result set is one less than the total number of rows.
LastCol	The last column that you want to include in the save, or -1 if all columns are included. The value of the last column in a result set is one less than the total number of columns.
Replace	Nonzero indicates that the specified data will replace any existing data in the table. Zero indicates that the specified data will be appended to any existing data in the table.
TableName	The name of the table in which the data will be stored. If the table doesn't exist, it is created.
TableSpaceName	The name of the table space in which the table exists or will be created. If TableSpaceName is NULL or an empty string, the default table space is used. If you have configured QMF for Windows to always use the default table space (see RSR_SDDIFFERENTTS in the description for GetResourceLimit()), this parameter is ignored.
ServerName	The name of the database server in which the table is stored. If ServerName is NULL or an empty string, the server you specify in the call to InitializeServer() is used, and UserID, Password, ForceDialog, and Account are ignored.
UserID	If you specified a different server in ServerName, UserID is the user ID used for that server. If you do not specify a user ID, QMF for Windows will use the one last specified for this server if available, or will display a dialog box if none is available. This parameter is ignored if ServerName is NULL or an empty string.

Name	Description
Password	If you specified a different server in ServerName, Password is the password used for that server. If you do not specify a password, QMF for Windows will use the one last specified for this server if available, or will display a dialog box if none is available. This parameter is ignored if ServerName is NULL or an empty string.
ForceDialog	If you specified a different server in ServerName, nonzero forces QMF for Windows to display a dialog box prompting for logon information, even if a user ID and password were specified or are otherwise available. Zero indicates that QMF for Windows will display this dialog box only if necessary. This parameter is ignored if ServerName is NULL or an empty string.
Account	If you specified a different server in ServerName, optionally, a string specifying accounting information to pass to that server when connecting. The server may use this information in a job accounting system. This parameter is ignored if ServerName is NULL or an empty string.
Comment	Optionally, a string that specifies a comment for the table in which the data is saved.
CommitScope	Optionally, how many rows to insert into the table at a time before committing the unit of work. Specifying zero indicates that all of the rows should be inserted before committing. Specifying 10 (for example), indicates that a commit should be performed after every ten rows are inserted.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information. If the result set is empty or no rows are retrieved from the database, nonzero is returned unless `FirstRow = 0`, and `LastRow = -1`. In this case, zero is returned and an empty table is created.

B.76 SaveQMFPProc()

short SaveQMFPProc(BSTR OwnerAndName, BSTR Text, BSTR Comment, BOOL Replace, BOOL Share)

Description

This function saves a procedure at a database server.

Parameters

Table 99 shows the parameters for this API.

Table 99. *SaveQMFPProc* parameters

Name	Description
OwnerAndName	A string containing the owner and name, separated by a period, of the procedure you want to save. For example, John.Proc2.
Text	A string containing the text that you want to save in the procedure.
Comment	A string containing any comment you want to save with the procedure. If there is no comment, pass this parameter as either an empty string or NULL.
Replace	Nonzero replaces an existing procedure with the same name. Zero aborts the operation if there is an existing procedure with the same name.
Share	Nonzero shares the procedure with other users. Zero does not share the procedure with other users.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

B.77 `SaveQMFQuery()`

short `SaveQMFQuery(BSTR OwnerAndName, BSTR Text, BSTR Comment, BOOL Replace, BOOL Share)`

Description

This function saves a query at a database server.

Parameters

Table 100 shows the parameters for this API.

Table 100. *SaveQMFQuery* parameters

Name	Description
OwnerAndName	A string containing the owner and name, separated by a period, of the query you want to save. For example, John.Query2.
Text	A string containing the text that you want to save in the query.

Name	Description
Comment	A string containing any comment you want to save with the query. If there is no comment, pass this parameter as either an empty string or NULL.
Replace	Nonzero replaces an existing query with the same name. Zero aborts the operation if there is an existing query with the same name.
Share	Nonzero shares the query with other users. Zero does not share the query with other users.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

B.78 SetBindOption()

short SetBindOption(BSTR CollectionName, BSTR PackageName, short Option, short Value)

Description

This function sets options for the collection and package prior to calling `EndBind()`.

Parameters

Table 101 shows the parameters for this API.

Table 101. *SetBindOption* parameters

Name	Description
CollectionName	The collection ID of the package for which you want to set the option.
PackageName	The name of the package for which you want to set the option.
Option	One of the options listed below.
Value	One of the values listed below for the specified option.

The meanings and values for the various options are shown in Table 102.

Table 102. Meanings and Values for the options

Option	Meaning	Values
DDM_PKGRPLOPT(0x211C)	Flag specifying whether or not to replace an existing package with the same collection ID and name.	DDM_PKGRPLALW (0x241F) Yes DDM_PKGRPLNA- (0x2420) No
DDM_STTDECDEL(0x2121)	The delimiter used for the decimal point in SQL statements in the package.	DDM_DECDELPRD (0x243C) Period DDM_DECDELCMA (0x243D) Comma
DDM_STTSTRDEL(0x2120)	The delimiter used for string values in SQL statements in the package.	DDM_STRDELAP (0x2426) Apostrophe DDM_STRDELDDQ (0x2427) Double Quote
DDM_PKGISOLVL(0x2124)	The isolation level for the package.	DDM_ISOLVLALL (0x2443) All DDM_ISOLVLCHG (0x2441) Change DDM_ISOLVLCS (0x2442) Cursor Stability DDM_ISOLVLNC (0x2445) No Commit DDM_ISOLVLR (0x2444) Repeatable Read
DDM_PKGATHOPT(0x211E)	Flag specifying whether or not to keep existing authorizations on the package.	DDM_PKGATHKP (0x2425) Keep DDM_PKGATHRVK (0x2424) Revoke
DDM_QRYBLKCTL(0x2132)	The method to use when fetching rows of data for queries in the package.	DDM_FIXROWPRC (0x2418) Row at a time DDM_LMTBLKPRC (0x2417) Block at a time

Option	Meaning	Values
DDM_RDBRLSOPT(0x2129)	When to release database resources acquired when running the package.	DDM_RDBRLSCMM (0x2438) Commit DDM_RDBRLSCNV (0x2439) Conversation deallocation
DDM_STTDATEFMT(0x2122)	Format for retrieved date values.	DDM_ISODATEFMT (0x2429) ISO DDM_USDATEFMT (0x242A) US DDM_EURDATEFMT (0x242B) European DDM_JISDATEFMT(0x242C) Japanese Industrial Standard
DDM_STTTIMEFMT(0x2123)	Format for retrieved time values.	DDM_ISOTIMEFMT (0x242E) ISO DDM_USATIMEFMT (0x242F) US DDM_EURTIMEFMT (0x2430) European DDM_JISTIMEFMT(0x2431) Japanese Industrial Standard

Return value

Zero if successful, nonzero is unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()` to get additional error information.

B.79 SetBindOwner()

short SetBindOwner(BSTR CollectionName, BSTR PackageName, BSTR OwnerID)

Description

This function enables you to specify an owner different from your user ID for the package you are binding. This may be necessary if your user ID does not have the required authorizations to bind the package, but the specified owner does.

Parameters

Table 103 shows the parameters for this API.

Table 103. *SetBindOwner* parameters

Name	Description
CollectionName	The collection ID of the package for which you want to specify the owner.
PackageName	The name of the package for which you want to specify the owner.
OwnerID	The desired owner ID for the package you are binding.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

B.80 SetBusyWindowButton()

```
void SetBusyWindowButton(BSTR Text)
```

Description

This function specifies the text displayed on the busy window's Cancel button.

Parameters

Table 104 shows the parameters for this API.

Table 104. *SetBusyWindowButton* parameters

Name	Description
Text	A string that specifies the text displayed on the busy window's Cancel button. The default value is "Cancel". If you specify an empty string the button is hidden. Regardless of the text you specify, the button always cancels, or closes the window.

Return value

None.

Related topics

`SetBusyWindowMessage()`, `SetBusyWindowMode()`, `SetBusyWindowTitle()`, `ShowBusyWindow()`

B.81 SetBusyWindowMessage()

void SetBusyWindowMessage(BSTR Message)

Description

This function specifies the text displayed in the busy window's message area.

Parameters

Table 105 shows the parameters for this API.

Table 105. SetBusyWindowMessage parameters

Name	Description
Message	A string that specifies the text displayed in the busy window's message area.

Return value

None.

Related topics

SetBusyWindowButton(), SetBusyWindowMode(), SetBusyWindowTitle(), ShowBusyWindow()

B.82 SetBusyWindowMode()

short SetBusyWindowMode(short Mode)

Description

This function determines whether or not QMF for Windows displays the busy window. The busy window is useful to provide feedback to the user and to enable the user to cancel a pending database action. Your changes take effect the next time QMF for Windows performs an operation that causes the busy window to display or hide.

Parameters

Table 106 shows the parameters for this API.

Table 106. SetBusyWindowMode parameters

Name	Description
Mode	Specifies when QMF for Windows displays the busy window.

Table 107 shows the valid values for parameter Mode.

Table 107. Valid values for the parameter mode

Value	Meaning
0 (RSM_NEVER)	The window does not display. This is the default.
1 (RSM_WHENBUSY)	The window displays when QMF for Windows is busy communicating with the database. QMF for Windows automatically displays this window as appropriate.
2 (RSM_CLIENTCONTROLLED)	The window displays after you call ShowBusyWindow(TRUE), and until you call ShowBusyWindow(FALSE). The client determines when the window displays.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString() or GetLastErrorType() to get additional error information.

Related topics

SetBusyWindowButton(), SetBusyWindowMessage(), SetBusyWindowTitle(), SetParent(), ShowBusyWindow()

B.83 SetBusyWindowTitle()

void SetBusyWindowTitle(BSTR Title)

Description

This function specifies the text displayed in the busy window's title bar.

Parameters

Table 108 shows the parameters for this API.

Table 108. SetBusyWindowTitle parameters

Name	Description
Title	A string that specifies the text displayed in the busy window's title bar.

Return value

None.

Related topics

SetBusyWindowMode(), SetBusyWindowButton(),
SetBusyWindowMessage(), ShowBusyWindow()

B.84 SetGlobalVariable()

short SetGlobalVariable(BSTR Name, BSTR Value)

Description

This function assigns a value to the specified global variable. This value is available for use in queries, forms, and procedures.

Parameters

Table 109 shows the parameters for this API.

Table 109. SetGlobalVariable parameters

Name	Description
Name	A string that contains the name of the variable you want to set.
Value	A string that contains the value you want to assign to the specified variable.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString() or GetLastErrorType() to get additional error information.

B.85 SetHostVariable()

short SetGlobalVariable(long QueryID, VARIANT Index, VARIANT Value)

Description

This function assigns a value to the specified host variable referenced by the query. The query must be a static query referencing host variables (either stored with the QMF query or created by AddHostVariable()). Index can specify either the numeric index of the host variable, or the name of the host variable.

Parameters

Table 110 shows the parameters for this API.

Table 110. *SetHostVariable* parameters

Name	Description
QueryID	The ID of the query as returned from InitializeStaticQuery().
Index	Either a number (variant type VT_I2) specifying the index of the host variable in the query, or a string (variant type VT_BSTR) specifying the name of the host variable.
Value	The value for the host variable. To specify a null value, the type of the variant should be set to VT_EMPTY

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString() or GetLastErrorType() to get additional error information.

B.86 SetOption()

short SetOption(short Option, VARIANT Value)

Description

This function sets the specified option value in QMF for Windows. For some options, the changes may not take affect until QMF for Windows restarts. Under normal conditions, you do not restart QMF for Windows until you have destroyed all instances of the QMF for Windows API object.

Parameters

Table 111 shows the parameters for this API.

Table 111. *SetOption* parameters

Name	Description
Option	Specifies which option to set.
Value	The value to which to set the option.

Table 112 shows the valid values for parameter Option.

Table 112. *Valid values for the parameter Option*

Value	Meaning
0 (RSO_SERVER_DEFINITION_FILE)	Server definition file name.

Value	Meaning
1 (RSO_CPIC_DLL)	CPI-C Provider DLL file name.
2 (RSO_CPIC_TIMEOUT_WARNING)	CPI-C warning timeout (in seconds). This limit is not used for the QMF for Windows API.
3 (RSO_CPIC_TIMEOUT_CANCEL)	CPI-C cancel timeout (in seconds).
4 (RSO_TCP_TIMEOUT_WARNING)	TCP warning timeout (in seconds). This limit is not used for the QMF for Windows API.
5 (RSO_TCP_TIMEOUT_CANCEL)	TCP cancel timeout (in seconds).
6 (RSO_DISPLAY_NULLS_STRING)	The string to use to display null values.
7 (RSO_ENTER_NULLS_STRING)	The string to use to enter null values.
8 (RSO_ENTER_DEFAULTS_STRING)	The string to use to enter default values.
9 (RSO_TRACE_FILE_1)	Trace file 1 name.
10 (RSO_TRACE_FILE_2)	Trace file 2 name
11 (RSO_TCP_TRACE_LEVEL)	TCP trace level.
12 (RSO_CPIC_TRACE_LEVEL)	CPI-C trace level
13 (RSO_DDM_TRACE_LEVEL)	DDM trace level.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()` to get additional error information.

Related topics

`GetOption()`

B.87 SetParent()

`short SetParent(long ParentWnd)`

Description

This function sets the parent window for dialogs. Normally, when QMF for Windows displays a dialog (in the busy window or the User Information dialog box), it is centered on and modal to QMF for Windows's main window. This

function enables you to force QMF for Windows's dialogs to be centered on and modal to your client application's window.

Parameters

Table 113 shows the parameters for this API.

Table 113. *SetParent* parameters

Name	Description
ParentWnd	The HWND of the new parent window. Specify NULL to use QMF for Windows's main window as the parent.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()` to get additional error information.

Related topics

`ShowBusyWindow()`

B.88 SetProcVariable()

`short SetProcVariable(long ProcID, BSTR Name, BSTR Value)`

Description

This function assigns a value to the specified variable. This value is substituted for the variable prior to running the procedure. If your procedure has one or more variables in it, you must call this function to set the variable values prior to calling `RunProc()`.

Parameters

Table 114 shows the parameters for this API.

Table 114. *SetProcVariable* parameters

Name	Description
ProcID	The ID of the procedure as returned from <code>InitializeProc()</code> .
Name	A string that contains the name of the variable you want to set.
Value	A string that contains the value you want to assign to the specified variable.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()` to get additional error information.

B.89 SetVariable()

`short SetVariable(long QueryID, BSTR Name, BSTR Value)`

Description

This function assigns a value to the specified variable. This value is substituted for the variable prior to running the SQL statement. If your SQL statement has one or more variables in it, you must call this function to set the variable values prior to calling either `Open()` or `Execute()`.

This function has an effect only for dynamic queries. For static queries, you should use the `GetHostVariableNames()`, `AddHostVariable()`, and `SetHostVariable()` functions.

Parameters

Table 115 shows the parameters for this API.

Table 115. SetVariable parameters

Name	Description
QueryID	The ID of the query as returned from <code>InitializeQuery()</code> .
Name	A string that contains the name of the variable you want to set.
Value	A string that contains the value you want to assign to the specified variable

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()` to get additional error information.

B.90 ShowBusyWindow()

`void ShowBusyWindow(BOOL Show)`

Description

This function tells QMF for Windows to either show or hide the busy window. The busy window is useful to provide feedback to the user and enables the

user to cancel a pending database action. This function only works if you called `SetBusyWindowMode()` with a mode of `RSM_CLIENTCONTROLLED`. If you set a parent window by calling `SetParent()`, the busy window will be modal to the specified window.

Parameters

Table 116 shows the parameters for this API.

Table 116. *ShowBusyWindow* parameters

Name	Description
Show	Nonzero shows the busy window; zero hides the busy window. If nonzero, the busy window displays until you call <code>ShowBusyWindow()</code> with Show set to zero.

Return value

None.

B.91 StartBind()

short StartBind(BSTR CollectionName, BSTR PackageName, BSTR ConsistencyToken)

Description

This function begins the process of binding a package in the database.

Parameters

Table 117 shows the parameters for this API.

Table 117. *StartBind* parameters

Name	Description
CollectionName	The desired collection ID for the package.
PackageName	The desired name for the package.
ConsistencyToken	A string 16 characters long containing the hexadecimal representation of an eight-byte token used to ensure consistency between the package bound in the database and an application using that package. When a section is executed within the package, you must provide this same value.

Return value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()`, `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

Related topics

`EndBind()`, `CancelBind()`

Appendix C. QMF for Windows tables and views

This section describes all of the tables and views that are added to the database when QMF for Windows is installed.

C.1 Tables

The following tables in Figure 144 through Figure 153 will be created in a database when QMF for Windows is configured to access it:

C.1.1 Q.OBJ_ACTIVITY_DTL

OBJ_ACTIVITY_DTL		
U1	OBJOWNER	C-Fixed Length(8)
U1	OBJNAME	C-Variable Length(18)
	SSID	C-Fixed Length(4)
U1	DATE	T-Date
U1	TIME	T-Time
	TSOID	C-Fixed Length(8)
	SQID	C-Fixed Length(8)
	ENMRO	C-Fixed Length(1)
	MODE	C-Fixed Length(1)
	ROWCT	N-Signed Integer
	CPUCT	N-Signed Integer
	SUCCESS	C-Fixed Length(1)
	SQLTEXT	C-Variable Length(3500)
	ETIME	T-Time
U1	IXTIME	N-Signed Integer
	BYTECT	N-Signed Integer

Figure 144. Object activity detail table

C.1.2 Q.OBJ_ACTIVITY_SUM

OBJ_ACTIVITY_SUMM		
U1	OBJOWNER	C-Fixed Length(8)
U1	OBJNAME	C-Variable Length(18)
	SSID	C-Fixed Length(4)
	FDATE	T-Date
	FTIME	T-Time
	FTSOID	C-Fixed Length(8)
	FSQLID	C-Fixed Length(8)
	FACTN	R-Fixed Length(1)
	FENVIRO	C-Fixed Length(1)
	FMODE	C-Fixed Length(1)
	LDATE	T-Date
	LTIME	T-Time
	LTSOID	C-Fixed Length(8)
	LSQLID	C-Fixed Length(8)
	LACTN	R-Fixed Length(1)
	LENVIRO	C-Fixed Length(1)
	LMODE	C-Fixed Length(1)
	USE_CT	N-Signed Integer
	CAN_CT	N-Signed Integer
	RUN_CT	N-Signed Integer
	LMDATE	T-Date
	LMTIME	T-Time
	LMTSOID	C-Fixed Length(8)
	LMSQLID	C-Fixed Length(8)
	LMACTN	R-Fixed Length(1)
	LMENVIRO	C-Fixed Length(1)
	LMMODE	C-Fixed Length(1)

Figure 145. Object activity summarization table

C.1.3 Q.OBJECT_DATA

OBJECT_DATA		
U1	OWNER	C-Fixed Length(8)
U1	NAME	C-Variable Length(18)
	TYPE	C-Fixed Length(8)
U1	SEQ	N-Signed Integer
	APPLDATA	R-Large Length

Figure 146. Object data table

C.1.4 Q.OBJECT_DIRECTORY

OBJECT_DIRECTORY		
U1	OWNER	C-Fixed Length(8)
U1	NAME	C-Variable Length(18)
	TYPE	C-Fixed Length(8)
	SUBTYPE	C-Fixed Length(8)
	OBJECTLEVEL	N-Signed Integer
	RESTRICTED	C-Fixed Length(1)
	MODEL	C-Fixed Length(8)
	CREATED	T-Date & Time
	MODIFIED	T-Date & Time
	LAST_USED	T-Date & Time

Figure 147. Object directory table

C.1.5 Q.OBJECT_REMARKS

OBJECT_REMARKS		
U1	OWNER	C-Fixed Length(8)
U1	NAME	C-Variable Length(18)
	TYPE	C-Fixed Length(8)
	REMARKS	C-Variable Length(254)

Figure 148. Object remarks table

C.1.6 Q.RAA_SUBTYPE

RAA_SUBTYPE		
U1	OWNER	C-Fixed Length(8)
U1	NAME	C-Variable Length(18)
	TYPE	C-Fixed Length(8)
	SUBTYPE	C-Fixed Length(8)

Figure 149. RAA subtype table

C.1.7 RDBI.AUTHID_TABLE

AUTHID_TABLE		
I1	PRIMARY_ID	C-Fixed Length(8)
I1	SECONDARY_ID	C-Fixed Length(8)

Figure 150. AuthID table

C.1.8 RDBI.PROFILE_TABLE

PROFILE_TABLE		
U1	CREATOR	C-Fixed Length(8)
	CASE	C-Fixed Length(18)
	DECOPT	C-Fixed Length(18)
	CONFIRM	C-Fixed Length(18)
	WIDTH	C-Fixed Length(18)
	LENGTH	C-Fixed Length(18)
	LANGUAGE	C-Fixed Length(18)
	SPACE	C-Fixed Length(50)
	TRACE	C-Fixed Length(18)
	PRINTER	C-Fixed Length(8)
U1	TRANSLATION	C-Fixed Length(18)
	PFKEYS	C-Variable Length(31)
	SYNONYMS	C-Variable Length(31)
	RESOURCE_GROUP	C-Fixed Length(16)
	MODEL	C-Fixed Length(8)
U1	ENVIRONMENT	C-Fixed Length(8)

Figure 151. Profile table

C.1.9 RDBI.RESERVED

RESERVED		
	DUMMY	N-Signed Integer

Figure 152. Reserved table

C.1.10 RDBI.RESOURCE_TABLE

RESOURCE_TABLE		
I1	RESOURCE_GROUP	C-Fixed Length(16)
I1	RESOURCE_OPTION	C-Fixed Length(16)
	INTVAL	N-Signed Integer
	FLOATVAL	N-Floating Point
	CHARVAL	C-Variable Length(80)

Figure 153. Resource table

C.2 Views

The following views in Figure 154 through Figure 161 will be created in a database when QMF for Windows is configured to access it. Each section shows the data definition language that creates the view and a graphical presentation.

C.2.1 Q.RAA_OBJECT_VIEW

```
CREATE VIEW Q.RAA_OBJECT_VIEW
(
    OWNER, NAME, TYPE, SUBTYPE, OBJECTLEVEL, RESTRICTED, MODEL, REMARKS
)
AS
SELECT A.OWNER, A.NAME, A.TYPE, A.SUBTYPE,
       A.OBJECTLEVEL, A.RESTRICTED, A.MODEL,
       B.REMARKS
FROM Q.OBJECT_DIRECTORY A, Q.OBJECT_REMARKS B
WHERE (A.OWNER = B.OWNER AND A.NAME = B.NAME)
      AND (A.RESTRICTED = 'N'
           OR A.OWNER IN (USER, CURRENT SQLID)
           OR A.OWNER IN (SELECT C.SECONDARY_ID
                          FROM RDBI.USER_AUTHID_VIEW C)
           OR EXISTS (SELECT D.AUTHID
                      FROM RDBI.USER_ADMIN_VIEW D));
```


RAA_OBJECT_VIEW	
OWNER	C-Fixed Length(8)
NAME	C-Variable Length(18)
TYPE	C-Fixed Length(8)
SUBTYPE	C-Fixed Length(8)
OBJECTLEVEL	N-Signed Integer
RESTRICTED	C-Fixed Length(1)
MODEL	C-Fixed Length(8)
REMARKS	C-Variable Length(254)

Figure 154. RAA object view

C.2.2 RDBI.ADMIN_VIEW

```
CREATE VIEW RDBI.ADMIN_VIEW
(
    "AUTHID"
)
AS
SELECT A.GRANTEE
FROM SYSIBM.SYSUSERAUTH A
WHERE A.SYSADMAUTH IN ('G', 'Y');
```

ADMIN_VIEW	
AUTHID	C-Fixed Length(8)

Figure 155. Admin view

C.2.3 RDBI.AUTHID_VIEW

```
CREATE VIEW RDBI.AUTHID_VIEW
(
    PRIMARY_ID,
    SECONDARY_ID
)
AS
SELECT A.PRIMARY_ID, A.SECONDARY_ID
FROM RDBI.AUTHID_TABLE A;
```

AUTHID_VIEW	
PRIMARY_ID	C-Fixed Length(8)
SECONDARY_ID	C-Fixed Length(8)

Figure 156. AuthID View

C.2.4 RDBI.PROFILE_VIEW

```
CREATE VIEW RDBI.PROFILE_VIEW
(
    CREATOR,
    "CASE",
    DECOPT,
    CONFIRM,
    WIDTH,
    LENGTH,
    LANGUAGE,
    SPACE,
    TRACE,
    PRINTER,
    TRANSLATION,
    PFKEYS,
    SYNONYMS,
    RESOURCE_GROUP,
    MODEL,
    ENVIRONMENT
) AS SELECT CREATOR, "CASE", DECOPT, CONFIRM,
           WIDTH, LENGTH, LANGUAGE, SPACE,
           TRACE, PRINTER, TRANSLATION,
           PFKEYS, SYNONYMS, RESOURCE_GROUP,
           MODEL, ENVIRONMENT
FROM RDBI.PROFILE_TABLE;
-- FROM Q.PROFILES;
```

PROFILE_VIEW	
CREATOR	C-Fixed Length(8)
CASE	C-Fixed Length(18)
DECOPT	C-Fixed Length(18)
CONFIRM	C-Fixed Length(18)
WIDTH	C-Fixed Length(18)
LENGTH	C-Fixed Length(18)
LANGUAGE	C-Fixed Length(18)
SPACE	C-Fixed Length(50)
TRACE	C-Fixed Length(18)
PRINTER	C-Fixed Length(8)
TRANSLATION	C-Fixed Length(18)
PFKEYS	C-Variable Length(31)
SYNONYMS	C-Variable Length(31)
RESOURCE_GROUP	C-Fixed Length(16)
MODEL	C-Fixed Length(8)
ENVIRONMENT	C-Fixed Length(8)

Figure 157. Profile view

C.2.5 RDBI.RESOURCE_VIEW

```
CREATE VIEW RDBI.RESOURCE_VIEW
(
    RESOURCE_GROUP,
```

```

RESOURCE_OPTION,
INTVAL,
FLOATVAL,
CHARVAL
) AS SELECT RESOURCE_GROUP, RESOURCE_OPTION,
          INTVAL, FLOATVAL, CHARVAL
FROM RDBI.RESOURCE_TABLE;
-- FROM Q.RESOURCE_VIEW;

```

RESOURCE_VIEW	
RESOURCE_GROUP	C-Fixed Length(16)
RESOURCE_OPTION	C-Fixed Length(16)
INTVAL	N-Signed Integer
FLOATVAL	N-Floating Point
CHARVAL	C-Variable Length(80)

Figure 158. Resource view

C.2.6 RDBI.TABLE_VIEW

```

CREATE VIEW RDBI.TABLE_VIEW
(
  OWNER, NAME, TYPE, SUBTYPE, OBJECTLEVEL, RESTRICTED, MODEL, REMARKS
)
AS
SELECT DISTINCT A.CREATOR, A.NAME, 'TABLE', A.TYPE, 0, 'Y', ' ', A.REMARKS
FROM SYSIBM.SYSTABLES A, SYSIBM.SYSTABAUTH B
WHERE (A.CREATOR = B.TCREATOR AND A.NAME = B.TTNAME)
      AND (B.GRANTEE IN (USER, CURRENT SQLID, 'PUBLIC', 'PUBLIC*')
          OR B.GRANTEE IN (SELECT B.SECONDARY_ID
                          FROM RDBI.USER_AUTHID_VIEW B)
          OR EXISTS (SELECT C.AUTHID
                    FROM RDBI.USER_ADMIN_VIEW C))
      AND (B.GRANTEETYPE IN (' ', 'U', 'G'))
      AND (B.DELETEAUTH IN ('Y', 'G')
          OR B.INSERTAUTH IN ('Y', 'G')
          OR B.SELECTAUTH IN ('Y', 'G')
          OR B.UPDATEAUTH IN ('Y', 'G'));

```

TABLE_VIEW	
OWNER	C-Fixed Length(8)
NAME	C-Variable Length(18)
TYPE	C-Variable Length(5)
SUBTYPE	C-Fixed Length(1)
OBJECTLEVEL	N-Signed Integer
RESTRICTED	C-Variable Length(1)
MODEL	C-Variable Length(1)
REMARKS	C-Variable Length(254)

Figure 159. Table view

C.2.7 RDBI.USER_ADMIN_VIEW

```
CREATE VIEW RDBI.USER_ADMIN_VIEW
(
    "AUTHID"
)
AS
SELECT A."AUTHID"
FROM RDBI.ADMIN_VIEW A
WHERE A."AUTHID" IN (USER, CURRENT SQLID)
OR A."AUTHID" IN (SELECT B.SECONDARY_ID
                  FROM RDBI.USER_AUTHID_VIEW B);
```

USER_ADMIN_VIEW	
AUTHID	C-Fixed Length(8)

Figure 160. User admin view

C.2.8 RDBI.USER_AUTHID_VIEW

```
CREATE VIEW RDBI.USER_AUTHID_VIEW
(
    PRIMARY_ID,
    SECONDARY_ID
)
AS
SELECT A.PRIMARY_ID, A.SECONDARY_ID
FROM RDBI.AUTHID_VIEW A
WHERE A.PRIMARY_ID = USER;
```

USER_AUTHID_VIEW	
PRIMARY_ID	C-Fixed Length(8)
SECONDARY_ID	C-Fixed Length(8)

Figure 161. User AuthID view

C.3 Table and view relationships

QMF for Windows lists are built using the views described above. The key view for queries, forms, and procedures is the Q.RAA_OBJECT_VIEW. The following diagram (Figure 162) shows the relationship and main use of all the views and tables that this view interacts with.

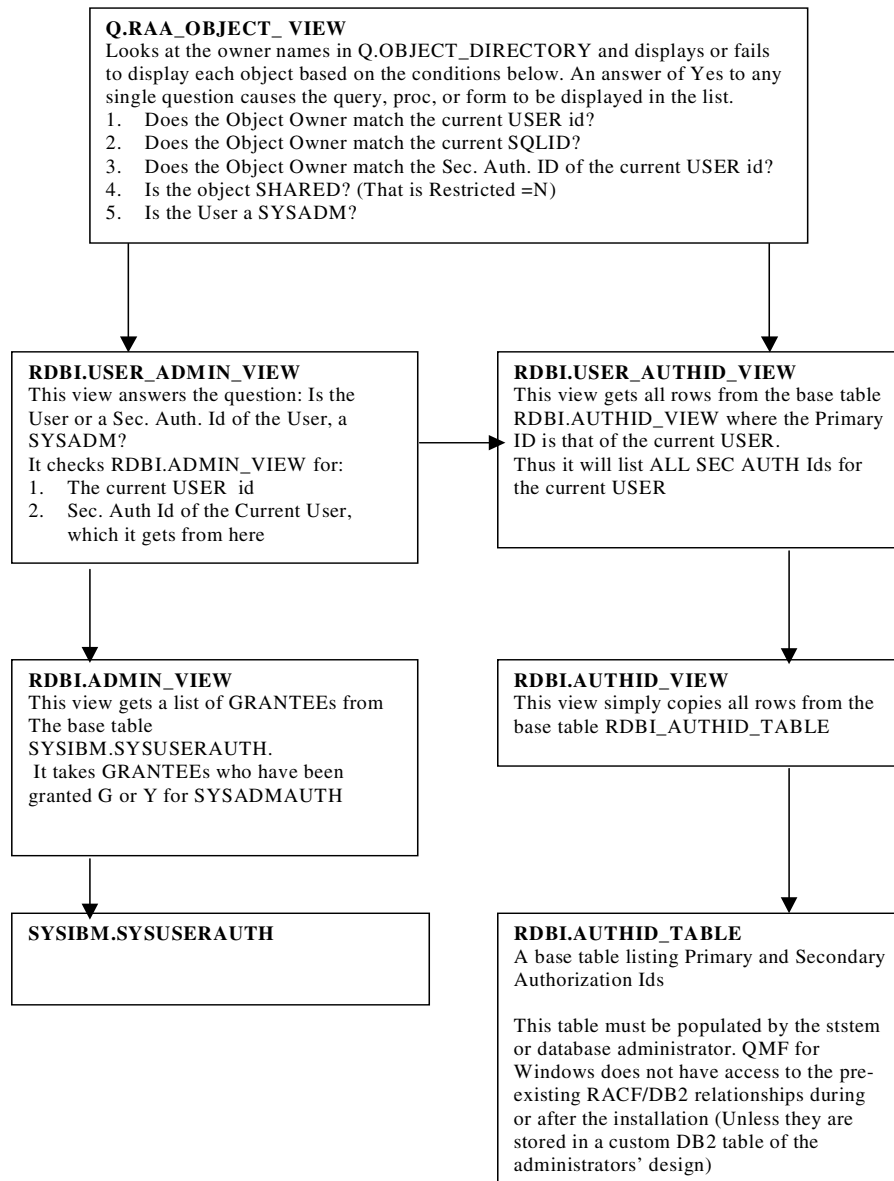


Figure 162. Q.RAA_OBJECT_VIEW relations

For the tables, the view called RDBL.TABLE_VIEW is the key and interacts with other views and tables as shown in Figure 163.

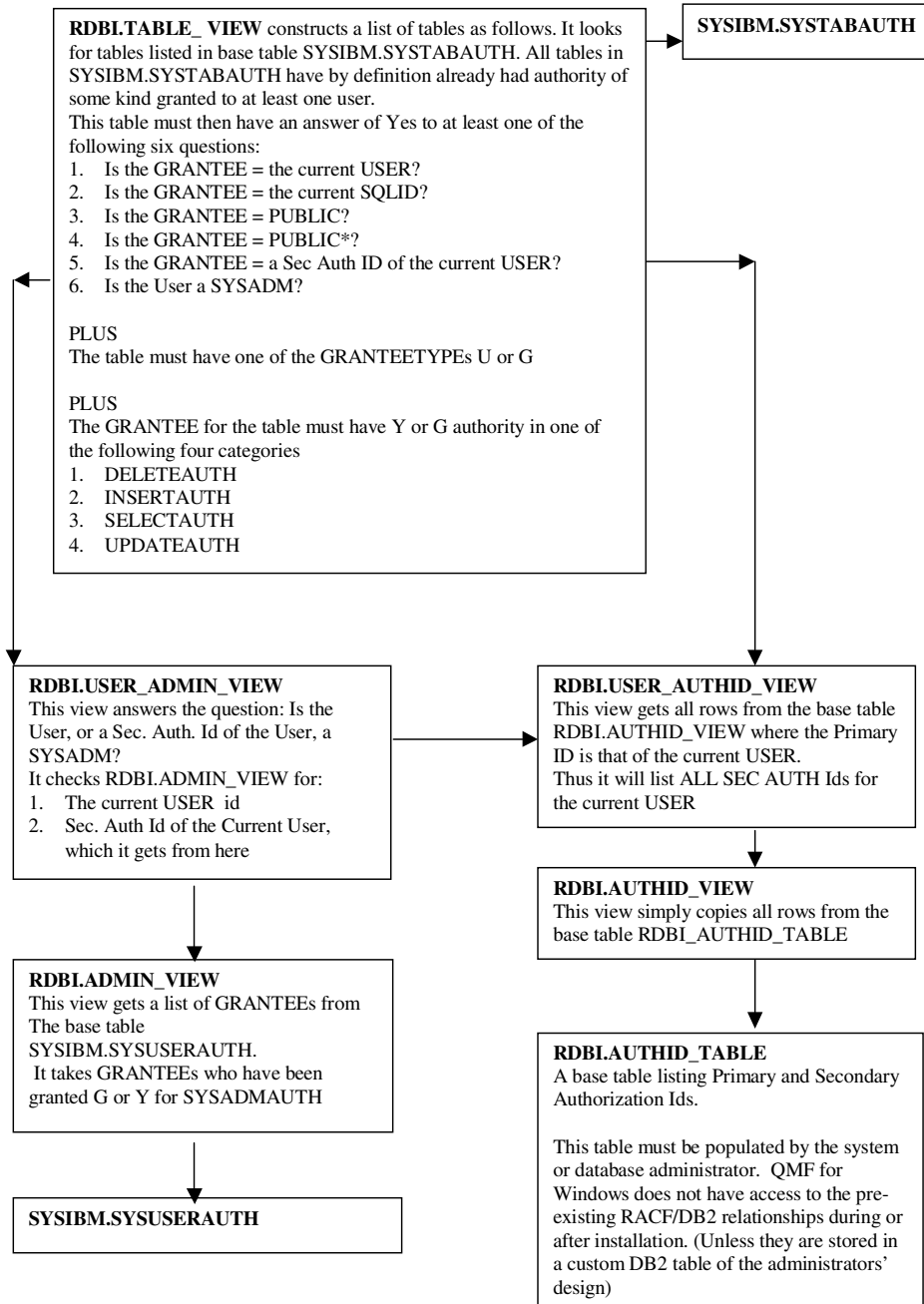


Figure 163. RDBI.TABLE_VIEW relations

Appendix D. Using the additional material

This redbook also contains additional material in CD-ROM or diskette format, and/or Web material. See the appropriate section below for instructions on using or downloading each type of material.

D.1 Using the CD-ROM or diskette

The CD-ROM or diskette that accompanies this redbook contains the following:

<i>Directory Name</i>	<i>Description</i>
Code	Product Trial Installation Code
Developer	Sample Code

D.1.1 System requirements for using the CD-ROM or diskette

The following system configuration is recommended for optimal use of the CD-ROM or diskette.

Hard disk space:	40MB minimum free space
Operating System:	Windows NT
Processor:	Intel Pentium 200 or higher
Memory:	64 MB
Other:	CD-ROM drive

D.1.2 How to use the CD-ROM or diskette

After inserting the CD-ROM into the CD-ROM drive, an installation window will show up automatically. If it does not start automatically, you can start it manually by double-clicking the AUTOSTART file in the CD-ROMs root directory. Alternatively, you can create a subdirectory (folder) on your workstation and copy the contents of the CD-ROM or diskette into this folder.

D.2 Locating the additional material on the Internet

The CD-ROM, diskette, or Web material associated with this redbook is also available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/sg245746>

Alternatively, you can go to the IBM Redbooks Web site at:

<http://www.redbooks.ibm.com/>

Select the **Additional materials** and open the directory that corresponds with the redbook form number.

Appendix E. Special notices

This publication is intended to help database administrators, application developers, and end users to understand the concepts of QMF for Windows. It explains how the needs for an Enterprise Query Environment can be solved using the QMF family of integrated tools. The information in this publication is not intended as the specification of any programming interfaces that are provided by QMF for Windows Version 6.1. See the PUBLICATIONS section of the IBM Programming Announcement for QMF for Windows Version 6.1 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

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, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer

responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

This document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain 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.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX	APPN
AS/400	AT
C/MVS	C/VM
CICS	CT
DATABASE 2	DataJoiner
DB2	Distributed Relational Database Architecture
DRDA	IBM
IMS	Intelligent Miner
MVS/ESA	Netfinity
OpenEdition	OS/2
OS/390	OS/400
QMF	RACF
RS/6000	S/390
SP	SQL/DS
System/370	System/390
VSE/ESA	VTAM
WIN-OS/2	XT
3090	400

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere., The Power To Manage., Anything. Anywhere., TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

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

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

SET and the SET logo are trademarks owned by SET Secure Electronic Transaction LLC.

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

Appendix F. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

F.1 International Technical Support Organization publications

For information on ordering these ITSO publications see “How to get ITSO redbooks” on page 399.

- *Getting Started with Data Warehouse and Business Intelligence*, SG24-5415
- *From Multiplatform Operational Data to Data Warehousing and Business Intelligence*, SG24-5174
- *My Mother Thinks I'm a DBA - Cross-Platform, Multi-Vendor, Distributed Relational Data Replication with IBM DB2 Data Propagator and IBM DataJoiner Made Easy*, SG24-5463
- *Intelligent Miner for Data - Enhance Your Business Intelligence*, SG24-5422
- *Accessing OS/390 OpenEdition MVS from the Internet*, SG24-4721

F.2 Redbooks on CD-ROMs

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at <http://www.redbooks.ibm.com/> for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

How to get ITSO redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** <http://www.redbooks.ibm.com/>

Search for, view, download, or order hardcopy/CD-ROM redbooks from the redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this redbooks site.

Redpieces are redbooks in progress; not all redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

Send orders by e-mail including information from the redbooks fax order form to:

	e-mail address
In United States	usib6fpl@ibmmail.com
Outside North America	Contact information is in the "How to Order" section at this site: http://www.elink.ibm.com/pbl/pbl

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the "How to Order" section at this site: http://www.elink.ibm.com/pbl/pbl

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the "How to Order" section at this site: http://www.elink.ibm.com/pbl/pbl

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and redbooks by accessing the IBM Intranet Web site at <http://w3.itso.ibm.com/> and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at <http://w3.ibm.com/> for redbook, residency, and workshop announcements.

Glossary

A

application programming interface (API). A functional interface supplied by the operating system or a separate orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program.

architecture. The number of processing units in the input, output, and hidden layer of a neural network. The number of units in the input and output layers is calculated from the mining data and input parameters. An intelligent data mining agent calculates the number of hidden layers and the number of processing units in those hidden layers.

attribute. Characteristics or properties that can be controlled, usually to obtain a required appearance. For example, color is an attribute of a line. In object-oriented programming, a data element defined within a class.

D

DATABASE 2 (DB2). An IBM relational database management system.

database table. A table residing in a database.

database view. An alternative representation of data from one or more database tables. A view can include all or some of the columns contained in the database table or tables on which it is defined.

data field. In a database table, the intersection from table description and table column where the corresponding data is entered.

data format. There are different kinds of data formats, for example, database tables, database views, pipes, or flat files.

data table. A data table, regardless of the data format it contains.

data type. There are different kinds of Intelligent Miner data types, for example, discrete numeric, discrete nonnumeric, binary, or continuous.

F

field. A set of one or more related data items grouped for processing. In this document, with regard to database tables and views, *field* is synonymous with *column*.

file. A collection of related data that is stored and retrieved by an assigned name.

file name. (1) A name assigned or declared for a file. (2) The name used by a program to identify a file.

flat file. (1) A one-dimensional or two-dimensional array; a list or table of items. (2) A file that has no hierarchical structure.

formatted information. An arrangement of information into discrete units and structures in a manner that facilitates its access and processing. Contrast with *narrative information*.

function. Any instruction or set of related instructions that perform a specific operation.

I

input data. The metadata of the database table, database view, or flat file containing the data you specified to be mined.

instance. In object-oriented programming, a single, actual occurrence of a particular object. Any level of the object class hierarchy can have instances. An instance can be considered in terms of a copy of the object type frame that is filled in with particular information.

M

metadata. In databases, data that describes data objects.

O

output data. The metadata of the database table, database view, or flat file containing the data being produced or to be produced by a function.

P

pass. One cycle of processing a body of data.

processing unit. A processing unit in a neural network is used to calculate an output by summing all incoming values multiplied by their respective adaptive connection weights.

R

record. A set of one or more related data items grouped for processing. In reference to a database table, *record* is synonymous with *row*.

rule. A clause in the form head<== body. It specifies that the head is true if the body is true.

S

Structured Query Language (SQL). An established set of statements used to manage information stored in a database. By using these statements, users can add, delete, or update information in a table, request information through a query, and display results in a report.

symbolic name. In a programming language, a unique name used to represent an entity such as a field, file, data structure, or label. In the Intelligent Miner you specify symbolic names, for example, for input data, name mappings, or taxonomies.

T

transaction. A set of items or events that are linked by a common key value, for example, the articles (items) bought by a customer (customer number) on a particular date (transaction identifier). In this example, the customer number represents the key value.

transaction ID. The identifier for a transaction, for example, the date of a transaction.

List of abbreviations

ADK	application development toolkit	DUW	distributed unit of work
ANSI	American National Standards Institute	DW	data warehouse
APPC	advanced program to program communication	EIS	executive information system
API	application programming interface	FTP	file transfer protocol
APPN	advanced peer to peer networking	GID	group ID
ASCII	American National Standard Code for Information Interchange	GUI	graphical user interface
BI	Business Intelligence	HTML	Hypertext Markup Language
CAE	client application enabler	HTTP	Hypertext Transfer Protocol
CP	control point	HLQ	high level qualifier
CORBA	Common Object Request Broker Architecture	IBM	International Business Machines Corporation
CPI-C	Common Programming Interface-Communications	IDS	intelligent decision support
DB2	database 2	ISO	International Organization for Standardization
DBA	Database Administrator	I/O	input/output
DBMS	database management system	IM	Intelligent Miner
DCL	data control language	IMS	Information Management System
DDL	data definition language	ISDN	integrated services digital network
DML	data manipulation language	IT	information technology
DR	distributed request	ITSO	International Technical Support Organization
DRDA	distributed relational database architecture	JCL	job control language
		JDBC	java database connectivity
		JDK	java developers kit
		JRE	java runtime environment
		LAN	local area network
		LOB	large object

LU	logical unit	UDP	user datagram protocol
ODBC	Open Database Connectivity	UDT	user-defined type
OEM	original equipment manufacturer	VSAM	Virtual Storage Access Method
OLAP	on-line analytical processing	WAN	wide area network
OLTP	on-line transaction processing		
OSA	open systems adapter		
OSI	open systems interconnection		
POS	Persistent Object Service		
QMF	Query Management Facility		
RACF	resource access control facility		
RAD	rapid application development		
RAM	random access memory		
RDBMS	relational database management system		
ROI	return on investment		
RUW	remote unit of work		
SDF	Server Definition File		
SMP/E	system modification program/enhanced		
SNA	shared network architecture		
SQL	structured query language		
TCP/IP	Transmission Control Protocol/Internet Protocol		
TP	transaction program		
UDB	universal database		
UDF	user-defined function		

Index

A

access
 existing QMF objects 183
 objects stored at a database 183
 objects stored in a file 186
 table 188
access controll 102
access path 243
ACROSS 221
Active Server Page 263
add from list 204
add row condition 208
add sort condition 207
AddDecimalHostVariable 293
AddHostVariable 294
API 126
APPC
 Conversations 51
 Logical Units 50
 LU 6.2 50
 Sessions 51
 Terminology 50
 Transaction Programs 51
application development 125
application requester 26
application server 26
application services 49
ARPANET 27
ASCII 85
ASP 131, 263
assign user 110
asynchronous application 128
AT command 123
authorization ID 85
AVERAGE 222

B

bind
 permissions 105
 static package 115
bind packages 88
bind privileges 88
BindDecimalHostVariable 295
BindHostVariable 296
BindSection 297

BOTTOM 229
BREAK 222
business intelligence 3, 10

C

C++ 172
 considerations 173
 setup 173
CALC 222
Call Level Interface 64
cancel query 189
CancelBind 298
CCSID 85
centralized administration 180
CGI 131, 263
 programming 263
 sample 263
change password 112, 246
ChangePassword 298
check
 resource limits 243
check form 227
ClearList 299
Close 299
collection name 81, 88
column
 add to form 214
 column sequence 225
command line interface 15
command line mode 120
Commit 300
commit 130
Common Gateway Interface 263
Common Object Request Broker Architecture 127
communication protocols 25
compact installation 178
CompleteQuery 301
CONNECT 229
connection timeout 98
control resource consumption 92
CONVERT 229
convert to SQL 210
CopyToClipboard 301
CORBA 127
cost estimation 119
COUNT 222
covert to HTML 194

- CPCT 222
- CPI-C 54
- create
 - empty form 212
 - global variable 281
 - lists 112, 247
 - new objects 197
 - procedure 227
 - prompted query 203
 - QMF objects 81
 - resource limits groups 92
 - sample tables 90
 - schedule 94
 - SQL query 199
 - substitution variable 279
 - table 198
- Cross Tab Report 233
- CSUM 222
- custom installation 178
- customize interface 249
- customize toolbar 250

D

- data 1
 - access 1
 - analysis 1
 - exporting 103
 - manipulation 1
 - retrieval 1
- data access infrastructure 11
- data communication 25
- data exchange protocols 25
- data flow control 49
- data link control 49
- database 1
 - heterogeneous 5
 - navigational 2
 - relational 2
- database connection
 - CLI 76
 - CPI-C 75
 - TCP/IP 74
 - trace 79
 - troubleshooting 79
- database name 73
- DB2 Connect 78
- DB2 DataJoiner 15
- decimal delimiter 82

- decision cycle 9
- decision support 14
- default form 220
- default table space 104
- delete server connection 92
- DeleteQMFObjct 303
- delivery technology 10
- Delphi 154
 - ClearGrid 160
 - delete QMF object 168
 - execute query 169
 - get QMF object info 167
 - get query text 167
 - initialize query 165
 - initialize server 164
 - list queries 162
 - list servers 163
 - refresh query list 162
 - sample 156
 - save on server 171
 - save QMF query 171
 - save to file 172
 - setup 154
- DHCP 47, 48
- Direct Routing 33
- DISPLAY 229
- display report 186, 219
- distributed reporting 5
- DRAW 229
- draw query 202
- DRDA 11, 25
 - distributed request (DR) 63
 - DUW 63
 - level 1 62
 - level 2 63
 - private protocols 63
 - RUW 62
- DSN_STATEMNT_TABLE 119
- DSQAO 284
- DSQCP 284, 285, 286
- DSQDC 284
- DSQEC 284, 286
- DSQQW 284, 287
- dynamic SQL 115, 243

E

- EBCDIC 85
- e-business 253

- edit codes 215
- e-mail 28
- encoding schema 85
- EndBind 303
- Enterprise Query Environment 11
 - requirements 11
- environment
 - multi vendor 15
- ERASE 229
- Execute 304
- ExecuteEx 304
- ExecuteStoredProcedure 305
- ExecuteStoredProcedureEx 307
- EXPORT 229
- Export 308
- export data 192
- ExportForm 311
- ExportReport 311

F

- FastSaveData 313
- fetch limit 98
- FetchNextRow 134, 314
- FetchNextRowEx 316
- FetchNextRows 316
- FetchNextRowsEx 318
- filter objects 185
- find 190
- FIRST 222
- FlushQMFCache 318
- footing 218
- form 211
 - add column 214
 - change 221
 - check 227
 - column sequence 225
 - column width 214
 - convert to HTML 256
 - create default 220
 - default 220
 - edit codes 215
 - HTML heading 256
 - indent 214
 - main window 213
 - save 218, 221
 - usage code 221
- Form calculations 15
- FORWARD 229

FTP 28

G

- gateway 16, 33
- GetColumnCount 134, 319
- GetColumnDataValue 319
- GetColumnHeader 320
- GetColumnHeaderEx 320
- GetColumnHeadings 134, 321
- GetColumnValue 322
- GetColumnValueEx 323
- GetDefaultServerName 324
- GetGlobalVariable 324
- GetHostVariableNames 325
- GetHostVariableTypeNames 325
- GetHostVariableTypes 326
- GetLastErrorString 326
- GetLastErrorType 327
- GetLastSQLCode 328
- GetLastSQLException 329
- GetLastSQLState 330
- GetOption 331
- GetOptionEx 332
- GetProcText 333
- GetProcVariables 333
- GetQMFObjectInfo 334
- GetQMFObjectInfoEx 336
- GetQMFObjectList 132, 338
- GetQMFObjectListEx 339
- GetQMFPProcText 340
- GetQMFPQueryText 341
- GetQueryText 132, 341
- GetQueryVerb 133, 342
- GetResourceLimit 343
- GetResourceLimitEx 348
- GetRowCount 348
- GetServerList 131, 349
- GetServerListEx 350
- GetStoredProcedureResultSets 351
- GetVariables 352
- GetVariablesEx 353
- Global Variables 277
- global variables 281
- Gopher 28
- governing 15
- grant permissions 89
- grid 272
- GROUP 223

grouping 221

H

heading 218
history report 107
host address 29
host name 74
hostname 46, 47
hostnames 48
hosts file 48
HTML
 dynamic reports 262
 edit form 258
 heading text 258
 preview 259
 response 267
 scheduling 260

I

implementation samples 20
IMPORT 229
indent 214
Indirect Routing 33
infrastructure
 solution 13
 typical 11
InitializeProc 353
InitializeQuery 354
InitializeServer 130, 355
InitializeStaticQuery 356
Internet 253
 environment 254
IP addressing 29
IP datagram 32
IP Routing
 algorithm 35
 table 34
IP routing 33
Isolation level 103
IsStatic 357

J

Java Database Connectivity 127
JDBC 127
join 205

L

Large Object 119
LAST 223
linear procedure 120
linear procedures 15
list
 display 185
 refresh 185
 remove from 186
 work with 247
list objects 184
LOB 119, 272
local host 36
local SDF 179
location name 73
logical unit 49
loopback network 30
Lotus 123
 import data 233
 Snap-In 232

M

MAXIMUM 223
maximum rows 98
Microsoft Access
 Access Report List 240
 Snap-In 238
Microsoft Excel
 Import Data 236
 Snap-In 235
middleware 16, 126
migrate from Query Manager 251
MINIMUM 223
modify data 192
multitasking 128
multithreading 128

N

Needs
 IS 10
needs
 end user 12
 power user 12
 travelling user 13
network address 29
networking protocols 11
NFSNET 27

O

object
 create new 197
 working with 187
Object REXX 15
object tracking 107
ODBC 11, 126
OMIT 223
Open 358
owner ID 81

P

packages 81
 authorization 82
 replacing 82
path control 49
PCT 223
Persistent Object Service 127
Personal Portal 273
 application support 274
 Favorite 275
physical control 49
port number 74
POS 127
predictive governing 118
Predictive Governor 118
Prepare 359
preprocessor 115
presentation services 49
PRINT 229
print 192
PrintReport 359
procedure 196
 Clear Grid 136
 commands 229
 create 227
 DataIntoGrid 136
 run 196, 230
 save 230
programming
 CGI 263
 considerations 173
programming concepts 125
prompted Query
 create 203
prompted query 175
 view 209
provider DLL 75

Q

QMF
 APIs 293
 architecture 25
 benefits 271
 CGI programming 263
 CGI sample 264
 components 178
 concepts 181
 customize interface 249
 existing installations 85
 future 272
 High Performance Option 7
 history 6
 HPO/Compiler 7
 HPO/Manager 7
 installation 177
 internet 254
 objects 83, 181
 packages 81
 prerequisites 25
 procedure 196
 Snap-In 231
 tables 84, 379
 views 379
 web publishing 254
QMFWinLibrary_TLB 155
queries
 predefined 12
query 1, 182, 188
 add column 206
 cancel 189
 create 199
 definition 1
 draw 202
 environment 3
 needs 10
 run 188, 201
 save 211
 types 188
 view prompted 189
Query Management
 challenge 9
 characteristics 3
 introduction 1
Query Manager 251
quick start 16

R

- RDB name 73
- reactive governing 118
- ReinitializeServer 360
- remote host 36
- remove from list 186
- report 212
 - column sequence 225
 - display 186, 219
 - footing 218
 - heading 218
 - preview 259
 - scheduling 260
- report distribution 5
- report management 5
- reporting
 - distributed 5
 - needs 10
- reporting environment 3
- reports
 - predefined 12
- resource governor 92
- resource group 87
- Resource Limit Facility 118
- resource limits 85
 - check 243
- result
 - order by 191
 - print 192
 - view 189
- Rocket Personal Portal 273
- Rollback 360
- rollback 130
- router 33
- routing 49
- row condition 208
- RowLimit 133
- RUN 230
- run a query 188
- run procedure 196, 230
- RunProc 361

S

- SAVE 230
- save
 - default form 221
 - form 218
 - query 211

- save procedure 230
- save user defaults 194
- SaveData 361
- SaveQMFPProc 363
- SaveQMFPQuery 133, 364
- schedule 94, 95
 - day range 96
 - number 95
 - status 96
 - time range 96
- schedule service 123
- scheduling 122
- SEND TO 230
- server
 - set 197
- server definition file 179
 - local 179
 - shared 179
- server name 73
- servlet 131
- SET GLOBAL 230
- set server 197
- SetBindOption 365
- SetBindOwner 367
- SetBusyWindowButton 368
- SetBusyWindowMessage 369
- SetBusyWindowMode 369
- SetBusyWindowTitle 370
- SetGlobalVariable 371
- SetHostVariable 371
- SetOption 372
- SetParent 373
- SetProcVariable 374
- SetVariable 375
- shared SDF 179
- SHOW 230
- ShowBusyWindow 375
- SMTP 28
- SNA
 - Layers 49
- Snap-In 231
 - Lotus 123 232
 - Microsoft Excel 235
 - Microsoft Access 238
- solution
 - cost 5
- sort 191
 - ascending 191
 - condition 191

- descending 191
- SQL 2
 - dynamic 243
 - permissions 100
 - static 16, 243
 - verbs 100
- SQL expression 206
- SQLDS 63
- StartBind 376
- static queries 115
- static SQL 16, 243
- STDEV 223
- stored procedures 15
- string delimiter 82
- subnet 30
- substitution variable 202
- Substitution Variables 277
- substitution variables 279
- SUM 223
- symbolic destination name 75
- SYNC_LEVEL 53
- synchronization 128
- synchronous application 128
- SYNCPOINT 53

T

- table 188
 - create new 198
 - join 205
- table editing 103
- TCP/IP 27, 47
 - Application Layer 28
 - architecture 28
 - Internet Layer 29
 - Network Interface Layer 29
 - Transport Layer 28
- TCP/IP Architecture 28
- TCPCT 223
- Telnet 28
- test connection 78
- thin client 13
- thread 128
- timeouts 96
- toolbar
 - customize 250
- totals 221
- TPCT 223
- Trace 79

- transaction services 49
- transmission control 49
- two phase commit 173
- typical installation 178

U

- UDP 28
- unit of work 130
- unused objects 107
- usage code 221
- user interface 5
- user profile 85, 86

V

- variables
 - global 277, 281
 - in registry 277
 - lifetime 277
 - pre-loaded global 283
 - structure 279
 - substitution 277, 279
 - user defined global 281
 - working with 277
- variant 173
- view
 - in Web Browser 259
 - prompted query 209
- view prompted query 189
- view result 189
- view SQL 188
- Visual Basic 135
 - export data 152
 - initialize query 140, 145, 150
 - initialize server 143, 148
 - list directory 144
 - list drives 144
 - list file list 144
 - list queries 139
 - list servers 138, 143, 148
 - sample 136
 - save query 151
 - setup 135

W

- web
 - environment 262
 - publishing 15, 254

static reports 255
Web Warehouse 261
wildcard 185
WINDOWS 230
Windows NT scheduler 122
WinSock 37
wrapper class 173

ITSO redbook evaluation

A DB2 Enterprise Query Environment - Build it With QMF For Windows !
SG24-5746-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at <http://www.redbooks.ibm.com/>
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?

Customer **Business Partner** **Solution Developer** **IBM employee**
 None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

What other redbooks would you like to see published?

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

SG24-5746-00
Printed in the U.S.A.

A DB2 Enterprise Query Environment - Build it With QMF For Windows !

SG24-5746-00

