

Query Management Facility



Getting Started with QMF for Windows

Version 7

Query Management Facility



Getting Started with QMF for Windows

Version 7

Note!

Before using this information and the product it supports, be sure to read the general information under "Appendix. Notices" on page 135.

Fifth Edition (September 2000)

This edition applies to Query Management Facility for Windows, a feature of Version 7 of DB2 Universal Database Server for OS/390 (DB2 UDB for OS/390), 5675-DB2 , and of Query Management Facility, a feature of Version 7 of DATABASE 2 Server for VM and VSE (DB2 for VM and VSE), 5697-F42, and of Query Management Facility for Windows for AS/400, 5697-G24, and of Query Management Facility for Windows for DB2 Workstation Databases, 5697-G22, and of DB2 Warehouse Manager, 5648-D35, and of DB2 Warehouse Manager for AS/400, 5697-G23, and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces and makes obsolete the previous edition, SC26-9582-02. The technical changes for this edition are indicated by a vertical bar to the left of the change. Editorial changes that have no technical significance are not noted.

© **Copyright International Business Machines Corporation 1997, 2000. All rights reserved.**

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

Contents

The QMF Library	vii	Chapter 3. Working with Prompted Queries	15
Chapter 1. Introduction	1	Building simple queries	15
Database servers	1	Opening new prompted queries	15
DB2 family of databases	1	Prompted query action buttons	15
User name vs. technical name	1	Adding tables to prompted queries	16
Setting the server name	1	Running prompted queries	16
Database security	2	Building Complex Queries	16
Logging on.	2	Adding columns to prompted queries	17
Correcting passwords	3	Using sort conditions	17
Changing passwords	3	Adding sort conditions	18
Specifying accounting strings.	4	Using row conditions	18
Governing	4	Adding row conditions	19
Viewing resource limits	4	Using multiple tables in prompted queries	19
Setting your own row limit	5	Creating prompted query join conditions	20
Customizing the toolbar	5	Prompted queries and SQL	20
Adding buttons to the toolbar	5	Viewing SQL for prompted queries	20
Moving buttons on the toolbar	6	Converting prompted queries to SQL	20
Removing buttons from the toolbar	6	Using Substitution Variables in Prompted	
Chapter 2. Working with SQL Queries	7	Queries	21
SQL queries	7	Saving Prompted Queries	21
Creating new SQL queries.	7	Saving prompted queries to files	21
Running SQL queries at a database server	7	Opening saved prompted query files.	21
Switching between the Results view and the		Saving prompted queries at the database	
SQL view	7	server	21
Working with fonts	8	Opening saved prompted queries at the	
Selecting the query display font.	8	database server	22
Multiple queries	8	Printing Prompted Queries	22
Displaying multiple queries simultaneously	8	Previewing prompted queries	22
Drawing queries	9	Chapter 4. Working with Query Results	25
Creating new SQL queries.	9	Sorting and sizing query results	25
Substitution variables in SQL queries	10	Selecting columns and rows.	25
Running SQL queries with substitution		Resizing columns and rows	25
variables	10	Auto fitting columns and rows.	25
Saving and Opening SQL queries	11	Sorting query results	26
Saving SQL queries to files	11	Reordering columns	26
Opening saved SQL query files.	11	Formatting query results.	26
Saving SQL queries at the database server	12	Selecting the query results display font	26
Opening saved SQL queries at the		Formatting numeric query results.	27
database server	12	Converting query results formatting to a	
Printing SQL queries	13	form	27
Previewing a query	13	Grouping and aggregating query results	27
Printing SQL queries	13	Grouping query results	27
		Summarizing query results	27
		Saving query results and formatting	27





Saving query results as a table	28	Opening saved list files	41
Saving query results to files	28		
Printing query results	28	Chapter 8. Working with Job Files	43
Previewing query results	28	Job files	43
Printing query results	28	Creating job files	43
Chapter 5. Working with Reports	29	Running job files	43
Forms	29	Auto fitting columns and rows	43
Understanding forms	29	Sorting query results	43
Producing a report using a form	30	Reordering columns	44
Editing a form	30	Formatting query results	44
Creating a form	30	Selecting the query results display font	44
Step 1: Create a form	30	Formatting numeric query results	44
Step 2: Change the column order	31	Converting query results formatting to a form	45
Step 3: Change the column headings	31	Grouping and aggregating query results	45
Step 4: Change the column format	31	Grouping query results	45
Step 5: Add summary information	31	Summarizing query results	45
Step 6: Add page headers and footers	32	Saving query results and formatting	45
Saving forms	32	Saving query results as a table	45
Saving a form to a file	32	Saving query results to files	45
Opening saved form files	33	Printing query results	46
Saving forms at the database server	33	Previewing query results	46
Opening saved forms at the database server	33	Printing query results	46
Printing reports	34	Chapter 9. Working with static queries	47
Exporting reports	34	Static queries	47
Chapter 6. Working with Procedures	35	Creating static queries	47
Running procedures	35	Replacing substitution variables with host variables	48
Creating a new linear procedure	35	Running a static query	49
Creating a new procedure with logic	35	Chapter 10. Working with the Table Editor	51
Running a procedure at a database server	35	Table editor	51
Saving procedures	36	Searching for rows using the table editor	51
Saving procedures to files	36	Adding a row	52
Opening a saved procedure file	36	Changing a row	52
Saving a procedure to the database server	36	Deleting a row	52
Opening saved procedures at the database server	37	Editing tables from the query results view	53
Printing procedures	37	Deleting a row from the query results view	53
Previewing a procedure	37	Updating columns from the query results view	53
Printing a procedure	38	DB2 Forms	53
Chapter 7. Working with Lists	39	Chapter 11. Distributing Data.	55
Objects	39	Exporting data	55
Listing objects	39	Exporting data to files	55
List window commands	40	Importing data	56
Creating lists	40	Saving data to a database server	56
Adding objects to lists	40	Using the Send To command	57
Removing objects from lists	41	Using the Microsoft Excel Add-In	58
Saving lists to files	41		

Using Sample Applications	58	GetColumnValue()	90
Chapter 12. Using QMF Report Center	59	GetColumnValueEx()	91
Getting Started in QMF Report Center	59	GetDefaultServerName()	91
QMF Report Center Window	60	GetGlobalVariable()	92
Connecting to the Server	61	GetHostVariableNames()	92
Working with Reports and Objects	61	GetHostVariableTypes()	92
Running Reports	62	GetLastErrorString()	93
Working with Folders and Favorites	62	GetLastErrorType()	93
Adding Reports to Favorites	63	GetLastSQLCode()	94
Chapter 13. Using the QMF for Windows		GetLastSQLError()	95
API	65	GetLastSQLState()	96
Controlling QMF for Windows through the		GetOption()	97
API	65	GetOptionEx()	98
Blocking calls	66	GetProcText()	98
Connecting to the database	66	GetProcVariables()	99
API Reference	67	GetQMFObjectInfo()	99
AddDecimalHostVariable()	67	GetQMFObjectInfoEx()	101
AddHostVariable()	67	GetQMFObjectList()	102
BindDecimalHostVariable()	68	GetQMFObjectListEx()	103
BindHostVariable()	69	GetQMFProcText()	104
BindSection()	70	GetQMFQueryText()	105
CancelBind()	71	GetQueryText()	105
ChangePassword()	71	GetQueryVerb()	106
ClearList()	71	GetResourceLimit()	106
Close()	72	GetResourceLimitEx()	110
Commit()	72	GetRowCount()	111
CompleteQuery()	73	GetServerList()	111
CopyToClipboard()	73	GetServerListEx()	112
DeleteQMFObject()	74	GetStoredProcedureResultSets()	112
EndBind()	75	GetVariables()	113
Execute()	75	GetVariablesEx()	114
ExecuteEx()	76	InitializeProc()	114
ExecuteStored Procedure()	76	InitializeQuery()	115
ExecuteStored ProcedureEx()	78	InitializeServer()	116
Export()	79	InitializeStaticQuery()	117
ExportForm()	81	IsStatic()	117
ExportReport()	81	Open()	118
FastSaveData()	83	Prepare()	119
FetchNextRow()	84	PrintReport()	119
FetchNextRowEx()	85	ReinitializeServer()	119
FetchNextRows()	85	Rollback()	120
FetchNextRowsEx()	86	RunProc()	120
FlushQMFCache()	87	SaveData()	121
GetColumnCount()	87	SaveQMFPProc()	123
GetColumnDataValue()	88	SaveQMFPQuery()	123
GetColumnHeader()	88	SetBindOption()	124
GetColumnHeaderEx()	89	SetBindOwner()	126
GetColumnHeadings()	89	SetBusyWindowButton()	126
		SetBusyWindowMessage()	127
		SetBusyWindowMode()	127

SetBusyWindowTitle()	128	ShowBusyWindow()	133
SetGlobalVariable()	129	StartBind()	133
SetHostVariable()	129	Appendix. Notices	135
SetOption()	130	Trademarks	138
SetParent()	131	Index	139
SetProcVariable()	131		
SetVariable()	132		

The QMF Library

You can order manuals either through an IBM representative or by calling 1-800-879-2755 in the United States or any of its territories.

Evaluating	<div data-bbox="521 395 643 545"><p>Introducing QMF</p><p>GC27-0714</p></div>			
Installing, planning for, administering, and diagnosing	<div data-bbox="521 564 643 713"><p>Installing and Managing QMF on OS/390</p><p>GC27-0719</p></div>	<div data-bbox="669 564 791 713"><p>Installing and Managing QMF on VM/ESA</p><p>GC27-0720</p></div>	<div data-bbox="817 564 939 713"><p>Installing and Managing QMF on VSE/ESA</p><p>GC27-0721</p></div>	<div data-bbox="964 564 1087 713"><p>Installing and Managing QMF for Windows</p><p>GC27-0722</p></div>
	<div data-bbox="521 725 643 874"><p>QMF Messages and Codes</p><p>GC27-0717</p></div>	<div data-bbox="669 725 791 874"><p>QMF High Performance Option User's Guide for OS/390</p><p>SC27-0724</p></div>		
Using	<div data-bbox="521 899 643 1048"><p>Using QMF</p><p>SC27-0716</p></div>	<div data-bbox="669 899 791 1048"><p>QMF Reference</p><p>SC27-0715</p></div>	<div data-bbox="817 899 939 1048"><p>Getting Started With QMF for Windows</p><p>SC27-0723</p></div>	
Application programming	<div data-bbox="521 1072 643 1222"><p>Developing QMF Applications</p><p>SC27-0718</p></div>			
Online libraries	<div data-bbox="542 1246 653 1355"></div> <div data-bbox="537 1364 655 1425"><p>SK2T-0730 OS/390, VM, & VSE</p></div>	<div data-bbox="720 1246 830 1355"></div> <div data-bbox="715 1364 833 1407"><p>SK2T-6700 OS/390 only</p></div>	<div data-bbox="897 1246 1008 1355"></div> <div data-bbox="892 1364 1010 1407"><p>SK2T-2067 VM only</p></div>	<div data-bbox="1075 1246 1185 1355"></div> <div data-bbox="1075 1364 1193 1407"><p>SK2T-0060 VSE only</p></div>

Chapter 1. Introduction

This chapter provides an overview of QMF for Windows and explains some of the basic tasks for getting started with QMF for Windows.

Database servers

Queries, forms, procedures, and tables are run and saved at a database server.

DB2 family of databases

QMF for Windows can connect to a wide range of DB2 databases.

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

Your QMF for Windows license determines which DB2 family products you can install on and connect to with your copy of QMF for Windows.

User name vs. technical name

Different versions and types of DB2 refer to a database by an RDB name, location name, or other technical name.

With QMF for Windows your administrator can assign an easily remembered name to a database name, for example, Purchasing Database instead of DB2P_01_PURCH.

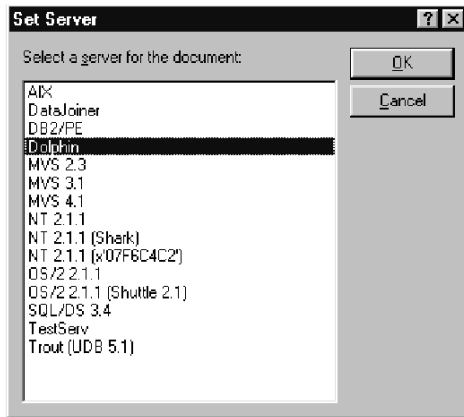
QMF for Windows refers to a database server or a DB2 database as a "server."

Setting the server name

Before you can query a database, QMF for Windows needs to know where the database is stored.

1. From the **File** menu, select **New SQL Query**. A new SQL query document opens.

2. From the **Query** menu, select **Set Server**. The Set Server dialog box opens.



3. From the list of available servers, select the one that you want to query and click **OK**. When you begin your next QMF for Windows session, QMF for Windows automatically reconnects to the same server.

Database security

You must provide a user ID and password before you can connect to a server.

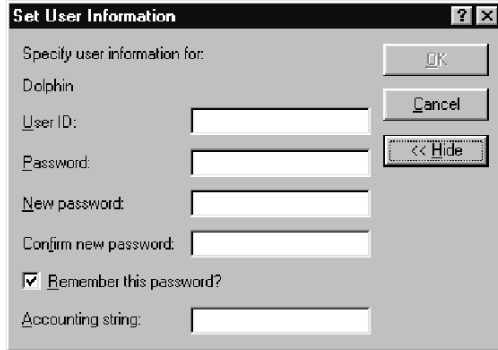
Logging on

You must specify a valid user ID and password for the database server that you are trying to access. The database server user ID and password is not necessarily the same as your local or network user ID and password.

If you are running Windows, you have the option of remembering server passwords across QMF for Windows sessions. If you are currently logged on to Windows, the Set User Information dialog box displays an additional check box labeled **Remember this password?**. If you select this check box, then the password you enter for that server is stored in the Windows password list. Whenever you are logged on to Windows, QMF for Windows can automatically retrieve that password so that you are not prompted. If you are not logged on when you run QMF for Windows, or if you are logged on as a different user, QMF for Windows prompts you for a user ID and password.

Note: If you choose to save a password, anyone who can log on to your Windows account can access your database servers with your (server) user ID and password.

1. From the **Query** menu, select **Set User Information**. The Set User Information dialog box opens.



2. Enter your user ID and password in the appropriate fields.

Note: The user ID and password are case sensitive. For example, if your user ID or password is uppercase, you must enter it in uppercase. Some types of database servers treat user IDs and passwords as case sensitive; others do not.

3. Check **Remember this password?** if you want to save your user ID and password.
4. Click **OK**. QMF for Windows stores this information in preparation for accessing the server.

Correcting passwords

If you entered an incorrect password, you can correct the error by reopening the Set User Information dialog box.

1. From the **Query** menu, click **Set User Information**. The Set User Information dialog box opens.
2. Type the password again and click **OK**. The password is corrected.

Changing passwords

You can change your password at the database server from QMF for Windows. This feature is currently supported only by DB2 for OS/390 Version 5 and later.

1. From the **Query** menu, select **Set User Information**. The Set User Information dialog box opens.
2. Click **Change**. The **New password** and **Confirm new password** fields appear.
3. Type your new password in the **New password** and **Confirm new password** fields and click **OK**. Your database server password is changed.

Specifying accounting strings

The database server uses accounting strings to track system usage. Ask your database administrator to find out if your system uses accounting strings.

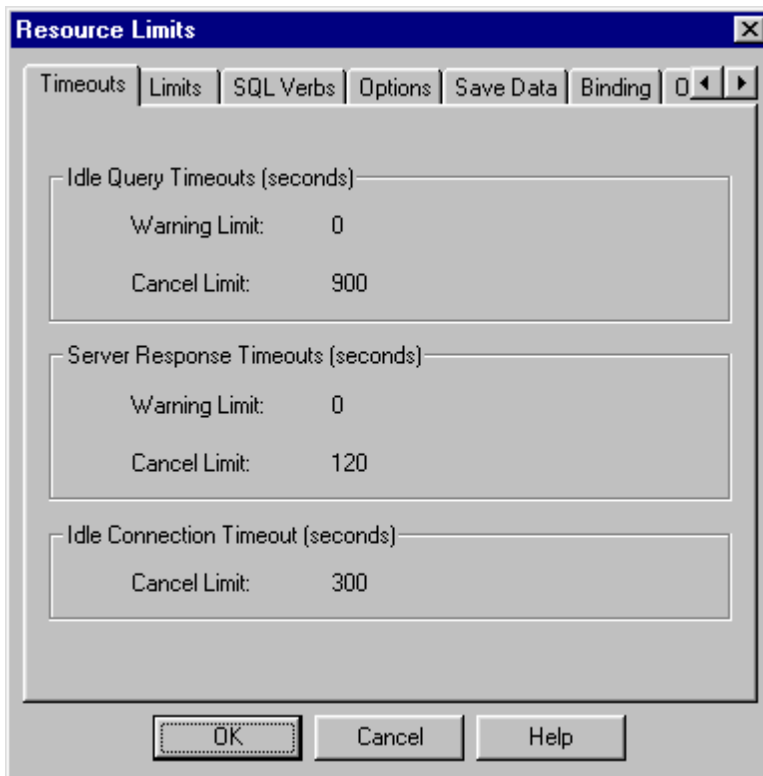
1. From the **Query** menu, select **Set User Information**. The Set User Information dialog box opens.
2. In the **Accounting string** field, type the accounting string you want to use and click **OK**. QMF for Windows stores the information in preparation for accessing the server.

Governing

The QMF for Windows governor is always running in the background, monitoring the usage of database and system resources. The governor also limits the type and size of queries you can run.

Viewing resource limits

From the **View** menu, select **Resource Limits**. The Resource Limits dialog box opens. All information in the Resource Limits dialog box is read only. Your system administrator sets these limits.



The types of limits and controls that can be in effect are:

- Timeouts
- Limits
- SQL Verbs
- Options
- Save Data
- Binding
- Object Tracking

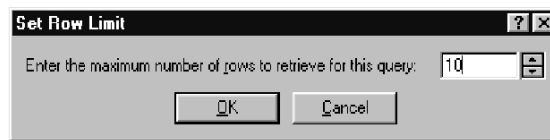
Setting your own row limit

You can specify the maximum number of rows to retrieve for this query. When this limit is reached, QMF for Windows cancels the query. The maximum authorized row limit specified in your resource limits group takes precedence over this parameter.

Enter 0 to specify no limit in this field.

Rows that QMF for Windows has already retrieved in excess of this limit are retained and available for viewing.

1. From the **Query** menu, click **Set Row Limit**. The Set Row Limit dialog box opens..



2. Enter the maximum number of rows that you want the query to return and click **OK**. The row limit is applied the next time you run the query.

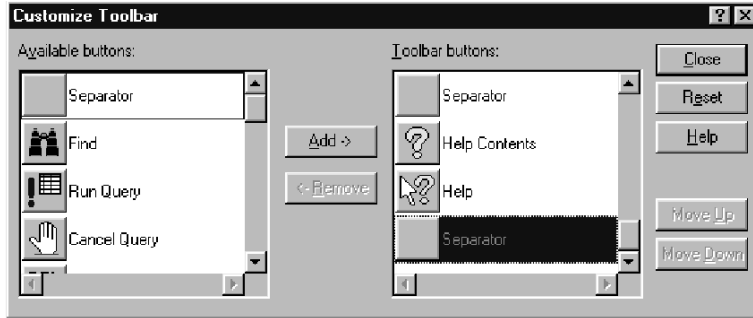
Customizing the toolbar

You can customize the toolbar to display only the buttons that you want to see.

Adding buttons to the toolbar

You have the option of adding buttons to the existing QMF for Windows toolbar. These buttons represent functions that not all users may need, but that are available for inclusion in the toolbar.

1. Double-click the gray area surrounding the toolbar. The Customize Toolbar dialog box opens.



2. From the **Available buttons** column, select the button that you want to add and click **Add**. The button is added to the toolbar.
3. When you finish adding buttons, click **Close**. The dialog box closes and the new buttons are added to the toolbar.

Moving buttons on the toolbar

You have the option of rearranging the buttons on the QMF for Windows toolbar.

1. Double-click on the gray area surrounding the toolbar. The Customize Toolbar dialog box opens.
2. From the **Available buttons** column, select the button that you want to move.
3. Use the **Move Up** and **Move Down** buttons to move the button within the toolbar.
4. When you finish moving buttons, click **Close**. The dialog box closes and the buttons appear in their new locations.

Removing buttons from the toolbar

You have the option of removing buttons from the QMF for Windows toolbar.

1. Double-click on the gray area surrounding the toolbar. The Customize Toolbar dialog box opens.
2. From the **Available buttons** column, select the button that you want to remove and click **Remove**. The button is removed from the toolbar.
3. When you finish removing buttons, click **Close**. The dialog box closes and the buttons are removed from the toolbar.

Chapter 2. Working with SQL Queries

Structured Query Language (SQL) is the most basic interface between a user and a database. Queries are written in SQL and processed by the database. Users can write QMF for Windows queries in SQL, or create queries using the "point and click" method.

SQL queries

Structured Query Language queries require knowledge of the commands and syntax of SQL. Users unfamiliar with SQL should try creating prompted queries.

Creating new SQL queries

Click the **New SQL Query** button on the toolbar.



A new query document opens.

Running SQL queries at a database server

1. Open a new query document and type in a query, or open an existing query file, or open a query from the database.
2. Click the **Run Query** button on the toolbar.



The query runs and the results are displayed.

Switching between the Results view and the SQL view

You can look at either the results of a query or the SQL statement itself.

From the SQL view of a query that has been run, click the **View Results** button on the toolbar.



The query results are displayed.

-or-

From the Results view of a query, click the **View SQL** button.



The SQL statement is displayed.

Working with fonts

You can change the font used to display queries. The choice of fonts varies according to what has been installed on your computer. For more information on adding fonts, refer to your operating system's help facility.

Note: If you save the query after selecting a new query display font, that query is always displayed using the new font.

Selecting the query display font

1. From the SQL view, click **Set Font** from the **Query** menu. The Font dialog box opens.
2. Select the font for displaying the text of the query and click **OK**. The query reappears in the new font.

Note: Click **Set As Default** to use the selected font as the default font for all new queries.

Multiple queries

You can have more than one query document open at the same time. You can also run more than one query at a time. You can use this feature to generate multiple reports, or to cut and paste SQL text from one query to another.

Displaying multiple queries simultaneously

1. Open at least two query documents.
2. From the **Window** menu, select one of the following commands:

Command	Result
Cascade	Displays queries in a staggered series.
Tile Horizontal	Displays query windows stacked vertically.
Tile Vertical	Displays query windows stacked side by side.

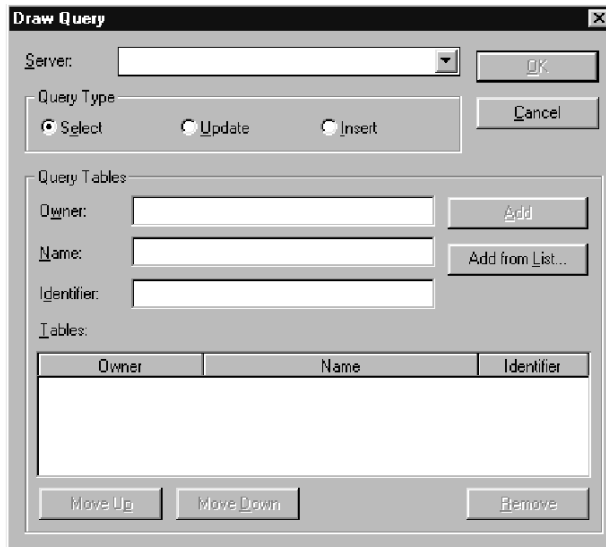
The query windows are arranged according to the option you select.

Drawing queries

Use the Draw Query command to create new SQL query documents. You specify one or more table names and the type of SQL statement that you want, and QMF for Windows automatically creates an SQL statement that references the names and data types of the columns in the table.

Creating new SQL queries

1. From the **File** menu, click **Draw Query**. The Draw Query dialog box opens.



2. Select the type of query that you want to create:

Query type	Result
Select	Retrieve rows from one or more tables.
Update	Change the information in a table.
Insert	Add new rows to a table.

3. Enter the owner and name of the table to be queried.

Note: You can use patterns to select table names from a list of matching tables.

- Use the percent character (%) to match a string of any length containing any characters. For example, to list all tables with a name beginning with the letter A, enter A%.

- Use the underscore character (`_`) to match a single character. For example, to list all tables with an owner that has the letter A in the second position, enter `_A%`.

After you enter a pattern, click **Add from List** and select a table from the resulting list.

4. Enter a unique identifier for the table.
5. Click **Add**. The table is added to the query.
6. When you have added the table or tables that you want to query, click **OK**. An SQL query for the selected tables is created and displayed.

Substitution variables in SQL queries

With substitution variables, you can use the same query to retrieve different information by supplying different values each time you run the query. To retrieve a different set of data, you do not need to rewrite the query. Rather, you just supply different values for the substitution variables in the query when you run it.

A substitution variable is text that you include in a query. It must begin with an ampersand character (&), and can contain up to 18 characters, which can be alphabetic, numeric, or one of the following special characters: `^ ! $ % & ' { } ? @ # % \` or `_`. For example, the following values are valid substitution variables:

```
&VARIABLE1  
&DEPARTMENT_NUMBER
```

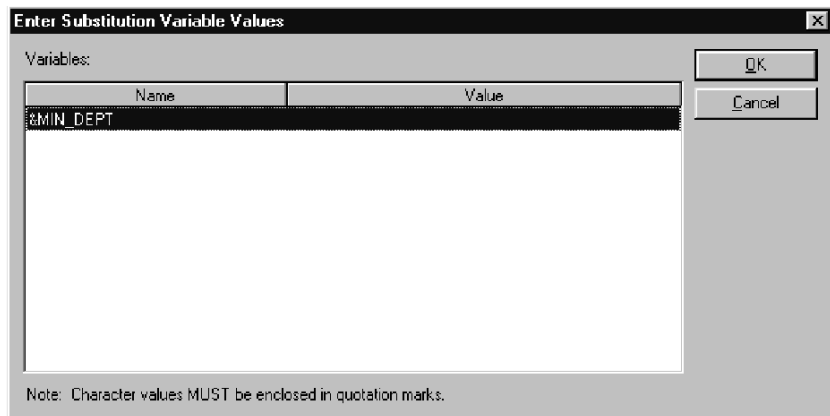
A substitution variable can appear anywhere in a query, and its value can be anything that you might write in a query (except a comment). For example, you can use a substitution variable in place of a column name, search condition, subquery, or any specific value.

Running SQL queries with substitution variables

1. Open a new query document and enter this SQL statement:

```
SELECT * FROM Q.STAFF WHERE DEPT >= &MIN_DEPT
```

2. Run the query. The Enter Substitution Variable Values dialog box opens.



3. In the **Value** field enter a value of 50 and click **OK**. The query runs and displays the query results.

Try experimenting with substitution variables by replacing values in the **SELECT** and **FROM** clauses. See what results your queries return with different inputs.

Saving and Opening SQL queries

You can save queries on your PC, on a file server, or at a database server.

Saving SQL queries to files

1. From an open query, click the **Save** button on the toolbar.



If the query has been saved before, it is saved again. If the query has not been previously saved, the Save As dialog box opens.

2. Enter the name of the file where you want the query stored and click **OK**. The query is saved.

Opening saved SQL query files

1. Click the **Open** button on the toolbar.



The Open dialog box opens.

2. Select the file that you want to open and click **OK**. The selected query opens in a new query document.

Saving SQL queries at the database server

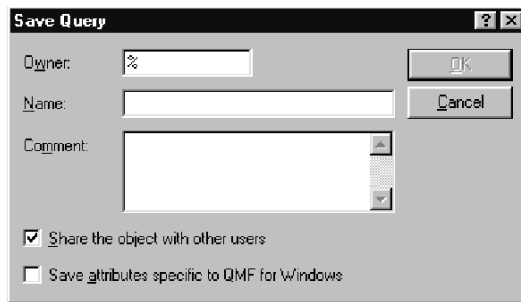
Queries saved at the server can be made accessible to other users. If you want to share your queries with other users, save them at the database server.

- 1.

From an open query, click the **Save at Server** button on the toolbar.



The Save Query dialog box opens.



2. Enter an owner, a name, select whether or not to share the saved query with other users, and click **OK**. The query is saved at the server.

If a query with this name already exists, you are prompted to overwrite the previously existing query.

Opening saved SQL queries at the database server

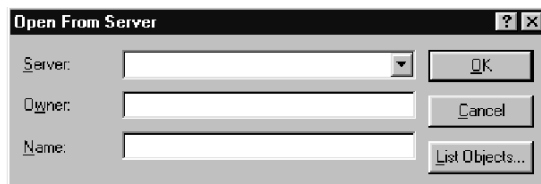
You can open queries that have been saved at the database server.

- 1.

Click the **Open From Server** button on the toolbar.



The Open From Server dialog box opens.



2. Enter a server, owner, and name, and click **OK**. The SQL query opens.

Printing SQL queries

You can preview and print your SQL queries.

Previewing a query

1. Open a query and activate the SQL view. The SQL statement appears.
2. From the **File** menu, click **Page Setup**. The Page Setup dialog box opens.
3. Make any changes you want to the layout of the page and click **OK**.
4. Click the **Print Preview** button on the toolbar.



A preview of the printed query appears.

Printing SQL queries

1. Open a query and activate the SQL view. The SQL statement appears.
2. From the **File** menu, click **Page Setup**. The Page Setup Dialog box opens.
3. Make any changes you want to the layout of the page and click **OK**.
4. Click the **Print** button on the toolbar.



The query is printed.

Chapter 3. Working with Prompted Queries

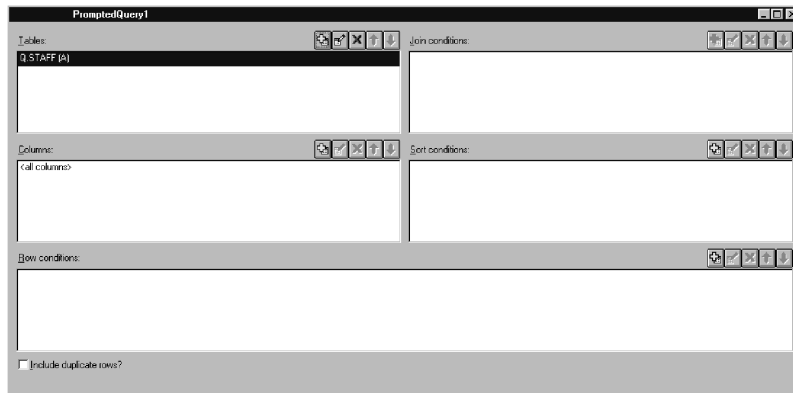
Prompted queries are an easy way for you to create a query by selecting options from menus and lists. Once you have created a prompted query, you can save it, or convert it to an SQL query.

Building simple queries

You can easily build simple queries using the prompted query interface.





Opening new prompted queries

- From the **File** menu, click **New Prompted Query**. A new prompted query document opens.



Prompted query action buttons

Edit prompted queries using the query action buttons. A set of buttons appears above the section that it controls.

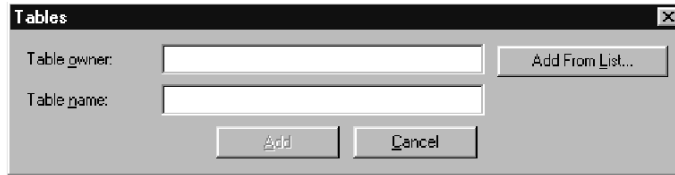
Prompted query action button	Appearance	Result
Add		Click to add an item to the prompted query.
Edit		Click to edit the highlighted item in the query.
Delete		Click to delete the selected item.
Move up and Move down		Click to move the selected item up and down in the prompted query.

Adding tables to prompted queries

1. In the Tables section of the prompted query document, click the **Add** button.



The Tables dialog box opens.



- 2.

Type the owner and name of the table that you want to add and click **Add**. The table is added to the query.

Note: You can use patterns to select objects from a list of matching objects.

- Use the percent character (%) to match a string of any length containing any characters. For example, to list all tables with a name beginning with the letter A, enter A%.
- Use the underscore character (_) to match a single character. For example, to list all tables with an owner that has the letter A in the second position, enter _A%.

After you enter a pattern, click **Add From List** and select a table from the resulting list.

3. Add any additional table conditions to the query and click **Close**. The prompted query document appears with the new tables listed.

Running prompted queries

You run a prompted query the same way that you run an SQL query. Click the Run Query button on the toolbar.



The prompted query runs.

Building Complex Queries

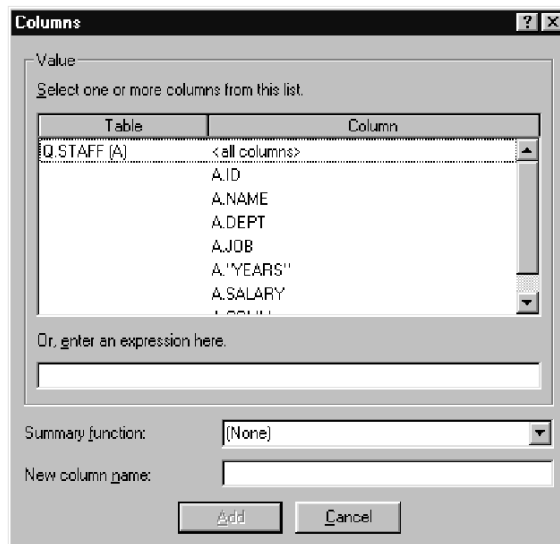
You can also build more complex queries using the prompted query interface.

Adding columns to prompted queries

1. In the Column section of the Prompted Query document, click the **Add** button.



The Columns dialog box opens.



2. Select the column you want to add and click **Add**. The column is added to the prompted query.
3. Add any additional columns to the query and click **Close**. The Prompted Query document appears with the new columns listed.

Note: You can apply a summary function to the column by selecting one in the Function field. Available summary functions include: AVERAGE, COUNT, MAXIMUM, MINIMUM, and SUM.

Note: You can rename a column in the query by typing a new column name in the **New column name** field.

Using sort conditions

Sort conditions are used to specify the way you want to sort the rows in the query. Rows can be sorted in ascending (A-Z) or descending (Z-A) order.

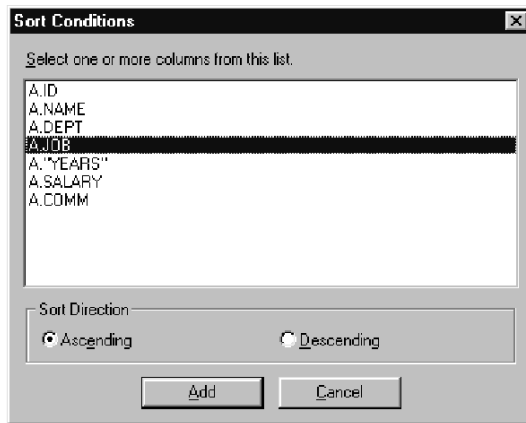
If you sort your rows by more than one column, the first column is ordered first, the second column is ordered within the order of the first column, and so on.

Adding sort conditions

1. In the Sort conditions section of the Prompted Query document, click the **Add** button.



The Sort Conditions dialog box opens.



2. Select the column you want to sort by, the direction in which to sort, and click **Add**. The sort condition is added to the prompted query.
3. Add any additional sort conditions to the query and click **Close**. The Prompted Query document appears with the new sort conditions listed.

Using row conditions

Many times you want to view only certain rows in a table. To select specific rows to view, add row conditions. If you do not use row conditions, all the rows in the table are displayed.

The following row conditions are available:

- Equal to
- Less than
- Less than or equal to
- Greater than
- Greater than or equal to
- Between
- Starting with
- Ending with
- Containing
- NULL

Row conditions are controlled by the following operators:

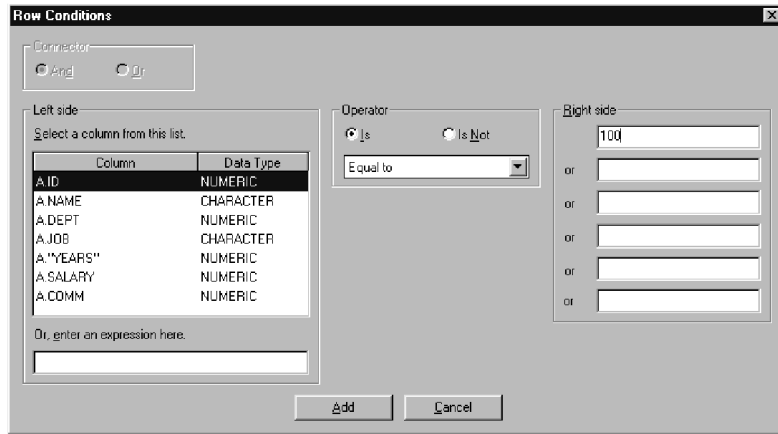
- Is
- Is Not

Adding row conditions

1. In the Row conditions section of the Prompted Query document, click the **Add** button.



The Row Conditions dialog box opens.



2. Select the parts of the conditional statement and click **Add**.

Part of the row condition	Function
Left side	Select the column you want to examine.
Operator	Determine the relationship between the left and right hand sides of the row.
Right side	Enter the condition for which you want to check.

The row condition is added to the prompted query.

3. Add any additional row conditions to the query and click **Close**. The Prompted Query document appears with the new row conditions listed.

Using multiple tables in prompted queries

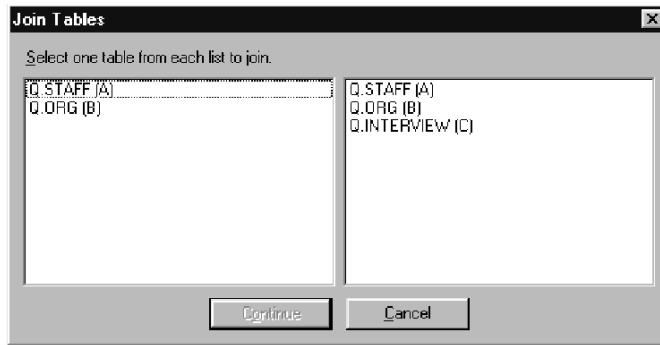
You can include information from more than one table in a prompted query.

You must relate the two tables, specifying one or more join conditions in each. Only rows from the tables where join columns are equal are included in the

results. The data type of each column in a join condition must match. Once you have specified a relationship between two columns, QMF for Windows remembers the relationship, and suggests it in future queries, making the creation of subsequent queries simpler and more efficient.

Creating prompted query join conditions

1. In the Tables section of the Prompted Query Window click the **Add** button to add at least two tables. If you have never joined the tables before, the Join Tables dialog box opens. If you have, QMF for Windows suggests the join condition that you used previously.



2. Select a column with the same data type from each table and click **Add**. The new join condition appears in the prompted query.

Prompted queries and SQL

You can use the prompted query interface to learn SQL.

Viewing SQL for prompted queries

From the prompted query view, click the **View SQL** button on the toolbar.



The equivalent SQL statement for the prompted query appears. You cannot modify the SQL statement from this view.

Converting prompted queries to SQL

You can convert a prompted query to a new SQL query document. The new SQL query can be modified, saved, printed, and run. From the **Query** menu, click **Convert to SQL**. The query is converted into a new SQL query document.

Using Substitution Variables in Prompted Queries

Substitution variables can be used in a prompted query in the same way as in an SQL query. See “Substitution Variables in SQL Queries” .

For example, substitution variables can be used in:

- a row condition
DEPT Is Greater Than Or Equal To &MinDept
- a column specification
&InputNum

Saving Prompted Queries

Prompted queries can be saved in files on your PC, on a file server, or at a database server.

Saving prompted queries to files

1. From an open prompted query, click the **Save** button on the toolbar.



Note: If the query has been saved before, it is saved again. If the query has not been previously saved, the Save As dialog box opens.

2. Enter the name of the file where you want the prompted query stored and click **OK**. The query is saved.

Opening saved prompted query files

1. Click the **Open** button on the toolbar.



The Open dialog box opens.

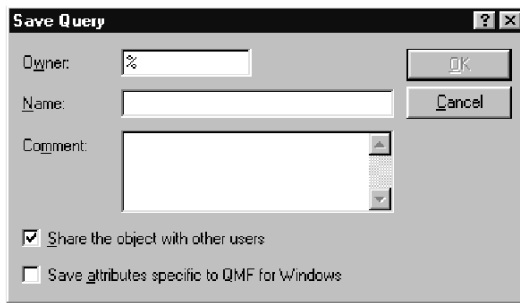
2. Select the file you want to open and click **OK**. The selected prompted query opens in a new query document.

Saving prompted queries at the database server

1. From an open prompted query, click the **Save at Server** button on the toolbar.



The Save Query dialog box opens.



2. Enter an owner, a name, select whether or not to share the saved query with other users, and click **OK**. The query is saved at the server.
If a query with this name already exists, you are prompted to overwrite the existing query.

Opening saved prompted queries at the database server

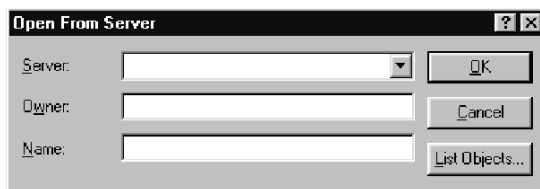
You can open prompted queries that have been saved at the database server.

- 1.

Click the **Open From Server** button on the toolbar.



The Open From Server dialog box opens.



2. Enter a server, owner, and name, and click **OK**. The prompted query opens.

Printing Prompted Queries

You can print your prompted query. You can also print the SQL text of a prompted query. See “Printing SQL queries” on page 13.

Previewing prompted queries

You can preview the results or text of a prompted query before you print it.

1. Open a query and activate the prompted view. The query appears.
2. From the **File** menu, click **Page Setup**. The Page Setup dialog box opens.

3. Make any changes you want to the layout of the page and click **OK**.
4. Click the **Print Preview** button on the toolbar.



A preview of the printed query appears.

Chapter 4. Working with Query Results

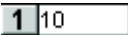



You can apply formatting, grouping, and aggregation directly to query results. This formatting can be saved with the query, or exported as a form.

Sorting and sizing query results

Users can select, resize, reorder, and sort the data results from a query.

Selecting columns and rows

Once you have run a query, you can use the controls in the Results view to edit and select information.

Column and row selectors	Appearance	Function
Row selector		Click to select all the data in a row.
Column selector		Click to select all the data in a column.
Cell		Click directly on the cell to select it.
Scroll-to-bottom and Scroll-to-top buttons		Click to scroll to the top or bottom of a set of query results.

Resizing columns and rows

You can change the appearance of a set of query results by resizing its columns and rows.

1. Using the mouse select the black dividing line between two columns or two rows.
2. Drag the divider line from side to side or up and down to resize the column or row.

Note: If you save the query after resizing its rows or columns, that query is always displayed using the new formatting.

Auto fitting columns and rows

You can automatically size columns and rows to fit the data they contain.

Using the mouse, select an entire column or row and double-click on the black dividing line between it and the adjacent object. The column or row is automatically resized to fit its data.

Note: If you save the query after resizing its rows or columns, that query is always displayed using the new formatting.

Sorting query results

Once you have run a query, you can sort the results alphabetically by column.

From the Results view of a query, select a column and select **Sort Ascending** from the **Results** menu.

The query results are sorted in ascending order.

-or-

From the Results view of a query, select a column and select **Sort Descending** from the **Results** menu.

The query results are sorted in descending order.

Note: To apply more complex sorting to the selected column, select **Sort** from the **Results** menu.

Reordering columns

You can change the order of columns in the query results.

From the Results view of a query, select a column and drag it to its new location.

The column appears in the new order.

Formatting query results

You can change the font used to display queries and query results. The choice of fonts varies according to what has been installed on your computer. For more information on adding fonts, refer to your operating system's help facility.

Note: If you save the query after selecting a new query results display font, those results are always displayed using the new font.

Selecting the query results display font

1. From the Results view, select **Set Font** from the **Results** menu. The Font dialog box opens.
2. Select the font and type size for displaying the results of the query and click **OK**. The query results are displayed in the format that you specified.

Note: Click **Set As Default** to use the selected font as the default font for all query results.

Formatting numeric query results

1. From the Results view, select a column containing numeric values and select **Format** from the **Results** menu. The Format dialog box opens.
2. Specify the formatting that you want to apply and click **OK**. The values are formatted according to your selection.

Note: Click **Set As Default** to use the selected font as the default font for all query results.

Converting query results formatting to a form

You can convert query results formatting to a form.

1. From the **Results** menu, select **Display Report**.
The Select Form dialog box opens.
2. Select **From Query** and click **OK**.
The query results formatting is converted to a form and opened in a new form window.

Grouping and aggregating query results

You can apply grouping, aggregation, and summary formatting to query results.

Grouping query results

You can group query results with or without summary information.

1. Select the column that you want to group.
2. From the **Results** menu, select the type of grouping that you would like to apply.
The column is grouped according to your selection.

Summarizing query results

You can summarize query results by column.

1. Select the column that you want to group.
2. From the **Results** menu, select the type of summary that you would like to apply.
The column is summarized according to your selection.

Saving query results and formatting

You can save query results and save the formatting as a form.

Saving query results as a table

You can save query results as a table at a database server.

1. From the **Results** menu, select **Save to Database**.

The Save Data dialog box opens.

2. Enter an owner and table name and click **OK**.

The query results are saved as a table at the database.

Saving query results to files

You can save query results to a file on your PC or a file server.

1. From the **Results** menu, select **Save to File**.

The Export Data dialog box opens.

2. Specify the location to which you want to save the file, any export options, and click **OK**.

The query results are saved to a file.

Printing query results

You can preview and print your query results.

Previewing query results

1. Open and run a query. The query results appear.
2. From the **File** menu, select **Page Setup**. The Page Setup Dialog box opens.
3. Make any changes you want to the layout of the page and click **OK**.
4. Click the **Print Preview** button on the toolbar.



A preview of the printed query results appears.

Printing query results

1. Open a query and activate the Results view. The query results appear.
2. From the **File** menu, select **Page Setup**. The Page Setup Dialog box opens.
3. Make any changes that you want to the layout of the page and click **OK**.
4. Click the **Print** button on the toolbar.



The query results are printed.

Chapter 5. Working with Reports

Reports are created by combining query results with formatting from a form.

Forms

Forms are sets of formatting instructions used to create, display, and print reports.

Understanding forms

Forms are composed of a number of components. These components can all be edited in a form document.

Main The primary components of a form, including headings, footings, and breaks.

Breaks

Characteristics, content, and placement of up to six subtotal lines in a report.

Calculations

Report calculation expressions.

Note: You must have IBM's ObjectREXX installed on your machine to use form calculations.

Columns

Appearance and formatting of columns in the report. Definable characteristics include column order, format, usage, indentation, and width.

Conditions

Conditional formatting constraints. For example, you can set the form to not display rows that do not meet certain characteristics.

Details

Report detail headings and body text. This is where you can combine or replace tabular data with free-form text to create form letters or address labels.

Final Content and placement of your report's final text. For example, you can choose to include final text and summary data at the end of the report.

HTML

Content and placement of HTML tags and formatting in HTML reports.

Options

Miscellaneous appearance options for your report.

Page Content and placement of the page heading and footing on your report.

Producing a report using a form

Reports are created by combining query results with the formatting options contained in a form. You can produce multiple reports from a single set of query results by repeating this process.

1. From a query results view, click the **Display Report** button.



The Select Form dialog box opens.



2. Depending on the type of form you select in the Select From dialog box, you are asked to provide additional information. Specify the file location or owner and name, or document title, as appropriate, and click **OK**. The report is generated using the selected form and current query results.

Editing a form

The Form window provides many options for editing and formatting forms.

From an open form, display the Form menu. The Form menu lists all your options for editing and formatting your form. You can also edit any of these components by clicking the corresponding button on the toolbar.

Creating a form

These steps all include sample data from the table Q.STAFF. Try experimenting with different settings to create your own custom forms.

Step 1: Create a form

1. Run the following SQL query to retrieve the data to display in the report:

```
SELECT * FROM Q.STAFF ORDER BY DEPT, NAME
```

The query results appear.

2. Click the **Display Report** button on the toolbar. The Select Form dialog box opens.
3. Specify that you want to use the default form and click **OK**. QMF for Windows displays the default report. To make changes to the default format, click one of the form component buttons on the toolbar. A button for each form component is displayed on the form toolbar.

Step 2: Change the column order

We want NAME to be the first column in the report and ID to be the second. The order of the columns is specified in the Columns component of the form.

1. Click **Columns...** on the **Form** menu to display the Columns tab of the Form dialog box.
2. Change the sequence of a column by typing over the existing sequence value. To make NAME the first column in the report, change its sequence number (the column in the list labeled Seq) to 1.
3. To make ID the second column in the report, change its sequence number to 2 and click **OK**. QMF for Windows displays the report with the new column order in the Form window.

Step 3: Change the column headings

We want the first column heading to be EMPLOYEE and the second to be COMMISSION. Column heading text is specified in the Columns component of the form.

1. Click **Columns...** on the **Form** menu to display the Columns tab of the Forms dialog box.
2. Change the column heading by typing over the existing column heading text. Change the first column heading to EMPLOYEE, and the last column heading to COMMISSION and click **OK**. QMF for Windows displays the report with the new column headings in the Form window.

Step 4: Change the column format

We want the SALARY column to be displayed with the appropriate currency symbol. The format of a column is determined by its edit code, which is specified in the Columns component of the form.

1. Click **Columns...** on the **Form** menu to display the Columns tab of the Forms dialog box.
2. Change the SALARY column edit code to D2 by typing over the existing edit code and click **OK**. QMF for Windows displays the report with the SALARY column using the appropriate currency symbol in the Form window.

Step 5: Add summary information

We want to divide the report into separate sections for each department. In addition, we want to see the total SALARY and COMMISSION for each department at the end of each section. To do this, we need to specify how we

want each column in the report to be used. The usage of a column is determined by its usage code and is specified in the Columns component of the form.

1. Click **Columns...** on the **Form** menu to display the Columns tab of the Forms dialog box.
2. To divide the report into sections based on DEPT, change the usage code of DEPT to BREAK1. Usage codes beginning with the word BREAK produce a section break for the specified column. The number that follows the word BREAK determines the break level; up to six break levels are supported in a report.
3. To specify that we want to include a total SALARY and COMMISSION for each DEPT, change the usage code of SALARY and COMMISSION to SUM.
4. The report will be easier to understand if we also include descriptive information at the end of each section break. To do this, click **Breaks...** on the **Form** menu.
5. You specify break footing text on the Breaks tab of the Form dialog box. Set the first break footing line to Department Total and click **OK**. QMF for Windows displays the Form window.

Step 6: Add page headers and footers

We want to add a page heading and footing to our report. Page headings and footings are specified in the Page component of the form.

1. Click **Page...** on the **Form** menu to display the Page tab of the Form dialog box.
2. The top portion of this dialog is used to specify the page heading. Set the first line of the page heading to Department Report and the second line to Total Salary and Commissions. Choose how you want the heading to be aligned.
3. The bottom portion of this dialog is used to specify the page footing. Set the first line of the page footing to End Of Page. Choose how you want the footing to be aligned and click **OK**. QMF for Windows displays the Form window.

Saving forms

You can save forms on your PC, on a file server, or at a database server.

Saving a form to a file

1. From an open form, click the **Save** button.
2. If the form has been saved before, select **Save**. If the form has not been previously saved, the Save As dialog box opens.
3. Enter the name of the file where you want the form stored, and click **OK**. The form is saved.

Opening saved form files

1. Click the **Open** button on the toolbar.



The Open dialog box opens.

2. Select the file that you want to open and click **OK**. The selected form opens in a new form document.

Saving forms at the database server

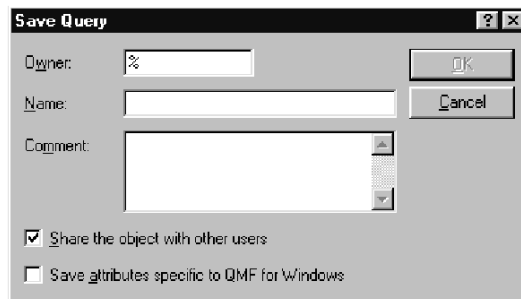
Forms saved at the server can be made accessible to other users. If you want to share your forms with other users, save them at the database server.

- 1.

From an open form, click the **Save at Server** button on the toolbar.



The Save Form dialog box opens.



The screenshot shows a dialog box titled "Save Query". It has a title bar with a question mark and a close button. The dialog contains the following fields and controls:

- Owner:** A text field with a dropdown arrow on the right.
- Name:** A text field.
- Comment:** A multi-line text area with scrollbars.
- Buttons:** "OK" and "Cancel" buttons are located to the right of the Name field.
- Checkboxes:** At the bottom, there are two checkboxes: " Share the object with other users" and " Save attributes specific to QMF for Windows".

2. Enter an owner, a name, select whether or not to share the saved form with other users, and click **OK**. The form is saved at the server.

If a form with this name already exists, you are prompted to overwrite the existing form.

Opening saved forms at the database server

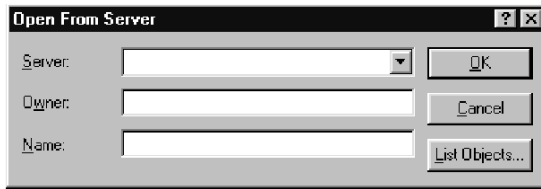
You can open forms that have been saved at the database server.

- 1.

Click the **Open From Server** button on the toolbar.



The Open From Server dialog box opens.



2. Enter a server, owner, and name, and click **OK**. The form opens.

Printing reports

You can produce printed reports.

1. Open a form and click **Page Setup**.
2. Make any changes that you want to the layout of the page and click **OK**.
3. Click **Print Report** on the **File** menu.

The report is printed.

Exporting reports

You can export a report to a file.

1. Open a form and click **Page Setup**.
2. Make any changes that you want to the layout of the page and click **OK**.
3. Click **Export Report** on the **File** menu. The Export Report dialog box opens.



4. Enter the name of the file where you want the report stored and click **OK**. The report is exported.

Chapter 6. Working with Procedures

Linear procedures enable you to run queries, generate reports, edit data, and perform other functions by executing a single command. For a complete listing of all the procedure commands supported by QMF for Windows, refer to the online help facility.

Procedures with logic, or REXX procedures, are similar to linear procedures, but contain IBM's Object REXX programming language as well as procedure commands. Object REXX must be installed locally in order to run procedures with logic

Running procedures

Procedures are used to execute multiple functions with one command.

Creating a new linear procedure

From the **File** menu, select **New Procedure**.

A new procedure document opens.

Creating a new procedure with logic

1. From the **File** menu, select **New Procedure**.

A new procedure document opens.

2. Type a REXX comment line as the first line of the procedure. REXX comment lines begin with `/*` and end with `*/`
3. Type any QMF procedure commands you want in the procedure. QMF commands must be entered in uppercase and enclosed in quote marks.
4. Type any REXX commands you want in the procedure.

Note: REXX commands are run locally, not at the database server. Object REXX must be installed locally.

Running a procedure at a database server

1. Open a new procedure document and type in a set of commands, or open an existing procedure from a file or the database server.
2. Click the **Run Procedure** button on the toolbar.



The procedure runs.

Saving procedures

You can save procedures on your PC, on a file server, or at a database server.

Saving procedures to files

1. From an open procedure, click the **Save** button on the toolbar.



If the procedure has been saved before, the procedure is saved. If the procedure has not been previously saved, the Save As dialog box opens.

2. Enter the name of the file where you want the procedure stored and click **OK**. The procedure is saved.

Opening a saved procedure file

1. Click the **Open** button on the toolbar.



The Open dialog box appears.

2. Select the file you want to open and click **OK**. The selected procedure opens in a new procedure document.

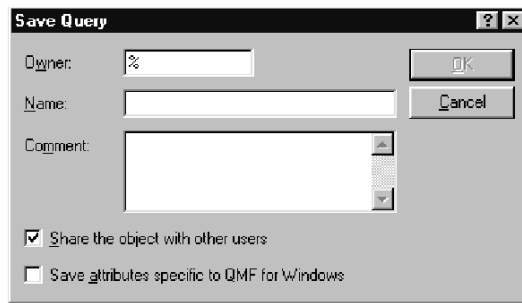
Saving a procedure to the database server

- 1.

From an open procedure, click the **Save at Server** button on the toolbar.



The Save Procedure dialog box opens.



2. Enter an owner, a name, select whether or not to share the saved procedure with other users, and click **OK**. The procedure is saved at the server.

If a procedure with this name already exists, you are prompted to overwrite the previously existing procedure.

Opening saved procedures at the database server

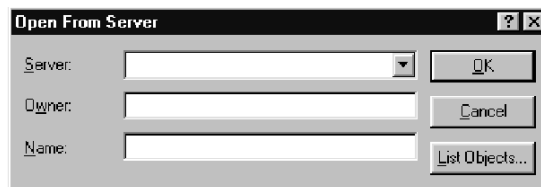
You can open procedures that have been saved at the database server.

- 1.

Click the **Open From Server** button on the toolbar.



The Open From Server dialog box opens.



2. Enter a server, owner, and name and click **OK**. The procedure opens.

Printing procedures

You can print the text of a procedure.

Previewing a procedure

1. Open a procedure. The procedure commands appear.
2. From the **File** menu, click **Page Setup**. The Page Setup Dialog box opens.
3. Make any changes you want to the layout of the page and click **OK**.

4. Click the **Print Preview** button on the toolbar:



A preview of the printed procedure appears.

Printing a procedure

1. Open a procedure. The procedure commands appear.
2. From the **File** menu, click **Page Setup**. The Page Setup Dialog box opens.
3. Make any changes you that want to the layout of the page and click **OK**.
4. Click the **Print** button on the toolbar:



The procedure is printed.

Chapter 7. Working with Lists

Lists provide you with an easy way to view collections of QMF objects.

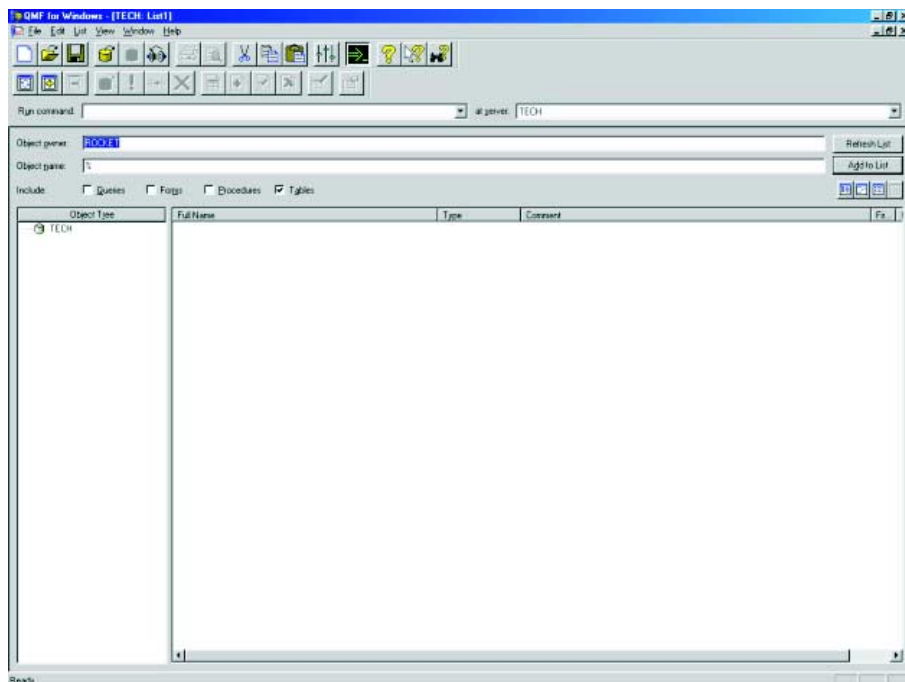
Objects

QMF for Windows recognizes four types of objects: queries, forms, procedures, and tables. You can use the List window to view the objects based on the object name, owner, and type.

Listing objects

1. From the **File** menu, select **New List**.

The List window opens.



2. Specify an owner and a name.

Note: You can use patterns to select objects from a list of matching objects.

- Use the percent character (%) to match a string of any length containing any characters. For example, to list all tables with a name beginning with the letter A, enter A%.

- Use the underscore character (`_`) to match a single character. For example, to list all tables with an owner that has the letter A in the second position, enter `_A%`.
3. Select the type of object for which you are searching.
 4. Click **Refresh List**. A list of matching objects saved at the database server is displayed.

List window commands

Right-clicking on an object in the List window displays a list of commands identical to those in the List menu.

Display object

Opens the selected object for viewing. Available for queries, forms, procedures, and tables.

Run object

Runs the selected object. Available for queries and procedures.

Draw object

Creates a query based on the selected table. You can choose to draw an SQL SELECT query, an SQL UPDATE query, an SQL INSERT query, or a prompted query. Available for tables.

Edit object

Opens the selected object for editing. Available for tables.

Properties

Displays the properties of the selected object, including comments, attributes, and historical usage information. Available for queries, forms, procedures, and tables.

Creating lists

You can create lists to serve as collections of objects. For example, you could create a list of all inventory related queries, forms, procedures and tables to keep your work in one place. Once created, you can add and remove objects from the list and save the list for future use.

Adding objects to lists

You can add objects to lists.

From an open list, specify the owner and name information of the objects that you want to add and click the **Add to List** button on the toolbar.



The objects matching the owner and name are added to the list.

Removing objects from lists

You can remove unrelated objects from lists.

From an open list, click the **Remove** button on the toolbar.



The object is removed from the list, but is not deleted.

Saving lists to files

1. From an open list, click the **Save** button on the toolbar.



If the list has been saved before, the list is saved. If the list has not been previously saved, the Save As dialog box opens.

2. Enter the name of the file where you want the list stored and click **OK**. The list is saved.

Opening saved list files

1. Click the **Open** button on the toolbar.



The Open dialog box opens.

2. Select the file that you want to open and click **OK**. The selected list opens in a list document.

Chapter 8. Working with Job Files

You can schedule and run procedures using job files. Job files use the Windows scheduler to run procedures according to your preset time and date.

Job files

You can create job files and store them locally or at the database server.

Creating job files

1. From the **File** menu, select **New Job**.
A new job document opens.

Running job files

You can run job files that have been saved locally.

1. Open a job file.
2. Click the **Run Job** button on the toolbar.



3. Drag the divider line from side to side or up and down to resize the column or row.

Note: If you save the query after resizing its rows or columns, that query is always displayed using the new formatting.

Auto fitting columns and rows

You can automatically size columns and rows to fit the data they contain.

Using the mouse, select an entire column or row and double-click on the black dividing line between it and the adjacent object. The column or row is automatically resized to fit its data.

Note: If you save the query after resizing its rows or columns, that query is always displayed using the new formatting.

Sorting query results

Once you have run a query, you can sort the results alphabetically by column.

From the Results view of a query, select a column and select **Sort Ascending** from the **Results** menu.

The query results are sorted in ascending order.

-or-

From the Results view of a query, select a column and select **Sort Descending** from the **Results** menu.

The query results are sorted in descending order.

Note: To apply more complex sorting to the selected column, select **Sort** from the **Results** menu.

Reordering columns

You can change the order of columns in the query results.

From the Results view of a query, select a column and drag it to its new location.

The column appears in the new order.

Formatting query results

You can change the font used to display queries and query results. The choice of fonts varies according to what has been installed on your computer. For more information on adding fonts, refer to your operating system's help facility.

Note: If you save the query after selecting a new query results display font, those results are always displayed using the new font.

Selecting the query results display font

1. From the Results view, select **Set Font** from the **Results** menu. The Font dialog box opens.
2. Select the font and type size for displaying the results of the query and click **OK**. The query results are displayed in the format that you specified.

Note: Click **Set As Default** to use the selected font as the default font for all query results.

Formatting numeric query results

1. From the Results view, select a column containing numeric values and select **Format** from the **Results** menu. The Format dialog box opens.
2. Specify the formatting that you want to apply and click **OK**. The values

Note: Click **Set As Default** to use the selected font as the default font for all query results.

Converting query results formatting to a form

You can convert query results formatting to a form.

1. From the **Results** menu, select **Display Report**.

The Select Form dialog box opens.

2. Select From query and click **OK**.

The query results formatting is converted to a form and opened in a new form window.

Grouping and aggregating query results

You can apply grouping, aggregation and summary formatting to query results.

Grouping query results

You can group query results with or without summary information.

1. Select the column that you want to group.
2. From the **Results** menu, select the type of grouping that you would like to apply.

The column is grouped according to your selection.

Summarizing query results

You can summarize query results by column.

1. Select the column that you want to group.
2. From the **Results** menu, select the type of summary that you would like to apply.

The column is summarized according to your selection.

Saving query results and formatting

You can save query results and save the formatting as a form.

Saving query results as a table

You can save query results as a table at a database server.

1. From the **Results** menu, select **Save to Database**.

The Save Data dialog box opens.

2. Enter an owner and table name and click **OK**.

The query results are saved as a table at the database.

Saving query results to files

You can save query results to a file on your PC or a file server.

1. From the **Results** menu, select **Save to File**.

The Export Data dialog box opens.

2. Specify the location to which you want to save the file, any export options, and click **OK**.

The query results are saved to a file.

Printing query results

You can preview and print your query results.

Previewing query results

1. Open and run a query. The query results appear.
2. From the **File** menu, select **Page Setup**. The Page Setup Dialog box opens.
3. Make any changes you want to the layout of the page and click **OK**.
4. Click the **Print Preview** button on the toolbar:



A preview of the printed query results appears.

Printing query results

1. Open a query and activate the Results view. The query results appear.
2. From the **File** menu, select **Page Setup**. The Page Setup Dialog box appears.
3. Make any changes that you want to the layout of the page and click **OK**.
4. Click the **Print** button on the toolbar:



The query results are printed.

Chapter 9. Working with static queries

A static query is an SQL query that has been previously passed to a database server and bound into a package. When a static query is run, the database server uses the SQL text bound in the package, rather than the SQL text currently appearing in the query window. Static queries are more resource efficient than dynamic queries, but static queries cannot be edited.

Static queries

Static queries are created from previously existing SQL and prompted queries.

Creating static queries

1. From the **Query** menu, select **Bind Static Package**. The Bind Static Package dialog box opens.

The screenshot shows the 'Bind Static Package' dialog box with the 'Package' tab selected. The 'Collection Name', 'Package Name', and 'Owner ID' fields are empty. The 'Replace existing package (if one exists)?' and 'Keep existing authorizations on package?' checkboxes are checked. The 'Decimal Delimiter' is set to 'Period (.)' and the 'String Delimiter' is set to 'Apostrophe (')'. The 'Isolation Level' is set to 'Cursor Stability'. The 'OK', 'Cancel', and 'Help' buttons are visible at the bottom.

2. Select the **Package** tab, enter a collection ID and package name and change any other desired options.
3. If the query contains any substitution variables, select the **Variables** tab. Replace any substitution variables with host variables.
4. Click **OK**. The static query is bound.

Note: After you bind a query, you must also save that query either to a file or to the database server.

Replacing substitution variables with host variables

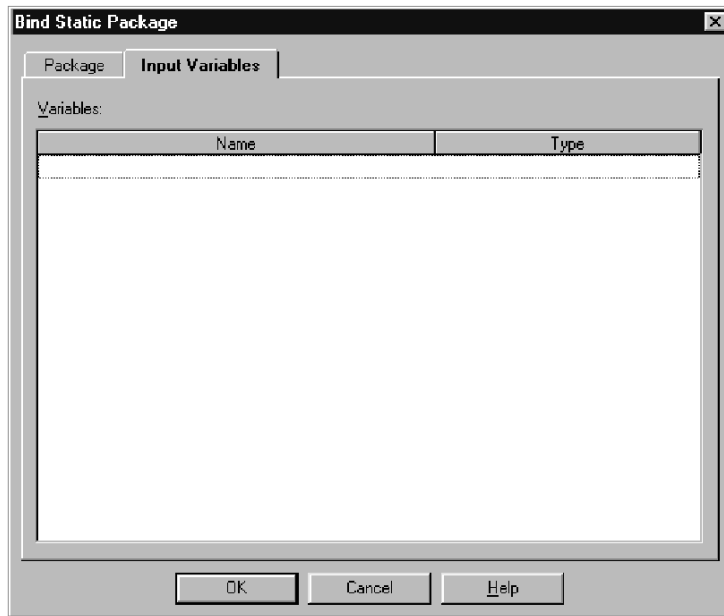
When you bind a package, you must specify a host variable to use in place of each substitution variable in the SQL text. However, a substitution variable cannot always be directly replaced by a host variable. Substitution variables provide direct text substitution in the query text before the text is sent to the database server. Host variables are sent as part of the query to the database server. Refer to the documentation for your database server for rules on where and how host variables can be used in queries.

After you specify a relationship between a substitution variable and a host variable, QMF for Windows remembers the relationship, and suggests it in future queries, making binding packages simpler.

The valid data types for host variables are:

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

1. From the Bind Static Package dialog box, select the Input Variables tab.



2. Enter the type of variable for each host variable and click **OK**. The substitution variables are converted to host variables.

Running a static query

You run static queries as you would any other queries. See "SQL Queries" on page "SQL queries" on page 7.

Chapter 10. Working with the Table Editor

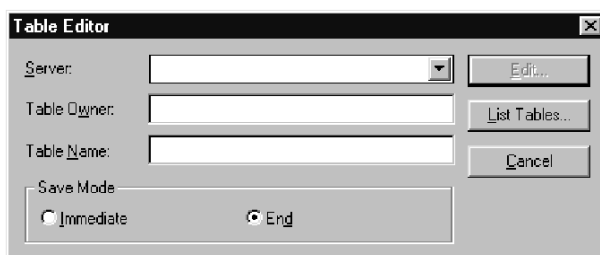
Use the table editor to search for, add, edit, or delete data stored in your tables without having to write SQL statements.

Table editor

The table editor gives you flexibility in editing and searching your data.

Searching for rows using the table editor

1. From the **File** menu, select **Table Editor**. The Table Editor dialog box opens.



2. Specify a table.

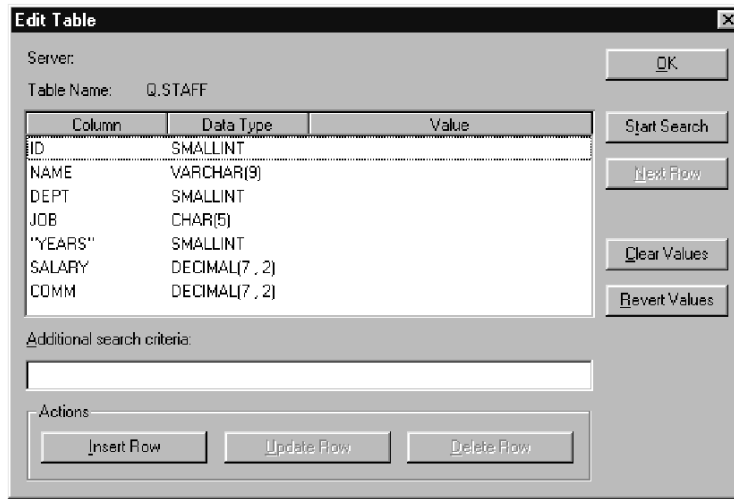
Note: You can use patterns to select table names from a list of matching tables.

- Use the percent character (%) to match a string of any length containing any characters. For example, to list all tables with a name beginning with the letter A, enter A%.
- Use the underscore character (_) to match a single character. For example, to list all tables with an owner that has the letter A in the second position, enter _A%.

Once you have entered a pattern, click **List Tables** and select a table from the resulting list.

3. Select a Save Mode.
 - Immediate - The table is updated at the database server immediately after each change.
 - End - The table is updated at the database server after you finish entering all changes. Other users will not be able to make changes to

the table while you are making your changes.



4. Click **Edit**. The Edit table dialog box opens.
5. Enter the values for which you want to search in the Value column, or type search criteria in the Additional search criteria field to specify more complex search conditions. You can enter any valid SQL predicate in the Additional search criteria field.
6. Click **Start Search**. The first matching row is displayed in the Value column.

Adding a row

1. In the Edit Table dialog box, enter the information for the new record.
2. Click **Insert Row**. The new row is added to the table.
3. Click **OK**. Your changes are saved.

Changing a row

1. From the Edit Table dialog box, search for the row you want to change.
2. Click **Next Row** until the row you want to change is displayed.
3. Edit the data in the Value column and click **Update Row**. The row is updated.
4. Click **OK**. Your changes are saved.

Deleting a row

1. From the Edit Table dialog box, search for the row that you want to delete.
2. Click **Next Row** until the row you want to delete is displayed.
3. Click **Delete Row**. The row is deleted.
4. Click **OK**. Your changes are saved.

Editing tables from the query results view

You can edit tables directly from the query results view.

Deleting a row from the query results view

You can delete individual rows from tables in the query results view.

From the query results view, select a row and select **Delete** from the **Edit** menu. The row is deleted.

Updating columns from the query results view

You can update the contents of individual columns in the query results view.

From the query results view, double click on a cell, enter a new value, and press Enter. The table is updated.

DB2 Forms

If you have the DB2 Forms User component installed on your machine, you can use it as the table editor for tables that do not contain LOB data. For more information on DB2 Forms, please visit the Resource Center for DB2 Forms at www.rocketsoftware.com/db2forms.

Chapter 11. Distributing Data

You can export your data to other databases and applications.

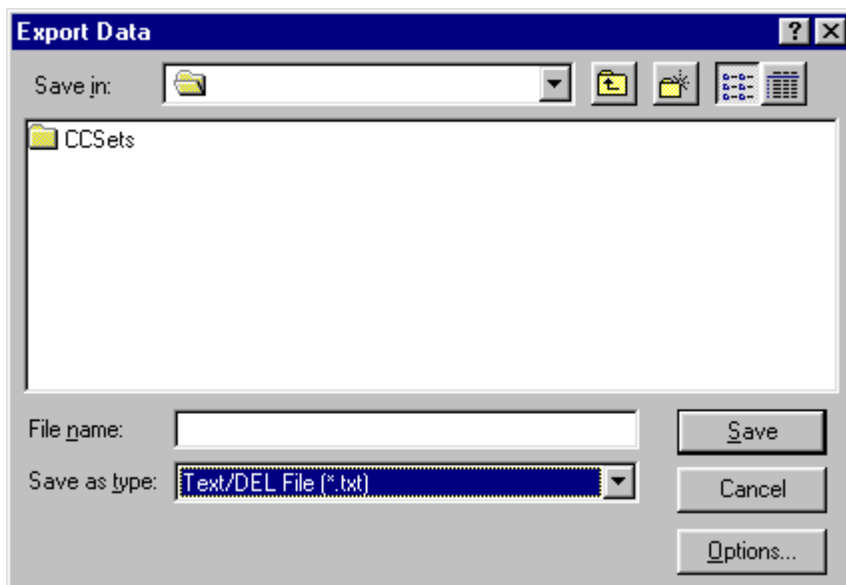
Exporting data

You can export data from QMF for Windows into other applications in the following ways:

- Export the data into a Text, CSV, IXF, or HTML file
- Save query results to a table
- Add query results directly into a Microsoft Excel spreadsheet

Exporting data to files

1. While viewing query results, select **Export Data** on the **File** menu. The Export Data dialog box opens.



2. Select the desired Output File Type and click the **Options** button. Depending on the type of output file you select, either the Export Text/DEL Options dialog box, the Export HTML Options dialog box, the Export IXF Options dialog box, or the Export CSV Options dialog box opens.

- You can produce a text file with a .TXT extension. This is a standard ASCII file with optional string and column delimiters (as specified in the Export Text/DEL Options dialog box).
 - You can produce an HTML file with an .HTM extension file. This is an HTML file that can be viewed by any web browser. All of the HTML tags are automatically generated in the file; it is ready to be published on your Internet or intranet web site. The options you choose on the Export HTML Options dialog box control the appearance of the exported data.
 - You can produce a .IXF file. An IXF export preserves all database information, including column headings and data types. It is typically used to transfer information from one database to another.
 - You can produce a .CSV file. A CSV export is very similar to a text export, using a comma as the column delimiter. This format is most commonly used by spreadsheet applications.
3. Select the options for the selected type of export file and click **OK**. The Options dialog box closes.
 4. Click **OK** on the Export Data dialog box. The data is exported.

Importing data

You can import data that has been saved in a IXF file. Once data is imported into a query window, it can be saved to a database server, exported to a new file, or used for reports. PCIXF and character mode System/370 IXF files are supported.

1. From the **File** menu, select **Import Data**. The Import Data dialog box opens.



2. Select the file you want to import and click **OK**. The imported data is displayed in a new query window.

Saving data to a database server

You can save imported query results to a database table.

1. While viewing imported query results, select **Save Data** on the **File** menu. The Save Data dialog box opens..

The 'Save Data' dialog box is shown with the following details:

- Server:** iDolphin
- Table owner:** q
- Table name:** (empty)
- Comment:** (empty)
- Table space:** (empty)
- Default table space:** (empty)
- Method:**
 - Regular (save data that is already retrieved)
 - Fast (retrieve and save data only at the server)
- Source Data Option:**
 - Save all of the data
 - Save only the currently selected data
- Existing Data Option:**
 - Replace any existing data
 - Add to any existing data
- Commit scope:** (empty)

2. Select a database server, enter a table owner and name, choose any other desired options, and click **OK**. The data is saved.

Using the Send To command

QMF for Windows includes a Send To command and a basic email client. You can use the Send To command in conjunction with job files to schedule queries and distribute their results.

1. From the **File** menu, select **Send To** and **Internet Mail Recipient**. The Message dialog box opens.
2. Specify a message recipient, a subject, the text of the message, and click **Next**. The Attachments dialog box opens.
3. Add or remove any attachments to the message and click **Next**. The Send Message dialog box opens.
4. Specify the name of your mail server and click **Finish**. The message is sent.

Using the Microsoft Excel Add-In

QMF for Windows includes an add-in for Microsoft Excel 7.0 or later. These add-ins enable you to run QMF for Windows from Excel and return query results directly into a spreadsheet. The appropriate add-in is automatically installed if you choose the "Typical" installation option, or if you choose the "Custom" installation option and select the Microsoft Excel Add-In option.

1. Click the **QMF for Windows** button on the Excel toolbar.



QMF for Windows opens.

2. From QMF for Windows, select and run a query. The query results appear.
3. Select the data you want to return to Excel.
4. From the **File** menu, select **Return data to Microsoft Excel**. Excel opens and displays the QMF for Windows Add-In dialog box.
5. Enter the destination range for the data and click **OK**. The data is added to the spreadsheet.

Using Sample Applications

Several sample applications and integration solutions are available for QMF for Windows. Visit the IBM web site at <http://www.ibm.com/qmf/> to find out more.

Chapter 12. Using QMF Report Center

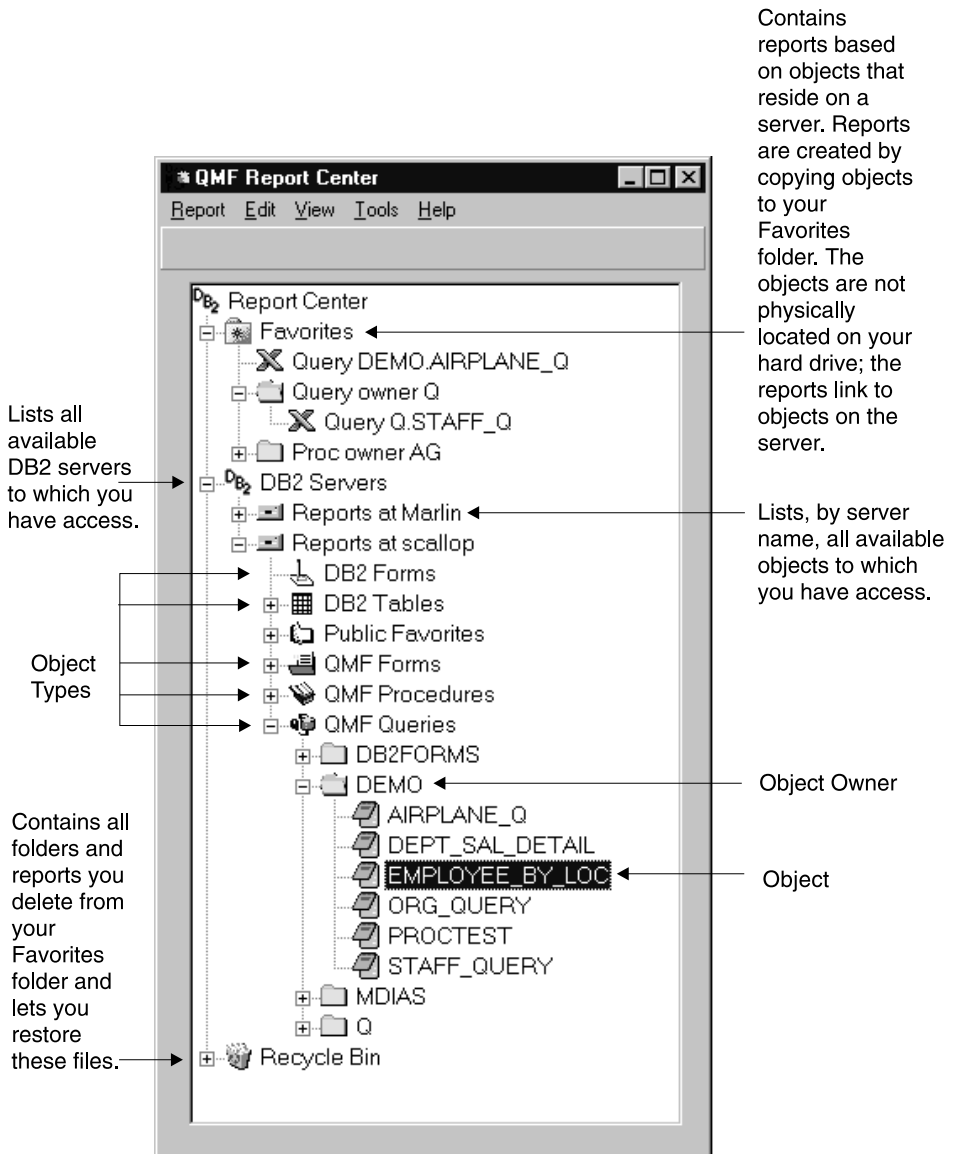
QMF Report Center lets you produce custom reports by using shared QMF queries, forms, procedures, and tables. With quick access to these objects, you can specify data format preferences and produce custom reports that can be viewed and manipulated in a variety of applications.

Getting Started in QMF Report Center

- Right-click on any object or folder to activate the same options that are available from the toolbar menus.
- Click the plus symbol (+) next to any folder to open the first level of contents. Hold the SHIFT key while clicking the plus symbol (+) to open all levels beneath the folder.

QMF Report Center Window

The QMF Report Center window contains a tree-like structure of available Favorites, DB2 servers, Public Favorites, objects, and a Recycle Bin.



Lists all available DB2 servers to which you have access.

Object Types

Contains all folders and reports you delete from your Favorites folder and lets you restore these files.

Contains reports based on objects that reside on a server. Reports are created by copying objects to your Favorites folder. The objects are not physically located on your hard drive; the reports link to objects on the server.

Lists, by server name, all available objects to which you have access.

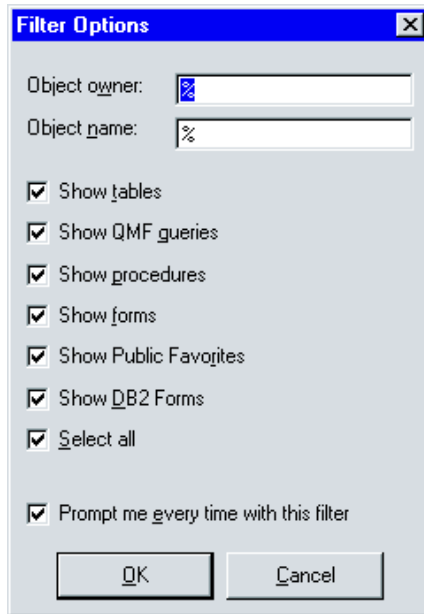
Object Owner

Object

Objects displayed in this window include an icon that represents the type of application with which the object output is associated.

Connecting to the Server

1. If no server names are displayed beneath DB2 Servers, click the plus symbol (+).
2. Click the plus symbol (+) next to a server. The Filter Options dialog box opens.



3. Select the object types that you want to see, then click **OK**. The available objects on the server are displayed, grouped by object type.

Working with Reports and Objects

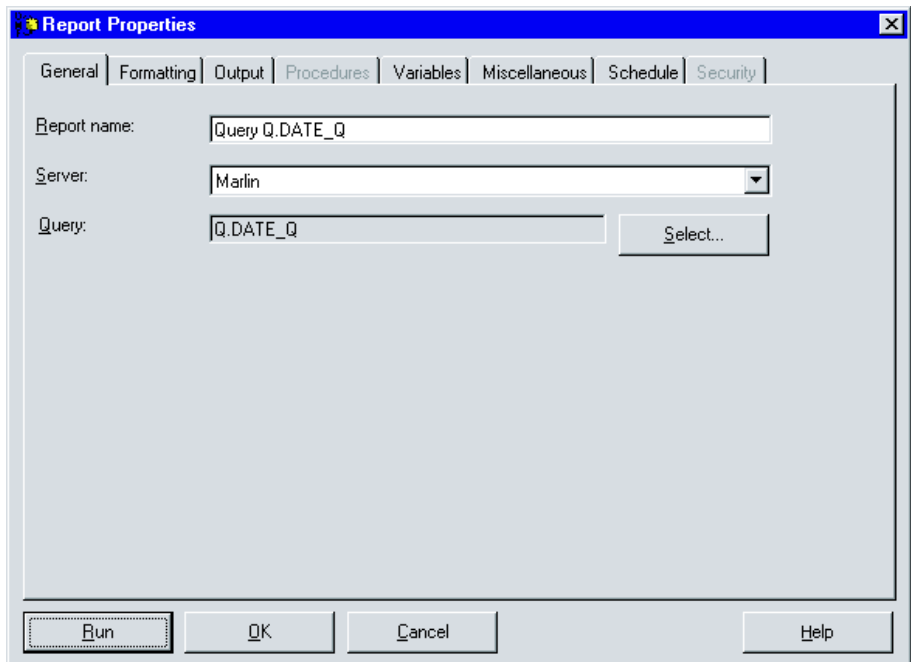
Reports are based on QMF objects. All items contained in your personal and Public Favorites folders are considered reports; you can manipulate formatting and display options for these items. The items contained in these Favorites folders link to the QMF objects that reside on the server. You do not actually modify a QMF object, you modify the link to the object that is referred to as a report. Since reports are based on objects, properties of objects also apply to reports.

You can create reports from objects that reside on a server; however, they are not saved to the server. This functionality allows you to quickly create one-time reports. After creating a report from objects on a server, however, you are given the option to save the report to your Favorites folder.

Running Reports

You can run reports from your Favorites folders or from objects located on the server.

1. With the report or object selected, choose *Properties* from the Report menu. The Report Properties dialog box opens.



2. Define properties, if desired.
3. Click the **Run** button. The report is processed and, if you selected the *View report after publishing* option in the Report Properties Output dialog box, the report is displayed in the application you specified.

You can also quickly run a report by any of the following methods:

- Select the report, then choose *Run* from the Report menu.
- Right-click on the report, then choose *Run*.
- Double-click on the report name.

Working with Folders and Favorites

Folders are used to group reports and QMF objects; folders are named according to object owner names. You can perform the same operations on folders that you do on reports, such as running reports and defining report properties. Performing these operations on a folder applies the operation to

every report contained within the folder. For example, if you want to consecutively run every report contained within a folder, select the folder, then select **Run** from the Report menu.

QMF Report Center contains two top-level folders in which you can store reports. The folders contain reports that point to objects on servers; the objects themselves are not contained within the Favorites folders. Your personal Favorites folder resides locally (on your PC), so you are the only user who can access the folder and its contents. The Public Favorites folder resides on the server and can be accessed by all authorized users. You may have access to several Public Favorites folders, depending on your resource limits, though there will never be more than one public Favorites folder on each server.

When you copy QMF objects to the Favorites folders, the folders are automatically renamed to include the object type and owner name. When you copy an entire object type (i.e., folder of same-type objects) from a server (e.g., all queries), the server name is also included in the new folder name.

Adding Reports to Favorites

You can add any object or report from a server to your personal Favorites folder, or to the Public Favorites folder on a server (provided you have been granted permission by your system administrator).

To add reports to personal Favorites:

With the report or object selected, choose *Add to favorites* from the Report menu, or drag the report or object to your personal Favorites folder. A report is added to the top of your personal Favorites folder with the following naming convention: ObjecttypeOWNERNAME.OBJECTNAME.

To add reports to Public Favorites:

Drag the QMF object or report to the Public Favorites folder on a server. You can add reports from your personal Favorites folder or from any server.

Note: When adding to Public Favorites or modifying reports in Public Favorites, you must select *Save changes to Public Favorites* from the Report menu before your updates are saved to the server.

For more information on using QMF Report Center, see the online help system.

Chapter 13. Using the QMF for Windows API

You can create custom applications using the QMF for Windows application programming interface.

Controlling QMF for Windows through the API

The following steps provide an overview of how you work with the API to control QMF for Windows.

1. Create an instance of the QMF for Windows API object. If you are using Microsoft Visual Basic, add a reference to the QMF for Windows type library, `qmfwin.tlb`. Then use the `Dim` statement:

```
Dim QMFWin As New QMFWin
```

Or the `CreateObject` statement:

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

- Note:** If you are using a different development environment, refer to your product documentation on how to complete this step.
2. Select the DB2 server you want to use and call `InitializeServer()` to initialize a connection to the database.

Note: You cannot initialize a server until a user ID and password are validated by DB2. You can have QMF for Windows prompt for the user ID and password, or you can prompt for them in your application and pass them as parameters in the `InitializeServer()` function call.

3. Select the query you want to run using `InitializeQuery()`. If the query contains variables, use the `SetVariable()` function to set the variable values.
4. Open or execute the query. Use the `Open()` function to open the query's cursor for `SELECT` statements, and use the `Execute()` function to execute the SQL for non-`SELECT` statements.
5. If the query is a `SELECT` statement, fetch rows of data by repeatedly calling `FetchNextRow()`. To fetch more than one row at a time, use `FetchNextRows()`, or use `CompleteQuery()` to direct QMF for Windows to fetch all of the rows.
6. If the query is a `SELECT` statement, close the query by using the `Close()` function.
7. Terminate the unit-of-work using the `Commit()` or `Rollback()` functions.

Blocking calls

All of the QMF for Windows API functions are synchronous. This means that they block, or do not return, until the requested database action completes. This implementation is desirable because it simplifies programming in the client application. However, if your client application is single-threaded, it cannot respond to user input or perform screen refreshes while it is waiting for a QMF for Windows API function to return.

The QMF for Windows API responds to one function call at a time from a client. If your client application is multi-threaded, 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).

Connecting to the database

Each instance of the QMF for Windows API object creates and uses a single connection to the database for all database activity that is subject to a subsequent rollback or commit, including opening a query, fetching data, and executing SQL statements.

If you create more than one query in a given instance of the QMF for Windows API object by calling `InitializeQuery()` two or more times, all the queries share the same single connection.

The QMF for Windows API responds to one function call at a time from a client. If your client application is multi-threaded, you must:

- `DeleteQMFObject()`
- `GetQMFOBJECTInfo()`
- `GetQMFOBJECTInfoEx()`
- `GetQMFOBJECTList()`
- `GetQMFOBJECTListEx()`
- `GetQMFQueryText()`
- `SaveQMFQuery()`

QMF for Windows creates and uses a second connection to the database in order to handle administrative database activity (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.

API Reference

This reference lists all the available commands for creating applications using the QMF for Windows API.

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

Name	Description
<i>QueryID</i>	The ID of the query as returned from InitializeStaticQuery().
<i>Type</i>	The SQL data type of the value to be passed to the database server. This value influences the conversion of <i>Value</i> from a VARIANT data type to the value actually passed. The only valid value for AddDecimalHostVariable() is 484 (RSDT_DECIMAL).
<i>Precision</i>	The precision of the decimal value.
<i>Scale</i>	The scale of the decimal value.
<i>Value</i>	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.

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

Name	Description
<i>QueryID</i>	The ID of the query as returned from <code>InitializeStaticQuery()</code> .
<i>Type</i>	The SQL data type of the value to be passed to the database server. This value influences the conversion of <i>Value</i> from a VARIANT data type to the value actually passed.
<i>Value</i>	The data value to substitute in the statement. To specify a null value, the type of the variant should be set to <code>VT_EMPTY</code> .

Valid values for *Type* include:

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.

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

Name	Description
<i>CollectionName</i>	The collection ID of the package you want to bind.
<i>PackageName</i>	The name of the package you want to bind.
<i>SectionNumber</i>	The section number of the statement within the collection and package you want to bind.
<i>Number</i>	The identifier for the variable you want to bind. The first variable in the SQL statement is variable 0, etc.
<i>Name</i>	Used by the database server for diagnostic purposes. This value is not validated nor required by QMF for Windows.
<i>DataType</i>	The SQL data type of the variable. The only valid value for <code>BindDecimalHostVariable()</code> is 484 (<code>RSDT_DECIMAL</code>).
<i>Precision</i>	The precision of the decimal value.
<i>Scale</i>	The scale of the decimal value.

Return Value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

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

Name	Description
<i>CollectionName</i>	The collection ID of the package you want to bind.
<i>PackageName</i>	The name of the package you want to bind.
<i>SectionNumber</i>	The section number of the statement within the collection and package you want to bind.
<i>Number</i>	The identifier for the variable you want to bind. The first variable in the SQL statement is variable 0, etc.
<i>Name</i>	Used by the database server for diagnostic purposes. This value is not validated nor required by QMF for Windows.

<i>DataType</i>	The SQL data type of the variable.
<i>Length</i>	The length of the variable.

Valid values for *DataType* include:

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
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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

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

Name	Description
<i>CollectionName</i>	The collection ID of the package you want to bind.
<i>PackageName</i>	The name of the package you want to bind.
<i>SectionNumber</i>	The section number of the statement within the collection and package you want to bind.
<i>SQLText</i>	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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

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

Name	Description
<code>CollectionName</code>	The collection ID of the package you want to bind.
<code>PackageName</code>	The name of the package you want to bind.

Return Value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

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

Name	Description
<code>NewPassword</code>	The new password.

Return Value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

ClearList()

short `ClearList(short Type)`

Description

This function re-initializes the internal list specified by the *Type* parameter.

Parameters

Name	Description
<i>Type</i>	Either the value RSL_SERVER or RSL_QUERY.

Return Value

Zero if successful, RS_ERROR_OUTOFRANGE if unsuccessful.

Related Topics

Open()

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 Close. If you are using MS Access 2.0, place square brackets [] around the function name.

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() or GetLastErrorType(), GetLastErrorCode(), GetLastErrorError(), or GetLastErrorState() to get additional error information.

Related Topics

Execute()

Open()

Commit()

short Close(long *QueryID*)

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 Commit. 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() or GetLastErrorType(), GetLastSQLCode(), GetLastSQLError(), or GetLastSQLState() to get additional error information.

Related Topics

Rollback()

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 can use FetchNextRow() or FetchNextRows() to retrieve the rows. Call Close() when you are done with this query.

Parameters

Name	Description
<i>QueryID</i>	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() or GetLastErrorType(), GetLastSQLCode(), GetLastSQLError(), or GetLastSQLState() to get additional error information.

CopyToClipboard()

short CopyToClipboard(long *QueryID*, long *FirstRow*, long *FirstCol*, long *LastRow*, long *LastCol*, BOOL *IncludeColHeadings*, [VARIANT *DateFomat*])

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 is returned if you attempt to copy rows that have not been retrieved from the database.

Parameters

Name	Description
<i>QueryID</i>	The ID of the query as returned from InitializeQuery().
<i>FirstRow</i>	The first row you want to include in the copy.
<i>FirstCol</i>	The first column you want to include in the copy.
<i>LastRow</i>	The last row you want to include in the copy, or -1 if all rows are included.
<i>LastCol</i>	The last column you want to include in the copy, or -1 if all columns are included.
<i>IncludeColHeadings</i>	Use nonzero to include the column headings in the first row and zero to not include them.
<i>DateTimeFormat</i>	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 the 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.

DeleteQMFObject()

short DeleteQMFObject(BSTR *OwnerAndName*)

Description

This function deletes a QMF object (query, form, procedure, or table).

Parameters

Name	Description
<i>OwnerAndName</i>	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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

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

Name	Description
<i>CollectionName</i>	The collection name used in the previous call to <code>StartBind()</code> .
<i>PackageName</i>	The package name used in the previous call to <code>StartBind()</code> .

Return Value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

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

Name	Description
<i>QueryID</i>	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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

Related Topics

`Execute()`

`Open()`

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

Name	Description
<i>QueryID</i>	The ID of the query as returned from <code>InitializeQuery()</code> .
<i>Result</i>	<p>A pointer to a <code>VARIANT</code> in which the result is stored. The result is an array (variant type <code>VT_ARRAY</code> <code>VT_VARIANT</code>) containing one value for each column in the result.</p> <p>Each value is specified in either its native data type or the closest variant data type. The supported return types are: string (variant type <code>VT_BSTR</code>), float (variant type <code>VT_R4</code>), double (variant type <code>VT_R8</code>), short (variant type <code>VT_I2</code>), long (variant type <code>VT_I4</code>), and binary (variant type <code>VT_UI1</code> <code>VT_ARRAY</code>).</p> <p>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>.</p>

Return Value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

ExecuteStored Procedure()

`short ExecuteStoredProcedure(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 variables.

Parameters

Name	Description
<i>QueryID</i>	The ID of the query as returned from <code>InitializeQuery()</code> . The SQL text for the query should specify a CALL statement.
<i>vaCommitOK</i>	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.
<i>vaMaxResultSets</i>	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.
<i>vaColumnNames</i>	An optional Boolean value specifying whether or not the database should return column names for the columns in each returned result set.
<i>vaColumnLabels</i>	An optional Boolean value specifying whether or not the database should return column labels for the columns in each returned result set.
<i>vaColumnComments</i>	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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

ExecuteStored ProcedureEx()

short ExecuteStoredProcedureEx(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 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 variables.

If the stored procedure returns result sets, call GetStoredProcedureResultSets() to retrieve the query IDs for the result sets.

Parameters

Name	Description
<i>QueryID</i>	The ID of the query as returned from InitializeQuery(). The SQL text for the query should specify a CALL statement.
<i>Result</i>	<p>A pointer to a VARIANT in which the result is stored. The result is an array (variant type VT_ARRAY VT_VARIANT) containing one value for each column in the result.</p> <p>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>You must properly initialize the VARIANT before calling this function. Visual Basic does this automatically. Visual C++ programmers must call VariantInit().</p>
<i>vaCommitOK</i>	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.

<i>vaMaxResultSets</i>	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.
<i>vaColumnNames</i>	An optional Boolean value specifying whether or not the database should return column names for the columns in each returned result set.
<i>vaColumnLabels</i>	An optional Boolean value specifying whether or not the database should return column labels for the columns in each returned result set.
<i>vaColumnComments</i>	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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

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 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 is returned if you attempt to copy rows that have not been retrieved from the database.

Parameters

Name	Description
<i>QueryID</i>	The ID of the query as returned from <code>InitializeQuery()</code> .
<i>FirstRow</i>	The first row you want to include in the export.
<i>FirstCol</i>	The first column you want to include in the export.
<i>LastRow</i>	The last row you want to include in the copy, or -1 if all rows are included.
<i>LastCol</i>	The last column you want to include in the copy, or -1 if all columns are included.

<i>IncludeColHeadings</i>	Use nonzero to include the column headings in the first row and zero to not include them.
<i>Filename</i>	A string containing the name of the file to which you want to write the export.
<i>DateTimeFormat</i>	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 the 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.

Name	Description
<i>Format</i>	Specifies the output format.

Value	Meaning
0 (RSEF_TEXT)	The output file is written in plain text format.
1 (RSEF_HTML)	The output file is written in HTML format, and the data is organized in an HTML table.
2 (RSEF_CSV)	The output file is written in CSV (comma separated values) format.
3 (RSEF_PCIXF)	The output file is written in PC/IXF format.
4 (RSEF_S370IXF)	The output file is written in System/370 IXF format.

Name	Description
<i>String Delimiter</i>	Specifies the string delimiter. This parameter is ignored if <i>Format</i> is RSEF_HTML.

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 (").

Name	Description
<i>Column Delimiter</i>	Specifies the column delimiter. This parameter is ignored if <i>Format</i> is RSEF_HTML.

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()`

ExportForm()

short `ExportForm(BSTR OwnerAndName, BSTR FileName)`

Description

This function exports the specified QMF form to the specified file.

Parameters

Name	Description
<i>OwnerAndName</i>	A string containing the owner and name, separated by a period, of the form that you want to export. For example, <code>John.Query2</code>
<i>FileName</i>	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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

Related Topics

`PrintReport()`

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

Name	Description
<i>QueryID</i>	The ID of the query as returned from <code>InitializeQuery()</code> .
<i>Source</i>	The name (either a filename or <code>Owner.Name</code>) of the form you want to use.
<i>OutputFileName</i>	The name of the file to which you want to write the report.
<i>PageLength</i>	The number of lines on each page of the report. A <i>PageLength</i> of -1 specifies continuous output (no page breaks unless the report is wider than <i>PageWidth</i>).
<i>IncludeDateTime</i>	Nonzero specifies that the date and time are included at the bottom of each page. Zero specifies that the date and time are not included.
<i>IncludePageNumbers</i>	Nonzero specifies that page numbers are included at the bottom of each page. Zero specifies that page numbers are not included.
<i>Format</i>	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.
<i>DateTimeFormat</i>	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.
<i>Format</i>	The format of the output file.
<i>UseFormPageSetup</i>	Optionally, if nonzero specifies that the <i>PageLength</i> , <i>PageWidth</i> , <i>IncludeDateTime</i> , and <i>IncludePageNumbers</i> 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.
Value	Meaning
0 (RSF_DEFAULT)	Use the default form. <i>FormName</i> should be an empty string.

1 (RSF_DATABASE)	Use a form from the database. Specify the form owner and name (Owner.Name) in the <i>FormName</i> 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 <i>SourceType</i> of RSF_FILE.
2 (RSF_FILE)	Use a form contained in a file. Specify the file name in the <i>FormName</i> parameter.

Return Value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call GetLastErrorString() or GetLastErrorType(), GetLastSQLCode(), GetLastSQLError(), or GetLastSQLState() to get additional error information.

Related Topics

ExportForm()

FastSaveData()

short FastSaveData(long *QueryID*, BOOL *Replace*, BSTR *Tablename*, BSTR *TableSpaceName*, [VARIANT *Comment*])

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

Name	Description
<i>QueryID</i>	The ID of the query as returned from InitializeQuery().
<i>Replace</i>	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.
<i>TableName</i>	The name of the table in which you want to store the data. If the table does not exist, QMF for Windows creates it.
<i>TableSpaceName</i>	The name of the table space in which the table exists or is created. If <i>TableSpaceName</i> is NULL or an empty string, QMF for Windows uses 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().
<i>Comment</i>	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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

FetchNextRow()

short `FetchNextRow(long QueryID, VARIANT* Row)`

Description

This function fetches the next row of data from the database.

Parameters

Name	Description
<i>QueryID</i>	The ID of the query as returned from <code>InitializeQuery()</code> .
Row	<p>A pointer to a <code>VARIANT</code> in which the result is stored. The result is an array (variant type <code>VT_ARRAY</code> <code>VT_VARIANT</code>) containing one value for each column in the result. Call <code>GetColumnCount()</code> to determine the number of values in the array.</p> <p>Each value is specified in either its native data type or the closest variant data type. The supported return types are: string (variant type <code>VT_BSTR</code>), float (variant type <code>VT_R4</code>), double (variant type <code>VT_R8</code>), short (variant type <code>VT_I2</code>), long (variant type <code>VT_I4</code>), and binary (variant type <code>VT_UI1</code> <code>VT_ARRAY</code>).</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 <code>VT_EMPTY</code>) instead of an array .</p> <p>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>.</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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

Related Topics

`FetchNextRows()`

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

Name	Description
<i>QueryID</i>	The ID of the query as returned from <code>InitializeQuery()</code> .

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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

Related Topics

`FetchNextRowsEx()`

FetchNextRows()

short `FetchNextRows(long QueryID, VARIANT* Rows, long* NumRows)`

Description

This function fetches the next *NumRows* of data from the database.

Parameters

Name	Description
<i>QueryID</i>	The ID of the query as returned from <code>InitializeQuery()</code> .

Row	<p>A pointer to a VARIANT in which the result is 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 contains extra, unused entries).</p> <p>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 row 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() or GetLastErrorType(), GetLastErrorCode(), GetLastErrorError(), or GetLastErrorState() to get additional error information.

Related Topics

FetchNextRow()

FetchNextRowsEx()

short FetchNextRowsEx(long QueryID, long* NumRows)

Description

This function fetches the next *NumRows* 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

Name	Description
<i>QueryID</i>	The ID of the query as returned from <code>InitializeQuery()</code> .
<i>NumRows</i>	A pointer to a long containing the number of row to fetch. If the number of un fetched rows in the result set is less than <i>NumRows</i> , <i>NumRows</i> 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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

Related Topics

`FetchNextRowEx()`

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 for Windows caches QMF information obtained from the database to reduce database traffic and improve performance. You call this function prior to calling `GetQMFOBJECTInfo()`, `GetQMFQueryText()`, or `GetQMFOBJECTList()` to ensure that the information returned is up to date.

Return Value

None.

GetColumnCount()

`long GetColumnCount(long QueryID)`

Description

This function returns the number of columns in the result set.

Parameters

Name	Description
<i>QueryID</i>	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() or GetLastErrorType(), GetLastSQLCode(), GetLastSQLError(), or GetLastSQLState() to get additional error information.

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

Name	Description
<i>QueryID</i>	The ID of the query as returned from InitializeQuery().
<i>Index</i>	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() or GetLastErrorType(), GetLastSQLCode(), GetLastSQLError(), or GetLastSQLState() to get additional error information.

GetColumnHeader()

BSTR GetColumnHeader(long *QueryID*, long *Index*, short* *Result*)

Description

This function returns the column header (column name) associated with the index *Index*.

Parameters

Name	Description
<i>QueryID</i>	The ID of the query as returned from InitializeQuery().
<i>Index</i>	The zero based index of the row of data to be retrieved.

Result	Zero if successful, nonzero if unsuccessful. If Result is nonzero, you can call GetLastErrorString() or GetLastErrorType(), GetLastErrorSQLCode(), GetLastErrorSQLError(), or GetLastErrorSQLState() 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.

GetColumnHeaderEx()

short GetColumnHeaderEx(long *QueryID*, long *Index*)

Description

This function returns 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

Name	Description
<i>QueryID</i>	The ID of the query as returned from InitializeQuery().
<i>Index</i>	The zero based index of the row of data to be retrieved.

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

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() or GetLastErrorType(), GetLastErrorSQLCode(), GetLastErrorSQLError(), or GetLastErrorSQLState() to get additional error information.

GetColumnHeadings()

short GetColumnHeadings(long *QueryID*, VARIANT* *Headings*)

Description

This function returns the column headings (also referred to as column names).

Parameters

Name	Description
<i>QueryID</i>	The ID of the query as returned from <code>InitializeQuery()</code> .
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 <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.

Note: Column headings are not available for static SQL statements. For query IDs returned from `InitializeStaticQuery()`, `GetColumnHeadings` returns the strings "Col1", "Col2", etc.

Return Value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

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

Name	Description
<i>QueryID</i>	The ID of the query as returned from <code>InitializeQuery()</code> .
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() or GetLastErrorType(), GetLastSQLCode(), GetLastSQLError(), or GetLastSQLState() to get additional error information.

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

Name	Description
<i>QueryID</i>	The ID of the query as returned from InitializeQuery().
<i>RowIndex</i>	The zero based index of the row to be retrieved.
<i>ColIndex</i>	The zero based index of the column to be retrieved.
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() or GetLastErrorType(), GetLastSQLCode(), GetLastSQLError(), or GetLastSQLState() to get additional error information.

GetDefaultServerName()

BSTR GetDefaultServerName()

Description

This function returns a string containing the default server name.

Return Value

A string that specifies the default server name.

GetGlobalVariable()

BSTR GetGlobalVariable(BSTR *Name*)

Description

This function retrieves the value of the specified global variable.

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.

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

Name	Description
<i>QueryID</i>	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.

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 AddHostVariableble()) See AddHostVariable() for a list of the data types that can be returned.

Parameters

Name	Description
<i>QueryID</i>	The ID of the query as returned from <code>InitializeQuery()</code> .
Types	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.

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 API object, NULL is returned.

Related Topics

`GetLastErrorType()`
`GetLastSQLCode()`
`GetLastSQLError()`
`GetLastSQLState()`

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 returned value indicates the type of error:

Value	Meaning
-------	---------

0 (RS_ERROR_NONE)	No errors have occurred since the QMF for Windows API object was created.
1 (RS_ERROR_SQL)	An SQL error occurred. If the error occurred during a call to a function that takes <i>QueryID</i> as an argument, call <i>Close()</i> 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 <i>InitializeServer()</i> or <i>ReinitializeServer()</i> if you want to continue.
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 <i>InitializeServer()</i> or <i>ReinitializeServer()</i> if you want 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 <i>QueryID</i> as an argument, call <i>Close()</i> 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\(\)](#)

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 codes 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()
GetLastSQLError()
GetLastSQLState()

GetLastSQLError()

VARIANT GetLastSQLError()

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 has the following format:

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
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\(\)](#)
[GetLastSQLCode\(\)](#)
[GetLastSQLState\(\)](#)

GetLastSQLState()

BSTR GetLastSQLState()

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\(\)](#)
[GetLastSQLCode\(\)](#)
[GetLastSQLError\(\)](#)

GetOption()

short GetOption(short *Option*, VARIANT* *Value*)

Description

Gets the specified option value in QMF for Windows.

Parameters

Name	Description
<i>Option</i>	Specifies which option to retrieve.

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 by 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 by 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.

Value	A pointer to a VARIANT in which the result is stored. The result is an array (variant type VT_ARRAY VT_VARIANT) containing one value for each column in the result. Call GetColumnCount() to determine the number of values in the 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 is unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()` to get additional error information.

Related Topics

`SetOption()`

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

Name	Description
Option	The option values are the same as those for the <code>GetOption()</code> call.

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.

Related Topics

`GetOption()`

`SetOption()`

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 use in the procedure before calling this function.

Parameters

Name	Description
------	-------------

ProcID	The ID of the procedure as returned from InitializeProc().
--------	--

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.

GetProcVariables()

short GetProcVariables(long ProcID, VARIANT* Variables)

Description

Gets the specified option value in QMF for Windows.

Parameters

Name	Description
ProcID	The ID of the procedure as returned from InitializeProc().
Value	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 is unsuccessful. If there are no variables in the procedure, the return value is RS_NO_ERROR_NO_DATA (-1). If the return value is nonzero, you can call GetLastErrorString() or GetLastErrorType() to get additional error information.

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

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
Value	A pointer to a VARIANT in which the result is stored. For RSI_TIMEUSED, 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.

Type	Specifies the type of information to get.
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_TIMESCANCELLED)	Number of times cancelled.

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.

<i>Time</i>	Specifies first used, last used, or last modified.
-------------	--

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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

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

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.

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_TIMESCANCELLED)	Number of times cancelled.
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.

Time Specifies first used, last used, or last modified.

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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

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

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.
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), each of format Owner.Name. If no matching QMF for Windows 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.

<i>Type</i>	Specifies the types of QMF objects that you want to include in the list. These values can be added together to specify multiple object types.
-------------	---

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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

GetQMFObjectListEx()

short `GetQMFObjectListEx(BSTR Owner, BSTR Name, 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

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.
Index	The index of the list of QMF objects that match the pattern.
Type	Specifies the types of QMF objects that you want to include in the list. These values can be added together to specify multiple object types.
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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

GetQMFProcText()

BSTR GetQMFProcText(BSTR *OwnerAndName*)

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 use in the procedure before calling this function.

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.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(), GetLastErrorSQLCode(), GetLastErrorSQLError, or GetLastErrorSQLState() to get additional error information.

GetQMFQueryText()

BSTR GetQMFQueryText(BSTR *OwnerAndName*)

Description

This function retrieves the SQL text stored in the specified query.

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

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(), GetLastErrorSQLCode(), GetLastErrorSQLError, or GetLastErrorSQLState() to get additional error information.

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

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().

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.

GetQueryVerb()

BSTR GetQueryVerb(long *QueryID*)

Description

This function returns a string containing the SQL verb you used in the query.

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 successful, a string containing the SQL verb is returned. If unsuccessful, NULL is returned. If the return value is NULL, you can call `GetLastErrorString()` or `GetLastErrorType()` to get additional error information.

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

Name	Description
Resource	The resource values include:

Value	Meaning
0 (RSR_IDLE_CONNECTION_TIMEOUT)	Idle connection timeout (in seconds).

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 interface?
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?
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 the Save Data command allowed?
36 (RSR_SAVE_DATA_TABLE_SPACE_NAME)	The default collection name for binding packages?
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.
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)	Allow exporting of data?
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?
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 RSR_SAVE_DATA_TABLE_SPACE_NAME, RSR_DEF_COLLECTION, and RSR_ACCOUNT_STRING, -1 is returned and the <i>ResourceLimit</i> 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.

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

Name	Description
Resource	The resource values are the same as those for the <code>GetResourceLimit()</code> call.

Note: The query verb is not available for static SQL statements. For query IDs returned from `InitializeStaticQuery()`, `GetQueryVerb()` returns an empty string.

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.

GetRowCount()

`long GetRowCount(long QueryID)`

Description

This function returns the number of rows currently in QMF for Windows' 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

Name	Description
QueryID	The ID of the query as returned from <code>InitializeQuery()</code> .

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.

GetServerList()

`short GetServerList(VARIANT* List)`

Description

This function returns an array containing the names of the database servers defined in QMF for Windows' 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

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.

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

Name	Description
Index	An index into the list of servers.

Return Value

Zero if successful, RS_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_OUTOFRANGE. If the return value is nonzero, you can call GetLastErrorString() or GetLastErrorType() to get additional error information.

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

Name	Description
QueryID	The ID of the original query as returned from <code>InitializeQuery()</code> .
ResultSets	A pointer to a VARIANT 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 VARIANT 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()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

GetVariables()

short `GetVariables(long QueryID, VARIANT* Variables)`

Description

This function returns an array of the names of the variables in the SQL text of the query. You must assign values to these variables by calling `SetVariable()` prior to running the query using either `Open()` or `Execute()`.

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.

GetVariablesEx()

short `GetVariablesEx(long QueryID, short Index)`

Description

This function returns the name of the variable in the SQL text of the query 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

Name	Description
QueryID	The ID of the query as returned from <code>InitializeQuery()</code> .
Variables	An index into the internal list of variables. Query the <i>Value</i> property for the string that corresponds with the index passed in. If there are no variables in the SQL statement, the function returns <code>RS_ERROR_NO_DATA</code> .

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.

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

Name	Description
SourceType	Specifies the source for the procedure text.

Value	Meaning
0 (RSS_STRING)	The text is contained in the <i>Source</i> parameter.
2 (RSS_FILE)	The text is contained in the text file whose name is specified by the <i>Source</i> parameter.
3 (RSS_QMFPROC)	The text is contained in the procedure whose owner and name are specified by the <i>Source</i> parameter.

<i>Source</i>	A string containing the text, the owner and name (Owner.Name) of the procedure, or the name of a file containing the procedure text.
---------------	--

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.

InitializeQuery()

long InitializeQuery(short *SourceType*, BSTR *Source*)

Description

This function sets the 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

Name	Description
SourceType	Specifies the source for the SQL statement text.

Value	Meaning
0 (RSS_STRING)	The SQL text is contained in the <i>Source</i> parameter.

1 (RSS_QMFQUERY)	The SQL text is contained in the query whose owner and name are specified by the <i>Source</i> parameter.
2 (RSS_FILE)	The SQL text is contained in the text file whose name is specified by the <i>Source</i> 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 *Query* parameter.

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

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 the QMF for Windows Server Definition File. Call GetServerList() to retrieve a list of valid servers.
UserID	A string containing the User ID you want to use. If UserID is NULL or an empty string, QMF for Windows attempts to use the User ID from the most recent query, if available. Otherwise, QMF for Windows displays the User Information dialog box to obtain a User ID and password.
Password	A string containing the password for the specified user ID. If a Password is NULL or an empty string, QMF for Windows attempts to use a memorized password if available (requires Windows for Workgroups). If no password is available, QMF for Windows displays the User Information dialog box to obtain a password.
ForceDialog	Nonzero indicates that QMF for Windows displays the User Information dialog box regardless of whether a User ID and Password are 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 does not display the User Information dialog box, even if a user ID and password have not been 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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

Related Topics

`SetParent()`

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

Name	Description
<code>CollectionName</code>	The name of a previously bound collection.
<code>PackageName</code>	The name of a previously bound package.
<code>ConsistencyToken</code>	The token used by the above named collection and package.
<code>SectionNumber</code>	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.

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

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.

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_FETCHALLROWS 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 Open. 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

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 the QMF for Windows 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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

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 is available after completing `Prepare()`.

Parameters

Name	Description
<i>QueryID</i>	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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

Related Topics

`Execute()`

`Open()`

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()`.

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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

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 `Rollback`. 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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

Related Topics

`Commit()`

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

Name	Description
<i>ProcID</i>	The ID of the procedure as returned from <code>InitializeProc()</code> .

Return Value

Zero if successful, nonzero if unsuccessful. If the return value is nonzero, you can call `GetLastErrorString()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

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 fails. 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()` has no effect on changes you make using this function.

Parameters

Name	Description
<i>QueryID</i>	The ID of the query as returned from <code>InitializeQuery()</code> .
<i>FirstRow</i>	The first row that you want to include in the save. The value of the first row in a result set is 0.
<i>FirstCol</i>	The first column that you want to include in the save. The value of the first column in a result set is zero.
<i>LastRow</i>	The last row that 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.
<i>LastCol</i>	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.
<i>Replace</i>	Nonzero indicates that the specified data replaces any existing data in the table. Zero indicates that the specified data is appended to any existing data in the table.
<i>TableName</i>	The name of the table in which the data is stored. If the table doesn't exist, it is created.

<i>TableSpaceName</i>	The name of the table space in which the table exists or is created. If <i>TableSpaceName</i> 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.
<i>ServerName</i>	The name of the database server in which the table is stored. If <i>ServerName</i> is NULL or an empty string, the server you specify in the call to InitializeServer() is used, and <i>UserID</i> , <i>Password</i> , <i>ForceDialog</i> , and <i>Account</i> are ignored.
<i>UserID</i>	If you specified a different server in <i>ServerName</i> , <i>UserID</i> is the user ID used for that server. If you do not specify a User ID, QMF for Windows uses the last user ID specified for this server, if available, or displays a dialog box if none is available. This parameter is ignored if <i>ServerName</i> is NULL or an empty string.
<i>Password</i>	If you specified a different server in <i>ServerName</i> , <i>Password</i> is the password used for that server. If you do not specify a password, QMF for Windows uses the last password specified for this server, if available, or displays a dialog box if none is available. This parameter is ignored if <i>ServerName</i> is NULL or an empty string.
<i>ForceDialog</i>	If you specified a different server in <i>ServerName</i> , 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 displays this dialog box only if necessary. This parameter is ignored if <i>ServerName</i> is NULL or an empty string.
<i>Account</i>	If you specified a different server in <i>ServerName</i> , 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 <i>ServerName</i> is NULL or an empty string.
<i>Comment</i>	Optionally, a string that specifies a comment for the table in which the data is saved.
<i>CommitScope</i>	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 GetLastErrorMessage() or 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.

SaveQMFPProc()

short SaveQMFPProc(BSTR *OwnerAndName*, BSTR *Text*, BSTR *Comment*, BOOL *Replace*, BOOL *Share*)

Description

This function saves a procedure at a database server.

Parameters

Name	Description
<i>OwnerAndName</i>	A string containing the owner and name, separated by a period, of the procedure that you want to save. For example, John.Proc2
<i>Text</i>	A string containing the text that you want to save in the procedure.
<i>Comment</i>	A string containing any comment that you want to save with the procedure. If there is no comment, pass this parameter as either an empty string or NULL.
<i>Replace</i>	Nonzero replaces an existing procedure with the same name. Zero aborts the operation if there is an existing procedure with the same name.
<i>Share</i>	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() or GetLastErrorType(), GetLastSQLCode(), GetLastSQLError(), or GetLastSQLState() to get additional error information.

SaveQMFPQuery()

short SaveQMFPQuery(BSTR *OwnerAndName*, BSTR *Text*, BSTR *Comment*, BOOL *Replace*, BOOL *Share*)

Description

This function saves a query at a database server.

Parameters

Name	Description
------	-------------

<i>OwnerAndName</i>	A string containing the owner and name, separated by a period, of the query that you want to save. For example, John.Query2
Text	A string containing the text that you want to save in the query.
Comment	A string containing any comment that 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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

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

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	Nonzero replaces an existing query with the same name. Zero aborts the operation if there is an existing query with the same name.
Share	One of the values listed below for the specified option.

Meanings and values for the various options are as follows:

Option	Meaning	Description
--------	---------	-------------

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_ISOLVLC (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
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_USADATEFMT (0x242A) US DDM_EURDATEFMT (0x242B) European DDM_JISDATEFMT (0x242C) Japanese Industrial Standard

DDM_STTTIMFMT(0x2123)	Format for retrieved time values.	DDM_ISOTIMFMT (0x242E) ISO DDM_USATIMFMT (0x242F) US DDM_EURTIMFMT (0x2430) European DDM_JISTIMFMT (0x2431) Japanese Industrial Standard
-----------------------	-----------------------------------	---

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.

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

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.
Comment	A string containing any comment that you want to save with the query. If there is no comment, pass this parameter as either an empty string or NULL.
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()` or `GetLastErrorType()`, `GetLastSQLCode()`, `GetLastSQLError()`, or `GetLastSQLState()` to get additional error information.

SetBusyWindowButton()

void `SetBusyWindowButton(BSTR Text)`

Description

This function specifies the text displayed on the busy window's Cancel button.

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()

SetBusyWindowMessage()

void SetBusyWindowMessage(BSTR *Message*)

Description

This function specifies the text displayed in the busy window's message area.

Parameters

Name	Description
Message	A string that specifies the text displayed on the busy window's message area.

Return Value

None.

Related Topics

SetBusyWindowButton()
SetBusyWindowMode()
SetBusyWindowTitle()
ShowBusyWindow()

SetBusyWindowMode()

void 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

Name	Description
Mode	Specifies when QMF for Windows displays the busy window:
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 after 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()

SetBusyWindowTitle()

void SetBusyWindowTitle(BSTR Title)

Description

This function specifies the text displayed in the busy window's title bar.

Parameters

Name	Description
------	-------------

Title	A string that specifies the text displayed on the busy window's title bar.
-------	--

Return Value

None.

Related Topics

SetBusyWindowButton()
 SetBusyWindowMode()
 SetBusyWindowMessage()
 ShowBusyWindow()

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

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.

SetHostVariable()

short SetHostVariable(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

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.

SetOption()

short `SetOption(short Mode, VARIANT Value)`

Description

This function sets the specified option value in QMF for Windows. For some options, the changes do not take effect 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

Name	Description
Option	Specifies which option to set:

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 used to display null values.
7 (RSO_ENTER_NULLS_STRING)	The string used to enter null values.
8 (RSO_ENTER_DEFAULTS_STRING)	The string used 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.

Name	Description
Value	The value to which to set the option.

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()`

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 the QMF for Windows main window. This function enables you to force the QMF for Windows dialog boxes to be centered on and modal to your client application window.

Parameters

Name	Description
ParentWnd	The HWND of the new parent window. Specify NULL to use the QMF for Windows 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()`

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

Name	Description
ProcID	The ID of the procedure as returned from InitializeProc().
Name	A string that contains the name of the variable that you want to set.
Value	A string that contains the value that 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.

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().

Parameters

Name	Description
QueryID	The ID of the query as returned from InitializeQuery().
Name	A string that contains the name of the variable that you want to set.
Value	A string that contains the value that 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.

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 call SetBusyWindowMode() with a mode of RSM_CLIENTCONTROLLED. If you set a parent window by calling SetParent(), the busy window is modal to the specified window.

Parameters

Name	Description
Show	Nonzero shows the busy window; zero hides the busy window. If nonzero, the busy window displays until you call ShowBusyWindow() with <i>Show</i> set to zero.

Return Value

None.

StartBind()

short StartBind(BSTR *CollectionName*, BSTR *PackageName*, BSTR *ConsistencyToken*)

Description

This function begins the process of binding a package in the database.

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() or GetLastErrorType(), GetLastSQLCode(), GetLastSQLError(), or GetLastSQLState() to get additional error information.

Related Topics

EndBind()

CancelBind()

Appendix. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10594-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will

be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this publication to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is as your own risk.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J74/G4
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

ACF/VTAM	IBMLink
Advanced Peer-to-Peer Networking	IMS
AIX	Language Environment
AIX/6000	MVS
AS/400	MVS/ESA
C/370	MVS/XA
CICS	OfficeVision/VM
CICS/ESA	OS/2
CICS/MVS	OS/390
CICS/VSE	PL/I
COBOL/370	PROFS
DATABASE 2	QMF
DataJoiner	RACF
DB2	S/390
DB2 Universal Database	SQL/DS
Distributed Relational Database Architecture	Virtual Machine/Enterprise Systems Architecture
DRDA	Visual Basic
DXT	VM/XA
GDDM	VM/ESA
IBM	VSE/ESA
	VTAM

Java or all Java-based trademarks and logos, and Solaris are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Lotus and 1-2-3 are trademarks of Lotus Development Corporation in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are trademarks or registered trademarks of Microsoft Corporation.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Index

A

- accounting strings 4
- action buttons
 - prompted query 15
- add-in
 - Excel 58
- AddDecimalHostVariable() 67
- AddHostVariable() 67
- adding
 - row conditions 19
 - sort conditions 18
- adding a row
 - table editor 52
- adding objects to lists 40
- API Reference 67

B

- between (row condition) 18
- BindDecimalHostVariable() 68
- BindHostVariable() 69
- BindSection() 70
- Blocking calls 66
- breaks
 - forms 29

C

- calculations
 - forms 29
- CancelBind() 71
- ChangePassword() 71
- changing a row
 - table editor 52
- changing passwords 3
- ClearList() 71
- Close() 72
- columns
 - adding to prompted queries 17
 - forms 29
- columns, reordering 26, 44
- columns, selecting 25
- Commit() 72
- CompleteQuery() 73
- complex queries
 - building 16
- conditions
 - forms 29
- Connecting to the database 66
- containing (row condition) 18
- converting formatting to a form 27, 45

- CopyToClipboard() 73
- creating
 - static queries 47
- creating a linear procedure 35
- creating job files 43

D

- database
 - security 2
- database server
 - exporting data to 56
- DB2 Forms 53
- DeleteQMFObject() 74
- details
 - forms 29
- display objects 40
- draw object 40
- draw queries
 - creating 9

E

- edit object 40
- EndBind() 75
- ending with 18
- equal to (row condition) 18
- Excel
 - add-in 58
- Execute() 75
- ExecuteEx() 76
- ExecuteStoredProcedure() 76
- ExecuteStoredProcedureEx() 78
- Export() 79
- ExportForm() 81
- exporting
 - reports 34
- exporting data
 - to a database server 56
 - to files 55
 - to other tables 56
- ExportReport() 81

F

- FastSaveData() 83
- FetchNextRow() 84
- FetchNextRowEx() 85
- FetchNextRows() 85
- FetchNextRowsEx() 86
- files
 - exporting data to 55
- final
 - forms 29

- FlushQMFCache() 87
- fonts
 - query display 8
 - results display 26, 44
- form
 - main 29
- formatting numeric query
 - results 27, 44
- forms
 - breaks 29
 - calculations 29
 - columns 29
 - conditions 29
 - details 29
 - final 29
 - HTML 29
 - opening a saved file 33, 36
 - options 30
 - page 30
 - producing a report 30
 - saving to files 32, 36
 - saving to the database server 33, 36

G

- GetColumnCount() 87
- GetColumnDataValue() 88
- GetColumnHeader() 88
- GetColumnHeaderEx() 89
- GetColumnHeadings() 89
- GetColumnValue() 90
- GetColumnValueEx() 91
- GetDefaultServerName() 91
- GetGlobalVariable() 92
- GetHostVariableNames() 92
- GetHostVariableTypes() 92
- GetLastErrorString() 93
- GetLastErrorType() 93
- GetLastSQLCode() 94
- GetLastSQLError() 95
- GetLastSQLState() 96
- GetOption() 97
- GetOptionEx() 98
- GetProcText() 98
- GetProcVariables() 99
- GetQMFOBJECTInfo() 99
- GetQMFOBJECTInfoEx() 101
- GetQMFOBJECTList() 102
- GetQMFOBJECTListEx() 103
- GetQMFPProcText() 104

- GetQMFQueryText() 105
 - GetQueryText() 105
 - GetQueryVerb() 106
 - GetResourceLimit() 106
 - GetResourceLimitEx() 110
 - GetRowCount() 111
 - GetServerList() 111
 - GetServerListEx() 112
 - GetStoredProcedureResultSets() 112
 - GetVariables() 113
 - GetVariablesEx() 114
 - governing 4
 - greater than (row condition) 18
 - greater than or equal to (row condition) 18
 - grouping query results 27, 45
- H**
- host variables
 - using in static queries 47
 - HTML
 - forms 29
- I**
- InitializeProc() 114
 - InitializeQuery() 115
 - InitializeServer() 116
 - InitializeStaticQuery() 117
 - Internet Mail 57
 - Is (row condition operator) 19
 - Is Not (row condition operator) 19
 - IsStatic() 117
- J**
- job files, creating 43
 - join conditions
 - creating in prompted queries 20
- L**
- less than (row condition) 18
 - less than or equal to (row condition) 18
 - listing
 - objects 39
 - lists
 - opening saved files 41
 - lists, adding objects 40
 - lists, removing objects 41
 - logging on 2
- M**
- main
 - forms 29
 - multiple queries
 - displaying simultaneously 8
 - multiple query documents 8
- multiple tables
 - in prompted queries 19
- N**
- new
 - draw queries 9
 - prompted queries 15
 - SQL query 7
 - Notices 135
 - null (row condition) 18
- O**
- objects
 - listing 39
 - Open() 118
 - opening
 - procedures at the database 37
 - prompted queries at the database server 22
 - prompted query files 21
 - saved forms at the database server 33
 - saved SQL files 11
 - SQL queries at the database server 12
 - options
 - forms 30
- P**
- page
 - forms 30
 - passwords
 - correcting 3
 - Prepare() 119
 - previewing
 - printed procedure 37
 - printed queries 13
 - reports 30
 - previewing query results 28, 46
 - print preview
 - prompted queries 22
 - printing
 - procedure 38
 - reports 34
 - SQL queries 13
 - printing query results 28, 46
 - PrintReport() 119
 - procedure
 - printing 38
 - procedure with logic 35
 - prompted queries
 - adding columns 17
 - adding tables 16
 - converting to SQL 20
 - creating 15
 - creating join conditions 20
- prompted queries (*continued*)
 - opening saved files 21
 - running 16
 - saving as files 21
 - saving to the database server 21
 - using multiple tables 19
 - using SQL in 20
 - using substitution variables 21
 - viewing SQL 20
- prompted query
 - action buttons 15
- Q**
- queries
 - building complex 16
 - query results, formatting 27, 44
 - query results, grouping 27, 45
 - query results, previewing 28, 46
 - query results, printing 28, 46
 - query results, saving 28, 45
 - query results, saving to files 28, 45
 - query results, sorting 26, 43
 - query results, summarizing 27, 45
- R**
- ReinitializeServer() 119
 - removing objects from lists 41
 - reordering columns 26, 44
 - reports
 - exporting 34
 - previewing 30
 - printing 34
 - producing a report using forms 30
 - resizing columns and rows 25
 - results view 7
 - REXX procedure 35
 - Rollback() 120
 - row condition operator
 - Is 19
 - Is Not 19
 - row conditions
 - adding 19
 - between 18
 - containing 18
 - ending with 18
 - equal to 18
 - greater than 18
 - greater than or equal to 18
 - less than 18
 - less than or equal to 18
 - null 18
 - starting with 18
 - using 18
 - rows, selecting 25
 - run object 40

- running
 - prompted queries 16
 - SQL query at a database server 7
 - static queries 49
- RunProc() 120
- S**
- sample applications 58
- SaveData() 121
- SaveQMFProc() 123
- SaveQMFQuery() 123
- saving
 - forms to files 32, 36
 - forms to the database server 33, 36
 - prompted queries as files 21
 - prompted queries to the database server 21
 - SQL queries to files 11
 - SQL queries to the database server 12
- saving query results 28, 45
- saving query results to files 28, 45
- searching
 - table editor 51
- Selecting columns and rows 25
- Send To 57
- server
 - setting 1
- SetBindOption() 124
- SetBindOwner() 126
- SetBusyWindowButton() 126
- SetBusyWindowMessage() 127
- SetBusyWindowMode() 127
- SetBusyWindowTitle() 128
- SetGlobalVariable() 129
- SetHostVariable() 129
- SetOption() 130
- SetParent() 131
- SetProcVariable() 131
- SetVariable() 132
- ShowBusyWindow() 133
- sort conditions
 - adding 18
 - using 17
- sorting query results 26, 43
- SQL
 - using in prompted queries 20
- SQL queries
 - opening a new document 7
 - opening saved files 11
 - print preview 13
 - printing 13
 - running at the database server 7
 - saving to files 11
 - SQL queries (*continued*)
 - saving to the database server 12
 - StartBind() 133
 - starting with (row condition) 18
 - static queries
 - creating 47
 - running 49
 - using substitution variables 47
 - substitution variables
 - in SQL queries 10
 - replacing with host variables 47
 - running SQL queries with 10
 - using in prompted queries 21
 - using in static queries 47
 - summarizing query results 27, 45
- T**
- table editor 51
 - adding a row 52
 - changing a row 52
 - searching for rows 51
- tables
 - adding to prompted queries 16
 - exporting data to 56
- toolbar
 - adding buttons 5
 - customizing 5
 - moving buttons 6
 - removing buttons 6
- V**
- viewing
 - results 7
 - SQL 7
 - SQL in prompted queries 20

Readers' Comments — We'd Like to Hear from You

Query Management Facility
Getting Started with QMF for Windows
Version 7

Publication No. SC27-0723-00

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

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

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

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? Yes No

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

Name

Address

Company or Organization

Phone No.



Fold and Tape

Please do not staple

Fold and Tape



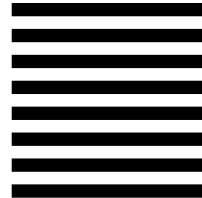
NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM CORPORATION
Department BWE/H3
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.



Fold and Tape

Please do not staple

Fold and Tape



File Number:

Program Number: 5675-DB2
5697-F42
5697-G24
5697-G22
5648-D35
5697-G23



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC27-0723-00



Spine information:



QMF

Getting Started with QMF for Windows

Version 7