Query Management Facility™

Developing QMF Applications

Version 7 Release 2

SC27-0718-01

Query Management Facility™

Developing QMF Applications

Version 7 Release 2

SC27-0718-01

Note!
Before using this information and the product it supports, be sure to read the general information under Appendix F, "Notices" on page 273.

Second Edition (March 2002)

This edition applies to Query Management Facility, a feature of Version 7 Release 1 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 Release 2 of DATABASE 2 Server for VM and VSE (DB2 for VM and VSE), 5697-F42, and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct editions.

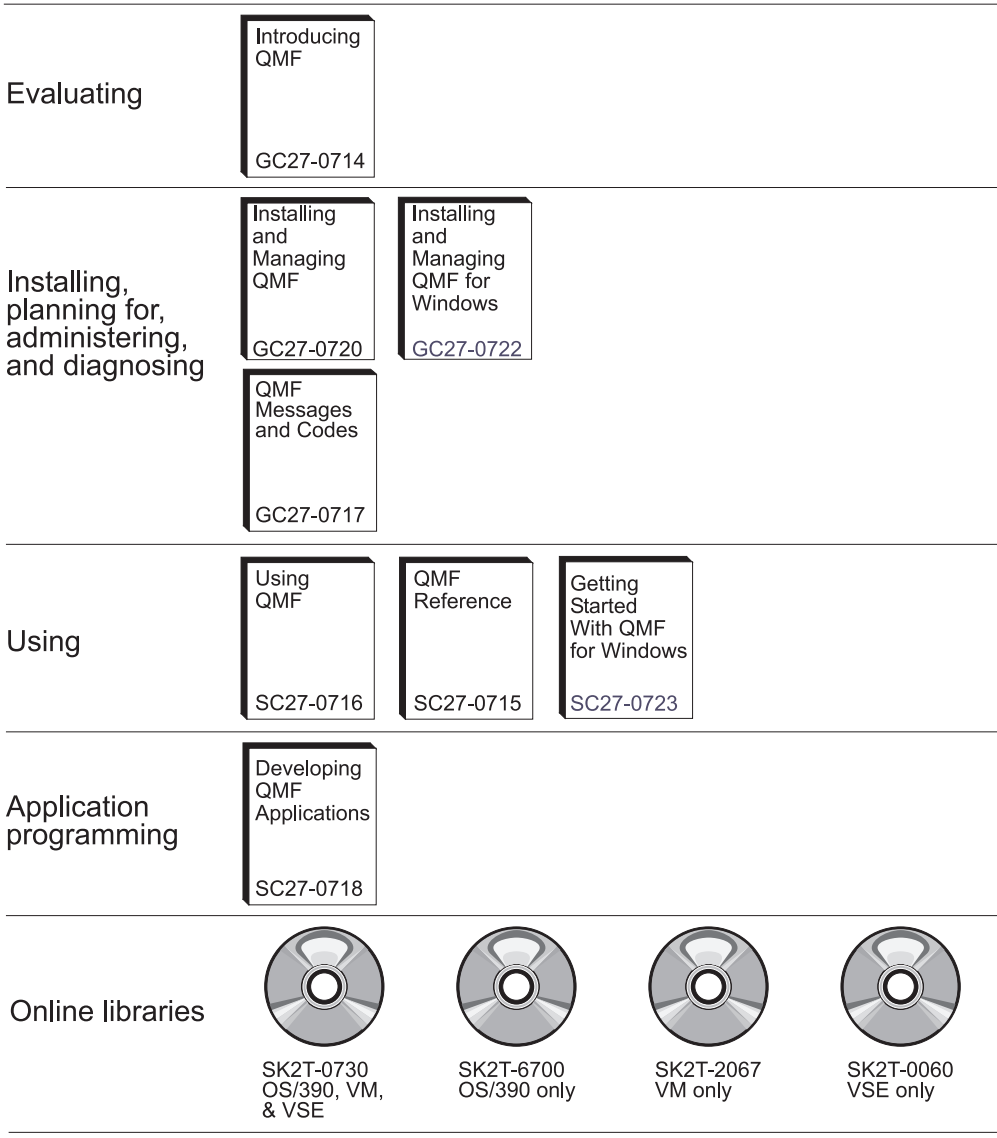
This edition replaces and makes obsolete the previous edition, SC27-0718-00.

The technical changes for this edition are indicated by a vertical bar to the left of a change. A vertical bar to the left of figure caption indicates that the figure has changed. Editorial changes that have no technical significance are not noted.

Contents

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.



About This Book

This book is intended to help application programmers write applications that use IBM® Query Management Facility (QMF).

How to use this book

The tasks in this book outline the design decisions that you need to make before you write a QMF application, show you different programming techniques, and provide some examples that highlight application programming using QMF. The appendixes provide reference information useful for application development.

This book serves OS/390®, VM® and VSE™ customers. Differences among systems, or among CICS®, CMS, TSO and native OS/390 batch , are highlighted when necessary. Otherwise, you can assume that QMF works the same in each system.

What you should know before you begin

QMF applications let you work with QMF objects and perform QMF functions from within an application program written in one of the languages QMF supports. This book assumes you already know how to write queries and procedures, format reports, and modify the database.

To write QMF applications using QMF command or callable interfaces, you might need to know one of the following programming languages:

Callable Interface
Command Interface

Assembler, PL/I, C, REXX, COBOL, FORTRAN
Any language that runs under ISPF

You might also need a panel display application, depending on the type of application you write.

For a list of books that provide information about QMF functions and administration, see "The QMF Library" on page v.

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information.

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

`http://www.ibm.com./qmf`

The Web site has a feedback page that you can use to enter and send comments.

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

Complete the readers' comment form
at the back of the book and return it by mail, by fax (800-426-7773 for the United States and Canada), or by giving it to an IBM representative.

How to order QMF books

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

For a list of QMF books, see "The QMF Library" on page v.

Chapter 1. QMF Application Development Overview

You can use many of the functions of QMF in your own applications. For example, you can write applications that:

- Run queries or procedures
- Export or import QMF objects
- Display or print reports or charts
- Enable the user to enter or change data in the database

You can also write applications that provide helpful functions to your users in QMF, such as a user-defined command that prints QMF reports at a remote location, or a function key that automatically generates a chart of the weekly sales results.

This chapter describes the two major types of QMF applications and the application development tools QMF provides to help you implement your application.

What Is application development in QMF?

The word *application* can have many meanings. In QMF, an application can be a procedure, a program, or an EXEC that lets you run QMF commands and alter QMF objects using the Export and Import QMF commands.

Application development refers to the process of creating an application. It includes:

- Understanding the problem that your application solves
- Designing the application
- Writing the code, associated messages, and help panels

Given these definitions, you can begin making the design decisions that affect how your end users use your application and what QMF application tools you use to enable your application to interact with QMF.

How can the end users use your application?

You might want end users to interact primarily with your application, or you might want them to use your application as a customized function in QMF.

- If your application is intended for end users who are unfamiliar with QMF, you probably want your end users to interact *primarily* with your application. In fact, you might not want your end users to know that QMF

is active. In this case, your application uses QMF services, but resides outside of QMF. Your program issues QMF commands only as needed.

- If your end users are familiar with QMF, you might want your end users to see your application as an extension or customization of QMF. In this case, you need to set up your application to run within QMF.

End users interacting primarily with the application

Suppose you write an application that uses QMF services. This application provides the end user with a menu-driven interface, as shown in Figure 1.

```
                        J & H Supply Company
                         Information System


Please select one of the following:

   1.  Print the monthly sales report

   2.  Create a new report

   3.  Modify information in the database

   4.  End the application


====> 1
```

Figure 1. An example of an application-defined panel

When the user selects an option, the application issues the appropriate QMF commands. If the user selects option 1, for example, the application runs a QMF procedure that might run a query and print the resulting report.

In the preceding example, your application controls QMF. Your user interacts only with your user interface and is not aware that QMF is active.

End users starting your application within a QMF session

Suppose you write an application that sends a QMF report from one user to another.

You expect your users to run your application from within the QMF environment, so you can assign the application a command synonym (SEND_TO) that the end users can issue from the command line, or you can assign the application to a function key instead, which automatically runs your application.

After the user generates a report, the user can send this report to Smith by entering SEND_TO SMITH on the QMF command line, as shown in Figure 2.

```
REPORT                                              LINE 1     POS 1     79


  NAME         DEPT  JOB        SALARY        COMM
  ---------  ------  -----  ----------  ----------
  DANIELS        10  MGR      19260.25           -
  JONES          10  MGR      21234.00           -
  LU             10  MGR      20010.00           -
  MOLINARE       10  MGR      22959.20           -
  HANES          15  MGR      20659.80           -
  KERMISCH       15  CLERK    12258.50      110.10
  NGAN           15  CLERK    12508.20      206.60
  ROTHMAN        15  SALES    16502.83     1152.00
  JAMES          20  CLERK    13504.60      128.20
  PERNAL         20  SALES    18171.25      612.45
  SANDERS        20  MGR      18357.50           -
  SNEIDER        20  CLERK    14252.75      126.50
  ABRAHAMS       38  CLERK    12009.75      236.50
  MARENGHI       38  MGR      17506.75           -
1=Help         2=            3=End       4=Print       5=Chart        6=Query
7=Backward     8=Forward     9=Form     10=Left       11=Right       12=
OK, here is your report.
COMMAND ===> SEND_TO SMITH
```

Figure 2. An example of a user entering a customized QMF command

What QMF application development tools are available?

Regardless of how your end users see your application, you can write applications using any of the following application development tools:

- QMF procedures
- QMF callable interface
- QMF command interface
- QMF externalized formats
- QMF command synonyms
- Other IBM products that bridge to QMF

QMF procedures

QMF procedures are QMF objects that run within QMF and issue QMF commands. QMF procedures can execute any QMF commands available at your installation. QMF provides two types of procedures: linear procedures and procedures with logic.

- *Linear procedures* contain only QMF commands and comments. You can use linear procedures in all environments supported in QMF.
- *Procedures with logic* combine QMF commands with REXX logic to allow you to create more powerful programs. You can use procedures with logic in all environments supported in QMF, except CICS. Procedures with logic can contain QMF commands and any statement that is valid in a REXX program.

For general information about writing linear procedures or procedures with logic, see *Using QMF* For specific information about using QMF procedures to write applications, see Chapter 2, "Using Procedures as Applications" on page 7.

Starting with Version 3.3, QMF provides a system initialization procedure that allows you to run commands and set global variables before the user sees the QMF home panel. For more information, see the version of *Installing and Managing QMF* for your platform.

QMF callable and command interfaces

If you choose not to use QMF procedures, you need to decide whether your program communicates with QMF through the callable interface or the command interface.

Callable interface

The QMF callable interface is a Systems Application Architecture (SAA) interface that you use to create an application that is invoked outside of QMF, starts a QMF session, and sends commands to QMF for execution.

The callable interface is available for all environments supported in QMF. It is the SAA Common Programming Interface for query in the VM, OS/390, and VSE environments, and is available for various languages as shown in Table 1.

Table 1. Callable interface support

	CICS under OS/390	CICS under VSE	CMS	TSO	APPC	SRPI	Native OS/390 batch
assembler [1]	×	×	×	×	×	×	×
C	×	×	×	×	×	×	×
COBOL	×	×	×	×	×	×	×
FORTRAN			×	×	×	×	×
PL/I	×	×	×	×	×	×	×
REXX			×	×	×	×	×

If you want to write SAA applications, you *must* use the callable interface in one of the SAA languages that QMF supports.

For more information about the callable interface, see Chapter 3, "The Callable Interface" on page 19.

Command interface

The QMF command interface allows you to create applications that submit commands to QMF from an ISPF dialog. QMF communicates with the ISPF dialog through the ISPF variable pool using this command interface.

The command interface is only available when ISPF is available. The command interface is not available in CICS.

For more information about the QMF command interface, see Chapter 4, "Using the Command Interface for Applications" on page 29.

Contrasting the callable and command interfaces

The differences between the callable interface and the command interface are:

Callable interface:

- Is available for all QMF-supported environments
- Does not require ISPF
- Does not require QMF to be started before you run your application
- Provides SAA Common Programming Interface for query

Command interface:

- Is available in all environments supported in QMF and ISPF
- Requires ISPF to be present and active
- Requires QMF to be started before the application is started
- Provides variables for communication between the ISPF application and QMF
- Requires the programming language to be supported by ISPF

External formats for QMF objects

Your application can export QMF objects to a file outside of the QMF product; for example, you can export a form to a CMS file, a TSO data set, or a CICS data queue. Each object has a particular format that your application can edit and transfer to another system, or import into QMF.

1. Assembler is not an SAA language.

For more information about the externalized formats of QMF objects, see Chapter 9, "Importing and Exporting QMF Objects" on page 77.

Command synonyms

QMF allows you to specify command synonyms for programs or procedures that you code. These command synonyms allow end users to use your programs and procedures just as they would use any QMF command.

For more information about command synonyms, see "Using command synonyms" on page 73.

Other IBM products that bridge to QMF

You can use the following IBM products with QMF to expand the function of QMF:

Application System (AS)
AS can issue QMF commands and define QMF queries. AS can then use the results of the QMF queries as input to AS processes.

Data Extract (DXT)
QMF can invoke DXT™ End User Dialogs to allow the end user to extract data from sources not directly supported by QMF.

ECF The Enhanced Connectivity Facility (ECF) allows a workstation user to access host relational data. The workstation user uses ECF facilities to send a request to the host to run a saved QMF query or procedure and to download the retrieved data to the workstation.

GDDM
The Interactive Chart Utility (ICU), used by QMF to display charts, is actually a feature of Graphical Data Display Manager (GDDM®).

ISPF Interactive System Product Facility allows the user to generate panels that can interact with QMF via the command interface.

Lotus 1-2-3/M
The host version of Lotus® 1-2-3® can access QMF to perform spreadsheet analysis on query results.

Chapter 2. Using Procedures as Applications

You can write many applications entirely as procedures. You can create procedures on your development system and either keep them for your personal use or move them to your production system for public use.

If you are using QMF in the CICS environment, you can use QMF linear procedures. If you are using QMF in the CMS, TSO, or native OS/390 batch environments, you can also use REXX statements and functions in your QMF procedures. REXX functions and procedures with logic are not available in the QMF CICS environment.

This chapter focuses on information you need to know to use QMF procedures to implement your application.

For information about how to create, build, and run a procedure, see *Using QMF*.

Using ISPF services in a QMF procedure requires a few extra steps. For information about running ISPF commands from a QMF procedure with logic running under ISPF, see "Using ISPF commands from a procedure with logic" on page 41.

Knowing when not to use pProcedures

If you are writing an application that operates on a procedure in QMF temporary storage, you cannot write your application as a procedure. This is because, when you run a procedure, that procedure becomes the current procedure in QMF temporary storage.

For example, if you write your application as a procedure, and code your application to save the current procedure in QMF temporary storage, the application saves itself, because it is the current procedure in QMF temporary storage when it is running.

Initial procedures

An initial procedure is a procedure that runs immediately after your QMF session starts. Use the DSQSRUN parameter to specify the name of this procedure. You can use DSQSRUN:

- With the DSQQMFE command, when QMF is started interactively

- With the QMF START command, when QMF is started through the callable interface

QMF runs the initial procedure differently depending on the type of QMF session used. For more information about how QMF uses the initial procedure, see "Interactive session with an Initial Procedure (DSQSRUN)" on page 53.

In TSO, and native OS/390 batch, applications can also set program parameters using a REXX EXEC as described by the DSQSCMD parameter of the QMF START command. Because QMF CICS does not support REXX, in CICS you must specify all program parameters on the START command using DSQSMODE=INTERACTIVE. The default mode from the callable interface is BATCH.

Considerations for writing initial Procedures

- By default, QMF reruns the initial procedure whenever the user issues the END command in an interactive session of QMF started by DSQQMFE. The DSQEC_RERUN_IPROC global variable specifies if the initial procedure is rerun. The default value of this variable is 1 to rerun the procedure; 0 prevents the initial procedure from being rerun.

  In callable interface programs, the initial procedure is never rerun, so this global variable does not affect your callable interface programs.
- If you are writing initial procedures for use in an interactive QMF session, you should avoid writing your initial procedure so that the current panel at the end of the procedure is the Home panel. If the Home panel is the current panel at the end of the initial procedure, QMF does not interactively display a panel at the end of the procedure. If no severe errors occurred and DSQEC_RERUN_IPROC is set to 1, QMF reruns the initial procedure without interacting with the user. This results in an uninterruptible loop that can appear as though QMF is not starting.

  To avoid creating an uninterruptible loop, do one of the following:
  - Make sure that the current panel at the end of the procedure is not the Home panel.
  - Make sure that the procedure contains either a QMF EXIT or an INTERACT command.
  - Set DSQEC_RERUN_IPROC to zero (0).
- When you specify values for substitution variables in initial procedures, the number of ampersands (&) you must use before the name of the variable can vary depending on your environment. For example, you can specify DSQSRUN as follows:

  ```
  DSQSRUN=INITPROC(&VAR1 = value)
  ```

The number of ampersands you need to specify with VAR1 depends on if QMF is running under CICS, CMS, TSO, or native OS/390 batch, if ISPF is present, and if the program starting QMF is written in REXX.

Initial Procedures and Remote Unit of Work

The initial procedure must be stored at the system on which you start QMF (the local system).

If you use the QMF CONNECT command from either your initial procedure or the command line during an interactive session set up by an initial procedure, you must reconnect to your original location before you can issue an END command to reinvoke your initial procedure.

If you are still connected to the remote location, you receive an error.

Using QMF CONNECT within a procedure

The QMF CONNECT command lets you connect to another user ID or to a remote DB2® database to use the remote unit of work support. You can use this command within a linear procedure or a procedure with logic.

When you write procedures that use the QMF CONNECT command to access remote databases, be aware of the following:

- If you are connected to a remote database and issue a RUN PROC command, that procedure and all the objects used in that procedure must be stored at the remote database.
- All QMF commands in the procedure are run in QMF temporary storage at the system where QMF is running (the local system). However, all objects used by these QMF commands (such as queries, procedures, or forms) must be defined in the database at the current location (the remote system).

  For more information about using the QMF CONNECT command and remote unit of work support, see *QMF Reference* (for command syntax).
- All commands that affect the database (for example, SQL statements, QMF queries, or EDIT TABLE updates) run at the current location.
- If the procedure contains system-specific commands (CICS, CMS, or TSO), these commands run at the system where QMF is running (the local system).

  If your procedures contain system-specific commands that do not run on the system where QMF runs, your procedure does not run successfully.
- Any files or data sets used in a system-specific command must exist on the system where QMF is running (the local system).

Substitution variables in procedures

You can use QMF substitution variables in linear procedures and procedures with logic.

A substitution variable is any variable that you can use in a QMF command; QMF manages these variables for you. A substitution variable is always preceded by an ampersand (&). You can assign a value to a substitution variable by setting global variables, by specifying values on the RUN command, or by specifying values on the RUN command prompt panel. For information on setting global variables, see "SET GLOBAL" on page 62.

See *Using QMF* if you need to learn more about using ampersands with substitution variables in QMF.

Specifying values on the RUN command

You can assign a value to a substitution variable using the RUN command:

- In your linear procedure:

```
RUN PROC SCHEDULE (&&TYPE='VACATION'
```

- In your procedure with logic:

```
"RUN PROC SCHEDULE (&&TYPE='VACATION'"
```

If you issue the QMF RUN command from within a PROC or QUERY panel, you do not need to specify the PROC or QUERY object types. RUN assumes these values when you invoke it from their respective panels.

The value of `&TYPE` is available only to the procedure called SCHEDULE.

In this example:

- The variable value `VACATION` is surrounded by single quotes because the value is a character string.
- TYPE is preceded by double ampersands (&&) to indicate that the value is being set on the RUN statement to be passed to the procedure named SCHEDULE. If the RUN statement specifies &TYPE, the procedure containing this statement prompts the user for the value.

This value for the substitution variable is active *only* within the procedure that defines it. The value is not active in any procedure or module called from the defining procedure.

Specifying values on the RUN command prompt panel

If you run a query or procedure that contains a substitution variable, and this variable is not assigned a value by a global variable or on the RUN command, QMF presents a RUN command prompt panel. You can specify the value for the variable on this panel.

This value for the substitution variable is active *only* within the procedure that defines it. The value is not active in any procedure or module called from the defining procedure.

Prompting for variables in linear procedures

In a linear procedure, QMF scans the procedure for substitution variables and resolves them before it processes any commands. The user is prompted for all variables before the procedure runs.

Prompting for variables in procedures with logic

In a procedure with logic, the user is not prompted for variables until REXX encounters the statement containing the variables. For example, if your procedure with logic contains three statements that contain variables that QMF must prompt you for, QMF prompts you three times—once for each statement.

If you want a procedure with logic to prompt you for all the necessary variable values at one time, like the linear procedure does, use a dummy procedure. Suppose you want to be prompted once for the substitution variables LASTNAME and DEPT_NUM, which occur on two different lines in your procedure with logic as shown in Figure 3.

```
/* This procedure runs two queries, displaying the report after each */
/* procedure has run.                        */

"RUN QUERY REG_QUERY (&&LASTNAME=&LASTNAME";
"INTERACT"
"RUN QUERY REG2_QUERY (&&DEPT_NUM=&DEPT_NUM";
```

Figure 3. Procedure with logic with variables

Add the following line to the beginning of your procedure with logic, immediately following the comment lines:

```
"RUN PROC PROMPT_ME (&LASTNAME, &DEPT_NUM";
```

where PROMPT_ME is a procedure with logic containing a comment line and no instructions, as shown in Figure 4 on page 12.

The completed procedure with logic looks like this:

```
/* This proc is a dummy proc that provides prompting. */
/* This procedure runs two queries, displaying the report after each */
/* procedure has run                        */

"RUN PROC PROMPT_ME (&LASTNAME, &DEPT_NUM";
"RUN QUERY REG_QUERY (&&LASTNAME=&LASTNAME";
"INTERACT"
"RUN QUERY REG2_QUERY (&&DEPT_NUM=&DEPT_NUM";
```

Figure 4. Procedure with logic that prompts for variables

Alternatively, you can use SET GLOBAL to prompt for all the values in your procedure at the same time, as in the following:

```
"SET GLOBAL (LASTNAME=&LASTNAME,DEPTNUM=&DEPT_NUM";
```

Using REXX variables in procedures with logic

You can use REXX variables in a procedure with logic. The values for these variables are known only within the procedure in which you defined them. You can:

- Copy a REXX variable to a QMF variable with the SET GLOBAL command
- Copy a global variable to a REXX variable with the GET GLOBAL command
- Use REXX variables in your REXX statements

For more information on REXX variables, see the REXX reference manual for your system. For details on the GET GLOBAL and SET GLOBAL commands, see *QMF Reference*.

QMF also provides a group of REXX variables for the SAA callable interface that QMF sets after processing each QMF command. These variables provide important information about the results of each command. You can use them in your procedures with logic. For more information about these variables, see "REXX language interface" on page 216.

Passing arguments to a procedure with logic

For procedures with logic, QMF provides an ARG option on the RUN PROC command. This option lets you pass *arguments*, or values, to a procedure with logic.

Use the ARG option when you are running a procedure that contains a REXX PARSE ARG or ARG statement, as in the following example:

```
PROC              WILDE.SHOW_ARGS                    MODIFIED    LINE 1

/*********************************************************************/
/* This procedure shows you how to use the 'ARG=' option on the RUN  */
/* PROC command.                                                     */
/*********************************************************************/
parse upper arg query_name form_name
"RUN QUERY" query_name "(FORM="form_name
```

The RUN command for this procedure is:

```
RUN PROC SHOW_ARGS (ARG=(query_name form_name)
```

Using the ARG option, you can also pass values between procedures.

Using REXX error-handling statements in procedures with logic

You can use REXX error handling techniques, such as the REXX SIGNAL instruction, in a procedure with logic. In addition, you can use QMF commands and variables with the REXX EXIT instruction to help clarify nonzero return codes.

Branching to error-handling subroutines

The REXX *signal on error* instruction tells REXX to leave the current line and branch to a label marked *error* when a nonzero return code is encountered. This statement requires two parts:

- *Signal on error*

  After every command, REXX puts the return code of the command in a variable called `rc`.

  If a command has a nonzero return code, REXX branches to the *error* label.

  Signal on error returns errors from the QMF REXX procedure (ADDRESS QRW) command environment, but not the REXX callable interface.

- *Error* label

  The *signal on error* instruction requires that you provide a label that the procedure can branch to if it encounters a nonzero return code. The label precedes your error handling code. The return code is in the variable `rc`. You can use this variable to branch to another subroutine, or you can use it in your EXIT instruction, as in the following:

  ```
  /*  error handling code for a procedure with logic */
  error:
    exit rc
  ```

Using messages with the REXX EXIT statement

As the previous section shows, you can use the REXX EXIT instruction to exit a procedure with logic. QMF always issues a message when it finishes running a procedure with logic. If you use the EXIT instruction, the message you see depends on these factors:

- If the last QMF command encountered an error
- If the return code was zero

Table 2 shows which message you see based on the given conditions.

Table 2. Messages returned from QMF

Nonzero return code from the last QMF command	Procedure return code	Message at completion of procedure
No	0	`OK, your procedure was run.`
No	nonzero	`The return code from your procedure was 8.`
Yes	0	The error message provided by QMF.
Yes	nonzero	The error message provided by QMF.

An error message takes precedence over the return code message if you have an incorrect QMF command and a nonzero return code.

If you want to show the error message from the last command *and* exit with a QMF return code, use the MESSAGE command and the EXIT DSQ_RETURN_CODE as in the following example:

```
⋮
"MESSAGE (TEXT='"dsq_message_text"'"
exit dsq_return_code
```

Figure 5. Showing the error message and return code

The variables `dsq_message_text` and `dsq_return_code` are QMF-provided REXX variables. (For a complete listing of these variables, see "REXX language interface" on page 216.) You can use the MESSAGE command and the `dsq_message_text` variable to store and display a message after further processing has occurred, as in Figure 6 on page 15.

```
 /* Monthly report                                          */
Signal on error
"DISPLAY TABLE JUNE_INFO"
"PRINT REPORT"
Exit(0);
Error:    Original_msg = dsq_message_text
/* Saves error message. */
"RUN PROC GENERAL_RECOVERY"
/* This proc generates  */
/* new dsq_message_text. */
"MESSAGE (TEXT='" Original_msg "'"
/* Display original error msg. */
Exit dsq_return_code;
```

Figure 6. Storing and retrieving messages in a procedure

For more information on the MESSAGE command, see "MESSAGE" on page 60.

Calling REXX programs from a procedure with logic

You might have procedures that call applications. When you call your REXX callable interface application from a procedure with logic, be careful about the number of ampersands (&) you specify for the substitution variables in your application. This is especially true if the program being called contains a RUN command with substitution variables, as in `RUN QUERY WEEKLY_Q (&&DEPT=58`.

Calling REXX programs without substitution variables

If your REXX program does *not* contain an imbedded RUN command that includes substitution variables, use one of the following commands to invoke your program:

- The ADDRESS instruction

  This instruction establishes a command environment. (For more information on command environments, see Chapter 5, "ADDRESS QRW: Using the QMF Command Environment" on page 37.) If your program is named PANDA, and you want to call it from within the CMS environment, your command is:

  ```
  ADDRESS CMS "PANDA"
  ```

- The CALL instruction

  This instruction invokes a program. For the program named PANDA, the command is:

  ```
  CALL PANDA
  ```

- A function

  You also can call the program PANDA as a function, as in the following:

  ```
  answer = PANDA()
  ```

For more information on any of these commands, see the REXX reference manual for your system.

You might consider removing the substitution variables from the RUN command if you want to call your programs using one of the REXX invocation calls. In that case, QMF prompts the user for the variables.

Calling REXX Programs that contain substitution variables

If your REXX application contains a QMF RUN command with a substitution variable, you must invoke it using either `CMS program_name` or `TSO program_name`.

Whether you are running a procedure with logic or a callable interface program invoked by a procedure with logic, commands come into QMF the same way. In this context, the callable interface program becomes a logical extension of the procedure itself.

Consider the command:

```
RUN QUERY WEEKLY_Q (&DEPT=58
```

In a procedure with logic, use two ampersands on the substitution variable to pass the variable to the query, as in the following:

```
"RUN QUERY WEEKLY_Q (&&DEPT=58"
```

If a substitution variable has only one ampersand, QMF resolves the variable for the procedure itself, and cannot pass the variable to the query.

If you call a REXX callable interface application from a procedure with logic, and that application contains the command `RUN QUERY WEEKLY_Q (&DEPT=58,` QMF resolves the variable just as it would for the calling procedure. Because only one ampersand is used, the variable is not passed to the query.

To pass variables to QMF from a REXX callable interface application called by a procedure with logic, you have three choices:

- Use the `CMS` or `TSO` command to call the application.

  When you call the application, QMF does not process any substitution variables it encounters. In the preceding command, `&DEPT=58` is passed to the query, where the substitution variable is resolved.
- Treat all substitution variables in your application as though you were using them in a procedure with logic.

  Add an ampersand to every substitution variable so the procedure with logic does not resolve it.
- Use global variables.

You can define global variables at the start of your application and use them throughout your QMF session.

Chapter 3. The Callable Interface

This chapter presents an overview of the QMF callable interface. For specific information about the QMF callable interface for a particular language, see the section in Appendix A, "Sample Code for Callable Interface Languages" that describes the callable interface for that language:

Assembler
"Assembler language interface" on page 127

C Language
"C Language Interface" on page 150

COBOL
"COBOL language interface" on page 167

FORTRAN
"FORTRAN language interface" on page 184

PL/I "PL/I language interface" on page 200

REXX "REXX language interface" on page 216

What is the Callable Interface?

Programming languages can use the QMF callable interface to run QMF commands. All SAA Query commands are supported through the callable interface. The QMF callable interface provides standard interfaces for different programming languages, and provides common storage and access to program variables.

When an application program needs to run a QMF command, it must first issue a call to a QMF-supplied routine to start communication between the program and QMF. This call is made to the QMF-supplied interface routine. QMF supplies a routine for each supported language.

The application program can issue one or more QMF commands after the initial start call. The application program calls the QMF-supplied routine to issue each QMF command.

After the QMF command finishes processing, QMF supplies a return code that indicates the status of QMF. The callable interface gathers other information about the processing of the command and stores this information in variables accessible to both QMF and the application program. These variables are contained in either a *variable pool* or in an *interface communications area*. When

the callable interface returns control to the calling application program, the application can refer to these variables but should not alter them.

When the application program no longer needs to use QMF, the program issues a call to terminate communication between the program and QMF. This call is made to the QMF-supplied routine.

Considerations for using the QMF Callable Interface

- A call to QMF returns control to the calling application program only after QMF finishes processing the QMF command.
- QMF is in an inactive state when not processing a call.
- The application program and QMF communicate with return codes and variable data stored in the variable pool or in the interface communications area.
- All QMF commands must be coded in uppercase English letters.

  If you are using a QMF national language feature (NLF), your QMF commands must be written in the NLF language specified as the presiding language, and written in (or folded to) uppercase.
- The maximum length of the passed commands is 256 bytes.

Figure 7 shows how the application passes commands through the callable interface to QMF.

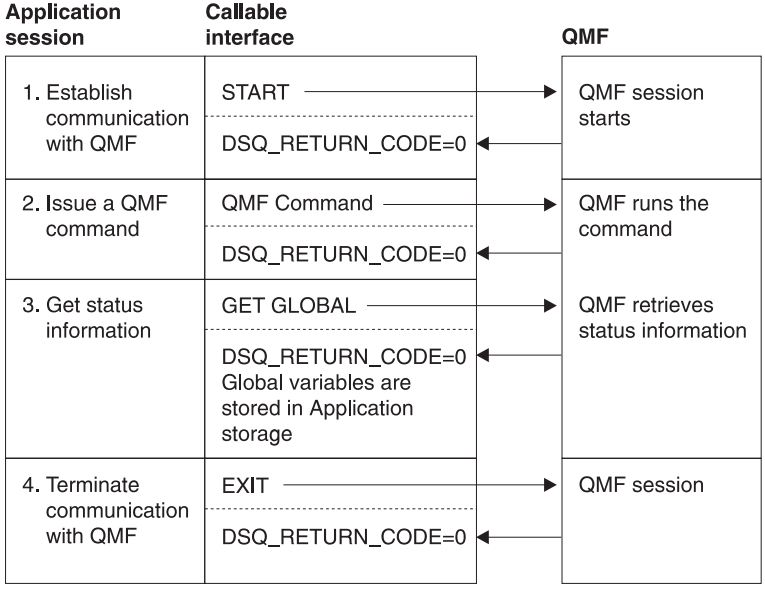


Figure 7. The application uses the QMF callable interface to communicate with QMF.

The results of issuing a command through the callable interface are generally the same as they are if you issue commands interactively.

Defining the Interface Communications Area (DSQCOMM)

QMF provides an interface communications macro for each supported programming language. This macro contains the following information:

- The interface communications area (DSQCOMM) or communications variables
- Definitions of return and reason codes
- Definition of the function calls to QMF

This macro defines some storage that contains the variables described in the preceding list. This storage is the callable interface communications area, and the variables stored in this area are accessible to both QMF and the callable interface application, although only QMF should alter the values. The application program should view these variables as read only.

The REXX callable interface uses interface communications variables provided by QMF rather than using a communications area.

The QMF callable interface communications area is required for all callable interface calls. Storage for the callable interface communications area is allocated by the program that is using QMF.

The START command establishes a unique instance or occurrence of a QMF session. The START command can establish only one QMF session:

- In a TSO address space
- In a single CMS virtual machine
- From a single CICS transaction

When running the START command, QMF updates the interface communications area or variables.

The interface communications area or variables must *never* be altered by the application program, with the following exceptions:

DSQ_COMM_LEVEL
: Set DSQ_COMM_LEVEL to the value of DSQ_CURRENT_COMM_LEVEL to identify the level of DSQCOMM. This does not apply to REXX.

DSQ_INSTANCE_ID
: If you call a callable interface program from within QMF, you need to

set the DSQ_INSTANCE_ID to zero (0) on the first call so that QMF resets the variable to the value set by the initial START command.

All calls that follow the START command must pass the address of the interface communications area that corresponds to the QMF instance. The application program is responsible for pointing to the correct interface communications area.

Each supported language has a unique communications macro that describes the interface communications area. Application programs must reference variables by variable name rather than value if they are to be portable, because the values can be different on other systems.

The interface communications area or variables contain the information in Table 3, which must *not* be altered by the calling program:

Table 3. DSQCOMM fields that must not be altered

Field	Description
Return code	Indicates the status of QMF processing after QMF processes a command.
Instance identifier	Identifies the instance of QMF that was started by the START command.
Completion message ID	Contains the message ID of the message that QMF displays at the user's terminal, if the command were issued there. This field is set at the completion of every QMF command. It contains the message QMF displays at the end of a command.
Query message ID	Contains the message ID of a QMF message resulting from a RUN QUERY command. This is the message ID of the message that is displayed in a user's query. This field is set when an error occurs while a query is running. It contains the message QMF displays within the query object at the end of a command.
START command parameter in error	Contains the parameter in error when the START command fails because of a parameter error.
Cancel indicator	Indicates if the user canceled the command processing while QMF was running the command.
Completion message	Contains the completion message that QMF displays at the user's terminal.

Table 3. DSQCOMM fields that must not be altered (continued)

Field	Description
Query message	Contains the query message text that resulted from a RUN QUERY command. This is the text that QMF displays in a user's query. For example, if you run a query object with an error, QMF displays a message describing the error that prevented the query from running. Query message then contains this error message text.

Return Codes

Return codes are returned after each call to the QMF callable interface. Return code values are described by the communications macro shipped with QMF.

If you want your applications to be portable across systems, the applications must reference the values of these codes by the variable names, because the values can be different on other systems.

Return codes from the callable interface indicate the following conditions:

- QMF successfully processed the request.
- QMF processed the request despite a warning condition.
- QMF did not process the command correctly.
- Due to a severe error, this instance of QMF has ended.

For a definition of each return code, see the appropriate programming language section of this guide.

Commands for using the callable interface

You can use the callable interface to use any QMF command that you would use in a procedure. However, there are three commands that have special syntax for the callable interface:

- START
- GET GLOBAL, extended syntax
- SET GLOBAL, extended syntax

START works only in the callable interface. To use GET GLOBAL and SET GLOBAL in a callable interface application written in a language other than REXX, use the *extended syntax*. The extended syntax of the SET GLOBAL command allows you to set global variables that have values up to 32 768 characters long. For more information on using the GET GLOBAL and SET GLOBAL commands in an application, see "GET GLOBAL" on page 56 and "SET GLOBAL: Extended syntax" on page 64.

For information about these and other commands you can use in a callable interface application, see Chapter 8, “QMF Commands in Applications” on page 51. To see examples of the START and SET GLOBAL commands for each language, see the sample program for each language:

Assembler
“Assembler programming examples” on page 131

C Language
“C language programming example” on page 154

COBOL
“COBOL programming example” on page 171

FORTRAN
“FORTRAN programming example” on page 187

PL/I “PL/I programming example” on page 203

REXX “REXX programming example” on page 221

Starting QMF from an application

Before you can run any other command from an application, you must start QMF. When using the callable interface, you start QMF by issuing the START command. You can have only one QMF session at a time.

Your application can issue a START command to test if QMF has already been started. If QMF has not been previously started, it starts. If QMF was previously started, the return code is nonzero, and you receive the following message number and message:

```
DSQ50719 QMF already active.  Secondary session not permitted.
```

With the REXX callable interface, you also can run the following program:

```
/*  test to see if QMF is active   */
"SUBCOM QRW"
if rc = 0
  then say "QMF is active"
else say "QMF is not active"
```

If your START command results in an error that is not severe (a return code of 4 or 8), QMF starts with errors. In this case, you can issue the EXIT command to stop QMF. You might want to issue the START command again. If the error persists, inspect the contents of the interface communications area or the QMF trace data output for sources of the error.

To pass parameters to QMF, specify the desired command keywords on the START command.

For details about the syntax and keywords used with the start command, see "START" on page 66.

Running your callable interface application

When you run your callable interface application, you must set up your running environment as though you were going to run interactive QMF.

For specific information about setting up your environment and compiling and running your callable interface application, see the appropriate coding sample for your language in Appendix A, "Sample Code for Callable Interface Languages" on page 127.

Using the callable interface from within QMF

> **Note to CICS users**
> You cannot use the callable interface from within QMF while in the CICS environment.

In all the environments supported in QMF, except CICS, you can use the callable interface from within QMF to run applications that modify QMF temporary storage areas. For example, you might want to export or import files through the callable interface even though you are in the middle of a QMF session.

You can do this by using the CMS or TSO command to call an application. From the application, you can run any valid QMF command. Because QMF is already active, you should not issue the START command.

You must set the DSQCOMM instance identifier (DSQ_INSTANCE_ID) to zero (0) before your first call to QMF. QMF determines the current instance and updates DSQ_INSTANCE_ID for use in subsequent QMF calls.

Error handling

Unless you are running QMF in a CICS environment, you can use the QMF-provided REXX variables or the similar values in the DSQCOMM communications area for error handling in your applications.

For example, the REXX variable `dsq_message_text` or the message text field in the DSQCOMM contains a QMF message.

In REXX, QMF assigns one of the following values to the variable `dsq_return_code` at the completion of every QMF command:

dsq_success
: Successful completion of the command

dsq_warning
: Normal completion with warnings

dsq_failure
: Command did not run correctly

dsq_severe
: Severe error; QMF session terminated

For the languages other than REXX, QMF places the same value in the return code field DSQ_RETURN_CODE of the DSQCOMM.

You can use these return codes and values in your applications. The following example shows how to use error-handling variables in a REXX callable interface application:

```
⋮
call dsqcix "CONVERT QUERY MYQUERY"
if dsq_return_code ¬= dsq_success then ...
⋮
call dsqcix "PRINT REPORT"
if dsq_return_code=dsq_severe | dsq_return_code=dsq_failure then ...
```

QMF also provides variables that contain message numbers and message text.

For a complete list of variables or fields in each DSQCOMM, see the appropriate section for each language in Appendix A, “Sample Code for Callable Interface Languages” on page 127.

Running callable interface programs under CICS

To run programs that use the QMF callable interface, install them into CICS using your normal method of installing CICS programs. For more information about applications in CICS, see *CICS for VSE/ESA Application Programming Guide*. For more information about installing QMF application programs, see *CICS for VSE/ESA System Definition Guide*.

In addition to the normal CICS requirements, the following considerations apply to all QMF callable interface programs running on CICS:

- Environment

When your program calls the QMF product, your program takes on the same characteristics as the interactive QMF product; it becomes a very large conversational program.

QMF is an assembler language program that contains CICS commands. It can be linked with other assembler language programs or with programs in one of the high-level languages (VS COBOL II, PL/I, or C/370). When you call QMF using a high-level language, the high-level language program must be linked first, and the resource definition online (RDO) program definition must specify that high-level language. Each high-level program has specific CICS considerations and restrictions. Refer to the high-level language programming guide and to the language considerations section in *CICS Application Programming Guide*.

In CICS, if you want to override any of the default QMF start parameters, you must specify these keywords on the START command. For example, the default mode from the callable interface is BATCH. To run an interactive QMF session you must issue the START command using DSQSMODE=INTERACTIVE.

- Program execution levels

  For QMF Version 3 Release 1 Modification 1, the interface between the QMF-supplied interface and the main QMF program was changed to run on a lower program level than the user's application program. Because of this change, user programs are not affected by environmental conditions such as the handle conditions set by QMF.

> **Note to CICS/OS/390 users**
> To use the QMF 6 callable interface after migrating from 3.1, you must link-edit the programs that currently use the QMF callable interface. If you are migrating from a later QMF release, you do not need to link-edit again.

- CICS region (OS/390) or partition (VSE) considerations

  The user program containing the QMF interface communications module and the main QMF module must run in the same region or partition. QMF resources, as described during QMF installation, must also be allocated to the CICS region or partition that runs QMF.

- Database
  - **DB2 for VSE or VM:**: When you invoke QMF through the callable interface, your CICS transaction runs QMF using the database packages that have already been installed, and no further action is required.
  - **DB2 UDB for OS/390:** The CICS transaction that invokes your program must also be described to DB2 by a Resource Control Table (RCT) entry. For more information about RCT entries, see *DB2 UDB for OS390 Administration Guide* and *CICS System Definition Guide*.

The RCT PLAN name should be the same for both the callable interface program and the QMF product.

Chapter 4. Using the Command Interface for Applications

QMF provides an application interface to use QMF services from an ISPF dialog. This interface is the *command interface*. The command interface allows you to issue QMF commands from an ISPF dialog running under QMF. Using this interface, QMF communicates with the dialog through the ISPF variable pool, as shown in Figure 8.

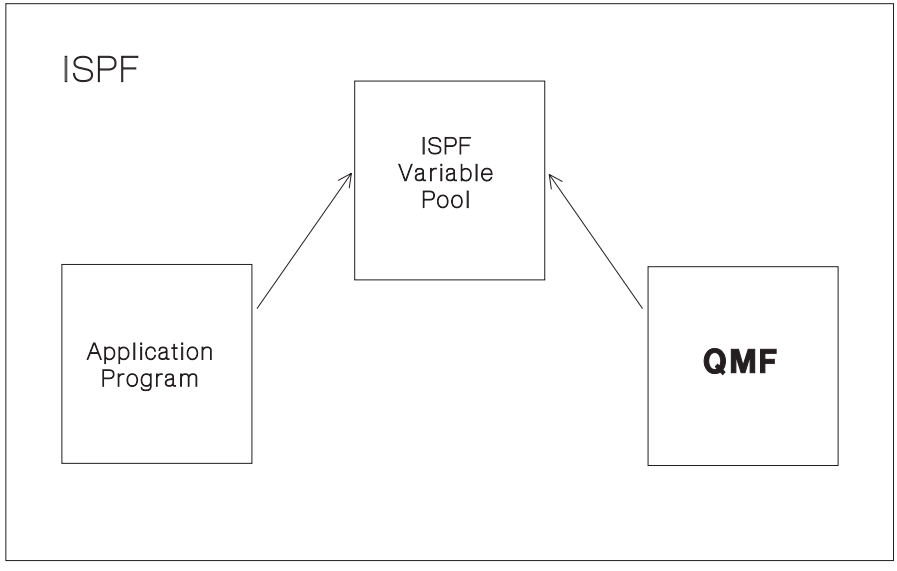


Figure 8. QMF command interface application interacting with QMF

Note to CICS users

The QMF command interface requires ISPF to run, but ISPF does not run in the CICS environment. Therefore, you need to use the QMF callable interface for application development under CICS.

To use the command interface effectively, you need to understand ISPF services and variable pools. See *ISPF: Dialog Management Guide and Reference* for more information on using ISPF.

Using the Command Interface for Applications

To use the command interface (DSQCCI), a QMF session must be running. You cannot start a QMF session using the command interface. You can start a QMF session under ISPF by using the ISPSTART command:

```
ISPSTART PGM(DSQQMFE) NEWAPPL(DSQE) PARM(...)
```

You use QMF commands from an ISPF dialog by calling the QMF command interface program DSQCCI using the ISPF SELECT PGM service. You pass the QMF command using the PARM option of the SELECT PGM service. To run a program that uses the command interface, you need to follow these steps:

1. Start ISPF.
2. Start QMF.
3. Run your program using the CMS or TSO command.

Important: If you omit any of these steps, your program fails.

Writing a program that uses the command interface: An example

Suppose you want to use the command interface to display an ISPF panel that prompts a user to specify a query name, runs the query, and displays a report.

For this scenario, you do the following:

1. Write your command interface REXX program. Your program does the following:
   a. Displays the ISPF panel QRYNAME using the DISPLAY services:
      ```
      ADDRESS ISPEXEC "DISPLAY PANEL(QRYNAME)"
      ```
   b. Runs a QMF query based on user input from the previous DISPLAY service. Here, the ISPF variable QNAME contains the name of the QMF query:
      ```
      ADDRESS ISPEXEC "SELECT PGM(DSQCCI) PARM(RUN QUERY" QNAME ")"
      ```
   c. Lets the user view the result of the query, using the following command:
      ```
      ADDRESS ISPEXEC "SELECT PGM(DSQCCI) PARM(INTERACT)"
      ```
2. Start ISPF.
3. Start QMF.
4. Call your program using the CMS or TSO command from the QMF command line. For example, if your program is named GETINFO, your command looks like one of the following depending on your system:
   ```
   CMS GETINFO
   TSO GETINFO
   ```

Invoking the command interface

The command interface is a program named DSQCCI. You can invoke it from a program through the ISPF SELECT service.

When you invoke the command interface through the ISPF SELECT service, pass the uppercase QMF command to be run in the PARM operand. Issue the following command:

```
SELECT PGM(DSQCCI) PARM(qmf_command)
```

All QMF commands specified as parameters to the command interface must be in uppercase, regardless of the QMF profile setting. ISPF does not automatically convert the commands from lowercase to uppercase, so if you specify your QMF commands in lowercase, QMF does not recognize them. If you wish prompting to be performed while QMF executes your QMF command, code the INTERACT command right in the front of your QMF command. Refer to "INTERACT" on page 57 for more information on the INTERACT command.

On the invocation, do not specify the NEWPOOL or NEWAPPL option. Omitting the NEWPOOL or NEWAPPL options ensures that the command interface can access your application's variables. The command interface uses the shared pool to communicate between QMF and your application.

The SELECT service requires you to use double ampersands on a RUN QUERY command. This prevents ISPF from interpreting the variable as one of its own.

The END command

When issued by the end user while the command interface (DSQCCI) is running, the END command terminates the DSQCCI invocation and returns control to the calling application. The QMF session remains active. Only an *abend* (abnormal termination) terminates a QMF session during a command interface invocation.

The EXIT command, or a severe error during a command interface invocation, causes QMF to set DSQCSESC to mark the session for termination. When the program that called DSQCCI ends and returns control to QMF, the QMF session then terminates.

Using variables in the command interface

The STATE command provides the current value for each QMF-provided variable. It can be used only in the command interface. When you issue this command, you can place the QMF variables in the ISPF variable pool through the VPUT command. Table 4 shows the subset of the available QMF variables that QMF places into the ISPF variable pool.

Table 4. QMF variables in the ISPF variable pool

Variable type	Variable name		Description
STATE command	DSQAAUTH	DSQAPLEN	QMF updates these variables when your application issues a STATE command.
	DSQABATC	DSQAPLNG	
	DSQACMDM	DSQAPPFK	
	DSQACRSR	DSQAPPRT	
	DSQADBCS	DSQAPRMP	
	DSQADBMG	DSQAPSPC	
	DSQAIACT	DSQAPSYN	
	DSQAITEM	DSQAPTRC	
	DSQAITLO	DSQAPWID	
	DSQAITMN	DSQAQMF	
	DSQAITMO	DSQAREVN	
	DSQALANG	DSQAROWS	
	DSQAMODL	DSQASUBI	
	DSQAMODP	DSQASUBP	
	DSQAOGRP	DSQATRAC	
	DSQAPCAS	DSQAVARN	
	DSQAPDEC		
CONVERT command	DSQCL*nnn*		QMF updates these variables when processing a CONVERT command.
	DSQCQ*nnn*		
	DSQCQCNT		
	DSQCQLNG		
	DSQCQTYP		

Table 4. QMF variables in the ISPF variable pool (continued)

Variable type	Variable name	Description
Command message	DSQCATTN	QMF updates these variables each time it processes a command issued by the command interface.
	DSQCIM00	
	DSQCIM*nn*	
	DSQCIMID	
	DSQCIMNO	
	DSQCIMSG	
	DSQCSESC	
Query message	DSQCIQ00	QMF updates these variables when RUN QUERY returns an error message.
	DSQCIQ*nn*	
	DSQCIQID	
	DSQCIQMG	
	DSQCIQNO	
	DSQCISQL	

To use QMF variables in the ISPF variable pool, use the 8-character name for the variable. For a description of the values and extended names of these variables, see Appendix E, "QMF Global Variable Tables" on page 257.

Command interface return codes

Return codes for the command interface are the same regardless of the language of your application. The return code can be positive or zero. A value of zero indicates successful execution. A positive value indicates that the execution failed or was in some way abnormal.

Return codes appear in a variable in the user's EXEC or CLIST. If you run a REXX EXEC, the return code is in the REXX variable called `RC`; if you run a CLIST, the return code is in the CLIST variable &LASTCC.

The following example shows an EXEC that examines a return code.

Example

Your application contains the following code:

```
ADDRESS ISPEXEC SELECT PGM(DSQCCI) PARM(RUN QUERYA (FORM=FORMA))
Select
  When (RC = 0)  Then nop
  When (RC = 64) Then
      Say "You must run QMF with ISPF to use command interface."
  When (RC = 100) Then
      Say "You need to start QMF before you begin your application"
  Otherwise
      Say "Unexpected error ("RC") from QMF command interface."
End
```

The code runs a query and then tests for an error using REXX RC.

You can place code for handling errors in program modules as well as in EXECs or CLISTs.

Return codes 0 through 16

Return codes 0 through 16 describe the QMF processing of the command passed with the command interface. When the command interface returns one of these codes, it also returns the values of the QMF command message variables in the application's ISPF shared pool. The codes are shown in Table 5.

Table 5. Return codes 0 through 16

Value	Explanation
0	Successful execution
4	QMF session marked for termination by an EXIT or END command
8	Execution failed, but error didn't mark the session for termination
16	Severe error: session marked for termination

A return code of 4 occurs *only* on the command that caused the session to be marked for termination. If the application then attempts to run another command, QMF returns another return code value to the user.

Return comes of 20 or more

These codes usually reflect some failure in the command interface (DSQCCI). The failure has made it impossible to copy a variable into the application's shared pool. As a result, the QMF variables might be invalid, or perhaps they haven't been set. The same can be true of the STATE variables if your

program uses the STATE command. (A variable has been "set" if it has been copied into the application's shared pool.)

These return codes usually indicate more serious errors than those in the 0 through 16 range. Some could require the services of your IBM Support Center.

The following table contains explanations of the return codes with values of 20 or more. *Shared variables* refers to the QMF variables (and the STATE variables, if the current command is the STATE command).

For some codes, the command was run but the shared variables weren't set. This might seem puzzling if the command was a STATE command. What this means is that QMF ran the STATE command properly. QMF then expected the command interfaceto set the updated shared QMF and STATE variables; however, the command interface failed to do this, for the reason given in the explanation of the error code. The codes are shown in Table 6.

Table 6. Return codes of 20 or more

Value	Explanation
20	A user exit routine called the command interface. These calls are always invalid. The command passed to the command interface was not run. The shared variables weren't set.
24	An error occurred in an ISPF VCOPY command. The command passed to the command interfacewas run. The shared variables weren't set.
32	An error occurred in an ISPF VREPLACE command. The command passed to the command interface was run. The shared variables weren't set.
36	An error occurred in an ISPF VPUT command. The command passed to the command interfacewas run. The shared variables weren't set.
40	An error occurred in an ISPF VREPLACE command. This code applies only to the execution of the STATE command. The command passed to the command interface was "run", but the shared variables weren't set.
44	An error occurred in an ISPF VPUT command. The code applies only to the execution of the STATE command. The QMF variables were set, but not the STATE variables.
60	Invalid call to the command interface. For example, the user might have invoked an application from a QMF prompt panel, and the application invoked the command interface. The command passed to the command interface was not run. The shared variables weren't set.
64	Not invoked in an ISPF environment. This error is issued when DSQCCI is run and ISPF is not active. For example, the user could have called DSQCCI without using an ISPF SELECT PGM command.
100	Failure to locate anchor. This error occurs when the application tries to issue a QMF command without having QMF active. You need to start QMF before you begin your application. The command passed to the command interface was not run. The shared variables weren't set.
104	Failure to locate anchor. The command passed to the command interface was not run. The shared variables were set but aren't valid.

Chapter 5. ADDRESS QRW: Using the QMF Command Environment

Note to CICS users
REXX is not supported in QMF CICS; therefore, ADDRESS QRW does not work in the CICS environment.

The REXX language always operates in a *command environment* that determines the default resolution of your commands. The default command environment is CMS or TSO, depending on your operating system.

When QMF is started, you can establish QMF as the default command environment through the REXX ADDRESS command. You can use this command alone or before a QMF command:

```
ADDRESS QRW
ADDRESS QRW command
```

With `ADDRESS QRW`, QMF remains the default command environment until you issue another ADDRESS command. With `ADDRESS QRW` *`command`*, QMF is the command environment for that command only.

When you are using a QMF procedure with logic, QRW is the default command environment.

Although QMF behaves the same whether you use the callable interface or the REXX command environment, `ADDRESS QRW` is *not* part of the SAA Query CPI. Use this command only if you do not plan to port your application to another SAA query environment.

The following example shows how to use the QMF command environment:

```
⋮
call dsqcix "START (DSQSMODE=INTERACTIVE"
if dsq_return_code=dsq_severe | dsq_return_code=dsq_failure
  then exit dsq_return_code

ADDRESS QRW
"RUN PROC MONDAY_P"
if dsq_return_code=dsq_severe | dsq_return_code=dsq_failure
  then exit dsq_return_code

"EXIT"
if dsq_return_code=dsq_severe | dsq_return_code=dsq_failure
  then exit dsq_return_code
⋮
```

Figure 9. Example of using the QMF command environment

Chapter 6. Writing QMF Applications that Use ISPF

You can write applications that have their own user interfaces and bypass all the QMF panels. One way to write these applications is to use ISPF to help you create your own panels, and pass the user's entries to QMF as variables. You can also take advantage of other ISPF services to create or read QMF objects.

Note to CICS users
ISPF does not run in the CICS environment, so ISPF services are not available under CICS.

ISPF helps you provide an end-user interface on mainframe systems. You can use ISPF with the QMF callable interface or command interface.

This chapter outlines considerations for using the callable interface with ISPF. For general information about using the callable interface, see Chapter 3, "The Callable Interface" on page 19. For information on using the command interface, see Chapter 4, "Using the Command Interface for Applications" on page 29.

Starting and running QMF from an ISPF application

The callable interface works with ISPF the same way it works with any other program. However, there are a few considerations:

The callable interface must match the language of Your ISPF dialog

If your ISPF dialog is a PL/I program, for example, you must use the QMF callable interface for PL/I.

You must use the correct language identifier

You must start your ISPF application with an ID of DSQ*n*, where *n* is a National Language Feature (NLF) identifier. This application ID prevents QMF from overriding your ISPF environment, such as the function key settings and labels. To start the application that starts QMF, use the following ISPF statement:

```
SELECT PGM(MYPROG) NEWAPPL(DSQn)
```

where *n* is the NLF identifier. The PL/I program `MYPROG` then starts QMF using the callable interface START command.

The ID DSQ*n* ensures that the ISPF environment remains intact even after QMF is started.

For a list of NLF identifiers, see Table 7 on page 68.

Use GET GLOBAL or SET GLOBAL Iistead of the STATE command

The GET GLOBAL and SET GLOBAL commands work for all the QMF global variables; the STATE command works only for variables containing state information. See tables of these variables in Appendix E, "QMF Global Variable Tables" on page 257.

Running queries that contain variables

Your applications can run queries that contain variables. You can run these queries from an application that uses ISPF services in one of three ways:

- Use ISPF file tailoring services.

  With this technique, you represent the query by an ISPF file tailoring skeleton. In that skeleton, the portions of the query that can change appear as ISPF dialog variables. After giving these variables the proper values, your program starts certain ISPF file-tailoring services. The result is a sequential file containing the query.

  The program can then import the query into QMF temporary storage and have QMF run it. The requisite IMPORT and RUN commands can be run through the callable interface or command interface.

  To use this technique, you must know how to define ISPF dialog variables in your program using the ISPF VDEFINE service. See *ISPF: Dialog Management Guide and Reference*

- Use the Program Development Facility (PDF) editor to create QMF objects

  You can use the PDF editor with PDF edit macros to design and control data entry to queries, procedures, forms, and profiles. You can write PDF macros using REXX programs.

- Create a query using an ISPF dialog.

  To create a file that contains an SQL query, your program can use ISPF display services to display a screen and create a file based on input from the user. This file can then be imported into QMF and run.

Invoking a program from a QMF procedure with logic under ISPF

If you are running QMF under ISPF, you must use the ISPF SELECT service to call your callable interface program or REXX programs from a procedure with logic. You must use the PGM keyword to tell ISPF that you are running your callable interface program as an ISPF dialog function. The syntax for this command is as follows:

```
ADDRESS ISPEXEC "SELECT PGM(programname)"
```

For REXX programs, you use the CMD keyword to tell ISPF that you are running your program as an ISPF dialog function. The syntax for this command is as follows:

```
ADDRESS ISPEXEC "SELECT CMD(cmdname)"
```

or

```
ADDRESS ISPEXEC "SELECT CMD(cmdname parameters)"
```

`cmdname` is the name of your callable interface or REXX program.

Using ISPF commands from a procedure with logic

Whenever you start QMF under ISPF, QMF is started as an ISPF program. Therefore, to run any ISPF commands from a QMF procedure with logic running under ISPF, you must transfer from the QMF program dialog to an ISPF command dialog. To do this, you must issue an ISPF `SELECT CMD` from your QMF procedure.

To set the correct ISPF environment and run a REXX program containing your ISPF commands, use the following ISPF SELECT command with the CMD keyword:

```
ADDRESS ISPEXEC "SELECT CMD(userprogram)"
```

`userprogram` is a REXX program that contains your ISPF commands.

For example, if the REXX program that contains your ISPF commands is called DIALOG, include the following command in your procedure with logic:

```
ADDRESS ISPEXEC "SELECT CMD(DIALOG)"
```

For more information on ISPF, see *ISPF: Dialog Management Guide and Reference*

You also can use a QMF CMS or TSO command to run your REXX program containing ISPF commands, for example **`CMS`** `DIALOG` or **`TSO`** `DIALOG`. QMF issues the ISPF SELECT CMD statement for you.

Writing QMF Applications that Use ISPF

If you are running QMF under ISPF and your procedure with logic starts a program requiring ISPF services, your procedure must start this program using the ISPF SELECT CMD environment as described in the preceding examples. For example, suppose you are running QMF under ISPF and your procedure with logic starts DB2's DSN command. Because the DSN command uses ISPF services, you should use one of the following commands to issue the DSN command:

```
ADDRESS ISPEXEC "SELECT CMD(DSN)"
```

or

```
ADDRESS ISPEXEC "SELECT CMD(DSNEXEC)"
```

where DSNEXEC contains the ADDRESS TSO DSN statement.

Callable interface considerations

If you want to use the LIBDEF function in your QMF applications that were link edited prior to QMF Version 7 and that use the callable interface, you must re-link edit your applications using the QMF Version 7 interface module.

Using the EDIT command with ISPF

When you run your QMF application under ISPF, you can edit your QMF SQL query or procedure using the following commands:

```
EDIT QUERY
EDIT PROC
```

If you issue the QMF EDIT command from within a PROC or QUERY panel, you do not need to specify the PROC or QUERY object types. EDIT assumes these values when you invoke it from their respective panels. By default, the QMF EDIT command places your procedure or query in a PDF editor session. QMF starts the PDF editor using the QMF application ID DSQ*n*, where *n* is the NLF identifier. QMF also sets the function keys and the location of the command line to match your QMF application.

To override this default, use the EDIT QUERY and EDIT PROC commands as follows:

```
EDIT QUERY (E=name
EDIT PROC (E=name
```

`name` can be either of the following:

- An editor available to you

- The name of a REXX program that specifies an application ID other than DSQE. You might want to use an application ID different from the QMF application ID if you want to have function keys different from those QMF provides.

If you are using PDF EDIT options that require PDF PROFILE data set members, you must create those members. For example, the PDF EDIT RECOVERY option requires a DSQ*n*EDRT PROFILE data set member (where *n* is the NLF character) that must exist before you use the EDIT command.

For more information about the QMF EDIT command, see online help and *QMF Reference*.

Using ISPF to debug applications

The QMF trace facility only traces QMF messages and commands. To trace the ISPF commands of your application, write the messages to the ISPF log file or data set. This ISPF service complements the QMF Trace facility described in Chapter 10, "Debugging Your QMF Applications" on page 123.

Using ISPF log service

Use the ISPF log service to write a message to the ISPF log file. For example, in REXX, the ISPF command to write a message to the ISPF log is:

```
ADDRESS ISPEXEC LOG MSG (message-id)
```

message-id is the identification of the message that is to be retrieved from the message library and written to the log.

Using PDF dialog test

If your installation has PDF, you can use the Dialog Test service (log option) to browse the contents of the log file or data set. You can also print the log file or data set when you exit ISPF.

The Dialog Test service has many other useful options for debugging your application. You can perform debugging interactively. You can run all or portions of your application, examine the results, make changes, and rerun it. You can also use Dialog Test to:

- Start selection panels, command procedures, and programs
- Display panels
- Add variables and modify variable values
- Run ISPF dialog services
- Add, modify, and delete breakpoint definitions
- Add, modify, and delete function and variable trace definitions

The trace (TRACES) option of the Dialog Test service enables you to create, change, and delete trace definitions. Therefore, you can monitor dialog service calls and dialog variable usage. During processing, if any of the trace definitions are satisfied, trace output is written to the ISPF log. You can use the LOG option of Dialog Test to browse the ISPF log, or you can examine the printed output when you exit ISPF.

For more information about ISPF services in general and Dialog Test in particular, refer to *ISPF Dialog Management Guide and Reference*

Chapter 7. Writing Bilingual Applications

Many businesses operate in several different countries, or in multilingual countries, where interactive applications need to run in several different national languages. Beginning with Version 3.2, you can write one English application and run it in any national language that QMF supports.

A QMF environment in a language other than English is a *National Language Feature* (NLF). An NLF provides a user with a QMF session that is tailored to a specific language. A German NLF, for example, allows you to operate QMF in a German language environment.

QMF provides bilingual support for commands and forms. You can run English QMF commands and display English forms in any NLF, or write translatable applications. This chapter provides information about working with QMF in multiple or non-English language environments.

Creating bilingual objects for your applications

The objects in a bilingual application are like any other QMF object. The key is that you either create or save them in English. How you do this depends on the specific object:

Queries
: You can create prompted and QBE queries in your native language, or you can create SQL queries in English.

Forms
: Always create forms in the presiding language, and then save them, either using the default language on the SAVE command (ENGLISH) or the presiding language.

  The global variable DSQEC_FORM_LANG controls which language is used for the SAVE command. The default value is 1 for English. A 0 value specifies that the forms are to be saved in the presiding session language.

Procedures
: You can create procedures in either English or the presiding language.

You can translate a form that you create and save in an NLF to English by issuing a SAVE command. For example, in French, the command to save a form called SEMAINE_F as WEEKLY_F in English is:

```
SAUVER FORMAT SEMAINE_F EN WEEKLY_F (LANGUE=ANGLAIS
```

This converts your NLF form to an English form that you can use in your bilingual application.

Using the command language variable

You can begin using English commands in an NLF session when you have the objects you need for your application. To do this, set the presiding language variable, DSQEC_NLFCMD_LANG, to English. This variable lets you switch between English and the presiding language of the NLF session.

Assuming your application is a procedure named WEEKLY_P, you would use the following commands:

```
"GET GLOBAL (CURR_LANG=DSQEC_NLFCMD_LANG"
"SET GLOBAL (DSQEC_NLFCMD_LANG='1'"
"RUN PROC WEEKLY_P"
"SET GLOBAL (DSQEC_NLFCMD_LANG=CURR_LANG"
```

These commands can be part of any valid QMF application, from an initial procedure to a high-level language program, but they must be in this order. The commands work in the following way:

Saving the presiding language value

The GET GLOBAL command saves the value for the presiding language in a variable called CURR_LANG. When that value is saved, you can reset DSQEC_NLFCMD_LANG to the value for English, 1.

Running your application

When your QMF session is set to English, you can run your English application. Any commands the user enters must be in English. However, if a user presses a function key, the underlying command is assumed to be in the presiding language.

QMF assumes that prompt panels are in the user's presiding language. For the EXPORT and IMPORT command prompt panels, the default file type is in the presiding language, too.

If the NLF provides uppercasing options in the profile, QMF adheres to the user's presiding language option, even when the user runs English commands.

Returning to the presiding language

After your application ends, you should reset the command language variable to the original value.

Using an initial Procedure in a bilingual application

If your application starts QMF and runs an initial procedure, QMF runs that procedure every time the user issues the END command. QMF terminates if this procedure encounters an error. For example, if the user is running in English and issues an END command in the presiding language, QMF interprets the command as an error and terminates.

You can avoid this situation in one of two ways:

- Change the initial procedure to handle bilingual applications.

  A bilingual initial procedure includes the commands shown in Figure 10.

```
"GET GLOBAL (CURR_LANG=DSQEC_NLFCMD_LANG"
"SET GLOBAL (DSQEC_NLFCMD_LANG=0"
:
/* QMF commands in the presiding language */
:
"SET GLOBAL (DSQEC_NLFCMD_LANG=CURR_LANG"
```

Figure 10. An initial procedure in a bilingual application

- Avoid running the initial procedure after the END command.

  You can set the variable DSQEC_RERUN_IPROC to 0 so that QMF does not run the initial procedure when the user issues the END command.

Using English commands

For most QMF commands, you must change the presiding language variable before you can run the command in English. To display a prompt panel or message in a presiding language, however, some English commands must run in any NLF, even when the presiding language variable is not set to English.

For example, if you have an interactive application that you want to write in English and run in an NLF, you need to use the MESSAGE command to give the user customized messages. In addition, you need the INTERACT command to display the message, as in the following example, which can be run in a French NLF session:

```
proceed_text = 'Continue...'
"RUN WEEKLY_Q"                    /* Use the English RUN command  */
"SET GLOBAL (DSQEC_NLFCMD_LANG=0" /* switch back to French        */
"MESSAGE (TEXT='"proceed_text"'"  /* message in French            */
"INTERACT"                        /* show the report with message */
```

The following commands work in any NLF:

- GET GLOBAL
- INTERACT
- MESSAGE
- SET GLOBAL
- START

Multilingual environments

When one or more NLFs are installed in your QMF installation, a *multilingual environment* is created. In such an environment, you can, with the proper authorization, have your choice of one presiding language for each QMF session. For example, you can choose English for one session and German for another, provided the German NLF is installed. Although you can't switch languages during a QMF session, you can switch the command language variable. Then you must end the current session and begin another to obtain the appropriate language environment.

QMF session environments

When no NLFs are installed, the only available QMF session environment is the English-language environment. When an NLF is installed, the NLF environment differs in some ways from the English-language environment.

Environmental similarities

In many aspects the QMF session environment is the same no matter which NLF is in operation. The most important similarities are:

Capabilities

In general, you can do anything in an NLF session that you can do in an English-language session. You can create and save all the temporary storage objects, format and print reports, and issue SQL commands. You can also run Prompted Query, SQL and QBE queries, and QMF procedures. The difference between the English and NLF environments lies not in what you can do, but in what you must enter at your terminal to get it done and what languages you see on your terminal screen.

SQL and QBE

The verbs, operators, and keywords of the SQL and QBE languages are not translated.

Usage codes for forms

These are identical; they are not translated.

System commands

CMS, TSO, or CICS and ISPF commands can still be issued from QMF through QMF's CMS, TSO, or CICS command. This command is unaffected by translation: you enter `CMS`, `TSO`, or `CICS` followed by the command to be run, and write the command exactly as you would if you were running it outside of QMF.

Environmental differences

Some of the more important differences between the NLF environment and the English-language environment are:

The QMF language

Every NLF has a complete set of verbs and keywords for the QMF language. These verbs and keywords must appear in your QMF commands when you are operating in the NLF's language environment. For a given NLF, these words might be translated.

For example, suppose that in the German NLF, the verb DISPLAY and the keyword PROC were translated into ANZEIGEN and PROZEDUR, respectively. During a German-language session, QMF understands the command ANZEIGEN PROZEDUR, but does not understand DISPLAY PROC.

Some elements of the QMF language are command synonyms and can be translated. As a result, each NLF has its own uniquely named command synonym table. When the NLF is installed, its command synonym table is created, and the profile for the NLF indicates the command synonym table name for that NLF.

QMF panels and messages

Every NLF has a complete set of QMF messages and panels. Like the verbs and keywords for QMF commands, these might not be translated, but in most cases they are translated. Within the panels and messages, the fixed portions of text can be translated. Variable information, such as a query name, is not translated.

Allowable panel input

Many QMF panels, such as prompt panels and form panels requiring user input, restrict the range of some entries to a small set of keywords. Most of the allowable values are translated. YES and NO responses in English, for example, are JA and NEIN in German.

Profile parameter values

Writing Bilingual Applications

In a multilingual environment, users have a separate profile for each NLF they can use for a QMF session. For each of these profiles, the parameters are the same and have the same meanings. But, as part of QMF's supply of keywords, their names can be translated. For certain parameters, the values they can assume are translated also.

For example, in an English profile, the CASE parameter can have the value UPPER, STRING, or MIXED. In a German profile, the CASE parameter is the SCHRIFT parameter, and the values it might assume are GROSS, KETTE, and GEMISCHT.

Exported and saved form objects

The SAVE, EXPORT, and IMPORT commands let you specify the language in which you want form objects saved. You can save them in English, or in the presiding language of your current session. For more information on these commands, see *QMF Reference*.

Sample tables and queries

IBM might supply translated versions of the English sample tables and queries with some of its NLFs. For example, Japanese users have sample tables translated from the English tables.

Creating translatable applications

You can save time in adapting an application to new languages by using variables for as many language-sensitive objects as you can. These variables can include:

- The verbs, object names, and option identifiers in a QMF command
- Installation-defined panel names

  If you are creating your own panels for your application, you need a set of translated panels for each language under which the application is to run. Give these panels unique names and make them available to the application users. The application can then use variables for the panel names.
- Installation-defined message identifiers

  Like panels, the messages should be translated into the appropriate NLF languages. The application can use variables for the message names.

Using variables lets you use the same program in several NLFs.

Chapter 8. QMF Commands in Applications

Any command that is valid on the QMF command line in a particular environment is valid in an application. In addition, QMF provides commands that are specially designed for applications.

This chapter describes QMF commands that users commonly include in their programs and describes their use in application development. For more information on commands and their syntax, see *QMF Reference*.

CONNECT

You can use the QMF CONNECT command to connect to a different system within your distributed network, with remote unit of work, during a QMF session. You can also use the QMF CONNECT command to access remote databases supported by QMF. When you connect to the remote system, this system becomes the *current location*.When you write applications, you can issue this command from:

- The callable interface
- The command interface
- Within a procedure (linear or with logic)

Certain aspects of your applications can be affected when you use the QMF CONNECT command to start remote unit of work. Be aware of the following considerations:

- When your application connects to a new location, the QMF profile, command synonyms, and function keys are reinitialized to the values at the new (current) location.
- All callable interface and command interface programs that start QMF and issue QMF commands must reside on the same system as the user (the local system). After the program starts QMF at the local system, the program can issue a QMF CONNECT command to connect to a remote database. Any subsequent QMF commands or SQL statements that affect database objects are run at the current location (the remote database).
- All programs started by QMF must follow the conventions of the operating system on which QMF is running (the local system).
- Different types of commands behave differently with remote unit of work. When your applications use remote unit of work, be aware that all system-specific and most QMF commands run at the system where QMF is running (usually, your local system). However, when a QMF command does either of the following:

- Sends SQL commands to the database
- Uses or alters QMF objects and data stored in the database

These commands affect the database at the *current location*.

•

An example

You are logged on to your local VM system (SANJOSE) running CMS. You want to write a REXX callable interface program that does the following:

1. Starts a QMF session

   ```
   CALL DSQCIX "START"
   ```

2. Connects to the remote DB2 database (DALLAS)

   ```
   CALL DSQCIX "CONNECT TO DALLAS"
   ```

3. Runs a procedure with logic(EARNINGS) that queries the remote database for data, formats the data, and prints the report

   ```
   CALL DSQCIX "RUN PROC EARNINGS"
   ```

   The procedure EARNINGS contains the following logic:

   ```
   ⋮
   "RUN QUERY EARNQ (FORM=EARNF"
   "PRINT REPORT"
   ⋮
   ```

   This procedure does not contain another CONNECT command.

4. Ends the QMF session

   ```
   CALL DSQCIX "EXIT"
   ```

When you write this program, be aware of the following:

- Your application program must reside on your local (SANJOSE) VM system.
- The QMF session starts on your local (SANJOSE) VM system.
- Your procedure must reside on the remote database (DALLAS); DALLAS is the current location when the application runs the procedure in step 3.
- Any QMF objects (in this case, the query and the form) used in the application or the procedure after the CONNECT command in step 2 must reside on the remote database (DALLAS).
- The SQL query EARNQ run by the procedure in step 3 runs against the DB2 database in DALLAS.
- The PRINT command in the procedure EARNINGS prints the report on the printer named by the profile at the current location (DALLAS). For this example, assume that the profile at the current location (DALLAS) defines the printer to be at the local VM system (SANJOSE).

For more information about connecting to a remote location with the QMF CONNECT command, see online help.

END

You can include the END command in your application to end the QMF session. You can also design your application so that your end user needs to press the End function key or type the END command at the command line to end the interactive QMF session and return control to the application.

The rules governing the END command depend on the type of session in which the END command is issued. This section discusses how the END command operates in each of the following types of QMF sessions:

- Session started by the callable interface
- Interactive session using ISPF with an initial procedure
- Interactive session using ISPF without an initial procedure
- Interactive session begun by an INTERACT command
- Batch mode session

Session started by the Callable Interface

When issued by the end user in an interactive session started by the callable interface, the END command terminates the interactive session and returns control to the calling application. Before QMF terminates the active session, QMF makes the Home panel the current panel. QMF remains active. Only the EXIT command or a severe error terminates QMF after it is started by a callable interfaceapplication.

Interactive session with an Initial Procedure (DSQSRUN)

QMF starts an interactive session that runs an initial procedure when QMF is started using these keywords:

```
DSQSRUN=xxxxx,DSQSMODE=I
```

where xxxxx (the value of the DSQSRUN keyword) is the name of a QMF initial procedure. This keyword is explained in “START command keyword” on page 67.

After QMF starts, QMF runs the initial procedure. After this procedure ends, the user is in an interactive session, unless the current panel is the Home panel. If the current panel is the Home panel, QMF does not start an interactive session. Instead, QMF immediately restarts the initial procedure if both of the following statements are true:

- No serious errors have occurred
- The DSQEC_RERUN_IPROC global variable is set to 1

You should avoid writing your initial procedure so that the current panel at the end of the procedure is the Home panel. If the Home panel is the current panel at the end of the initial procedure, an uninterruptible loop occurs; QMF can appear as though it is not starting or running the initial procedure. To avoid this, make sure that either of the following occurs:

- The current panel at the end of the procedure is not the Home panel.
- The procedure contains either a QMF EXIT or INTERACT command.

When the end user issues an END command in the interactive session and DSQEC_RERUN_IPROC= 1, QMF simply restarts the initial procedure. Use the EXIT command to terminate the session.

When QMF is not started through the callable interface, you can use DSQ_RERUN_IPROC to control whether QMF reruns the initial procedure. If you set DSQEC_RERUN_IPROC = 0, then QMF terminates instead of rerunning the initial procedure when the END command runs. This variable has no effect on callable interface applications.

Interactive session without an Initial procedure

In this situation, the DSQSRUN parameter is not specified when QMF is started. This ensures that no procedure runs before the user receives control.

When the end user issues the END command from within this interactive session, QMF does one of the following:

- Makes the Home panel the current panel if the current panel is any other panel.
- Marks the session for termination if the current panel is the Home panel. If issued online, the END command terminates the session immediately. If issued in an application, the session ends whenever the application ends.

Interactive session begun by an INTERACT command

An application can begin a new interactive QMF session within the current interactive QMF session by using the INTERACT command described in "INTERACT" on page 57. The old session can be a primary session, with or without an initial procedure, or it can be a session begun by another application.

In the new session, the user can enter an END command online, or an application can issue one. Either way, running the END command marks the interactive session for termination, no matter what the current panel is.

If issued online, the END command terminates the session immediately. If issued in an application, the session ends whenever the application ends. When that session ends, control returns to the application that started it.

Batch mode session

A QMF batch mode session runs in a noninteractive session on all environments supported in QMF. You can start QMF and prevent screen display by specifying batch mode (DSQSMODE=BATCH), which is the default. You must specify an initial procedure using DSQSRUN when using DSQQMFE; however, when using the callable interface, you do not have to specify an initial procedure.

During the batch QMF session, the initial procedure can issue an END command, or it can start an application that issues an END command. The results are like those for an interactive session without an initial procedure. The END command:

- Makes the Home panel the current panel if any other panel is the current panel.
- Marks the session for termination if the current panel is the Home panel.

If issued by the initial procedure, the END command terminates the session immediately. If issued in an application, the session ends whenever the application issues the QMF EXIT command.

During the session no interaction is allowed. Therefore, the session cannot begin a new session.

EXIT

The EXIT command works the same regardless of how the QMF session was started: it marks all the user's sessions for termination. In batch mode, there is just one session. For an interactive session, this includes the primary session and every session begun by the INTERACT command.

When the EXIT command is entered on the command line, the session in which it is entered is terminated immediately. Each session begun by the INTERACT command terminates as the application that started it completes. When the EXIT command is issued in an application, the session ends when the original QMF session ends. All interactive sessions begun by the INTERACT command must end before QMF terminates.

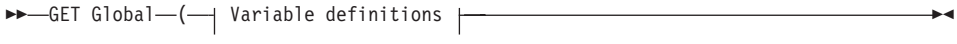
In a callable interface program, it is important to include the QMF EXIT statement when the application is done using QMF. If you forget to include this command, your QMF session remains active until you log off or until your batch job completes.

When the user or an application issues the EXIT command, QMF sets DSQAO_TERMINATE to 1 (marked for termination). Only an application running within QMF can test and use this global variable. If

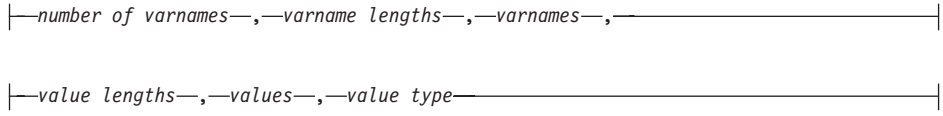
DSQAO_TERMINATE is set to 1 when QMF returns to the main QMF session, QMF immediately terminates and releases resources.

GET GLOBAL

You can use the GET GLOBAL command to access QMF global variables in your application. For languages other than REXX, QMF provides an extended syntax of the GET GLOBAL command.



Variable definitions:



The parameters specified on the GET GLOBAL command define the storage that your application program uses to store the variable names and values returned by the GET GLOBAL command.

number of varnames
The number of variables requested.

varname lengths
A list of lengths for each variable name specified.

The length of the variable name should be equal to the actual length of the global variable name in your storage area. An 18-character area padded with trailing blanks is allowed.

varnames
A list of names of the QMF variables.

Because QMF deletes trailing blanks, you should not specify trailing blanks in global variable names.

value lengths
A list of the lengths of the values of the variables.

The following rules apply to the variable value:

- If the value length you supply is less than that of the value stored in QMF, QMF truncates on the right and returns a truncated value.
- If the value length you supply is greater than that of the value stored in QMF, QMF returns a value padded with trailing blanks.
- Integer lengths should always be 4 bytes.

values
A list of variable values.

value type
The data type of the storage area that contains the values. It must be either character or integer.

INTERACT

The INTERACT command places end users into interactive QMF or GDDM ICU sessions. While in these sessions, the end users can enter commands as though they are in normal interactive sessions of these products.

INTERACT has two forms: session and command.

The session form of INTERACT

When you issue the INTERACT command, QMF places the user on the current panel and allows the user to issue QMF commands interactively. The INTERACT command provides another QMF "session" within your current session. The INTERACT command can place the user in either an interactive QMF session or an interactive GDDM ICU session.

- For an interactive QMF session:

  Issue the INTERACT command following a QMF command that would normally display a QMF panel. In this session, the user can enter any commands that are valid for interactive QMF.
- For an interactive GDDM ICU session:

  Issue the INTERACT command following a command that normally makes QMF start GDDM ICU and display the ICU panel. In this session, the user can enter any commands that are valid for the ICU.

A scenario

If you run a procedure that requires only one step to produce your report, like the following:

```
/* This procedure prints the weekly sales report. */
"RUN QUERY WEEKLY_SALES_Q (FORM=WEEKLY_SALES_F"
"PRINT REPORT"
```

Figure 11. A simple procedure, without the INTERACT command

QMF displays the REPORT panel containing your formatted data with a message that says, `"OK, your procedure was run."`

However, you might decide to write a procedure involving several steps. If you want to see the intermediate results of a procedure, you must use the INTERACT command. To see the intermediate result of a procedure that runs

more than one query, insert an INTERACT command immediately following the first RUN command:

```
/* This procedure generates a report showing annual sales.  */
"RUN QUERY WEEKLY_SALES_Q (FORM=WEEKLY_SALES_F"
"INTERACT"
"RUN QUERY YEAR_TOTAL_Q (FORM=YEAR_TOTAL_F"
```

Figure 12. Using INTERACT in a procedure

Then, when you run this procedure from the home panel, QMF displays the REPORT panel containing your formatted data. Next, you enter the END command from the REPORT panel, and the procedure continues, running the second query and displaying the final report. If you omit the INTERACT command, QMF displays only the final report without showing the result of the first query.

The INTERACT command produces the same effect when it is issued through the callable interface, although in REXX the same commands look like this:

```
⋮
call dsqcix "RUN QUERY WEEKLY_SALES_Q (FORM=WEEKLY_SALES_F"
call dsqcix "INTERACT"
call dsqcix "RUN QUERY YEAR_TOTAL_Q (FORM=YEAR_TOTAL_F"
⋮
```

Figure 13. Using INTERACT in a REXX application

The `Call dsqcix "INTERACT"` line is the REXX syntax for issuing the INTERACT command through the callable interface. You need to use the syntax required by your programming language to issue the INTERACT command through the callable interface.

Suppressing the display of reports

If you run a query in a QMF callable interface application, QMF by default displays the resulting report. However, you can tell QMF not to automatically display the resulting report by setting the DSQDC_DISPLAY_RPT global variable to zero (0). You can also set this global variable on the START command by specifying `DSQADPAN=0`.

This global variable is valid only when the RUN QUERY command is issued from an application. It does not affect the display of reports when RUN QUERY is issued from the QMF command line.

Ending an INTERACT session

When the user issues the END command, control returns to the process that issued the INTERACT command; however, the two sessions are not

independent. Anything done during the INTERACT session remains in effect when the old session resumes. For example, if the user modifies the current form object in the new interactive session, the current form object in the old session contains these modifications when the new session ends.

If you want your application to display the QMF Home panel after the user issues an END command from a QMF object panel (the way interactive QMF does), add the logic from "A REXX example of using an INTERACT l'QMF720oop" on page 224.

The command form of INTERACT

The command interface (DSQCCI) runs QMF commands interactively only when the command interface application uses the command form of INTERACT and QMF is running an interactive session (DSQSMODE=I).

The command form of INTERACT has no effect on a command issued through the callable interface. In the callable interface, the only way to control whether commands are run interactively is to set the START command keyword DSQSMODE. For more information about the DSQSMODE keyword, see Table 7 on page 68.

Use the following syntax to request interactive execution of a designated command. Issue the command:

```
INTERACT command
```

where *command* is the designated command. QMF runs this command interactively when any dialog between QMF and the user about the command's execution actually takes place. Various QMF prompt and status panels can appear in this dialog.

For example, the following command displays the command prompt panel for RUN QUERY command options:

```
INTERACT RUN QUERY ABC ?
```

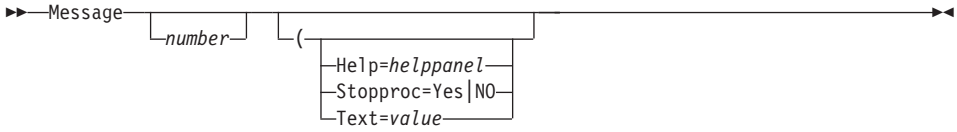
If interactive execution is not allowed, as in a QMF batch session, the command form of INTERACT has no effect on the command it is preceding.

You can check to see if interactive execution is allowed in the current session by examining a variable named DSQAO_INTERACT; a value of 1 means that INTERACT is allowed. A batch application, for example, does not allow interactive execution. See 259 in Appendix E, "QMF Global Variable Tables" for details on DSQAO_INTERACT.

MESSAGE

When you are writing applications, you often want to give specific messages to your users about the information displayed for them or the function they should perform next. You can write your own messages and display them on QMF panels through the MESSAGE command. In ISPF, you can also specify that QMF display the message help for an ISPF error message.

The MESSAGE command syntax:



number (with ISPF only)
number is only valid under ISPF. This parameter is the identification number of a message definition in an ISPF message library.

HELP
Use this parameter to specify a help panel other than the one defined with the message normally displayed in this situation. Replace *helppanel* with the appropriate panel ID.

If you want to display a QMF panel, the panel's definition is in DSQPNLE, so you cannot modify the panel.

In ISPF, if you want to create and display your own panel, the panel's definition must be in an ISPF panel library, and this library must be concatenated to your ISPPLIB file or data set. The panel must be a help panel, not a menu or a data-entry panel.

In ISPF, if you have specified *number*, *helppanel* defaults to the help-panel indicator for the message definition specified by *number*.

In ISPF, if the message definition specified by *number* does not define a help-panel indicator, then the MESSAGE command does not provide message help. Instead, the QMF help for the object panel appears on the user's screen when the user requests help.

STOPPROC
Use `Stopproc` to suppress the execution of linear procedures by setting the *procedureterminationswitch.* The following command sets the procedure termination switch:

```
Message (Stopproc=Yes
```

When `Stopproc=Yes`, the procedure termination switch is on. The default value is `No` (off). This switch only affects linear procedures.

While this switch is on, any QMF procedure receiving control ends its execution immediately. While the switch is off, procedures run normally.

When the switch is off, only a MESSAGE command can turn it on. When the switch is on, it stays on until one of the following happens:

- Another QMF command is issued. This can be any QMF command, except a MESSAGE command with the option to turn the switch on.
- Control returns to the user when the application ends. A user can always issue online commands that run QMF procedures.

You can check to see whether the procedureterminationswitch is on by examining the variable DSQCM_MESSAGE. If the termination option is in effect, this variable contains the message for the MESSAGE command that turned on the termination switch.

TEXT option

Use **TEXT=** to define a message or to override the text in an ISPF message definition. Replace *value* with the character string to be used for the message. A value that contains blank characters must be surrounded with delimiters. Valid delimiters for a message value are single quotes, parentheses, and double quotes. When the delimeters are double quotes, the double quotes are displayed as part of the message. The maximum length for message value is 78 single-byte characters. A value longer than 78 characters is truncated to contain only the first 78 characters. QMF does not fold the text into uppercase; however, ISPF might fold the text into uppercase if MESSAGE is issued through DSQCCI (the command interface).

If your message contains quotes, you need to double the quotes in the **TEXT=** specification.

In ISPF, the default is the *long message text* of the ISPF message specified by *number*, which becomes the generated message. The text is left as it is; no folding takes place whatever the value of the CASE setting for the user's QMF profile.

Examples of using the MESSAGE command to generate messages

Suppose that you want to write an application, using a procedure, that runs two queries and displays two reports. When QMF displays the first report, you want to display a message that tells users to end the interactive session when they are ready to continue to the second report. You can write a linear procedure like that in Figure 14 on page 62, which includes a message defined by the MESSAGE command on the REPORT panel. To have your message appear on the REPORT panel, place the MESSAGE command immediately before the INTERACT command:

```
⋮
   RUN QUERY WEEKLY_SALES_Q (FORM=WEEKLY_SALES_F
   MESSAGE (TEXT='OK, press END when you are finished viewing this report.'
   INTERACT
   RUN QUERY YEAR_TOTAL_Q (FORM=YEAR_TOTAL_F
⋮
```

Figure 14. Example of using the MESSAGE command

If you use a procedure with logic, you can use a REXX variable in place of the text string, as in Figure 15. When you use REXX variables, you must use double quotes around the variable name in the *messagetext* text string.

```
oktext = 'OK, press END when you are finished viewing this report.'
"RUN QUERY WEEKLY_SALES_Q (FORM=WEEKLY_SALES_F"
"MESSAGE (TEXT='"oktext"'"
"INTERACT"
"RUN QUERY YEAR_TOTAL_Q (FORM=YEAR_TOTAL_F"
```

Figure 15. Using REXX variables with the MESSAGE command in a procedure

Examples of MESSAGE Commands with ISPF Available

- `MESSAGE MSG011X`
  - The message text is the long message in MSG011X.
  - The message help panel is the panel identified (if any) in MSG011X.
  - Whether the procedure termination switch is set after QMF processes the command is determined by the procedure termination switch in MSG011X.
- `MESSAGE MSG011X (HELP=PANELX STOPPROC=YES`
  - The message text is the long message in MSG011X.
  - The message help panel is a panel named PANELX.
  - The procedure termination option switch is turned on, which suppresses the execution of QMF linear procedures in the application.

SET GLOBAL

You can create your own global variables with the SET GLOBAL command and use them in QMF commands as substitution variables. You can use your own global variables, or you can use the ones QMF provides. For a list of the QMF-provided global variables, see Appendix E, “QMF Global Variable Tables” on page 257.

To set a global variable for a procedure, do one of the following:

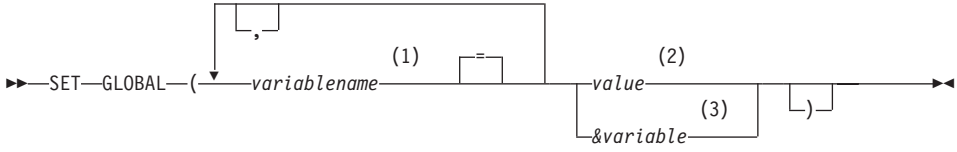
- Set the variable on the SHOW GLOBALS panel.

The variable name can be up to 18 characters long, and the length of the value can be up to 32 768 characters.

- Use the linear syntax of the SET GLOBAL command in your procedure, on the command line, or on the SET GLOBAL prompt panel.
- Use *extended syntax* for the callable interface languages other than REXX. For information about using the extended syntax of the SET GLOBAL command in the callable interface, see SET GLOBAL: Extended syntax.

SET GLOBAL: linear syntax

The name of the global variable can be up to 17 characters long, and the length of the value must be 55 characters or less. The linear syntax for the SET GLOBAL command is as follows:



Notes:

1 Identifies the global variable to which a value is assigned.

2 The character string that makes up the content of the global variable. A value that contains blank characters must be surrounded with delimiters. Valid delimiters for a global value are single quotes, parentheses, and double quotes. When the delimters are double quotes, the double quotes are included as part of the global variable.

3 A global variable name which contains the content of the global variable.

varname=value
Assigns a value to a variable name.

For example, to set a global variable called DEPT, issue the following command:

- In your linear procedure:
  ```
  SET GLOBAL (DEPT=38
  ```
- In your procedure with logic:
  ```
  "SET GLOBAL (DEPT=38"
  ```

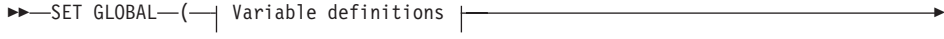
For more information about the SET GLOBAL command, see *QMF Reference* .

When you define a global variable, it remains defined until you reset the variable or end your QMF session. For information about using the RESET GLOBAL command, see *QMF Reference*.

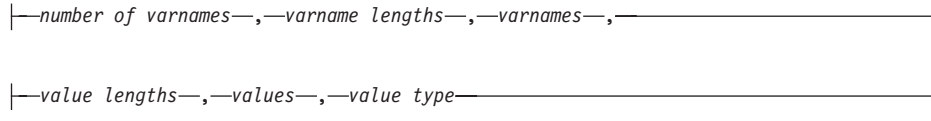
SET GLOBAL: Extended syntax

To change the value of any of these variables in an application written in a language other than REXX, (assembler, C, COBOL, FORTRAN, or PL/I), you must use the SET GLOBAL command with *extended syntax*. For examples of this command, see the sample program for the appropriate language in Appendix A, "Sample Code for Callable Interface Languages" on page 127.

The maximum length of a variable name used with a SET GLOBAL extended syntax command is 17 characters. The maximum length of a variable value is 32 768 characters.



Variable definitions:



number of varnames
The number of variables requested.

varname lengths
A list of lengths for each variable name specified.

The length of the variable name should be equal to the actual length of the global name in your storage area. An 18-character area padded with trailing blanks is allowed.

varnames
A list of names of the QMF variables.

value lengths
A list of lengths of the values of the variables.

The following rules apply to the variable value:

- If the value length you supply is less than the length of the value stored in your storage area, the value is truncated on the right when it is stored in QMF.
- If the value length you supply is greater than the length of the value stored in your storage area, the value might appear to have unrecognizable characters in it when it is stored in QMF.
- Integer lengths should always be 4 bytes.

QMF uses whatever value is in storage, starting at the address you assign for the length you assign. If the length is too long, QMF might abend.

values
A list of variable values.

value type
The data type of the storage area that contains the values. It must be either character or integer.

If you are using SET GLOBAL in the REXX callable interface, you can use only the *linear syntax* for the SET GLOBAL command, shown in "SET GLOBAL" on page 62. With this linear syntax, the maximum length for the global variable name is 17 characters, and the maximum length for the variable value is 55 characters.

Rules for using global variables

- On the SET GLOBAL command, variable names are not preceded with an ampersand as they are on the RUN and CONVERT commands.
- The QMF form does not recognize global variables set to form variable names or aggregation variable names.
- The QMF form does not recognize global variables with question marks in the names.

Rules for defining global variable names

- Global variable names are limited to 17 characters when entered on the command line and 18 characters when entered through the callable interface. You should use 17-character names, however, because of limitations of the SET GLOBAL command.
- A global variable name can contain a numeric character, but the first character of a global variable name cannot be numeric.
- Global variables cannot begin with DSQ because QMF reserves these letters for QMF predefined global variables.
- The first character of a global variable name must be an alphabetic character (A through Z) or one of these special characters:

  ¢ ! $ ~ { } ? @ # % \

- A global variable name cannot contain blanks or any of the following characters:

  * ( ) – + ¬ | : ; " ' < > / . , = &

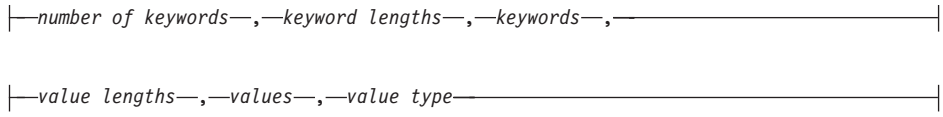
- Trailing blanks are not recognized in global variable names.

START

When you start QMF through the callable interface, you need to use the START command. Only one QMF session can be active at one time. If you want your application to test whether QMF has already been started, see "Starting QMF from an application" on page 24.

This section contains information on the START command syntax and keywords, including a table of keyword descriptions.

Keyword definitions:



Assembler, C, COBOL, FORTRAN, and PL/I use the following specifications for the START command:

number of keywords
The number of start command keywords you are using in your START command.

keyword lengths
The length of each start command keyword specified.

keywords
Names of the start command keywords.

There are three SAA start command keywords (DSQSCMD, DSQSMODE, and DSQSRUN). QMF provides other start command keywords in addition to these three. For more information on the start command keywords, see "START command keyword" on page 67.

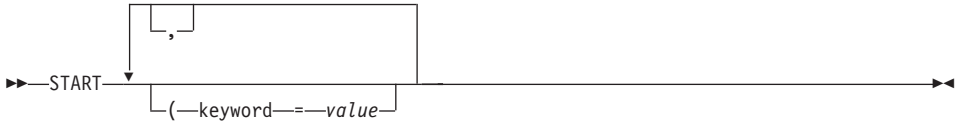
value lengths
A list that contains the lengths of the values for each start command keyword.

values
A list of values for the start command keywords specified in this command.

value type
The data type of the storage area that contains the value. The value type must be `character` for the START command.

START command syntax for the REXX Callable Interface

For the REXX callable interface, the START command has the following syntax:



START command keyword

Specify any of the following keywords on the START command:

DSQADPAN
DSQSIROW

DSQALANG
DSQSMODE [2]

DSQSBSTG
DSQSPILL

DSQSCMD [2] (CMS and TSO only)
DSQSPLAN (TSO only)

DSQSDBCS
DSQSPRID (TSO only)

DSQSDBNM
DSQSRSTG (CMS and TSO only)

DSQSDBQN (CICS only)
DSQSRUN [2]

DSQSDBQT (CICS only)
DSQSSPQN (CICS only)

DSQSDBUG
DSQSSUBS (TSO only)

DSQSDCSS (CMS only)
DSQSUSER (CICS/VSE only)

These keywords are described in Table 7 on page 68.

QMF allows you to specify START command keywords with the following conve

2. This keyword is an SAA command keyword.

- You can specify any start command keyword on the START command. In all environments supported in QMF, except CICS, you can also specify any keyword in the REXX program named by the DSQSCMD parameter *except* DSQSCMD. Because QMF CICS does not support REXX, you must specify all keywords on the START command.
- If you do not specify any keywords, QMF uses the values of the START command keywords as they appear in the program specified by the DSQSCMD keyword. If you do not use this program, QMF uses the default values of each keyword.
- If your application or the initial procedure specifies keywords that are not supported in a particular environment, those keywords are ignored. This way, you can compile a single program to run in multiple QMF environments without changing the environment-specific keywords.

For detailed information about these keywords and how they are affected by environmental dependencies, see *Installing and Managing QMF* for your platform. In Table 7, a superscript SAA ([2]) by the keyword name denotes an SAA start command keyword.

Table 7. START command keywords, descriptions, and default values

START command keywords	Description	Default value
DSQADPAN	Sets the DSQDC_DISPLAY_RPT global variable. This variable controls whether QMF displays the report when a query is run from within an application program. A value of 1 displays the report when a query is run. Set the value to 0 to specify that the report not be displayed.	In the callable interface: 1 In batch mode, or if QMF started interactively with DSQQMFE: 0

Table 7. START command keywords, descriptions, and default values (continued)

START command keywords	Description	Default value
DSQALANG	Determines the presiding language for the session you are starting. The value for this parameter is a one-character language identifier. Enter or specify QMF commands in the presiding language specified by this keyword. If you want to enter English commands when the presiding language is a language other than English, you can use the QMF bilingual support (see Chapter 7, "Writing Bilingual Applications" on page 45). The following table shows a complete list of language identifiers that are valid values for this variable: **Identifier** — **Language** **D** German **E** English **F** French **H** Hangeul (Korea) **I** Italian **K** Kanji (Japan) **P** Brazilian Portuguese **Q** Danish (not available in QMF VSE) **S** Spanish **U** Uppercase English **V** Swedish (not available in QMF VSE) **Y** Swiss French **Z** Swiss German	E, for English

Table 7. START command keywords, descriptions, and default values (continued)

START command keywords	Description	Default value
DSQSBSTG	Tells QMF how many bytes of storage to use for report generation. It lets you limit the amount of storage when there are multiple users in the same address space in CICS, and has similar uses in TSO and CMS if this variable is specified. In TSO and CMS, this keyword overrides the DSQSRSTG keyword if you specify both. If you do not specify this keyword, the value of the DSQSRSTG keyword is used. **Note to CICS users:** DSQSBSTG is always used for CICS; DSQSRSTG is never used for CICS. If you set the value of DSQSBSTG to below the minimum amount of storage required to produce a report, QMF automatically allocates the minimum amount of storage required; this minimum depends on your environment. A large report can require more than the minimum amount of storage.	In CICS: 500 000 bytes In CMS or TSO: zero bytes
DSQSCMD [2] (CMS and TSO only)	Specifies the REXX program that sets the QMF program parameters. When QMF receives the START command from a callable interface application, QMF calls the REXX program specified by this keyword. This REXX program provides values for QMF program parameters that QMF uses as defaults for those keywords not specified on the START command. `START (DSQSCMD=`*`yourprogram`* **Note to CICS users:** QMF CICS does not support REXX; therefore, DSQSCMD is not supported under CICS. If you start QMF using the callable interfaceunder CICS and you want to set QMF program parameters, you must specify the keywords on the START command.	DSQSCMDE
DSQSDBCS	Determines whether QMF allows double-byte characters when the terminal does not support DBCS. Values are YES and NO. You should set the value to YES when you intend to print double-byte character set (DBCS) data from a non-DBCS terminal or run a QMF batch job that prints DBCS data. Otherwise, the value should be NO.	NO

Table 7. START command keywords, descriptions, and default values (continued)

START command keywords	Description	Default value
DSQSDBNM	Specifies the location to connect to when starting a QMF session. A null value means that QMF connects to the default database (the database it normally connects to without remote unit of work). If your installation does not have a communication database set up and you attempt to specify a value other than null or the name of the default database, an error occurs.	NULL
DSQSDBQN (CICS only)	Specifies that CICS storage is used for QMF trace data. The name must conform to CICS name specifications for the type of CICS queue selected by DSQSDBQT. For more information about CICS name specifications, see the CICS application programming guide for your system.	DSQD
DSQSDBQT (CICS only)	Specifies the type of CICS storage to be used for QMF trace data. The values are: **TD** Use CICS transient data. **TS** Use CICS auxiliary temporary storage. Use caution when specifying temporary storage, because QMF can generate a large amount of trace data.	TD
DSQSDBUG	Specifies whether product tracing occurs during QMF initialization. The values are: **ALL** Specifies most detailed QMF tracing. **NONE** Specifies minimal QMF tracing.	NONE
DSQSDCSS (CMS only)	Specifies the name of the DCSS (discontiguous shared segment) that contains QMF execution modules.	QMF720*n*, where *n* is the national language identifier. For English, the default is QMF720E.
DSQSIROW	Indicates the number of rows QMF fetches before displaying the first screen of data for a RUN QUERY, IMPORT DATA, or DISPLAY command.	100

Table 7. START command keywords, descriptions, and default values (continued)

START command keywords	Description	Default value
DSQSMODE [2]	Tells QMF which mode you want to work in. **I** Specifies interactive mode. **B** Specifies batch mode. When the value of DSQSMODE is B, panel display is inhibited so that QMF can run in a background job.	B (batch)
DSQSPILL	Specifies whether QMF uses the spill file or data set. Possible values are YES or NO.	For CICS: NO For CMS and TSO: YES
DSQSPLAN (TSO only)	Specifies the DB2 application plan ID assigned to QMF.	QMF720
DSQSPRID (TSO only)	Specifies whether to use the TSO logon ID or the primary authorization ID to select the appropriate row from Q.PROFILES and to qualify Q.ERROR_LOG entries. Allowable values are PRIMEID or TSOID.	PRIMEID
DSQSRSTG (CMS and TSO only)	Determines the number of bytes of virtual storage you want to reserve for your application and other applications called by your application. Use this parameter if you plan to run applications from within the QMF environment. If you do not reserve storage for your application, QMF can use all the virtual storage available to produce a large report.	Zero (0)
DSQSRUN [2]	Specifies the name of the QMF initial procedure to run after QMF is started. The initial procedure runs only once with the callable interface. In this procedure, you can include commands to set global variables and profile variables to customize the user's session. This procedure can place the user in QMF if the application is going to run QMF interactively, or prepare the QMF session for a batch operation.	NULL
DSQSSPQN (CICS only)	Specifies the name of the CICS temporary storage queue that is used for QMF spill data. When the program parameter DSQSPILL has a value of YES, this spill area is used to contain report data.	DSQS*id*, where *id* is the CICS terminal ID
DSQSSUBS (TSO only)	Specifies the ID of the DB2 subsystem on which QMF is installed.	DSN

Table 7. START command keywords, descriptions, and default values (continued)

START command keywords	Description	Default value
DSQSUSER (CICS/VSE only)	Specifies the SQL/DS authorization ID and password on the CONNECT command. To specify the DSQSUSER keyword, enter the following: `DSQSUSER=SQL ID/password` where SQL ID is the SQL/DS authorization ID for the user who is starting QMF.	The 3-byte VSE operator ID and the password defined in the system catalog. **Must specify** if starting QMF with DSQSMODE=B.

Using command synonyms

QMF lets you create command synonyms, which are commands that resemble QMF commands. Command synonyms give you a lot of flexibility, and they are extremely useful for end users. For example, command synonyms perform the function of a command or start an application. To enable QMF users to access your command synonyms, you must enter your command synonyms into one or more command synonym tables. When a user issues a command synonym, QMF runs a TSO, RUN, CICS, or CMS command that starts the user's application.

Creating a command synonym

To create a command synonym:

1. Create a descriptive command.

   QMF commands follow the *verb object* format. Every command is a verb (action word) and many commands also have an object (descriptive noun) following them. For example, END is a verb-only command, while CONVERT QUERY is a verb-object command.

   You can create command synonymswith verbs that are identical to existing QMF commands. If you do, you can still use the original QMF command by preceding it with the command `QMF`. See *QMF Reference* for information on the command `QMF`.

   For example, suppose your procedure runs a report to see if the weekly sales figures have been entered. If the data for the current week is missing, the procedure calls the Table Editor to add the latest information to the table. Regardless of what you named the procedure, you want the command synonym to be descriptive for the user. You could choose a verb-object pair like `UPDATE SALES`.

   If your command synonym needs parameters or options, you can use the substitution variable `&ALL`.

2. Update the appropriate command synonym table with your new command synonym.

   You need to know the name of the command synonym table you want to use. The variable DSQAP_SYNONYM_TBL contains the name of the command synonym table for each user.

   Your database administrator has access to the command synonymtables. If you want to create a command synonym for your personal use, you might want to have a view defined that lets you add command synonyms.

   A command synonym table would contain the following information for the command synonym UPDATE SALES:

```
ADD               Q.COMMAND_SYNONYMS

                                                                1 to 4 of 4
 VERB. . . . . . . . . ( UPDATE             )
 OBJECT. . . . . . . . ( SALES                          )
 SYNONYM_DEFINITION. . ( RUN PROC WEEKLY_SALES                      >
 REMARKS . . . . . . . ( procedure that checks to see if the weekly sales fi>
```

   After you press the ADD function key, QMF adds this command synonym to your table. Before you can use the command, however, you must reconnect to QMF.

3. Update your profile, if necessary.

   If you added this command synonym to a new table or view, add the name of the new table or view to your profile.

4. End your QMF session.

   QMF does not recognize the changes you made to your command synonym and profile tables until you start a new QMF session.

SAA RUN QUERY report minisession

When you write applications that produce QMF reports, you can limit users' access to QMF by using *report minisessions*. In a report minisession, QMF limits the commands that a user can issue while viewing a report. Valid and invalid commands for a report minisession are listed in Table 8 on page 75 and Table 9 on page 76.

A report minisession behaves as a nested session (a session within a session). In minisessions, your initial QMF session remains intact, but becomes temporarily unavailable while you are viewing a report. The minisession becomes your current, active session until you issue the END command (or press the End function key). When you end a minisession, you either return to the initial QMF session or to the calling application, depending on how you write the application. The application cannot continue to issue subsequent commands until the report minisession is ended.

The QMF global variable DSQDC_DISPLAY_RPT, in effect, determines whether QMF starts a report minisession. This is because DSQDC_DISPLAY_RPT determines whether QMF displays a report after running a query (set to 1 to display the report, 0 to suppress display).

When you start QMF using the callable interface:

- The default value for global variable DSQDC_DISPLAY_RPT is 1. (When QMF is started with DSQQMFE, whether interactively or in Batch mode, the default value of the global variable is 0).
- If you run a procedure or an application that runs a query, QMF starts a report minisession; it is in this minisession that QMF displays the report resulting from the query.
- If your procedure or application does not run a query, or if you run a query from the SQL panel, QMF does not start a report minisession.

If you don't want QMF to start a report minisession, do one of the following:

- Change the value of DSQDC_DISPLAY_RPT to 0.
- Set the DSQADPAN parameter to 0 when you start QMF from the callable interface.

See "SET GLOBAL" on page 62 for more information about global variables.

From a report minisession, you can issue the following commands and synonyms for those commands (restrictions noted in parentheses):

Table 8. Valid commands in a minisession

• BACKWARD	• FORWARD	• RETRIEVE
• BOTTOM	• GET GLOBAL	• RIGHT
• CANCEL (when pop-up window active)	• HELP	• SAVE (Data)
• CICS	• INTERACT	• SET (Profile, Global)
• CMS	• ISPF	• SHOW (Report, Chart)
• DISPLAY (Report, Chart)	• LEFT	• SWITCH (when Help active)
• END	• MESSAGE	• TOP
• ENTER	• PRINT (Report, Chart)	• TSO
	• QMF	

Table 9 contains a list of commands that are *not* valid in the minisession:

Table 9. Invalid commands in a minisession

- ADD
- CANCEL
- CHANGE
- CHECK
- CLEAR
- CONNECT
- CONVERT
- DELETE
- DESCRIBE
- DISPLAY (Query, Proc, Profile, Form)
- DRAW
- EDIT
- ENLARGE
- ERASE
- EXIT
- EXPORT
- EXTRACT
- GETQMF
- IMPORT
- INSERT
- INTERACT
- IRM
- LIST
- NEXT
- PREVIOUS
- PRINT (Query, Proc, Profile, Form)
- REDUCE
- REFRESH
- RESET GLOBAL
- RESET (Query, Proc, Form)
- RUN
- SAVE
- SEARCH
- SHOW
- SORT
- SPECIFY
- START
- SWITCH

QMF returns an error message when you run an exec, a CLIST, or a procedure that issues a restricted command.

Chapter 9. Importing and Exporting QMF Objects

You can write applications that use QMF objects outside of the QMF environment. To place QMF objects outside of the QMF environment, you need to use the QMF EXPORT and IMPORT commands.

You can export the following objects:

chart data

form procedure (proc)

query report

table

When you export an object, QMF converts the object to an *externalized* format and places it in a file, data set, or CICS data queue. The externalized format of QMF objects is a powerful element of QMF application development. The IMPORT command reads the externalized format from a file, data set, or CICS data queue and places the object either in QMF temporary storage or in the database (depending on how you issue the command).

You can export data and table objects in either the QMF or IXF format. The format for form, Prompted Query, and report objects is a more complex format called the *encoded format.* Charts are exported in Graphics Data Format (GDF), a GDDM format.

This chapter describes all the QMF export formats and shows how you can use them in your applications. Appendix B, "Export/Import Formats" on page 227 describes the QMF format for data and defines the table and field numbers for encoded format objects. For information about IXF format, see Appendix C, "Integrated Exchange Format (IXF)" on page 241.

CICS users: If you write applications that use the IMPORT or EXPORT commands, read "Rules and considerations when using CICS queues" on page 121.

To see the syntax of the IMPORT and EXPORT commands, see *QMF Reference* , and for information about importing and exporting QMF objects, see online help.

What you can do with an exported file, data set, or CICS data queue

The import/export facility lets you:

- Provide query results to your application

  The purpose of many applications is to use the data produced by a QMF query. Use the QMF EXPORT command to get data out of the database and into your application.

- Create objects within your application and use them in QMF

  You can create an object outside of the QMF environment using the appropriate format for the object. When you import the file, data set, or CICS data queue containing the object into QMF, a new QMF object is created.

  You cannot import reports and charts into QMF.

- Store non-QMF objects in the database

  When you import an object as a procedure or query object, QMF brings it into the QMF environment as is; it does not insert additional records or fields into the imported file. You can import any program or file that has a record length of 79 bytes or less.

- Make QMF objects available to other environments or products.

  CAUTION:
  Exported objects transfered between systems or environments are exposed to translation risks that can alter or destroy the exported object. IBM does not recommend transfering exported objects between environments running with different CCSIDs or character sets, such as between EBCDIC and ASCII systems, or between different NLF environments.

  You can use the CONVERT QUERY command to convert a prompted query or QBE query to an SQL query that you can export and use in other products. For more information on the CONVERT command, see *QMF Reference*.

  You can transfer QMF objects:

  – Between CMS sessions in VM
  – Between QMF under TSO, or native OS/390 batch and QMF under CICS using CICS extrapartition transient data queues
  – Over networks with SENDFILE

- Save objects and data outside of the database

  For example, in the middle of a program, you can export your data so that an external program can manipulate it.

- Create bilingual applications

  You can create a QMF form in your presiding language, and translate it to English using the `LANGUAGE=` option on the EXPORT command. You can also

use the LANGUAGE= option on the IMPORT and EXPORT commands to translate an English form to your presiding language.

Exporting versus saving data

The difference between EXPORT DATA and SAVE DATA is in where and how the object is stored, which affects what you can do with the results:

- Exporting a data object produces a file, data set, or CICS data queue. You can read, modify, or print each sequentially, but you do all these operations through QMF application programs or other external applications.
- The SAVE DATA command produces a database table. Whatever actions you perform using saved data must be done through the database.

Data and table objects

When you run a query, QMF displays the result in a report that you can export as either a data object or report object. When you export the report object, the object maintains the format of the data specified in its form object. When you export the report object as an HTML report, it is packaged with appropriate HTML 3.0 coding. You can put the report on a web server for display on the World Wide Web. QMF data and table objects are exported in the form of *raw data.* See "Report objects" on page 111 for details on the report object.

The raw data for the tabular display is stored in the temporary storage area as a *data object*. Relational tables and views stored in the database are referred to as *table objects*. The exported formats of a table in temporary storage (DATA) and a table stored in the database (TABLE) are identical. An object exported as data can be imported as a table, and vice versa.

Data and table objects can be exported in QMF format or Integrated Exchange Format (IXF).

You can specify either DATAFORMAT=QMF or DATAFORMAT=IXF on your EXPORT command to tell QMF the export format you want. The QMF format is the default. The QMF format is described in "QMF format for data" on page 227.

IXF has two formats: binary and character, which are described in "Binary versus character" on page 83. The IXF format is described in Appendix C, "Integrated Exchange Format (IXF)" on page 241.

You can create your own tables in a &file by specifying the QMF or IXF format and importing the file, data set, or CICS data queuethat contains the data you need. Include the required fields and add your own data as

appropriate. Then import this file, data set, or CICS data queue into QMF as a table object. Here is an example of a command to import a file, data set, or CICS data queue into the database as a table object:

```
IMPORT TABLE MYTABLE FROM MYDATA
```

For more information about the EXPORT and IMPORT commands, see *QMF Reference* . CICS/VSE users should read "Rules and considerations when using CICS queues" on page 121.

Interpreting a data object in QMF format: an example

You can calculate the length of the header record when you have the length of the data records. In this example, each data record is 23 bytes long. "QMF format for data" on page 227 explains that the first 12 bytes contain level and number information. There are 24 bytes for each column of data, and there are three columns. Thus, for this three-column data object, the header is 84 bytes:

(12 + (24 X 3) = 84).

If you export the following data from Q.STAFF:

```
ID  NAME     COMM
___ _______ _______
 10 SANDERS   -
 20 PERNAL   612.45
```

You would use the following table to calculate the widths of each column:

Table 10. Calculating column widths

Column name	Data type	Data type width (length in header)	Width of column
ID	SMALLINT	2	2 + 2 = 4
NAME	VARCHAR	9	2 + 2 + 9 = 13
COMM	DECIMAL (7,2)	7	(7 + 1)/2 + 2 = 6
		Length of data record:	23

Each header record is the same length as the data records: 23 bytes. Those 84 bytes are spread across four 23-byte header records; the last record is padded with blanks.

Figure 16 on page 81 shows the header from the report and its hexadecimal representation. The reversed-type numbers indicate notes following the figure.

```
      R  E  L      1  .  0              I  D
1     D9 C5 D3 40 F1 4B F0 40 0004 0003 C9 C4 40 40 40 40 40 40 40 40
       1                          2    3    4
                                    N      N  A M  E
2     40 40 40 40 40 40 40 01F4 0002  D5 00 D5 C1 D4 C5 40 40 40 40 40 40
                               5     6    7
                                     Y      C  O  M  M
3     40 40 40 40 40 40 40 40 01C0 0009 E8 00 C3 D6 D4 D4 40 40 40 40 40
                                          Y
4     40 40 40 40 40 40 40 40 40 01E4 07 02 E8 00 40 40 40 40 40 40 40
```

Figure 16. Sample header records for exported data object in QMF format. 40 is the hexadecimal code for a blank character.

Figure 17 shows the data from the report and the hexadecimal representation of that data. For information about what the byte positions mean, see "QMF format for data" on page 227.

```
             10             S  A  N  D  E  R  S
1    00 00 00 0A 00 00 00 07 E2 C1 D5 C4 C5 D9 E2  00 00 FF FF 00 00 00 40 40
             8              9                                    10
             20               P  E  R  N  A  L
2    00 00 00 14 00 00 00 06 D7 C5 D9 D5 C1 D3 00 00 00 00 00 00 61 24 5C
```

Figure 17. Sample data records for exported data object in QMF format

1 REL 1.0
Object format level: 1.0

The object format level tells QMF which version of the object format this object is using. Every time a QMF object format is changed, the level number is changed; object formats are *not* changed with every new release.

2 X'0004'
Number of header records: 4

3 X'0003'
Number of data columns: 3

4 X'C9 C4'
Column name: ID

5 X'1F4'
Data type: SMALLINT

6 X'0002'
Column width: 2

7 X'D5'
Nulls allowed: N signifies no

8 **X'0A'**
Value for first column of first data record: 10

9 **X'07'**
Length of name in second column of first data record: 7

10 **X'FFFF'**
Indicator information: column contains a null value

If you want more information about the files, data sets, or CICS data queues that are generated when data or table objects are exported, see "Specifications for externalized QMF objects" on page 120.

For an example of the IXF format, see Appendix C, "Integrated Exchange Format (IXF)" on page 241.

Rules and information for export/import of data and table objects

Here are some general considerations for importing and exporting data or table objects.

The file, data set, or CICS data queue stays allocated

The QMF IMPORT DATA command appears to store the data in the QMF temporary storage area and display the report on the screen. Actually, only a portion of the data is stored and displayed. The file, data set, or CICS data queue remains open and allocated to QMF. QMF reads records when the user scrolls through the file, data set, or CICS data queue.

This connection is maintained until the data object is replaced or reset, or QMF has read all the records. At this point, the file, data set, or CICS data queue is closed and is no longer considered allocated to QMF. This means that an application should not attempt to delete or alter a file, data set, or CICS data queue allocated to QMF with an IMPORT DATA command. The application needs to either start using another data source or empty the QMF temporary data storage area (RESET DATA) before it tries to alter or delete a data set it has been reading.

During the execution of the IMPORT command, QMF does not *lock* the file, data set, or CICS data queue while it is being read. It does not take steps to prevent the file, data set, or CICS data queuefrom being altered while it is being read. If the file, data set, or CICS data queue is erased or altered in any way before QMF has finished reading it, the results are unpredictable and can cause a system error.

An incomplete data prompt can occur when QMF needs to complete the object and there isn't enough storage for the data object. QMF needs to complete the data object if, for example, you requested the export of an object to the same file, data set, or CICS data queue. This situation implies that you

previously performed an IMPORT DATA command from the same file, data set, or CICS data queue now named on the EXPORT command. For more information about the incomplete data prompt and actions to take, see either:

Binary versus character

When you export a data or table object using the QMF format or the binary form of the IXF format (OUTPUTMODE=BINARY), the data is in a *raw* binary form. However, when you use the character form of IXF (OUTPUTMODE=CHARACTER), the exported data is in EBCDIC form. Exported data for form, report, procedure, and SQL query objects is also in EBCDIC form.

Application programs written in languages such as PL/I, COBOL, and assembler can usually read and process binary data faster and more efficiently than character data. Interchanging data from one IBM product to another is more efficiently done in binary. However, if your application programs are written in REXX, or if you're processing the data with an editor, you'll find EBCDIC (character) data to be more efficient.

Errors

After QMF imports data from a file, data set, or CICS data queue, QMF displays the report panel and a confirmation message. If the file, data set, or CICS data queue contains format errors, QMF does not display the report panel; instead, QMF displays an error message on the object panel that was current before QMF processed the IMPORT command. However, if the current object panel was the Report panel, and QMF finds errors in the imported data, QMF displays the Home panel and an error message.

Unlike the form object, when a data or table object is imported, the format of the input file, data set, or CICS data queue must be precisely the same as the format of the output file, data set, or CICS data queue that would be generated if the same object were exported using the EXPORT DATA or EXPORT TABLE command.

Procedures and SQL queries

The format of the file, data set, or CICS data queue representing these objects is the simplest of all the objects. Each record in the file, data set, or CICS data queue is essentially an image of a line as it appears on the screen (fixed length record of 79 bytes).

This is an SQL query:

```
SQL query                                        MODIFIED  LINE    1

SELECT *
FROM Q.STAFF
```

This is the query in its externalized format:

```
* * * Top of File * * *
SELECT *
FROM Q.STAFF



* * * End of File * * *
```

Because of the simplicity of the record format, creating or editing an SQL query or procedure outside of QMF is very straightforward. An SQL query or procedure consists of a fixed-length file, data set, or CICS data queue containing 79-byte query or procedure records. Import the resulting file, data set, or CICS data queue, and your query or procedure is now in the QMF temporary storage area ready to be run.

Chart objects

You can export a chart object for processing outside of the QMF environment. A chart cannot be saved as a QMF object in the database or retrieved from the database. You cannot import charts into QMF.

When QMF exports a chart object, it converts the data from the report to a Graphics Data Format (GDF). GDF, a GDDM format, is an existing standard for data interchange. You can print the exported chart data using GDDM utilities, or include it in documents—script files, for example. Refer to a GDDM application programming guide for details about the GDF format.

You can use an exported chart object just as you would any GDF formatted file or data set. For example:

- Using the Document Composition Facility (DCF), an application can combine a QMF report (using a printed or exported report) with a QMF chart (using an exported chart) and send the formatted information to a printer.
- Using a graphics editor such as the GGXA graphics editor, an application can make further modifications and refinements to an exported QMF chart.

Encoded objects

The form and prompted query objects are exported and imported in an encoded format, which is a format that translates an object to a tabular structure. The encoded format helps you manipulate individual parts of an object more easily. The report objects are also exported in an encoded format; however, reports cannot be imported.

The encoded format of a form, report, or prompted query (relational or entity-relationship) consists of the following records:

- Fixed format records: Header records (H) (see page85)
- Variable format records
  - Data value records (V) (see page89)
  - Data table description records (T) (see page91
  - Table row records (R) (see page95)
  - End of object record (E) (see page97)

An application data record, denoted by an asterisk (*), can be used by application programs to store information and comments associated with the object in the exported file. See "Application data record (*)" on page 97 for details.

In addition to the preceding records, an exported report can contain the following records:

- Report line records (L) (see page114)
- Data continuation records (C) (see page116)

For specifications about the exported files, data sets, or CICS data queues, see "Specifications for externalized QMF objects" on page 120.

Fixed format records

Most records have a variable format. However, header records have a fixed format, even though the file or data set containing the records can be of variable format.

Header records (H)

These records are used to identify the contents of the exported form, report, or prompted query. A header record is the first record of the exported file. It describes the characteristics of the object.

A header record contains the information described in Table 11 (an asterisk indicates that the field is required for import):

Table 11. Header record information

Byte Position	Information and Type
01*	Header record indicator (H)
02	Blank
03-05*	Product identifier (QMF)
06	Blank

Table 11. Header record information (continued)

Byte Position	Information and Type
07-08	QMF release level in which the form, report, or prompted query was exported: 11 for QMF Version 7
09	Blank
10*	Type of object: F for form R for report T for relational prompted query E for ER prompted query
11	Blank
12-13*	QMF object level 01 for report 04 for form 01 for prompted query (relational or ER)
14	Blank
15*	Format of object E for format used to export form, report and prompted query (relational or ER) objects
16	Blank
17	Status of the object: E - Contains errors (for form only) W - Contains warning V - Valid
18	Blank
19	Whole or partial object indicator W for whole object
20	Blank
21	National language in use when object was exported: E for English
22	Blank
23*	Action against object in the temporary storage area when importing (R for replace object)
24	Blank
25-26	Length of control area in the beginning of each following record: 01 for form 02 for report 01 for prompted query (relational or ER)
27	Blank
28-29	Length of integer length fields specified in V and T records (03)
30	Blank
31-38	Date stamp: yy/mm/dd
39	Blank
40-44	Time stamp: hh:mm
45	Blank
46-50	SSSSS, for OS/2 objects
51	Blank
52-56	DDDDD, for OS/2 objects

The object level in the H record denotes a change in an object's format. All object formats begin with object level 01; if a later release of QMF changes an object format, the object level is increased by 1. The object level increases only when the change in the format could potentially create an error in your application. Check for level changes in the object types of reports, forms, and prompted queries, which have the encoded format in Figure 17 on page 81.

For example, the externalized format for form objects handles break information differently in Version 3.2 than in previous releases. Because of this change, the object level for form objects increased from 03 to 04 for Version 3.2. In general, the following changes cause the object level to be incremented:

- Field numbers in V or R records are removed or replaced.
- The layout of a particular record type is redefined.

However, new values for a field or new field numbers do *not* create errors in your application. Check the object level value to ensure that objects you import do not create problems for your application.

Example of an H record for a prompted query:

```
H QMF 11 T 01 E V W E R 01 03 98/11/20 17:12
```

Value from example	Description
H QMF 11 T	A Version 7 QMF relational Prompted Query header record
01	Structure of the prompted query is at object level 1
E	Format type is that for forms, reports, and prompted queries
V	The exported prompted query does not contain any errors or warnings
W	The file contains the entire prompted query
E	The national language in use when object was exported is English
R	When importing, object in temporary storage area is replaced
01	Length of control area is 1 byte
03	Length of integer length fields is 3 bytes
98/11/20	Date stamp
17:12	Time stamp

See Figure 19 on page 101 for a complete example of the Prompted Query encoded format.

Example of an H record for a form:

```
H QMF 11 F 04 E V W E R 01 03 98/12/16 22:08
```

Value from example	Description
H QMF 11 F	A Version 7.2 QMF form header record
04	Structure of the form is at object level 4
E	Format type is that for forms, reports, and prompted queries

Value from example	Description
V	The exported form does not contain any errors or warnings
W	The file contains the entire form
E	The national language in use when object was exported is English
R	When importing, object in temporary storage is replaced
01	Length of control area is 1 byte
03	Length of integer length fields is 3 bytes
98/12/16	Date stamp
22:08	Time stamp

See Figure 20 on page 104 for a complete example of the form encoded format.

Example of an H record for a report:

```
H QMF 11 R 01 E V W E R 02 03 98/10/14 16:20
```

Value from example	Description
H QMF 11 R	A Version 7.2 QMF report header record
01	Structure of the report is at object level 1
E	Format type is that for forms, reports, and prompted queries
V	The exported report does not contain any errors or warnings
W	The file contains the entire report
E	The national language in use when object was exported is English
R	Ignored
02	Length of control area is 2 bytes
03	Length of integer length fields is 3 bytes
98/10/14	Date stamp
16:20	Time stamp

See Figure 22 on page 113 for a complete example of the report encoded format.

Variable format records

With the exception of H records, which are fixed format records, all records are variable format records:

Indicator

Record type

V Data value (see "Data value records (V)")

T Data table description (see "Data table description records (T)" on page 91)

R Table row (see "Table row records (R)" on page 95)

E End of object (see "End-of-object record (E)" on page 97)

***** Application data (see "Application data record (*)" on page 97)

L Report line (see "Report line records (L)" on page 114)

C Data continuation (see "Data continuation records (C)" on page 116)

Variable format records are accepted on input. This refers to the records themselves, not the files, data sets, or CICS data queuesthat contain the records. Variable format records have the following general form:

The control area is:

Byte position
Description

01 Record identifier (H,V,T,R,E,*,L,C)

02 Blank (sometimes omitted; see specific type of variable format record)

The record data area is a variable length area containing information about that specific record. Fields in this area are separated by a delimiter (a blank character is used in this book).

Data value records (V)

Value records are used to provide a value for a single field in an object, such as blank lines before the heading in the form. V records contain:

- A field number unique to the object
- The field's length
- The field's value

Appendix B, "Export/Import Formats" on page 227 lists the assignments of field numbers to the fields contained in the prompted query, form, and report objects.

The contents of a V record are:

Control area for V records:

Byte Position	Description
01	Value record identifier (V)
02	Blank (used only for reports, omitted for forms and prompted query)

Record data area for V records

Byte position	Description
01	Blank
02-05	Field number (1001-9999)
06	Blank
07-09	Length of the data value (000-999). Can also be an asterisk (*) followed by two blanks. An asterisk indicates that the data value is delimited by the end of the record.
10	Blank
11-end	Data

Notes:

1. Record data area byte positions are *offset* from the end of the control area, the length of which is indicated in the header record.
2. An omitted data value (an end-of-record or only blanks following the length field) indicates that the field contains a null value.
3. If the length field is zero, the default value for the field is applied and a warning message is issued.
4. If the specified length is different from the actual data that follows, QMF issues a warning.

Examples of V records

Form: `V 1511 *  NONE`

(See page 232 for a complete list of field numbers.)

Field Width of wrapped report lines

Value 'NONE'

Report: `V  1001 006 PERIOD`

(See page 238 for a complete list of field numbers.)

Field Profile DECIMAL option

Length
6

Value 'PERIOD'

Prompted query: `V 1501 001 K`

(See page 230 for a complete list of field numbers.)

Field Duplicate rows

Length
1

Value keep

Data table description records (T)

In the encoded format, most data in an object appears in *tables.* These are not relational tables in the database, but rather a means of grouping information *within the encoded format.*

Each T record defines one table, and each table corresponds to a particular part of an object, such as summary calculations in the form. Thus, one exported file can contain many of these encoded tables. See Appendix B, "Export/Import Formats" on page 227 for information about field numbers for encoded tables and their columns.

A T record is always followed by R records. The T record describes the R records that follow it. If there are no R records following a T record, the table is omitted.

Be sure your application program refers to the contents of tables of an exported form, report, or prompted query by using the encodings in the T record to correctly locate the values in the R records. Your application program should not use fixed offsets to locate information in R records.

The contents of a T record are as follows:

Control area for T records:

Byte position
Description

01 Table record identifier (T)

02 Blank (used only for reports, omitted for forms and prompted queries)

Record data area for T records

The byte positions in the following list are *offsets* following the end of the control area, the length of which is indicated in the header record.

Byte position
Description

01 Blank

02-05 Table number (1001-9999)

06 Blank

07-09 The number of rows (R records) in this table. An asterisk (*) used instead of a numeric value means that the table consists of all the R records that follow.

10 Blank

11-13 The number of columns in the record (000-999)

14 Blank

15-18, 24-27, ...
The field number for this column (repeating field)

19, 28, ...
Blank (repeating field)

20-22, 29-31, ...
The length of the data values in this column (repeating field)

Bytes 11-13 (number of columns) indicate how many field number/data value length pairs follow; this means that the information in bytes 15 through 22 is repeated for each column.

Examples of T records (Form)

```
T 1110 *   002 1112 007 1113 018
```

(See page 232 for a complete list of field numbers.)

Field Column heading table

Rows All

Columns
2

Column field
Column data type

Length
7

Column field
Column heading

Length
18

Examples of T records (Prompted Query)

```
T 1110 008 002 1112 001 1113 027
```

(See page 230 for a complete list of field numbers.)

Field Table definitions table

Rows 8

Columns
2

Column field
Table ID

Length
1

Column field
Table name

Length
27

Examples of T records (Report)

```
T  1010 005 003 1012 008 1013 003 1014 006
```

(See page 238 for a complete list of field numbers.)

Field Formatted report table

Rows 5

Columns
3

Column field
BREAKn

Length
8

Column field
Edit code for data

Length
3

Column field
Starting position for field contain data

Length
6

Rules and notes:

1. When a form or prompted query is imported, the number of R records must match the row count specified in bytes 07-09 of the record data area of the T record. Otherwise, QMF issues a warning.
2. When a form or prompted query is imported, the number of columns indicated in bytes 11-13 must agree with the field number/length pairs in the bytes that follow. If not, QMF issues a warning.
3. The number of field number/length pairs is limited to the number of columns in the table, and their order is arbitrary.
4. Columns with a length of zero (or not included in this table) are set to their default values when the object in the temporary storage area is updated and a warning is issued. This is not always true for Prompted Query. Where possible, a default is supplied; otherwise an error occurs.
5. To set a column field to blank, the column must have a positive length in the T record and a blank value in the R record.

Table row records (R)

R records provide a set of values for a single row in an encoded table. R records contain a list of values arranged in an order described by the associated T record. An R record matches the description of the positions and lengths of the data values specified in the T record. The contents of an R record are as follows:

Control area for R records:

Byte position	Description
01	Row record identifier (R)
02	Blank (used only for reports, omitted for forms and prompted queries)

Record data area for R records

Following the control area, the data area for R records consists of a series of values separated by a delimiter (blank character). The format is as follows:

```
_val.._val..._val..
```

where `val...` is the data value for this row and column and _ is the delimiter.

Examples of R records

In these examples, the length of the column value is always given in the T record for that column.

Form: `R 2 SALARY`

(See page 232 for a complete list of field numbers.)

Column Value
' 2'

Column Value
'SALARY'

Report: `R GROUP L2`

(See page 238 for a complete list of field numbers.)

Column Value
'GROUP '

Column Value
'L2'

Prompted query: `R C A.DEPT`

(See page 230 for a complete list of field numbers.)

Column Value
'C'

Column Value
'A.DEPT'

Rules and notes:

1. An R record must immediately follow another R or a T record.
2. The number of data values (v..v) must match the description in the associated T record.
3. A data value length of zero in the associated T record indicates that *no* value is to be applied to this row and column of the object; that is, it is set to its default value. However, the presence of the field in the T record requires that the R record contain an extra blank for this field (a zero-length value results in one blank followed by another in the R record).

End-of-object record (E)

The E record specifies the end of an exported object. It is the last record of an exported file, appearing as the character E. For an exported report, an E record is followed by a blank character to complete its control area. For a form, the blank is omitted.

Any records following the E record are ignored. If an E record is not included with the file being imported, QMF assumes that end-of-file implies the end of the object.

Application data record (*)

Application data records allow application programs to include their own data, such as comments, associated with a given object in the external file. Application programs frequently use these records as comment records to further describe the object in the file. The information following the asterisk is essentially ignored and has no effect on the input process.

Application data records can appear anywhere in the external file except before the header (H) record. QMF does not write out application data (*) records on export. However, you can use these records in the file, data set, or CICS data queue you create. They are useful as comment records. The contents of an application data record are as follows:

Byte position
Description

01 Application data record identifier (*)

02-end of record
Data

Example of an application data record

```
*This is the form that groups by DEPT.

This comment would be in an exported form.
```

Exporting encoded format objects

When you export an object with the encoded format:

- All table and field numbers are written out as four-digit numbers.
- The table columns are written out in the order in which they normally appear in the object, except that the column with the maximum length is moved to the right end of the table record and associated row records.
- Numeric lengths are three digits long, including leading zeroes, if necessary.
- The blank character is used as a delimiter in all records.
- The delimiter is not written following the last character of each record.
- Blanks are written in all reserved fields.
- An E record is the last record written to the output file.

Importing encoded format objects

When you import a form, report, or prompted query:

- The file can consist of variable or fixed-length records. See "Specifications for externalized QMF objects" on page 120 and Appendix B, "Export/Import Formats" on page 227.
- The record identifier (H, V, T, R, E, *, L, or C) must be in the first position of every record.
- The first two bytes are reserved for control information (the control area).
- Every data field (including field numbers, lengths, and values) must be preceded and followed by one delimiter. Exception: The last data field in a record need not be followed by a delimiter because the end-of-record acts like a delimiter. (The examples in this book use the blank character as the delimiter.)
- If QMF encounters a duplicate data value or table during IMPORT, it replaces the previous value or table. However, duplicates are not allowed where they would violate the rules for a particular object. For example, the number of columns provided for a form can't be changed after the first COLUMNS table has been processed.
- Table numbers, field numbers, and numeric lengths, can contain leading zeroes or leading blanks. However, trailing blanks (except for the blank delimiter) are not allowed; fields must be right-justified.
- When * is used instead of a length or count, it must be left-justified and padded with trailing blanks.

- If the value supplied for a data entry field is shorter than the field, it is padded with trailing blanks. If it's longer, it is truncated.
- If the record is shorter than its fixed-format length, those fields left unspecified are assumed to be blank.

Prompted query objects

This section refers to the external format of relational prompted queries.

An exported prompted query object contains the information displayed in the echo area of the Prompted Query primary panel. Exported Prompted Query files, data sets, or CICS data queues can either be imported into the QMF temporary storage area or directly into the database. When you import a prompted query, QMF checks to see whether the incoming query is consistent with the data in the database. For example, if the prompted query being imported has columns A, B, and C in table XYZ, QMF verifies that table XYZ with columns A, B, and C exists in the database.

For a list of table and field numbers associated with prompted query objects, see "Table and field numbers for the prompted query object" on page 230.

Exporting a prompted query object

This section illustrates an example of an exported prompted query. Figure 18 on page 100 shows the Prompted Query base panel echo text of a prompted query to be exported.

```
Tables:
  Q.STAFF(A)
  Q.ORG(B)
  Q.STAFF(C)

Join Tables:
  A.DEPT And B.DEPTNUMB
  And A.ID And C.ID

Columns:
  A.ID
  A.DEPT
  A.JOB
  A.SALARY
  DEPTNUMB
  C.SALARY
  C.SALARY+A.COMM

Row Conditions:
  If A.SALARY Is Greater Than 10000
  And A.DEPT Is Equal To 84 or 96

Sort:
  Descending by C.SALARY+A.COMM

Duplicate Rows:
  Keep duplicate rows
```

Figure 18. Sample prompted query to be exported

Figure 19 on page 101 shows the format of the exported prompted query.

```
H QMF 11 T 01 E V W E R 01 03 98/11/20 17:12
T 1110 003 002 1112 001 1113 050
R A Q.STAFF
R B Q.ORG
R C Q.STAFF
T 1150 002 002 1152 020 1153 020
R A.DEPT               B.DEPTNUMB
R A.ID                 C.ID
T 1210 007 002 1212 001 1213 255
R C A.ID
R C A.DEPT
R C A.JOB
R C A.SALARY
R C B.DEPTNUMB
R C C.SALARY
R C C.SALARY+A.COMM
T 1310 009 003 1312 001 1313 008 1314 255
R 1 C        A.SALARY
R 2 IS       GT
R 3          10000
R 4 I
R 1 C        A.DEPT
R 2 IS       EQ
R 3          84
R 3          96
R 4 A
T 1410 001 002 1412 001 1413 255
R D C.SALARY+A.COMM
V 1501 001 K
E
```

Figure 19. The exported file, data set, or CICS data queue

Importing a prompted query

If you want to import a prompted query object that your application edited or generated, be aware of the following:

- When a prompted query file is imported, the incoming records must be in a specific order following the header (H) record. The order should be:
  1. T records for table definitions
  2. R records for table names
  3. T records for column definitions
  4. R records for columns
  5. Row condition records (field number 1310) must be in order within each condition according to the entry type sequence number (field number 1312), that is, the same order that row data appears in the Prompted Query echo area.
  6. The remaining records can be in any order.

- The Tables table must appear before any other tables or V records.
- The value of row count in the Tables T record must be * or an integer from 0 through 15. A zero value in the row count causes everything in the query to be ignored, which means that an empty query is imported.
- QMF does not issue warnings for Prompted Query imports.
- If a second Tables table (table 1110) is specified, QMF issues an error, and the contents of the table are ignored.
- Prompted Query does not supply default values on import.
- If there is a Sort table, there must be a Columns table preceding it.
- QMF accepts duplicate records in the import file. The most recent value for the record is used.
- All column names must be qualified by the table identifier during import.
- When a prompted query is exported to a preallocated data set, the minimum logical record length (LRECL) allowed is 259 bytes.
- The exported format of a prompted query is the same regardless of the national language used; the format is language independent. The language byte in the header record is ignored during import. You can see the codes used in exporting a prompted query in "Table and field numbers for the prompted query object" on page 230.

  Summary functions and expressions are not translated; thus summary functions COUNT, AVG, SUM, MIN, and MAX remain unchanged. They are SQL symbols, which are not translated.

Form objects

The form object contains all the information specified in all the QMF form panels. When you export a form, QMF converts the form panels you changed to the encoded format. The following panels are in the encoded format *only* if you modified the panel:

- FORM.BREAKn, where n = 3 to 6
- FORM.CALC
- FORM.CONDITIONS
- All variation panels greater than 1 for FORM.DETAIL

Eliminating unused panels from the externalized format helps you save space on your system.

Creating a default form: an example

The LAYOUT command allows the user to view a sample report based on a form (in QMF temporary storage or in the database) without requiring any data to be in QMF temporary storage. LAYOUT generates sample data, imports it into QMF, and applies the form to it to create a report.

Note to CICS users
The LAYOUT command requires ISPF, which is not available in CICS.

Users can look at the form without running a query by creating a form, exporting it, and then importing it into QMF as part of initialization. If you import the form during the initial procedure, users can access the form by entering `SHOW FORM.`

The minimal form you can import is a header and end record. However, to use FORM.COLUMNS, you need to have at least one column of information.

You can create a default form by running a query that creates an empty report:

```
SQL query

SELECT JOB
FROM Q.STAFF
WHERE NAME='empty_set'
```

When QMF displays the report, enter `EXPORT FORM TO DEFAULT` (including the (`QUEUETYPE=`*xx* parameter in CICS). Your file, data set, or CICS data queue named DEFAULT contains the information shown in Figure 20 on page 104.

```
H QMF 11 F 04 E V W E R 01 03 98/12/16 22:08

T 1110 001 011 1112 007 1113 040 1114 007 1115 006 1116 005 1117 005 1118 003 1119 008 1120 008
  1122 006 1121 050
R CHAR    JOB                                              2      5     C     1   DEFAULT
  DEFAULT  NO
V 1201 001 0
V 1202 001 2
T 1210 001 003 1212 004 1213 006 1214 055
R 1    CENTER
V 1301 001 2
V 1302 001 0
T 1310 001 003 1312 004 1313 006 1314 055
R 1    CENTER
V 1401 002 NO
V 1402 001 1
V 1403 001 0
T 1410 001 003 1412 004 1413 006 1414 055
R 1    RIGHT
V 1501 001 1
V 1502 003 YES
V 1503 003 YES
V 1504 003 YES
V 1505 003 YES
V 1506 003 YES
V 1507 003 YES
V 1508 003 YES
V 1509 003 YES
V 1510 003 YES
V 1511 004 NONE
V 1512 002 NO
V 1513 007 DEFAULT
V 1514 002 NO
V 1515 004 NONE
V 2790 001 1
V 2791 003 YES
V 2805 003 YES
T 2810 001 003 2812 004 2813 006 2814 055
R 1    LEFT
V 2901 002 NO
V 2902 001 1
V 2904 001 0
V 2906 002 NO
V 2907 002 NO
T 2910 001 003 2912 004 2913 006 2914 055
R 1    LEFT
V 3080 001 1
V 3101 002 NO
V 3102 002 NO
V 3103 001 0
V 3104 001 0
T 3110 001 003 3112 004 3113 006 3114 055
```

Figure 20. Sample format of an exported form (Part 1 of 2)

You can import your default file, data set, or CICS data queueevery time a

```
R 1    LEFT
V 3201 002 NO
V 3202 001 1
V 3203 001 0
V 3204 001 1T 3210 001 003 3212 004 3213 006 3214 055
R 1    RIGHT
V 3080 001 2
V 3101 002 NO
V 3102 002 NO
V 3103 001 0
V 3104 001 0
T 3110 001 003 3112 004 3113 006 3114 055
R 1    LEFT
V 3201 002 NO
V 3202 001 1
V 3203 001 0
V 3204 001 1
T 3210 001 003 3212 004 3213 006 3214 055
R 1    RIGHT
E
```

Figure 20. Sample format of an exported form (Part 2 of 2)

user logs on by issuing the command `IMPORT FORM FROM DEFAULT` (including the `(QUEUETYPE=`*xx* parameter in CICS) in your initial procedure

Considerations for QMF form objects in applications

When using a QMF form in an application, you need to keep a few things in mind:

- **Creating a form file, data set, or CICS data queue outside of QMF**

  If you create a form &file outside of QMF (that is, you *don't* create it using EXPORT FORM), it is not necessary to have a complete form object to import it successfully into QMF. All you really need is the header (H) record followed by the T and R records of the COLUMNS table. Default values are applied for the rest of the form when it's imported.

  You have more flexibility when you create your own form file, data set, or CICS data queue—it doesn't have to be exactly like the file, data set, or CICS data queue you get if you use EXPORT FORM. For example, when QMF exports a form, all data values in a value (V) record are preceded by a length, whereas you can use an asterisk (*) signifying that the data value is delimited by the end of the record when you import a form.

  If an R record count in an imported form is less than the number of default lines QMF has already allocated for the associated area in the default form, QMF keeps the excess lines.

- **Checking the object level in the header record**

The object level in the header record of a form file, data set, or CICS data queue tells you the level of the format structure at the time the form was generated. (Object level is indicated in bytes 12 and 13 of the header record as described in 85.) You can make sure that your application properly interprets the contents of the form file, data set, or CICS data queue by checking that the object level represents the format upon which your application is based.

- **Using application data records**

  The application data records mentioned in "Application data record (*)" on page 97 can be useful in your application program. They allow you to include your own comments within a file, data set, or CICS data queue for a form object. You can place them anywhere in the file, data set, or CICS data queue following the header record. When QMF reads such a record it ignores all data in the record following the *. The record therefore, has no effect on the import process.

- **Importing and exporting date/time information**

  If your installation supports date/time data types and you export a form with date/time information, you cannot subsequently import that form using a QMF installation that does not support date/time data types. If you do, the IMPORT command processing stops and QMF issues an error message.

- **Break field numbers changed for QMF Version 3.2**

  Instead of including table and field numbers for each Break panel, QMF Version 3.2 now uses one field number (3080) to act as a "trigger" to indicate which Break panel receives the information that follows it.

  If you create Break panels in an exported file, data set, or CICS data queue, you can set field 3080 to the number of the Break panel you want. Valid values for this field are 1 through 6 only.

  You can use any of the six break panels in an encoded file, data set, or CICS data queue without having to define all the panels. For example, you can create a Break5 panel without creating Break panels 1 through 4.

- **Form application migration aid**

  Because QMF introduced new field numbers for the Break panels in Version 3.2, and these new numbers are not compatible with earlier versions of QMF, the object level of an exported form has increased to object level 4. The form objects were not changed between QMF Version 3 Release 1 and Version 3 Release 1 Modification 1 (Version 3.1.1) or between Version 3.1.1 and Version 3 Release 2 (Version 3.2).

  When upgrading from QMF Version 2.4 or earlier (including QMF VSE Version 1) to QMF Version 3.2, you should upgrade your current applications to reflect the new Break field numbers. However, QMF Version 3.2 provides a Form Application Migration Aid that lets you install Version 3.2 and still use existing applications that use the old Break field numbers.

When you export QMF Version 3.2 form objects, this aid converts the new Break field numbers in these form objects to their QMF Version 2.4 counterparts. Your existing application can then run with QMF Version 3.2 without requiring an immediate upgrade.

Important: The Form Application Migration Aid does not allow you to export QMF Version 3.2 forms and use them with QMF Version 2.4.

- **Restrictions for using forms in CICS**

  Because REXX is not available under QMF CICS, the areas on the QMF form that rely on REXX do not work if you try to run the form in the CICS environment. These areas include anything entered on the FORM.CALC panels, the FORM.CONDITIONS panels, and the Specify Definition window. Therefore, REXX calculations, conditional row formatting, and column definitions are not available to QMF CICS users.

For additional information and rules governing the form files, data sets, or CICS data queues for input and output, see "Importing encoded format objects" on page 98.

Importing a form object

When you import a form, these fields must be in uppercase:

- Record identifier for all records
- The following fields in the header record:
  - Product identifier (QMF)
  - Type of object (F)
  - Format of object (E)
  - Action against object (R)
- Data type values (NUMERIC, CHAR, GRAPHIC, UNKNOWN) in the R records for the COLUMNS table. If your installation supports date/time data types, data type values DATE, TIME, and TIMEST must also be in uppercase.
- All the form keywords and substitution variables used in the form panels. When a form is imported, all the input in the form is left intact. If a form keyword is in lowercase, the error indicator in the form panel is turned on. To correct the error, the field must be typed over. If the data type value is not in uppercase, an error occurs, and the IMPORT ends.

The T record of the COLUMNS table (field number 1110) must immediately follow the header record, and it must include a numeric count of the number of rows in the table (an * row count is not allowed).

If the entire COLUMNS table has been read in, unspecified fields are set to their default values, and the form is displayed.

Variation panels

The variation number field (field number 2790) determines which variation panel is updated by all the variation panel information that follows the field. This V record should precede all other V, T, and R records for a given variation panel.

If a value for a particular variation appears more than once in the encoded format, the later values replace the original values. The number of variations in the form are equal to the highest variation number in the form. There is no required order for variation numbers when importing.

Translated forms

When you import an English language form into a non-English session, QMF automatically translates the reserved words in the form into your current session's language if the national language identifier in the H record is an E.

Omitting data type, edit code, and width in an imported form

In the COLUMNS table, data type (field number 1112), edit code (field number 1117), and width (field number 1116) can optionally be omitted when the following rules are observed:

- Edit code must be included if data type and width are omitted. Based on the specified edit code, QMF inserts appropriate defaults for data type and width.
- Data type must be included if edit code and width are omitted. QMF provides default values for edit code and width.
- Width must be accompanied by either data type or edit code.

Table 12 contains information about values for the column data type field.

Table 12. Values for Column Data Type field

Data Type Value (QMF form)	Data Type Number (Database form)	Character String (Database form)	Meaning
DATE	384	DATE	Date
TIME	388	TIME	Time
TIMEST	392	TIMESTAMP	Time stamp
	496	INTEGER	Integer
	500	SMALLINT	Small integer
	484	DECIMAL	Decimal
NUMERIC	480	FLOAT	Floating point
	448	VARCHAR	Varying character
	452	CHAR	Fixed character
	456	LONG VARCHAR	Long varying character
CHAR	904	ROWID	Row identifier

Table 12. Values for Column Data Type field (continued)

Data Type Value (QMF form)	Data Type Number (Database form)	Character String (Database form)	Meaning
			Varying graphic
	464	VARGRAPHIC	Fixed graphic
	468	GRAPHIC	Long varying
GRAPHIC	472	LONG VARGRAPHIC	graphic

In addition to the preceding data type values, there is an UNKNOWN data type that QMF uses in response to a U, V, or invalid edit code.

Detecting errors during import

If QMF detects an error in the format of the form file during import, the import function is ended with a message describing the error and its location in the file.

If an error is encountered in the header record and a form already exists in the temporary storage area, the existing form is displayed. If the form is successfully imported, QMF displays the form panel.

If an error is encountered after the header record is read, any existing form in the temporary storage area is discarded, and the Home panel is displayed. However, if the data object exists, QMF generates a default form for the data, but does not display it.

Certain minor errors detected by QMF do not terminate the import. In such cases, QMF issues a warning message and, where appropriate, applies defaults. Some examples are:

- V records
  - Zero length fields.
  - Specified length field does not match the length of data actually supplied.
- T records
  - Zero column length.
  - The number of columns specified does not match the following field number/length pairs.

You can respond to errors and warnings in either of two ways:

- Fix one problem at a time.
- Run the IMPORT FORM command with SET PROFILE (TRACE=L2).

Running the IMPORT FORM command with SET PROFILE (TRACE=L2) gives you a list of all the message numbers related to the IMPORT command. The command:

```
HELP message_number
```

displays the error message for that message number.

Exporting a form object

The format of forms exported before QMF Version 1 Release 2 is different from the format described in this section; however, QMF continues to accept forms exported with an earlier release of QMF. When such forms are imported or displayed in later releases, there can be some performance degradation.

"Table and field numbers for the form object" on page 232 lists the assignment of field numbers to the various parts of the form object. It also shows which parts of the form are tables and which are individual values in the exported file. Column data type (field 1112) is not displayed on the form panels but is associated with the form in its external format.

The column data type is not required when a form is imported. If it is missing during import, QMF provides default data type information from the edit codes. See "Importing a form object" on page 107 for more information. During export, the data type keyword (field number 1112) QMF provides is based on the specified edit code. For a U, V, or invalid edit code, QMF specifies the data type keyword UNKNOWN. Table 13 shows the data type keywords QMF generates for the edit codes specified. In this table, x represents the number of decimal places to be displayed, where x is an integer from 0 to 99.

Table 13. Data type keywords generated for edit codes specified

Edit Code Specified	Data Type Keyword
B, BW, C, CW, CT, CDx, X, XW	CHAR
G, GW	GRAPHIC
E, D, I, J, K, L, P, EZ, DZ, IZ, JZ, KZ, LZ, PZ, DZC, Dx, Ix, Jx, Kx, Lx, Px	NUMERIC
TDXx	DATE
TTXx	TIME
TSI	TIMEST
U, V	UNKNOWN
None of the above (invalid)	UNKNOWN

Variation panels

When you export a form, QMF only exports those variation panels with values that have been changed from the default. Therefore, the total number of variations in the external form can be less than that shown in the variation count indicator on the panel. QMF can alter the individual variation numbers to put the variations back into a continuous sequence.

Translated forms

When you export a form from a non-English session, you can either export the form in the current session language or in English. Because of this, the national language identifier in the H record might not reflect the language of the session from which you exported the form.

Report objects

When QMF displays a report, you see the result of interaction between the form and data objects in temporary storage. A report object does not exist in temporary storage. When you export a report, QMF is really exporting the interaction of a form and a data object. A report cannot be saved in the database, and an exported report cannot be imported back to QMF. However, you can use exported reports to:

- Extract data from the report and use it in the application
- Modify the appearance of the report for printing or redisplay by the application

Example of exporting a report

This example illustrates a report with a level 1 break. Figure 21 on page 112 shows the report to be exported. For an example of an across report, see 117.

For a list of the field numbers, see "Table and field numbers for the report object" on page 238.

```
REPORT                                              LINE 1      POS 1     79

                          J & H SUPPLY COMPANY
                     AVERAGE SALARIES (DEPTS 10, 15, 20)
                                REPORT 17


                    AVERAGE
   DEPT  JOB         SALARY
 ------  -----  ----------
     10  MGR      20865.86
                ----------
             *    20865.86

     15  CLERK    12383.35
         MGR      20659.80
         SALES    16502.83
                ----------
             *    15482.33

     20  CLERK    13878.68
         MGR      18357.50
         SALES    18171.25
                ----------
             *    16071.53

                ==========
                  17473.24


                                 COMPANY NAME
                                   REPORT 17
```

Figure 21. A tabular QMF report

Figure 22 on page 113 shows the format of the exported report shown in Figure 21.

```
H QMF 11 R 01 E V W E R 02 03 98/10/14 11:24
V  1001 006 PERIOD
V  1002 003 016
T  1010 003 006 1013 005 1014 006 1015 006 1016 006 1017 006 1012 008
R  L     000001 000003 000008 000001 BREAK1
R  C     000009 000011 000015 000001 GROUP
R  L2    000016 000018 000027 000001 AVERAGE
L  110 10100000                              J & H SUPPLY COMPANY
L  110 10100000                        AVERAGE SALARIES (DEPTS 10, 15, 20)
L  110 10100000                                 REPORT 17
L  110 10000000
L  110 10000000
L  170 10000000                      AVERAGE
L  170 10000000     DEPT  JOB        SALARY
L  170 10010000   ------  -----  ----------
L  181 11000000       10  MGR      20865.86
L  151 10010000                   ----------
L  151 11100000               *    20865.86
L  151 10000000
L  181 11000000       15  CLERK    12383.35
L  181 11000000           MGR      20659.80
L  181 11000000           SALES    16502.83
L  151 10010000                   ----------
L  151 11100000               *    15482.33
L  151 10000000
L  181 11000000       20  CLERK    13878.67
L  181 11000000           MGR      18357.50
L  181 11000000           SALES    18171.25
L  151 10010000                   ----------
L  151 11100000               *    16071.52
L  151 10000000
L  190 10010000                   ==========
L  190 11000000                     17473.24
L  120 10000000
L  120 10000000
L  120 10100000                                    COMPANY NAME
L  120 10100000                                      REPORT 17
E
```

Figure 22. Format of the exported sample report

When exporting a report, QMF writes the full text of the formatted report with additional information to interpret the contents of the report.

The header record is the first record of the exported file. It is followed by the appropriate V, T, and R records. If the report is an across-style report, it has another group of V, T, and R records that follows the first group.

In addition to H, V, T, R, and E records, exported reports also require two additional types of records:

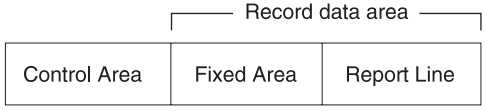
- Report line records (L)
- Data continuation records (C)

These two records follow the last group of V, T, and R records.

If you want to use only the formatted data of the report in your application, you can have QMF send print output to a file, data set, or CICS data queue. This file, data set, or CICS data queue contains only the formatted data without any layout information.

Report line records (L)

Each formatted line in a report is described by an L record. There is one L record for each line in the report. Like other variable format records (V, T, R), L records consist of a control area followed by a record data area. The format of the control area is similar to the other records; the record data area is composed of a fixed area that precedes the formatted report line itself. The fixed area provides information about the report line that follows it.



The contents of an L record are as follows:

Control area for L records:

Byte position
Description

01 Value record identifier (L)

02 Continuation indicator. Indicates whether the current record is continued to a data continuation record (see "Data continuation records (C)" on page 116):

- C for continued
- D for continued with DBCS delimiters `SO` and `SI` inserted at the end of the current record and the beginning of the data portion of the next record
- Blank if not continued

(See notes 1 and 2 on page 115 following the descriptions.)

Record data area for L records (fixed area):

Byte position
Description

01 Blank

02-04 Report part indicator 110 = Page heading 120 = Page footing 13n = Break heading (n is break number, 1-6) 15n = Break footing (n is break number, 1-6) 170 = Column heading 171 = Detail heading 180 = Detail line 181 = Group summary line 190 = Final footing

05 Blank

06-13 Line type attributes. Byte 06 is always 1. Each byte in bytes 7 through 13 indicates the presence or absence of the corresponding line type attribute in the formatted report line (1 = attribute present, 0 = attribute absent).

Byte position
Description

06 1

07 Data

08 Text

09 Separator

10 Column wrap. See note 3.

11 Line wrap. See note 3.

12 Second data line (across reports only). See note 4.

13 Reserved

14 Blank

id=repline.Record data area for L records (report line):

Byte position
Description

01-end
The actual formatted report line

Example of an L record:

```
L  151 11100000    DEPARTMENT TOTALS     93,659.45
```

(Break1 footing line containing text and data)

Notes:

1. A C record immediately follows an L record marked with a continuation character in byte 2 of the control area.
2. When D is specified for the continuation indicator in the control area, it means that the current record is too long to fit into a single physical record, and that, in the process of splitting up the record, SO (shift out) and SI (shift in) characters were added to the current and next records to preserve the integrity of the DBCS data being continued.
3. Attributes for column wrap (byte 10) and line wrap (byte 11) are used to indicate the continuation of a single logical report line to multiple *physical*

report lines. The presence of either attribute in a given L format record means that the column data or wrapped line is continued on a following L format record.

4. Across reports containing percent or cumulative sum columns can contain two data lines for each group (also break and final) summary. The first summary data line contains the cumulative percent or cumulative sum values of the column as computed *across* each unique across value. The second summary data line contains the cumulative percent or cumulative sum values of the column as computed *down* each group (in the report or within a control break). The *second data line* (byte 12) line type identifies the second data line in exported reports of this nature.

Data continuation records (C)

A C record is used to continue a value or set of values across more than one record. It immediately follows the record being continued. The format of a C record corresponds to the format of the original record being continued. QMF uses C records to continue L records only. The C record contains the following:

Control area for C records:

Byte position
Description

01 Value record identifier (C)

02 Continuation indicator. Indicates whether the current record is continued to another C record:

- C for continued
- D for continued with DBCS delimiters `SO` and `SI` inserted at the end of the current record and the beginning of the data portion of the next record
- Blank if not continued

(See Notes 1 on page 115 and 2 on page 115.)

Record data area for C records

The byte positions in the following list are *offset* from the end of the control area, the length of which is indicated in the header record.

Byte position
Description

01 Blank

02-end
Value or set of values being continued

Examples of C records

A report line continuation, in which a single report line value is split in the middle of the text, but is not continued to another continuation record:

```
C  ARS --->  <---- TOTAL --->
```

A report line continuation, in which a single report line value is split in the middle of the text and the record is continued to another C record:

```
CC ERK --->  <---- MGR ---> <---- SAL
```

Example of Exporting an across report

Figure 23 illustrates an exported across-style report.

```
REPORT                                          LINE 1      POS 1     79

                          J & H SUPPLY COMPANY
                          DEPT AVERAGE SALARIES
                        REPORT 18 (ACROSS REPORT)


         <---------------- JOB ----------------->
         <- CLERK -->  <-- MGR --->  <- SALES -->  <- TOTAL -->
              AVERAGE       AVERAGE       AVERAGE       AVERAGE
   DEPT       SALARY        SALARY        SALARY        SALARY
  ------    ----------    ----------    ----------    ----------
      10                    20865.86                    20865.86
      15      12383.35      20659.80      16502.83      15482.33
      20      13878.68      18357.50      18171.25      16071.53
      38      12482.25      17506.75      17407.15      15457.11
            ==========    ==========    ==========    ==========
              12914.76      19998.21      17372.10      16880.26


                               COMPANY NAME
                                 REPORT 18
                                  PAGE 1
```

Figure 23. Sample across report. This report uses QMF across report functions.

Figure 24 on page 118 shows the resulting encoded format from Figure 23.

```
H QMF 11 R 01 E V W E R 02 03 98/10/14 16:20

V  1001 006 PERIOD
V  1002 003 016
T  1010 002 006 1013 005 1014 006 1015 006 1016 006 1017 006 1012 008
R  L     000001 000003 000008 000001 GROUP
R  L2    000003 000005 000014 000001 AVERAGE
V  2001 005 C
V  2002 003 001
V  2003 003 YES
T  2010 004 003 2012 006 2013 006 2014 006
R  000014 000018 000009
R  000029 000031 000023
R  000042 000046 000037
R  000056 000060 000051
L  110 10100000                              J & H SUPPLY COMPANY
L  110 10100000                              DEPT AVERAGE SALARIES
L  110 10100000                            REPORT 18 (ACROSS REPORT)
L  110 10000000
L  110 10000000
L  170 10000000           <----------------- JOB ----------------->
L  170 11000000           <- CLERK -->  <-- MGR --->  <- SALES -->  <- TOTAL -->
L  170 10000000                AVERAGE       AVERAGE       AVERAGE       AVERAGE
L  170 10000000     DEPT       SALARY        SALARY        SALARY        SALARY
L  170 10010000   ------    ----------    ----------    ----------    ----------
L  181 11000000       10                   20865.86                    20865.86
L  181 11000000       15      12383.35      20659.80      16502.83      15482.33
L  181 11000000       20      13878.68      18357.50      18171.25      16071.53
L  181 11000000       38      12482.25      17506.75      17407.15      15457.11
L  190 10010000             ==========    ==========    ==========    ==========
L  190 11000000               12914.76      19998.21      17372.10      16880.26
L  120 10000000
L  120 10000000
L  120 10100000                                   COMPANY NAME
L  120 10100000                                     REPORT 18
L  120 10100000                                      PAGE 1
E
```

Figure 24. Format of the exported across-style report

HTML reports

When you export a report for HTML, QMF places the necessary HTML tags before and after the body of your report so that you can place it on a web server and displayit in an HTML 3.0 compliant web browser. Figure 25 on page 119 illustrates the HTML coding that QMF places around the report.

```
<HTML>
<HEAD>
<TITLE>
Report
</TITLE>
</HEAD>
<BODY>
<PRE>

                J & H SUPPLY COMPANY
          AVERAGE SALARY  (DEPTS  10,  15,  20)
                      REPORT 17

                        AVERAGE
        DEPT  JOB         SALARY
        ----  -----    ---------
        10    MGR       20865.86
                       ---------
                    *   20865.86

        15    CLERK     12383.53
              MGR       20659.80
              SALES     16052.83
                       ---------
                    *   15482.33

        20    CLERK     13878.67
              MGR       18357.50
              SALES     18171.25
                       ---------
                    *   16071.52
                       =========
                       17473.52

                      COMPANY NAME
                       REPORT  17




</PRE>
</BODY>
</HTML>
```

Figure 25. Sample HTML Report Coding

QBE queries

QBE query objects are exported using a format internal to QMF. This format cannot be altered in any way.

Specifications for externalized QMF objects

Table 14 contains specifications for TSO and CMS IMPORT and EXPORT files.

For CICS, record sizes are the same as those indicated in Table 14; however, they are not enforced. For example, you could import an SQL query from a temporary storage queue with a record size of 32k and QMF would truncate it to 79 bytes.

Record format is not a factor for CICS temporary storage or transient data queues. A temporary storage queues holds records without regard to their format. A transient data queue is defined to a destination control table (DCT) and ignores the record format.

Queue names are user generated and have no default prefix or suffix. CICS TS queue names are 8 bytes. TD queue names are 4 bytes.

Table 14. File and data set attributes

Object	Reord Size	Record format (CMS/TSO)
Data or table (QMF format)	Maximum size: 7,000 bytes	Records must be fixed length
Data or table (IXF format)	Maximum size: 32,756 (see note 2) The minimum LRECL that QMF accepts for an IXF file, data set, or CICS data queue during import is 49 bytes.	Records must be variable length
Prompted query	Maximum: 7,290 bytes Minimum: 266 bytes on EXPORT; 41 bytes on IMPORT.	Records must be variable length on EXPORT; can be either fixed or variable on IMPORT.
SQL query	Must be 79 bytes on EXPORT; must be less than 256 bytes on IMPORT, but is truncated to 79 bytes.	Records must be fixed length on EXPORT; can be either fixed or variable on IMPORT.
QBE query	Must be 1,024 bytes (see note 3).	Records must be variable length.
Form	Maximum: 7,290 bytes Minimum: 161 bytes on EXPORT; 23 bytes on IMPORT.	Records must be variable length on EXPORT; can be either fixed or variable on IMPORT.
Proc	Must be 79 bytes on EXPORT; can be any size on IMPORT, but is truncated to 79 bytes.	Records must be fixed length on EXPORT; can be either fixed or variable on IMPORT.

Table 14. File and data set attributes (continued)

Object	Reord Size	Record format (CMS/TSO)
Report	Maximum: 7,290 bytes Minimum: 65 bytes	Records must be variable length.
HTML Report	Maximum: 32,000 bytes	Records must be variable length.

Notes:

1. You must specify a name for your file, data set, or CICS data queuein the EXPORT or IMPORT command. For more information about names, see *QMF Reference*.
2. The minimum LRECL for an exported form that includes defined columns is 161 bytes. This minimum accommodates Version 3.2 enhancements to QMF forms, including column heading alignment, column data alignment, column definition expression, and information about passing nulls. If the form does not contain column definition information, the minimum LRECL for CMS is 113 bytes.
3. An empty QBE query is 828 bytes.
4. Record size is normally the length of a row of data in the table being exported (including space for null indicators and DBCS delimiters), plus the length of the IXF D-type record count field (5 bytes). If the record size derived from the row length is less than the length of the longest IXF header record (81 bytes), then the record size is set to 81 bytes.

Rules and considerations when using CICS queues

Rules:

1. In CICS, both IMPORT and EXPORT require that you specify the QUEUETYPE option. There is no default.
2. When importing an object from a transient data (TD) queue in CICS, you must specify the correct object type; the queue is emptied once QMF retrieves its contents. For example, if you specify “Form” when the object type in the transient data queue is a procedure, QMF issues an error message. However, you cannot successfully issue the IMPORT command again (even with the corrected object type) using the same queue, because that queue is now empty.
3. In CICS, the transient data or temporary storage (TS) queue must contain a single, completed QMF object before you issue the IMPORT command.
4. If you export to a transient data queue, the queue must be open, enabled and empty before you issue the EXPORT command. For information about CICS transient data queues, see *CICS for VSE/ESA Application Programming Guide*

Considerations:

QMF handles CICS transient data queues differently than temporary storage queues.

- **Transient data queues:** QMF imports the entire transient data queue prior to displaying the object on the screen. This means that the contents of the entire queue must fit into your storage or spill area. It also means that, if the object to be displayed is large, there may be a delay before QMF displays the object on the screen.

  A CICS intrapartition transient data queue can hold up to 32K rows of data; an extrapartition transient data queue can be as large as it needs to be to hold the object.
- **Temporary storage queues:** QMF reads approximately 100 rows of temporary storage before displaying them to the user. A temporary storage queue can hold up to 32K rows of data.
- **Adding a QMF object to the queue:** QMF uses the SUSPEND parameter on the IMPORT and EXPORT commands to let CICS regulate when the command is run.

  The SUSPEND parameter on the IMPORT and EXPORT commands determines the action to be taken if a queue is busy. When the SUSPEND parameter is set to `YES`, QMF issues a CICS ENQ (enqueue) for the CICS data queue name. This tells CICS to wait until the queue is available before writing the QMF object to the queue, thus ensuring that the QMF transaction does not interfere with any other jobs being handled by the queue.

  When the SUSPEND parameter is set to `NO`, the EXPORT command is canceled and a message is returned. The default value of SUSPEND is `NO`. That QMF issues an automatic ENQ is reflected in the SUSPEND option of the EXPORT and IMPORT commands.

Chapter 10. Debugging Your QMF Applications

In addition to error-handling and application support commands, QMF provides debugging facilities for your programs. The techniques described in this chapter apply to callable interface applications.

For information on ISPF debugging techniques, see Chapter 6, "Writing QMF Applications that Use ISPF" on page 39. You can use the REXX trace facility through the TRACE statement. For more details on this statement, see *REXX Reference.*

Debugging your callable interface applications

QMF provides two trace options, L and A, and several different levels of tracing for debugging your applications.

Using the L-option for tracing

The L-option lets you tell QMF to write messages and commands to an external QMF trace data output that you allocate before you begin your QMF session. There are two L-options you can choose:

L1 Every QMF message is written to the QMF trace data output.

L2 Every QMF message *and* command is written to QMF trace data output. For example, you can use L2 to trace and debug Q.SYSTEM_INI system initialization procedure commands and messages.

You can set the L-option in one of two ways:

1. Issue the `DISPLAY PROFILE` command, and when the PROFILE object appears, change the TRACE option to either L1 or L2.
2. Issue the command:

   ```
   SET PROFILE (TRACE=x
   ```

   where x is either L1 or L2.

If you allocate the trace data output yourself, you can arrange for the trace information to be printed or subsequently viewed at a terminal. In either case, you can then examine the data after the QMF session. See "Allocating the QMF trace data output" on page 125 for details on allocation, or consult your information center.

Using the A-option for tracing

The A-option allows you to specify a level of tracing for QMF application support services.

The A-option setting can be A0, A1, or A2. A0 is the default and is interpreted as the signal for no A-tracing at all. A1 and A2 can then call for increasingly detailed results. This is the pattern used for the other QMF trace options.

You specify the A-option in the same way you specify the L-option: through a QMF SET command, or by entering it on the screen after you execute the DISPLAY PROFILE command. For example, you can enter the following just before you invoke the application you are debugging:

```
SET PROFILE (TRACE=L2A1)
```

Then, when you begin your application, both L2 and A1 tracing are in effect.

To determine the current A-option setting, look at the variable DSQAO_APPL_TRACE. Its value is 0, 1, or 2, respectively, for the settings A0, A1, or A2. You can use the value of DSQAO_APPL_TRACE to select the kind of tracing you want in your application, as in Figure 26.

```
/*  REXX program to set tracing                */
call dsqcix "GET GLOBAL(A_TRACE=DSQAO_APPL_TRACE"
if a_trace > 0 then
  do
    /* trace code for both A1 and A2 */

⋮
    if a_trace = 2 then
      do
         /* trace code for just A2   */

⋮
      end
  end
```

Figure 26. Sample REXX program to set tracing

Nested DO-groups like the ones in Figure 26 can appear throughout an application. Where they appear, they take “snapshot” dumps of certain data areas, print the values of certain critical variables, load a debugging module, or perform any other diagnostic procedure that can help you debug the application. Precisely what is done depends on the setting in effect for the A-option while the application is running.

A good place for A-option code is in a large application. Consider leaving this code in the application after you finish your debugging. Doing this does not produce A-trace output if you run the application with an A0 setting. If you modify the application, and in the process introduce a bug, you can run this code again.

Turning the tracing off

To turn the tracing off after you test the application, issue the following command:

```
SET PROFILE (TRACE=NONE
```

This discontinues tracing for the rest of your QMF session, but does not affect your permanent QMF profile.

Allocating the QMF trace data output

You must allocate the QMF trace data output *before* you invoke QMF if tracing is to be used. It is possible that the output was allocated automatically through your startup procedure. Even so, you might want to reallocate the output if the original allocation does not meet your needs.

For examples of how to allocate QMF trace data output for CMS or TSO, see the assemble or compile and execution coding example in the chapter discussing the appropriate language:

Assembler
Assembler language interface, beginning on page 127.

C Language
C Language Interface, beginning on page 150.

COBOL
COBOL language interface, beginning on page 167.

FORTRAN
FORTRAN language interface, beginning on page 184.

PL/I PL/I language interface, beginning on page 200.

REXX REXX language interface, beginning on page 216.

The commands in the examples allocate a sequential trace data output that you can examine at a terminal after your QMF session is over. The output consists of fixed length, 80-character records. The trace information is formatted to 80 characters per line. You can view an entire line of output on a terminal screen.

For CICS, you can use program parameters DSQSDBQT and DSQSDBQN to specify where QMF puts your trace data. Use caution when using CICS

temporary storage because QMF can produce a large amount of trace data. CICS temporary storage is recommended for message or small application trace data only.

Using tracing with the QMF MESSAGE command

You can use the QMF MESSAGE command to do more than display a message when an application ends. You can also use it to record messages in the QMF trace data output. To do this, run the application with the L-Option for TRACE set to L1 or L2. (For information on how to do this, refer to "Using the L-option for tracing" on page 123.) Every message processed through the MESSAGE command is then recorded, along with other QMF messages (and commands if L2 is used), in the QMF trace data output.

By placing MESSAGE commands in strategic places in your program, you can log useful information in the QMF trace file. After the QMF session, you can examine it, either on the terminal or in printed output. For more information about the QMF trace data output, see "Allocating the QMF trace data output" on page 125.

Example

An application issues the commands shown in the following example:

```
call dsqcix "SET PROFILE (TRACE=L2"
⋮
call dsqcix "MESSAGE (TEXT='QUERYA COMPLETED SUCCESSFULLY'"
⋮
call dsqcix "MESSAGE (TEXT='EXECB ENTERED WITH VALUE OF 7'"
⋮
```

Records containing the messages 'QUERYA COMPLETED SUCCESSFULLY' and 'EXECB ENTERED WITH VALUE OF 7' are written into the QMF trace data output.

Because QMF messages can change from one release to the next, you should not use the QMF trace data output as input to an application.

Debugging errors on the START and other QMF commands

Depending on the level of your DSQCOMM, you might have message text in your DSQCOMM. If your START command (or any QMF command) fails, this message text is invaluable for debugging. If you are working with the current level of DSQCOMM, the message text is available to you. See *QMF Messages and Codes* for information about all QMF error messages.

Appendix A. Sample Code for Callable Interface Languages

This appendix contains sample code for each of the QMF callable interface languages:

Assembler
"Assembler language interface"

C Language
"C Language Interface" on page 150

COBOL
"COBOL language interface" on page 167

FORTRAN
"FORTRAN language interface" on page 184

PL/I "PL/I language interface" on page 200

REXX "REXX language interface" on page 216

This appendix contains a sample program for each language supported by QMF. Each sample program:

- Starts QMF
- Sets three global variables
- Runs a query called Q1
- Prints the resulting report using form F1
- Ends the QMF session

QMF does not supply query Q1 or form F1, but the sample programs, as written, use these objects.

This appendix also shows how to assemble or compile, link-edit, and run the programs using the callable interface. QMF does not ship the REXX EXECs, JCL, or CLISTs in these examples, but you can copy them and alter them to suit your installation.

Assembler language interface

If you use assembler language, you must use Assembler H or High Level Assembler (HLASM) with the callable interface. QMF provides one function call, DSQCIA, for the assembler language.

For CICS/VSE, you must use HLASM to construct 31-bit addressing.

Interface communications area mapping for Assembler (DSQCOMMA)

DSQCOMMA provides DSQCOMM mapping for assembler language and is shipped with the product. Table 15 shows the values for DSQCOMMA.

Table 15. Interface communications area for DSQCOMMA

Structure Name	Data Type	Description
DSQ_RETURN_CODE	DS F	Indicates the status of a QMF command after it runs. Its values are: **DSQ_SUCCESS** Successful execution of the request. **DSQ_WARNING** Normal completion with warnings. **DSQ_FAILURE** Command did not execute correctly. **DSQ_SEVERE** Severe error; QMF session terminated.
DSQ_INSTANCE_ID	DS F	Identifier established by QMF during execution of the START command
DSQ_COMM_LEVEL	DS CL12	Identifies the level of the DSQCOMM. You should set this to the value of DSQ_CURRENT_COMM_LEVEL before issuing the QMF START command.
DSQ_PRODUCT	DS CL2	Identifies the IBM query product in use.
DSQ_PRODUCT_RELEASE	DS CL2	Identifies the release level of the query product in use.
DSQ_RESERVE1	DS XL28	Reserved for future use
DSQ_MESSAGE_ID	DS CL8	Completion message ID
DSQ_Q_MESSAGE_ID	DS CL8	Query message ID
DSQ_START_PARM_ERROR	DS CL8	Parameter in error when START failed due to a parameter error
DSQ_CANCEL_IND	DS C	Contains one of two values, depending if the user canceled while a QMF command was running: • DSQ_CANCEL_YES • DSQ_CANCEL_NO
DSQ_RESERVE2	DS XL23	Reserved for future use
DSQ_RESERVE3	DS XL156	Reserved for future use

Table 15. Interface communications area for DSQCOMMA (continued)

Structure Name	Data Type	Description
DSQ_MESSAGE_TEXT	DS CL128	Completion message text
DSQ_Q_MESSAGE_TEXT	DS CL128	Query message text

Function calls for Assembler language

The function call for assembler language has two formats: DSQCIA and DSQCIA extended syntax.

DSQCIA

This call is for QMF commands that do not require access to application program variables. Use this call for most QMF commands.

```
CALL DSQCIA,(DSQCOMM,CMDLTH,CMDSTR),VL
```

The parameters have the following values:

DSQCOMM
The interface communications area

CMDLTH
Length of the command string, CMDSTR; a FULLWORD parameter

CMDSTR
QMF command to execute; an uppercase character string of the length specified by CMDLTH

`VL` is the assembler VARIABLE LIST statement.

DSQCIA, extended syntax

This extended syntax format of the DSQCIA function is for the three QMF commands that require access to application program variables: START and the extended formats of GET GLOBAL and SET GLOBAL.

```
CALL DSQCIA,(DSQCOMM,CMDLTH,CMDSTR,
       PNUM,KLTH,KWORD,VLTH,VALUE,VTYPE),VL
```

The parameters have the following values:

DSQCOMM
The interface communications area

CMDLTH
Length of the command string, CMDSTR; a FULLWORD parameter

CMDSTR
QMF command to execute; an uppercase character string of the length specified by CMDLTH

PNUM
Number of command keywords; a FULLWORD parameter

KLTH
Length of each specified keyword; a FULLWORD parameter or array of FULLWORD parameters

KWORD
QMF keyword or keywords; a character or structure of characters whose lengths are the same as specified by KLTH

VLTH
Length of each value associated with the keyword; a FULLWORD parameter or array of FULLWORD parameters

VALUE
Value associated with each keyword. Its type is specified in the VTYPE parameter, and can be a character, structure of characters, FULLWORD parameter, or array of FULLWORD parameters.

VTYPE
QMF data type of the value string VALUE. This type has one of two values, which are provided in the communications macro, DSQCOMMA:

- DSQ_VARIABLE_CHAR for character values. If VTYPE is DSQ_VARIABLE_CHAR, then VALUE is not validated.
- DSQ_VARIABLE_FINT for integer values. If VTYPE is DSQ_VARIABLE_FINT, then VALUE is validated, and VALUE must be an integer.

All values specified in the VALUE field must have the data type specified in VTYPE.

`VL` is the assembler VARIABLE LIST statement.

Migration information

The DSQCOMM changed between Version 2 Release 4 and Version 3.2.

- If you want to continue using the old DSQCOMM, you do not have to reassemble your program.
- If you want to use Version 3.2 of DSQCIA, you must again link-edit your Version 2 Release 4 program.

The Version 3.2 DSQCOMM provides message text that is especially useful if there is an error in your START command. If you want to use the new DSQCOMM, you need to reassemble your program and initialize DSQ_COMM_LEVEL (in DSQCOMM) to DSQ_CURRENT_COMM_LEVEL. If you do *not* set this value, QMF treats your DSQCOMM as a Version 2 Release 4 level.

The Version 3.2 level of DSQCOMM is 512 bytes long, which is an increase from the 256 bytes available in Version 2 Release 4. Any instructions that are used to move or initialize this structure that had a 256-byte limit (such as MVC) must be changed to use instructions that work on a larger data area.

Note to CICS/MVS users

The DSQCIA changed between Version 3 Release 1 Modification 1 and Version 3 Release 2. The interface between the QMF-supplied function call and the main QMF program changed from a CALL interface to an EXEC CICS LINK interface. The new interface provides better isolation from the user program and the QMF product. Because the interface has changed, you need to link-edit your programs that used the callable interface again.

Assembler programming examples

You can look at the sample source code listings here or you can access them online.

- For MVS, the sample program is a member of the library QMF720.SDSQSAPE.
- For VM, the sample program is on the production disk.
- For VSE, the sample program is in the QMF sublibrary and is named DSQABFAC.Z.

The sample programs for the assembler callable interface perform the following functions:

- Start QMF
- Set three global variables
- Run a query called Q1
- Print the resulting report using form F1
- End the QMF session

QMF does not supply query Q1 or form F1, but the sample programs use these objects.

This section also shows how to assemble, link-edit, and run an assembler program using the callable interface. QMF does not ship the REXX EXEC, JCL, or CLIST in these examples, but you can copy them from here, altering them to suit your installation.

Sample Assembler program for CICS/MVS and CICS/VSE

The program DSQABFAC is shipped with QMF for CICS.

```
         TITLE 'Sample HLASM Query Callable Interface'                  00001000
********************************************************************** 00002000
*                                                                    * 00003000
*  Sample Program:  DSQABFAC                                         * 00004000
*  Assembler Version of the SAA Query Callable Interface             * 00005000
*                                                                    * 00006000
********************************************************************** 00007000
DSQABFAC DFHEIENT CODEREG=(12),DATAREG=(13),EIBREG=(11)                 00008000
         SPACE 1                                                        00009000
********************************************************************** 00010000
* Start a query interface session                                   * 00011000
********************************************************************** 00012000
         LA    R4,CICOMM             ESTABLISH ACCESS TO DSQCOMM       00013000
         USING DSQCOMM,R4                                               00014000
         SPACE 1                                                        00015000
         MVC   DSQ_COMM_LEVEL,DSQ_CURRENT_COMM_LEVEL                    00016000
         ST    R4,QMFP1              Address of DSQCOMMA               00017000
         LA    R1,STARTQIL           Address of START command length   00018000
         ST    R1,QMFP2                                                 00019000
         LA    R1,STARTQI            Address of START command          00020000
         ST    R1,QMFP3                                                 00021000
         LA    R1,1                  One Start command parameter       00022000
         ST    R1,NUMPARMS                                              00023000
         LA    R1,NUMPARMS           Address of number of parameters   00024000
         ST    R1,QMFP4                                                 00025000
         LA    R1,STARTKYL           Address of keyword lengths        00026000
         ST    R1,QMFP5                                                 00027000
         LA    R1,STARTKY            Address of keywords               00028000
         ST    R1,QMFP6                                                 00029000
         LA    R1,STARTVL            Address of value lengths          00030000
         ST    R1,QMFP7                                                 00031000
         LA    R1,STARTV             Address of values                 00032000
         ST    R1,QMFP8                                                 00033000
         LA    R1,DSQ_VARIABLE_CHAR  Address of value data type        00034000
         ST    R1,QMFP9                                                 00035000
         OI    QMFP9,X'80'           Set end of parameter list         00036000
         LA    R1,QMFPLIST           Address of parameter list         00037000
         CALL  DSQCIA                                                   00038000
         SPACE 1                                                        00039000
```

Figure 27. DSQABFAC, sample HLASM program for CICS/MVS and CICS/VSE (Part 1 of 5)

```
*********************************************************************** 00040000
* Set numeric values into query using SET command                     * 00041000
*********************************************************************** 00042000
         SPACE 1                                                        00043000
         LA    R1,20                Set values for SET GLOBAL command  00044000
         ST    R1,VVAL1                                                 00045000
         LA    R1,40                                                    00046000
         ST    R1,VVAL2                                                 00047000
         LA    R1,84                                                    00048000
         ST    R1,VVAL3                                                 00049000
         LA    R1,SETGL             Addr of SET GLOBAL command length  00050000
         ST    R1,QMFP2                                                 00051000
         LA    R1,SETG              Address of SET GLOBAL command      00052000
         ST    R1,QMFP3                                                 00053000
         LA    R1,3                 Three SET GLOBAL variables         00054000
         ST    R1,NUMPARMS                                              00055000
         LA    R1,NUMPARMS          Address of number of parameters    00056000
         ST    R1,QMFP4                                                 00057000
         LA    R1,VNAME1L           Address of variable name lengths   00058000
         ST    R1,QMFP5                                                 00059000
         LA    R1,VNAME1            Address of variable names          00060000
         ST    R1,QMFP6                                                 00061000
         LA    R1,VVAL1L            Address of value lengths           00062000
         ST    R1,QMFP7                                                 00063000
         LA    R1,VVAL1             Address of values                  00064000
         ST    R1,QMFP8                                                 00065000
         LA    R1,DSQ_VARIABLE_FINT Address of value data type         00066000
         ST    R1,QMFP9                                                 00067000
         OI    QMFP9,X'80'          Set end of parameter list          00068000
         LA    R1,QMFPLIST          Address of parameter list          00069000
         CALL  DSQCIA                                                   00070000
         SPACE 1                                                        00071000
*********************************************************************** 00072000
* Run a query                                                         * 00073000
*********************************************************************** 00074000
```

Figure 27. DSQABFAC, sample HLASM program for CICS/MVS and CICS/VSE (Part 2 of 5)

```
         LA    R1,QUERYL            Addr of RUN QUERY command length   00075000
         ST    R1,QMFP2                                                 00076000
         LA    R1,QUERY             Address of RUN QUERY command       00077000
         ST    R1,QMFP3                                                 00078000
         OI    QMFP3,X'80'          Set end of parameter list          00079000
         LA    R1,QMFPLIST          Address of parameter list          00080000
         CALL  DSQCIA                                                   00081000
         SPACE 1                                                        00082000
********************************************************************** 00083000
* Print the result of the query                                     * 00084000
********************************************************************** 00085000
         LA    R1,REPTL             Addr of PRINT Report command lth   00086000
         ST    R1,QMFP2                                                 00087000
         LA    R1,REPT              Address of PRINT Report command    00088000
         ST    R1,QMFP3                                                 00089000
         OI    QMFP3,X'80'          Set end of parameter list          00090000
         LA    R1,QMFPLIST          Address of parameter list          00091000
         CALL  DSQCIA                                                   00092000
         SPACE 1                                                        00093000
********************************************************************** 00094000
* End the query interface session                                   * 00095000
********************************************************************** 00096000
         LA    R1,ENDQIL            Address of EXIT command length     00097000
         ST    R1,QMFP2                                                 00098000
         LA    R1,ENDQI             Address of EXIT command            00099000
         ST    R1,QMFP3                                                 00100000
         OI    QMFP3,X'80'          Set end of parameter list          00101000
         LA    R1,QMFPLIST          Address of parameter list          00102000
         CALL  DSQCIA                                                   00103000
         SPACE 1                                                        00104000
********************************************************************** 00105000
* Return                                                            * 00106000
********************************************************************** 00107000
         SPACE 1                                                        00108000
```

Figure 27. DSQABFAC, sample HLASM program for CICS/MVS and CICS/VSE (Part 3 of 5)

```
         XR    R15,R15            ZERO RETURN CODE                      00109000
         DFHEIRET RCREG=15                                              00110000
********************************************************************** 00111000
* Data Areas                                                          * 00112000
********************************************************************** 00113000
         SPACE 1                                                        00114000
*  Query Interface commands                                            00115000
         SPACE 1                                                        00116000
STARTQI  DC    C'START'              START FUNCTION                     00117000
SETG     DC    C'SET GLOBAL'         SET GLOBAL FUNCTION                00118000
QUERY    DC    C'RUN QUERY Q1'       RUN QUERY                          00119000
REPT     DC    C'PRINT REPORT (FORM=F1)'  PRINT REPORT                  00120000
ENDQI    DC    C'EXIT'               END INTERFACE                      00121000
         SPACE 1                                                        00122000
         DS    0F                                                       00123000
STARTQIL DC    AL4(L'STARTQI)        LENGTH OF START FUNCTION           00124000
SETGL    DC    AL4(L'SETG)           LENGTH OF SET GLOBAL FUNCTION      00125000
QUERYL   DC    AL4(L'QUERY)          LENGTH OF RUN QUERY COMMAND        00126000
REPTL    DC    AL4(L'REPT)           LENGTH OF PRINT REPORT COMMAND     00127000
ENDQIL   DC    AL4(L'ENDQI)          LENGTH OF END INTERFACE COMMAND    00128000
         SPACE 1                                                        00129000
*  START command keyword                                               00130000
         SPACE 1                                                        00131000
STARTKY  DC    C'DSQSMODE'                                              00132000
STARTV   DC    C'INTERACTIVE'                                           00133000
         DS    0F                                                       00134000
STARTKYL DC    AL4(L'STARTKY)                                           00135000
STARTVL  DC    AL4(L'STARTV)                                            00136000
         SPACE 1                                                        00137000
*  SET GLOBAL command variable names                                   00138000
         SPACE 1                                                        00139000
VNAME1   DC    C'MYVAR01'                                               00140000
VNAME2   DC    C'SHORT'                                                 00141000
VNAME3   DC    C'MYVAR03'                                               00142000
         DS    0F                                                       00143000
VNAME1L  DC    AL4(L'VNAME1)                                            00144000
VNAME2L  DC    AL4(L'VNAME2)                                            00145000
VNAME3L  DC    AL4(L'VNAME3)                                            00146000
         SPACE 1                                                        00147000
*  SET GLOBAL command values                                           00148000
         SPACE 1                                                        00149000
VVAL1L   DC    AL4(L'VVAL1)                                             00150000
VVAL2L   DC    AL4(L'VVAL2)                                             00151000
VVAL3L   DC    AL4(L'VVAL3)                                             00152000
* Callable interface communications definition                         00153000
         DSQCOMMA                                                       00154000
```

Figure 27. DSQABFAC, sample HLASM program for CICS/MVS and CICS/VSE (Part 4 of 5)

```
* Equates for registers 0-15                                          00155000
R0       EQU   00                                                     00156000
R1       EQU   01                                                     00157000
R2       EQU   02                                                     00158000
R3       EQU   03                                                     00159000
R4       EQU   04                                                     00160000
R5       EQU   05                                                     00161000
R6       EQU   06                                                     00162000
R7       EQU   07                                                     00163000
R8       EQU   08                                                     00164000
R9       EQU   09                                                     00165000
R10      EQU   10                                                     00166000
R11      EQU   11                                                     00167000
R12      EQU   12                                                     00168000
R13      EQU   13                                                     00169000
R14      EQU   14                                                     00170000
R15      EQU   15                                                     00171000
* Local variables located in CICS working storage                     00172000
DFHEISTG DSECT                                                        00173000
         ORG   DFHEIUSR                                               00174000
NUMPARMS DS    F                     NUMBER OF KEYWORDS              00175000
* QMF SET GLOBAL command values                                       00176000
VVAL1    DS    F                                                      00177000
VVAL2    DS    F                                                      00178000
VVAL3    DS    F                                                      00179000
* QMF Callable interface parameter list                               00180000
QMFPLIST DS   0D                                                      00181000
QMFP1    DS    F                                                      00182000
QMFP2    DS    F                                                      00183000
QMFP3    DS    F                                                      00184000
QMFP4    DS    F                                                      00185000
QMFP5    DS    F                                                      00186000
QMFP6    DS    F                                                      00187000
QMFP7    DS    F                                                      00188000
QMFP8    DS    F                                                      00189000
QMFP9    DS    F                                                      00190000
* Callable interface communications area                              00191000
CICOMM   DS    CL(DSQCOMM_LEN)                                        00192000
         CSECT                                                        00193000
         SPACE 1                                                      00194000
         END   DSQABFAC                                               00195000
```

Figure 27. DSQABFAC, sample HLASM program for CICS/MVS and CICS/VSE (Part 5 of 5)

Sample Assembler program for TSO and CMS

For TSO and CMS, QMF ships the following program with the product. It is named DSQABFA.

```
DSQABFA  TITLE 'SAMPLE SAA QUERY CALLABLE INTERFACE'
DSQABFA  CSECT
*********************************************************************
*                                                                   *
*  Sample Program:  DSQABFA                                         *
*  Assembler Version of the SAA Query Callable Interface            *
*                                                                   *
*********************************************************************
         SPACE 1
         STM   R14,R12,12(R13)       SAVE ENTRY REGISTERS
         BALR  R12,0                 INITIALIZE BASE REGISTER
         USING *,R12
         LA    R2,SAVEAREA           CHAIN SAVE AREAS
         ST    R2,8(R13)
         ST    R13,SAVEAREA+4
         LR    R13,R2                ESTABLISH SAVE AREA
         SPACE 1
*********************************************************************
* Start a query interface session                                   *
*********************************************************************
         LA    R4,CICOMM             ESTABLISH ACCESS TO DSQCOMM
         USING DSQCOMM,R4
         SPACE 1
         MVC   DSQ_COMM_LEVEL,DSQ_CURRENT_COMM_LEVEL
         LA    R1,1                  1 PARAMETER
         ST    R1,NUMPARMS
         CALL  DSQCIA,                                              X
               (CICOMM,              QI COMMON AREA                 X
               STARTQIL,             START COMMAND LENGTH           X
               STARTQI,              START COMMAND                  X
               NUMPARMS,             NUMBER OF KEYWORDS             X
               STARTKYL,             KEYWORD LENGTHS                X
               STARTKY,              KEYWORDS                       X
               STARTVL,              VALUE LENGTHS                  X
               STARTV,               VALUES                         X
               DSQ_VARIABLE_CHAR),VL VALUES ARE CHARACTERS
         SPACE 1
```

Figure 28. DSQABFA, sample assembler program for TSO and CMS (Part 1 of 4)

```
*********************************************************************
* Set numeric values into query using SET command                   *
*********************************************************************
         SPACE 1
         LA    R1,20                SET VALUES TO BE MODIFIED
         ST    R1,VVAL1
         LA    R1,40
         ST    R1,VVAL2
         LA    R1,84
         ST    R1,VVAL3
         LA    R1,3                 3 PARAMETERS
         ST    R1,NUMPARMS
         SPACE 1
         CALL  DSQCIA,                                            X
               (CICOMM,                                           X
               SETGL,               SET GLOBAL COMMAND LENGTH     X
               SETG,                SET GLOBAL COMMAND            X
               NUMPARMS,            NUM OF VARIABLES TO BE SET    X
               VNAME1L,             VARIABLE NAME LENGTHS         X
               VNAME1,              VARIABLE NAMES                X
               VVAL1L,              VALUE LENGTHS                 X
               VVAL1,               VALUES                        X
               DSQ_VARIABLE_FINT),VL VALUES ARE INTEGERS
         SPACE 1
*********************************************************************
* Run a query                                                       *
*********************************************************************
         SPACE 1
         CALL  DSQCIA,                                            X
               (CICOMM,                                           X
               QUERYL,              QUERY COMMAND LENGTH          X
               QUERY),VL            TEXT OF QUERY COMMAND
         SPACE 1
*********************************************************************
* Print the result of the query                                     *
*********************************************************************
         SPACE 1
         CALL  DSQCIA,(CICOMM,REPTL,REPT),VL
         SPACE 1
*********************************************************************
* End the query interface session                                   *
*********************************************************************
         SPACE 1
         CALL  DSQCIA,(CICOMM,ENDQIL,ENDQI),VL
         SPACE 1
```

Figure 28. DSQABFA, sample assembler program for TSO and CMS (Part 2 of 4)

```
*********************************************************************
* Return                                                            *
*********************************************************************
         SPACE 1
         SR    R15,R15               SET RETURN CODE
         L     R13,4(R13)
         L     R14,12(R13)           RESTORE CALLER REGISTERS
         LM    R0,R12,20(R13)
         BR    R14
         EJECT
*********************************************************************
* Data Areas                                                        *
*********************************************************************
         SPACE 1
*  Query Interface commands
         SPACE 1
STARTQI  DC    C'START'              START FUNCTION
SETG     DC    C'SET GLOBAL'         SET GLOBAL FUNCTION
QUERY    DC    C'RUN QUERY Q1'       RUN QUERY
REPT     DC    C'PRINT REPORT (FORM=F1)'  PRINT REPORT
ENDQI    DC    C'EXIT'               END INTERFACE
         SPACE 1
         DS    0F
STARTQIL DC    AL4(L'STARTQI)        LENGTH OF START FUNCTION
SETGL    DC    AL4(L'SETG)           LENGTH OF SET GLOBAL FUNCTION
QUERYL   DC    AL4(L'QUERY)          LENGTH OF RUN QUERY COMMAND
REPTL    DC    AL4(L'REPT)           LENGTH OF PRINT REPORT COMMAND
ENDQIL   DC    AL4(L'ENDQI)          LENGTH OF END INTERFACE COMMAND
         SPACE 1
*  START command keyword
         SPACE 1
STARTKY  DC    C'DSQSMODE'
STARTV   DC    C'INTERACTIVE'
         DS    0F
STARTKYL DC    AL4(L'STARTKY)
STARTVL  DC    AL4(L'STARTV)
         SPACE 1
```

Figure 28. DSQABFA, sample assembler program for TSO and CMS (Part 3 of 4)

```
*  SET GLOBAL command variable names
         SPACE 1
VNAME1   DC    C'MYVAR01'
VNAME2   DC    C'SHORT'
VNAME3   DC    C'MYVAR03'
         DS    0F
VNAME1L  DC    AL4(L'VNAME1)
VNAME2L  DC    AL4(L'VNAME2)
VNAME3L  DC    AL4(L'VNAME3)
         SPACE 1
*  SET GLOBAL command values
         SPACE 1
VVAL1    DS    F
VVAL2    DS    F
VVAL3    DS    F
VVAL1L   DC    AL4(L'VVAL1)
VVAL2L   DC    AL4(L'VVAL2)
VVAL3L   DC    AL4(L'VVAL3)
         SPACE 1
NUMPARMS DS    F                     NUMBER OF KEYWORDS
         SPACE 1
*  callable interface communications area
         SPACE 1
CICOMM   DS    CL(DSQCOMM_LEN)
         SPACE 1
SAVEAREA DS    18F
         EJECT
         DSQCOMMA
         SPACE 1
R0       EQU   00                    EQUATES FOR REGISTERS 0-15
R1       EQU   01
R2       EQU   02
R3       EQU   03
R4       EQU   04
R5       EQU   05
R6       EQU   06
R7       EQU   07
R8       EQU   08
R9       EQU   09
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         SPACE 1
         END   DSQABFA
```

Figure 28. DSQABFA, sample assembler program for TSO and CMS (Part 4 of 4)

DSQCOMM for Assembler

This communications area changed between Version 2 Release 4 and Version 3.2. QMF ships this file as DSQCOMMA.

```
         MACRO                                                          00001000
         DSQCOMMA                                                       00002000
*********************************************************************** 00003000
*  Callable Interface - variable constants                          *  00004000
*********************************************************************** 00005000
*                                                                       00006000
*  Communications Level ID                                              00007000
*                                                                       00008000
DSQ_CURRENT_COMM_LEVEL   DC    CL12'DSQL>001002<'                       00009000
*                                                                       00010000
*  Query Product IDs                                                    00011000
*                                                                       00012000
DSQ_QRW            DC    C'01'                                         00013000
DSQ_QMF            DC    C'02'                                         00014000
DSQ_QM4            DC    C'03'                                         00015000
*                                                                       00016000
*  Query Product Release IDs                                            00017000
*                                                                       00018000
DSQ_QRW_V1R2       DC    C'01'                                         00019000
DSQ_QRW_V1R3       DC    C'02'                                         00020000
DSQ_QMF_V2R4       DC    C'01'                                         00021000
DSQ_QMF_V3R1       DC    C'02'                                         00022000
DSQ_QMF_V3R1M1     DC    C'03'                                         00023000
DSQ_QMF_V3R2       DC    C'04'                                         00024000
DSQ_QMF_V3R3       DC    C'05'                                         00025000
DSQ_QMF_V6R1       DC    C'06'                                         00026000
DSQ_QM4_V1R1       DC    C'01'                                         00027000
*                                                                       00028000
*  Extended parameter data types                                        00029000
*                                                                       00030000
DSQ_VARIABLE_CHAR  DC    C'CHAR'                                       00031000
DSQ_VARIABLE_FINT  DC    C'FINT'                                       00032000
*                                                                       00033000
*  Return codes                                                         00034000
*                                                                       00035000
DSQ_SUCCESS        EQU   0                                             00036000
DSQ_WARNING        EQU   4                                             00037000
DSQ_FAILURE        EQU   8                                             00038000
DSQ_SEVERE         EQU   16                                            00039000
*                                                                       00040000
*  Instance ID values                                                   00041000
*                                                                       00042000
DSQ_CONTINUE       EQU   0                                             00043000
*                                                                       00044000
```

Figure 29. DSQCOMMA, assembler communications area (Part 1 of 2)

```
*  Cancel indicator                                                      00045000
*                                                                        00046000
DSQ_CANCEL_YES     EQU   C'1'                                          00047000
DSQ_CANCEL_NO      EQU   C'0'                                          00048000
*                                                                        00049000
*                                                                        00050000
DSQ_INTERACTIVE    EQU   C'1'                                          00051000
DSQ_BATCH          EQU   C'2'                                          00052000
*                                                                        00053000
DSQ_YES            EQU   C'1'                                          00054000
DSQ_NO             EQU   C'2'                                          00055000
*                                                                        00056000
************************************************************************ 00057000
*  Callable Interface Communications Area                            *  00058000
************************************************************************ 00059000
DSQCOMM            DSECT                                               00060000
DSQ_RETURN_CODE    DS    F            FUNCTION RETURN CODE             00061000
DSQ_INSTANCE_ID    DS    F            ID ESTABLISHED IN START CMD      00062000
DSQ_COMM_LEVEL     DS    CL12         COMMUNICATIONS LEVEL ID          00063000
DSQ_PRODUCT        DS    CL2          QUERY PRODUCT ID                 00064000
DSQ_PRODUCT_RELEASE DS   CL2          QUERY PRODUCT RELEASE ID         00065000
DSQ_RESERVE1       DS    CL28         RESERVED                         00066000
DSQ_MESSAGE_ID     DS    CL8          COMPLETION MESSAGE ID            00067000
DSQ_Q_MESSAGE_ID   DS    CL8          QUERY MESSAGE ID                 00068000
DSQ_START_PARM_ERROR DS  CL8          START PARAMETER IN ERROR         00069000
DSQ_CANCEL_IND     DS    C            CMD CANCEL INDICATOR             00070000
DSQ_RESERVE2       DS    CL23         RESERVED                         00071000
DSQ_RESERVE3       DS    CL156        RESERVED                         00072000
DSQ_MESSAGE_TEXT   DS    CL128        COMPLETION MESSAGE               00073000
DSQ_Q_MESSAGE_TEXT DS    CL128        QUERY MESSAGE                    00074000
         SPACE 1                                                        00075000
DSQCOMM_LEN        EQU   *-DSQCOMM    LENGTH OF DSQCOMM AREA           00076000
         MEND                                                           00077000
```

Figure 29. DSQCOMMA, assembler communications area (Part 2 of 2)

Running Your Assembler programs in CICS

After you write your program, you need to translate, assemble, and link-edit it before you can run it. The examples listed in this section show the steps necessary to do so. QMF does not ship the REXX EXEC, JCL, or CLIST in these examples, but you can copy them from here, altering them to suit your installation.

Translating, assembling, and link-editing for CICS in MVS

When you translate, assemble, and link-edit a program that uses the QMF callable interface, be aware of the following:

- The communications area macro DSQCOMMA must be available to the assemble step or copied into your program as a DSECT.

- The QMF interface module DSQCIA must be made available during the link-edit phase of your program.

The following JCL shows an example of the CICS-supplied procedure DFHEBTAL. For instructions on how to use this procedure, see *CICS for VSE/ESA System Definition Guide*

```
//sampasm   JOB
//          EXEC PROC=DFHEBTAL
//TRN.SYSIN DD  *
*ASM      XOPTS(CICS translator options .....)
          .
          Your program or copy of QMF sample DSQABFA
          .
/*
//* Provide Access to QMF Communications Macro DSQCOMM
//ASM.SYSLIB   DD DSN=QMF720.SDSQSAPE,DISP=SHR
//* Provide Access to QMF Interface Module
//LKED.QMFLOAD DD DSN=QMF720.SDSQLOAD,DISP=SHR
//LKED.SYSIN   DD *
          INCLUDE CICSLOAD(DFHEAI)
          INCLUDE CICSLOAD(DFHEAI0)
          INCLUDE QMFLOAD(DSQCIA)
          ORDER DFHEAI,DFHEAI0
          ENTRY sampasm
          MODE  AMODE(31) RMODE(ANY)
          NAME  sampasm(R)
/*
```

Figure 30. JCL for running the CICS translator, assembler, and linkage editor

Translating, assembling, and link-editing for CICS in VSE

The following VSE job control is an example of installing an HLASM program into CICS running in VSE/ESA. This example is shipped with QMF and is located in the QMF sublibrary under the name of DSQ3CIAC.Z. See *CICS for VSE/ESA System Definition Guide* for more information.

If your installation is using HLASM, make sure that your system administrator has established a VSE library exit that handles the macro processing of E-Decks. This exit reads the DSQCOMMA. For a description of how to set up this exit, see *VSE Guide to System Functions* and *IBM High-Level Assembler Programmer's Guide for OS/390, VM and VSE* for detailed information.

Use the following HLASM compiler options to assemble the program:

```
'LIBMAC,USING(NOLIMIT,NOWARN),EXIT(LIBEXIT(EDECKXIT(ORDER=EA)))'
```

The LIBEXIT parameter includes CICS macro definitions created by the CICS translation process.

```
// JOB DSQ3CIAC
*  ---------------------------------------------------------------------
*  Install QMF Callable Interface (HLASM)
*  ---------------------------------------------------------------------
// SETPARM VOLID=volid      *-- update volid for syspch
// SETPARM START=rtrk       *-- update start track/block (syspch)
// SETPARM SIZE=ntrks       *-- update number of tracks/blocks (syspch)
*  ---------------------------------------------------------------------
// DLBL   IJSYSPH,'ASM.TRANSLATION',0
// EXTENT SYSPCH,,1,0,&START,&SIZE.
ASSGN SYSPCH,DISK,VOL=&VOLID.,SHR
*  Library search chain must contain the QMF, CICS and HLASM sublibrary
// LIBDEF *,SEARCH=(PRD2.PROD,PRD1.BASE,PRD2.CONFIG)
// LIBDEF PHASE,CATALOG=PRD2.PROD
*  ---------------------------------------------------------------------
*  Step 1: Translate Callable Interface program
*  ---------------------------------------------------------------------
*  You may need to update or remove the SLI statement for your program.
*  ---------------------------------------------------------------------
// EXEC   DFHEAP1$
* $$ SLI MEM=DSQABFAC.Z,S=PRD2.PROD
/*
```

Figure 31. Job control for running assembler and linkage editor in VSE (Part 1 of 2)

```
*  ---------------------------------------------------------------------
*  Step 2: Assemble Callable Interface program
*  ---------------------------------------------------------------------
CLOSE SYSPCH,00D
// DLBL   IJSYSIN,'ASM.TRANSLATION',0
// EXTENT SYSIPT
ASSGN SYSIPT,DISK,VOL=&VOLID.,SHR
// OPTION CATAL,DECK,SYM,ERRS
   PHASE DSQABFAC,*
         INCLUDE DFHEAI
         INCLUDE DFHEAI0
         INCLUDE DSQCIA
         INCLUDE DSQCLOD2
         INCLUDE DSQCMCVP
// EXEC ASMA90,SIZE=(ASMA90,50K),                                       C
               PARM='LIBMAC,USING(NOLIMIT,NOWARN),EXIT(LIBEXIT(EDECKXITC
               (ORDER=EA)))'
CLOSE SYSIPT,SYSRDR
/*
*  ---------------------------------------------------------------------
*  Step 3: Link-edit Callable Interface program
*  ---------------------------------------------------------------------
// EXEC LNKEDT,PARM='AMODE=31,RMODE=ANY'
/*
/&
// JOB  RESET
ASSGN SYSIPT,SYSRDR    IF 1A93D, CLOSE SYSIPT,SYSRDR
ASSGN SYSPCH,00D       IF 1A93D, CLOSE SYSPCH,00D
/&
```

Figure 31. Job control for running assembler and linkage editor in VSE (Part 2 of 2)

Assembling and running your programs under CMS in VM

The following sample program assembles and runs your callable interface application using the Assembler H compiler. QMF does not ship the REXX EXEC in this example, but you can copy it from here, altering it to suit your installation.

```
/*********************************************************************/
/* Assemble your program and execute it.          */
/*********************************************************************/
TRACE off
ADDRESS CMS

/*********************************************************************/
/* Assemble the program                                              */
/*********************************************************************/
"ERASE TEMPP MACLIB A"
"MACLIB GEN TEMPP DSQCOMMA"
Maclist = "TEMPP DMSSP CMSLIB OSMACRO"
"GLOBAL MACLIB" Maclist
"HASM yourname"

/*********************************************************************/
/* Access SQL/DS and initialize database                             */
/*********************************************************************/
"EXEC PRODUCT SQLDS"
"EXEC SQLINIT DBNAME(SQLDBA)"

/*********************************************************************/
/* Access GDDM product disk                                          */
/*********************************************************************/
"EXEC PRODUCT GDDM"

/*********************************************************************/
/* Issue Filedefs for QMF product                                    */
/*********************************************************************/
/* DEBUG = DDNAME FOR QMF DIAGNOSTICS OUTPUT     */
"FILEDEF DSQDEBUG PRINTER ( LRECL 80  BLKSIZE 80  RECFM FBA PERM"
/* PRINT = DDNAME FOR QMF PRINTED OUTPUT                          */
"FILEDEF DSQPRINT PRINTER ( LRECL 133 BLKSIZE 133 RECFM FBA PERM"
/* EDIT  = DDNAME FOR QMF EDIT TRANSFER FILE                      */
"FILEDEF DSQEDIT DISK QMFEDIT FILE A (PERM"
/* DSQSIDE = DDNAME FOR QMF SPILL FILE                            */
"FILEDEF DSQSPILL DISK DSQSIDE DATA A1 (PERM"
/* DSQPNLE = DDNAME FOR PANEL FILE                                */
"FILEDEF DSQPNLE DISK DSQPNLE FILE * (PERM"
"FILEDEF ISPLLIB CLEAR"
"FILEDEF ISPLLIB DISK DSQLDLIB LOADLIB *"
```

Figure 32. REXX program to assemble and run your program (Part 1 of 2)

```
/*****************************************************************/
/* Provide access to QMF and GDDM program libraries              */
/*****************************************************************/
"GLOBAL LOADLIB DSQLDLIB "
"GLOBAL TXTLIB ADMRLIB ADMPLIB ADMGLIB"

Say "Starting to execute 'ASSEMBLER' program"
ADDRESS CMS "RUN yourname"

Exit 0
```

Figure 32. REXX program to assemble and run your program (Part 2 of 2)

Running your Assembler programs in TSO

You must assemble and link-edit your program before you can run it in TSO. The following sections provide sample jobs that assemble and link-edit your programs and sample programs for running your assembled program in TSO either with or without ISPF.

Assembling and link-editing in TSO

The following sample job assembles and link-edits your program using Assembler H. Some parameters might vary from one installation to the next. See your QMF administrator for details.

```
//sampasm    JOB
//STEP1      EXEC PROC=ASMHCL
//* Provide Access to QMF Communications Macro DSQCOMM
//C.SYSLIB   DD  DSN=QMF720.SAMPLIB,DISP=SHR
//C.SYSIN    DD  *
          .
          Your program or copy of QMF sample DSQABFA
          .
/*
//* Provide Access to QMF Interface Module
//L.QMFLOAD  DD DSN=QMF720.SDSQLOAD,DISP=SHR
//L.SYSIN    DD  *
          INCLUDE QMFLOAD(DSQCIA)
          ENTRY sampasm
          MODE  AMODE(31) RMODE(ANY)
          NAME  sampasm(R)
/*
```

Figure 33. JCL for running assembler and linkage editor in TSO

After your program is assembled successfully, you can run it.

Running in TSO under ISPF

After your program is assembled successfully, you can run it under ISPF by writing a program similar to the following:

```
PROC 0
CONTROL ASIS
/*************************************************************/
/* Specify attribute list for dataset allocations            */
/*************************************************************/
ATTR PRINTDCB LRECL(133) RECFM(F B A) BLKSIZE(1330)
ATTR DEBUGDCB LRECL(80)  RECFM(F B)   BLKSIZE(3120)
ATTR UDUMPDCB LRECL(125) RECFM(V B A) BLKSIZE(1632)
ATTR EDITDCB  LRECL(79)  RECFM(F B A) BLKSIZE(4029)
/*************************************************************/
/* Datasets used by TSO                                      */
/*************************************************************/
ALLOC FI(SYSPROC) DA('QMF720.SDSQCLTE','ISR.ISRCLIB')
ALLOC FI(SYSEXEC) DA('QMF720.SDSQEXCE')
/*************************************************************/
/* Datasets used by ISPF                                     */
/*************************************************************/
ALLOC FI(ISPLLIB) SHR REUSE +

      DA('QMF720.SDSQLOAD','ADM.GDDMLOAD','DSN.DSNEXIT','DSN.DSNLOAD')
ALLOC FI(ISPMLIB) SHR REUSE +
      DA('QMF720.SDSQMLBE','ISR.ISRMLIB','ISP.ISPMLIB')
ALLOC FI(ISPPLIB) SHR REUSE +
      DA('QMF720.SDSQPLBE','ISR.ISRPLIB','ISP.ISPPLIB')
ALLOC FI(ISPSLIB) SHR REUSE +
      DA('QMF720.SDSQSLBE','ISR.ISRSLIB','ISP.ISPSLIB')
ALLOC FI(ISPTLIB) SHR REUSE +
      DA('ISR.ISRTLIB','ISP.ISPTLIB')
/*************************************************************/
/* QMF/GDDM Datasets                                         */
/*************************************************************/
ALLOC FI(ADMGGMAP) DA('QMF720.QMFMAPS')  SHR REUSE
ALLOC FI(ADMCFORM) DA('QMF720.DSQCFORM') SHR REUSE
ALLOC FI(DSQUCFRM) DA('QMF720.DSQUCFRM') SHR REUSE
ALLOC FI(ADMSYMBL) DA('ADM.GDDMSYM')  SHR REUSE
ALLOC FI(ADMGDF)   DA('ADM.GDDM.CHARTLIB') SHR REUSE
ALLOC FI(ADMDEFS)  DA('ADM.GDDM.NICKNAME') SHR REUSE
/*************************************************************/
/* Datasets used by QMF                                      */
/*************************************************************/
ALLOC FI(DSQPRINT) SYSOUT(X) USING(PRINTDCB)
ALLOC FI(DSQDEBUG) SYSOUT(X) USING(DEBUGDCB)
ALLOC FI(DSQUDUMP) SYSOUT(X) USING(UDUMPDCB)
ALLOC FI(DSQSPILL) NEW UNIT(SYSDA) SPACE(1,1) TRACKS
ALLOC FI(DSQEDIT)  NEW UNIT(SYSDA) USING(EDITDCB)
ALLOC FI(DSQPNLE)  DA('QMF720.DSQPNLE') SHR REUSE
/*************************************************************/
/* Start your program as the initial ISPF dialog             */
/*************************************************************/
ISPSTART PGM(sampasm) NEWAPPL(DSQE)
EXIT CODE(4)
```

Figure 34. CLIST for running your program in TSO under ISPF

The EXIT CODE(4) suppresses the ISPF disposition panel.

Running in TSO without ISPF

After your program is assembled successfully, you can run it in TSO without ISPF by writing a program similar to the following:

```
PROC 0
CONTROL ASIS
/*************************************************************/
/* Note: QMF, DB2 and GDDM load libraries must be allocated */
/*       before executing this CLIST.                        */
/*       Name of QMF load library is  "QMF720.SDSQLOAD".     */
/*************************************************************/
/* Specify attribute list for dataset allocations            */
/*************************************************************/
ATTR PRINTDCB LRECL(133) RECFM(F B A) BLKSIZE(1330)
ATTR DEBUGDCB LRECL(80)  RECFM(F B)   BLKSIZE(3120)
ATTR UDUMPDCB LRECL(125) RECFM(V B A) BLKSIZE(1632)
ATTR EDITDCB  LRECL(79)  RECFM(F B A) BLKSIZE(4029)
/*************************************************************/
/* Datasets used by TSO                                      */
/*************************************************************/
ALLOC FI(SYSPROC) DA('QMF720.SDSQCLTE')
ALLOC FI(SYSEXEC) DA('QMF720.SDSQEXCE')
/*************************************************************/
/* QMF/GDDM Datasets                                         */
/*************************************************************/
ALLOC FI(ADMGGMAP) DA('QMF720.QMFMAPS')  SHR REUSE
ALLOC FI(ADMCFORM) DA('QMF720.DSQCFORM') SHR REUSE
ALLOC FI(DSQUCFRM) DA('QMF720.DSQUCFRM') SHR REUSE
ALLOC FI(ADMSYMBL) DA('ADM.GDDMSYM')  SHR REUSE
ALLOC FI(ADMGDF)   DA('ADM.GDDM.CHARTLIB') SHR REUSE
ALLOC FI(ADMDEFS)  DA('ADM.GDDM.NICKNAME') SHR REUSE
/*************************************************************/
/* Datasets used by QMF                                      */
/*************************************************************/
ALLOC FI(DSQPRINT) SYSOUT(X) USING(PRINTDCB)
ALLOC FI(DSQDEBUG) SYSOUT(X) USING(DEBUGDCB)
ALLOC FI(DSQUDUMP) SYSOUT(X) USING(UDUMPDCB)
ALLOC FI(DSQSPILL) NEW UNIT(SYSDA) SPACE(1,1) TRACKS
ALLOC FI(DSQEDIT)  NEW UNIT(SYSDA) USING(EDITDCB)
ALLOC FI(DSQPNLE)  DA('QMF720.DSQPNLE') SHR REUSE
/*************************************************************/
/* Start your program using TSO CALL command                */
/*************************************************************/
CALL sampasm
EXIT CODE(0)
```

Figure 35. CLIST for running your program under TSO without ISPF

C Language Interface

The C Language callable interface provided here corresponds to that provided for other SAA languages.

For the C language, QMF provides a DSQCOMMC communications macro and two function calls, DSQCICE and DSQCIC.

Note: To access QMF from a C++ application, you need an interface written in C.

Interface communications area mapping for C language (DSQCOMMC)

DSQCOMMC provides DSQCOMM mapping for C language and is shipped with the product. Table 16 shows the values for DSQCOMMC.

Table 16. Interface communications area for DSQCOMMC

Structure Name	Data Type	Description
DSQ_RETURN_CODE	signed long integer	Indicates the status of a QMF command after it has been run. Its values are: **DSQ_SUCCESS** Successful execution of the request. **DSQ_WARNING** Normal completion with warnings. **DSQ_FAILURE** Command did not run correctly. **DSQ_SEVERE** Severe error; QMF session terminated.
DSQ_INSTANCE_ID	signed long integer	Identifier established by QMF during execution of the START command
DSQ_COMM_LEVEL	character, length 12	Identifies the level of the DSQCOMM. You should set this to the value of DSQ_CURRENT_COMM_LEVEL before issuing the QMF START command.
DSQ_PRODUCT	character, length 2	Identifies the IBM query product in use.
DSQ_PRODUCT_RELEASE	character, length 2	Identifies the release level of the query product in use.
DSQ_RESERVE1	character, length 28	Reserved for future use
DSQ_MESSAGE_ID	character, length 8	Completion message ID
DSQ_Q_MESSAGE_ID	character, length 8	Query message ID
DSQ_START_PARM_ERROR	character, length 8	Parameter in error when START failed due to a parameter error
DSQ_CANCEL_IND	character, length 1	Contains one of two values, depending if the user canceled while a QMF command was running: • DSQ_CANCEL_YES • DSQ_CANCEL_NO
DSQ_RESERVE2	character, length 23	Reserved for future use

Table 16. Interface communications area for DSQCOMMC (continued)

Structure Name	Data Type	Description
DSQ_RESERVE3	character, length 156	Reserved for future use
DSQ_MESSAGE_TEXT	character, length 128	Completion message text
DSQ_Q_MESSAGE_TEXT	character, length 128	Query message text

Function calls for the C language

QMF provides two function calls for the C language: DSQCIC and DSQCICE.

DSQCIC

This call is for QMF commands that do not require access to application program variables. Use this call for most QMF commands.

```
DSQCIC (&DSQCOMM,&CMDLTH,&CMDSTR)
```

The parameters have the following values:

DSQCOMM
The interface communications area

CMDLTH
Length of the command string, CMDSTR; a long type parameter

CMDSTR
QMF command to run, specified as an array of unsigned character type of the length specified by CMDLTH. The QMF command must be uppercase.

DSQCICE

This call has an extended syntax for the three QMF commands that do require access to application program variables: START and the extended formats of GET GLOBAL and SET GLOBAL.

```
DSQCICE (&DSQCOMM,&CMDLTH,&CMDSTR,
         &PNUM,&KLTH,&KWORD,
         &VLTH,&VALUE,&VTYPE);
```

The parameters have the following values:

DSQCOMM
The interface communications area.

CMDLTH
Length of the command string, CMDSTR; a long integer parameter.

CMDSTR
QMF command to run. It is an array of unsigned character type. The QMF command must be uppercase.

PNUM
Number of command keywords. It is a long integer parameter.

KLTH
Length of each specified keyword, &KWORD. It is a long integer parameter or an array of long integer parameters.

KWORD
QMF keyword or keywords. Each is an unsigned character array.

VLTH
Length of each value associated with the keyword; a long integer parameter or array of long integer parameters

VALUE
Value associated with each keyword. Its type is specified in the VTYPE parameter, and can be an unsigned character array, a long integer parameter, or array of long integer parameters.

VTYPE
Data type of the value string VALUE. This type has one of two values, which are provided in the communications macro, DSQCOMMC:

- DSQ_VARIABLE_CHAR for unsigned character type
- DSQ_VARIABLE_FINT for long integer

All of the values specified in the VALUE field must have the data type specified in VTYPE.

The C language interface is similar to the others. There are, however, the following parameter considerations:

- Command strings, START, GET, and SET command parameters are all input character strings. For these, C requires you to pass a storage area that is terminated with a null value, which must be included in the parameter's length. The compile-time length function should be used to obtain the parameter length that is passed to the QMF interface.
- If the string is not terminated by a null before reaching the end of the string, an error is returned by QMF. The null value (X'00') indicates the end of a character string.
- For C parameters that are output character strings, including values obtained by the GET command, QMF moves data from QMF storage to the application's storage area and sets the null indicator at the end of the string. If the character string does not fit in the user's storage area, a warning message is issued and the data is truncated on the right. A null indicator is always placed at the end of the data string.

Migration information

The DSQCOMM changed from Version 2 Release 4 to Version 3.2.

- If you want to continue using the old DSQCOMM, you do not have to recompile your program.
- If you want to use the Version 3.2 version of DSQCICX, you must link-edit your Version 2 Release 4 program again.

However, the Version 3.2 DSQCOMM provides message text that is especially useful if you have an error in your START command. If you want to use the new DSQCOMM, you need to recompile your program, and initialize DSQ_COMM_LEVEL (in DSQCOMM) to DSQ_CURRENT_COMM_LEVEL. If you do *not* set this value, QMF treats your DSQCOMM as a Version 2 Release 4 level.

Note to CICS users in MVS

The DSQCICX changed from Version 3 Release 1 Modification 1 to Version 3 Release 2. The interface between the QMF-supplied function call and the main QMF program has changed from a CALL interface to an EXEC CICS LINK interface. The new interface provides better isolation from the user program and the QMF product. Because the interface has changed, programs that used the callable interface must be link-edited again.

C language programming example

This example shows the SAA callable interface for IBM C language.

The following program, DSQABFC, is shipped with the QMF product. You can look at the sample source code listing here or you can access it online.

- For OS/390, the sample program is a member of the library QMF720.SDSQSAPE.
- For VM, the sample program is on the production disk.
- For VSE, the sample program is in the QMF sublibrary and named DSQABFC.Z.

The sample program for the C language callable interface performs the following function:

- Starts QMF
- Sets three global variables
- Runs a query called Q1
- Prints the resulting report using form F1
- Ends the QMF session

QMF does not supply query Q1 or form F1, but the sample program uses these objects.

This section also shows how to compile, link-edit, and run a C language program that uses the callable interface. QMF does not ship the REXX EXEC, JCL, or CLIST in these examples, but you can copy them from here, altering them to suit your installation.

```
/*********************************************************************/
/* Sample Program:  DSQABFC                                          */
/* C Version of the SAA Query Callable Interface                     */
/*********************************************************************/

/*********************************************************************/
/* Include standard and string "C" functions                         */
/*********************************************************************/
#include <string.h>
#include <stdlib.h>

/*********************************************************************/
/* Include and declare query interface communications area           */
/*********************************************************************/
#include <DSQCOMMC.H>

int main()
     {

 struct dsqcomm communication_area;      /* DSQCOMM from include   */

 /*********************************************************************/
 /* Query interface command length and commands                       */
 /*********************************************************************/
 signed long command_length;
 static char start_query_interface[] = "START";
 static char set_global_variables[] = "SET GLOBAL";
 static char run_query[] = "RUN QUERY Q1";
 static char print_report[] = "PRINT REPORT (FORM=F1";
 static char end_query_interface[] = "EXIT";

 /*********************************************************************/
 /* Query command extension, number of parameters and lengths         */
 /*********************************************************************/
 signed long number_of_parameters;    /* number of variables       */
 signed long keyword_lengths[10];     /* lengths of keyword names  */
 signed long data_lengths[10];        /* lengths of variable data  */
```

Figure 36. DSQABFC, sample C program (Part 1 of 3)

```
/*****************************************************************/
/* Variable data type constants                                  */
/*****************************************************************/
static char char_data_type[] = DSQ_VARIABLE_CHAR;
static char int_data_type[]  = DSQ_VARIABLE_FINT;

/*****************************************************************/
/* Keyword parameter and value for START command                 */
/*****************************************************************/
static char start_keywords[] = "DSQSMODE";
static char start_keyword_values[] = "INTERACTIVE";
/*****************************************************************/
/* Keyword parameter and values for SET command                  */
/*****************************************************************/
#define SIZE_VAL 8
char set_keywords [3][SIZE_VAL];   /* Parameter name array       */
signed long set_values[3];         /* Parameter value array      */

/*****************************************************************/
/* MAIN PROGRAM                                                  */
/*****************************************************************/

/*****************************************************************/
/* Start a Query Interface Session                               */
/*****************************************************************/
    strncpy (communication_area.dsq_comm_level,
             DSQ_CURRENT_COMM_LEVEL,
             sizeof(communication_area.dsq_comm_level));
    number_of_parameters = 1;
    command_length = sizeof(start_query_interface);
    keyword_lengths[0] = sizeof(start_keywords);
    data_lengths[0] = sizeof(start_keyword_values);
    dsqcice(&communication_area,;
            &command_length,;
            &start_query_interface[0],
            &number_of_parameters,;
            &keyword_lengths[0],
            &start_keywords[0],
            &data_lengths[0],
            &start_keyword_values[0],
            &char_data_type[0]);
```

Figure 36. DSQABFC, sample C program (Part 2 of 3)

```
  /*****************************************************************/
  /* Set numeric values into query using SET command               */
  /*****************************************************************/
      number_of_parameters = 3;
      command_length = sizeof(set_global_variables);
      strcpy(set_keywords[0],"MYVAR01");
      strcpy(set_keywords[1],"SHORT");
      strcpy(set_keywords[2],"MYVAR03");
      keyword_lengths[0] = SIZE_VAL;
      keyword_lengths[1] = SIZE_VAL;
      keyword_lengths[2] = SIZE_VAL;
      data_lengths[0] = sizeof(long);
      data_lengths[1] = sizeof(long);
      data_lengths[2] = sizeof(long);
      set_values[0] = 20;
      set_values[1] = 40;
      set_values[2] = 84;
      dsqcice(&communication_area,;
              &command_length,;
              &set_global_variables[0],
              &number_of_parameters,;
              &keyword_lengths[0],
              &set_keywords[0][0],
              &data_lengths[0],
              &set_values[0],
              &int_data_type[0]);

  /*****************************************************************/
  /* Run a Query                                                   */
  /*****************************************************************/
      command_length = sizeof(run_query);
      dsqcic(&communication_area,&command_length,;
                  &run_query[0]);

  /*****************************************************************/
  /* Print the results of the query                                */
  /*****************************************************************/
      command_length = sizeof(print_report);
      dsqcic(&communication_area,&command_length,;
                  &print_report[0]);

  /*****************************************************************/
  /* End the query interface session                               */
  /*****************************************************************/
      command_length = sizeof(end_query_interface);
      dsqcic(&communication_area,&command_length,;
                  &end_query_interface[0]);
      exit(0);
  }
```

Figure 36. DSQABFC, sample C program (Part 3 of 3)

DSQCOMM for C

This include file, DSQCOMMC, is shipped with the QMF product.

```
/*****************************************************************/
/* C Include for Query Callable Interface (MVS/VM)               */
/*****************************************************************/

/* Structure declare for Communications Area                     */

struct dsqcomm {
        long int dsq_return_code;   /* Function return code      */
        long int dsq_instance_id;   /* id established in start cmd*/
        char dsq_comm_level[12];    /* communications level id   */
        char dsq_product[2];        /* query product id          */
        char dsq_product_release[2]; /* query product release    */
        char dsq_reserve1[28];      /* reserved                  */
        char dsq_message_id[8];     /* completion message ID     */
        char dsq_q_message_id[8];   /* query message ID          */
        char dsq_start_parm_error[8]; /* start parameter in error */
        char dsq_cancel_ind[1];     /* cmd cancelled indicator   */
                            /* 1 = cancelled, 0 = not cancelled*/
        char dsq_reserve2[23];      /* RESERVED AREAS            */
        char dsq_reserve3[156];
        char dsq_message_text[128]; /* Message text              */
        char dsq_q_message_text[128]; /* Query message text      */
     } ;

/* RETURN CODES                                                  */

#define DSQ_SUCCESS                 0
#define DSQ_WARNING                 4
#define DSQ_FAILURE                 8
#define DSQ_SEVERE                 16

/* Communications Level                                          */

#define DSQ_CURRENT_COMM_LEVEL    "DSQL>001002<"

/* Query Product Codes                                           */

#define DSQ_QRW                 "01"
#define DSQ_QMF                 "02"
#define DSQ_QM3                 "03"

/* Query Product Release Levels                                  */

#define DSQ_QRW_V1R2            "01"
#define DSQ_QRW_V1R3            "02"
#define DSQ_QMF_V2R4            "01"
#define DSQ_QMF_V3R1            "02"
#define DSQ_QMF_V3R1M1          "03"
#define DSQ_QMF_V3R2            "04"
#define DSQ_QMF_V3R3            "05"
#define DSQ_QMF_V6R1            "06"
#define DSQ_QM4_V1R1            "01"
```

Figure 37. DSQCOMMC, C communications area (Part 1 of 2)

```
/* INSTANCE CODES                                                  */

#define DSQ_CONTINUE                0

/* CANCELLED INDICATOR                                             */

#define DSQ_CANCEL_YES             "1"
#define DSQ_CANCEL_NO              "0"

/* VARIABLE TYPES                                                  */

#define DSQ_VARIABLE_CHAR        "CHAR"
#define DSQ_VARIABLE_FINT        "FINT"

#define DSQ_INTERACTIVE          "1"
#define DSQ_BATCH                "2"

#define DSQ_YES                  "1"
#define DSQ_NO                   "2"

/* Call Interface structure                                        */

/* Calling format for normal call with 3 parameters                */
#define dsqcic(parm1, parm2, parm3 )\
               dsqcicx( parm1, parm2, parm3)

/* Calling format for call with CMD_EXT area 9 parameters          */
#define dsqcice(parm1, parm2, parm3,\
               parm4, parm5, parm6, parm7, parm8, parm9 )\
               dsqcicx( parm1, parm2, parm3,    \
               parm4, parm5, parm6,  \
               parm7, parm8, parm9 )

/* DECLARE OS LINKAGE FORMAT                                       */

#pragma linkage(dsqcicx, OS)
```

Figure 37. DSQCOMMC, C communications area (Part 2 of 2)

Running your programs in CICS

After writing a program, you need to translate, compile, and link-edit it before you can run it. The examples in this section show the necessary steps. QMF does not ship the REXX EXEC, JCL, or CLIST in these examples, but you can copy them from here, altering them to suit your installation.

When you translate, compile, and link-edit a program that uses the QMF callable interface under CICS, consider the following:

- The communications area macro DSQCOMMC must be available to the compile step or copied into your program.
- The QMF interface module DSQCICX must be available during the link-edit phase of your program.

Translating, compiling, and link-editing for CICS in MVS

The following example uses the CICS-supplied procedure DFHEBTDL. For instructions on how to use this procedure, see *CICS for VSE/ESA System Definition Guide*

```
//sampleC   JOB
//          EXEC PROC=DFHEBTDL
//TRN.SYSIN DD  *
#pragma   XOPTS(CICS translator options .....)
          .
          Your program or copy of QMF sample DSQABFC
          .
/*
//* Provide Access to QMF Communications Macro DSQCOMMC
//ASM.SYSLIB   DD DSN=QMF720.SDSQSAPE,DISP=SHR
//* Provide Access to QMF Interface Module
//LKED.QMFLOAD DD DSN=QMF720.SDSQLOAD,DISP=SHR
//LKED.SYSIN   DD *
          INCLUDE CICSLOAD(DFHELII)
          INCLUDE QMFLOAD(DSQCICX)
          ORDER DFHELII
          ENTRY sampleC
          MODE  AMODE(31) RMODE(ANY)
          NAME  sampleC(R)
/*
```

Figure 38. JCL for running the CICS translator, C compiler, and linkage editor

C/370 language programs must be link-edited with AMODE=31.

Translating, compiling, and link-editing for CICS in VSE

During the C/370 pre-link step, the IBM-supplied interface objects (DSQCICX.OBJ, DSQCLOD2.OBJ, and DSQCMCVP.OBJ) located in sublibrary PRD2.PROD (the QMF default install sublibrary) must be available in the `LIBDEF *` search chain.

During the link-edit step, the CICS assembler interface DFHEAI0 must be in the `LIBDEF *` search chain, as shown in Figure 39 on page 161.

This sample job control is stored in PRD2.PROD as DSQ3CIC.Z.

```
// JOB DSQ3CIC    Sample job to Install QMF Callable Interface (C/370)
*  ------------------------------------------------------------------
*  Install QMF Callable Interface Example (C/370)
*  ------------------------------------------------------------------
// SETPARM VOLID=volid         *-- update volid for syspch
// SETPARM START=rtrk          *-- update start track/block
// SETPARM SIZE=ntrks          *-- update number of tracks/blocks
// SETPARM VOLID2=volid2       *-- update volid for work area
// SETPARM START2=rtrk         *-- update start track/block
// SETPARM SIZE2=ntrks         *-- update number of tracks/blocks
*  ------------------------------------------------------------------
// DLBL   IJSYSPH,'c.translation',0
// EXTENT SYSPCH,,1,0,&START,&SIZE
ASSGN SYSPCH,DISK,VOL=&VOLID,SHR
*  Library search chain must contain the QMF, CICS and C/370 sublibrary
// LIBDEF *,SEARCH=(PRD2.PROD,PRD1.BASE,PRD2.CONFIG)
// LIBDEF PHASE,CATALOG=PRD2.PROD
*  ------------------------------------------------------------------
*  Step 1: Translate callable interface program (C/370)
*  ------------------------------------------------------------------
*  You may need to update or remove the SLI statement for your program.
*  ------------------------------------------------------------------
// EXEC   DFHEDP1$,SIZE=256K
..*  $$ SLI MEM=DSQABFC.Z,S=PRD2.QMFD
/*
CLOSE SYSPCH,00D
*  ------------------------------------------------------------------
*  Step 2: Compile callable interface program (C/370)
*  ------------------------------------------------------------------
// DLBL   IJSYSIN,'c.translation',0
// EXTENT SYSIPT
ASSGN SYSIPT,DISK,VOL=&VOLID,SHR
// DLBL   IJSYSPH,'compiler.output',0
// EXTENT SYSPCH,,1,0,&START2,&SIZE2
ASSGN SYSPCH,DISK,VOL=&VOLID2,SHR
// OPTION DECK
// EXEC EDCCOMP,SIZE=EDCCOMP,PARM='RENT'
CLOSE SYSIPT,SYSRDR
CLOSE SYSPCH,00D
*  ------------------------------------------------------------------
*  Step 3: Pre-link callable interface program (C/370)
*  ------------------------------------------------------------------
// DLBL   IJSYSIN,'compiler.output',0
// EXTENT SYSIPT
ASSGN SYSIPT,DISK,VOL=&VOLID2,SHR
// OPTION CATAL,NODECK
   PHASE DSQABFC,*
         INCLUDE DFHELII
         INCLUDE DFHEAI0
// EXEC EDCPRLK,SIZE=EDCPRLK
CLOSE SYSIPT,SYSRDR
/*
```

Figure 39. Job control to run the CICS/VSE translator, C compiler, and linkage editor (Part 1 of 2)

```
*  ---------------------------------------------------------------------
*  Step 4: Link-edit callable interface program (C/370)
*  ---------------------------------------------------------------------
// EXEC LNKEDT,PARM='AMODE=24,RMODE=24'
/*
/&
// JOB  RESET
ASSGN SYSIPT,SYSRDR    IF 1A93D, CLOSE SYSIPT,SYSRDR
ASSGN SYSPCH,00D       IF 1A93D, CLOSE SYSPCH,00D
/&
```

Figure 39. Job control to run the CICS/VSE translator, C compiler, and linkage editor (Part 2 of 2)

Compiling and running your programs under CMS in VM

The following program compiles and runs your callable interface application using the IBM C compiler.

QMF does not ship the REXX EXEC in this example, but you can copy it from here, altering it to suit your installation.

```
/*********************************************************************/
/* Compile your program and run it.                                  */
/*********************************************************************/
TRACE off
ADDRESS CMS
/*********************************************************************/
/* Access C product disk using an exec, PRODUCT, that you write.  */
/*********************************************************************/
EXEC PRODUCT ADC370
/*********************************************************************/
/* Compile the program                                               */
/*********************************************************************/
"GLOBAL TXTLIB IBMLIB SCEELKED"
"GLOBAL LOADLIB EDCLINK SCREERUN"
"CC" PNAME "(SOURCE SHOWINC"
/*********************************************************************/
/* Create an executable "C" module file                              */
/*********************************************************************/
"GLOBAL LOADLIB DSQLDLIB SCREERUN"
"GLOBAL TXTLIB EDCBASE ADMRLIB ADMPLIB ADMGLIB"
"CMOD yourname DSQCICX DSQCLOD2 DSQCMCVP"
/*********************************************************************/
/* Access SQL/DS and initialize database                             */
/*********************************************************************/
"EXEC PRODUCT SQLDS"
"EXEC SQLINIT DBNAME(SQLDBA)"
/*********************************************************************/
/* Access GDDM product disk                                          */
/*********************************************************************/
"EXEC PRODUCT GDDM"
/*********************************************************************/
/* Issue Filedefs for QMF product                                    */
/*********************************************************************/
/* DEBUG = DDNAME FOR QMF DIAGNOSTICS OUTPUT     */
"FILEDEF DSQDEBUG PRINTER ( LRECL 80  BLKSIZE 80  RECFM FBA PERM"
/* PRINT = DDNAME FOR QMF PRINTED OUTPUT                          */
"FILEDEF DSQPRINT PRINTER ( LRECL 133 BLKSIZE 133 RECFM FBA PERM"
/* EDIT  = DDNAME FOR QMF EDIT TRANSFER FILE                      */
"FILEDEF DSQEDIT DISK QMFEDIT FILE A (PERM"
/* DSQSIDE = DDNAME FOR QMF SPILL FILE                            */
"FILEDEF DSQSPILL DISK DSQSIDE DATA A1 (PERM"
/* DSQPNLE = DDNAME FOR PANEL FILE                                */
"FILEDEF DSQPNLE DISK DSQPNLE FILE * (PERM"
"FILEDEF ISPLLIB CLEAR"
"FILEDEF ISPLLIB DISK DSQLDLIB LOADLIB *"
/*********************************************************************/
/* Provide access to QMF and C program libraries                     */
/*********************************************************************/
"GLOBAL LOADLIB DSQLDLIB SCREERUN"
"GLOBAL TXTLIB EDCBASE ADMRLIB ADMPLIB ADMGLIB"
Say "Starting to run 'C' program"
"yourname"

Exit 0
```

Figure 40. REXX program to compile and run your program

You might have to modify this program to suit your installation.

Running your C programs in TSO

The following sections provide sample jobs for compiling and link-editing your callable interface application and sample programs for running your compiled programs either with or without ISPF.

Compiling and link-editing in TSO

The following job compiles and link-edits your callable interface application using the IBM C compiler for MVS. Some parameters might vary from one installation to the next. See your QMF administrator for details.

```
//sampleC    JOB
//STEP1      EXEC PROC=EDCCL,LPARM='MAP'
//* Provide Access to QMF Communications Macro DSQCOMM
//COMPILE.SYSLIB   DD  DSN=QMF720.SAMPLIB,DISP=SHR
//COMPILE.SYSIN    DD  DATA,DLM='<>'
          .
          Your program or copy of QMF sample DSQABFC
          .
<>
//* Provide Access to QMF Interface Module DSQCICX
//LKED.SYSLIB   DD DSN=QMF720.SDSQLOAD,DISP=SHR
/*
```

Figure 41. JCL for running the C compiler and linkage editor in TSO

Running your programs in TSO without ISPF

After your program has been compiled successfully, you can write a program similar to the following to run it:

```
PROC 0
CONTROL ASIS
/*************************************************************/
/* Note: QMF, DB2, GDDM and C load libraries must be         */
/*       allocated before running this CLIST.                */
/*       Name of QMF load library is  "QMF710.SDSQLOAD".     */
/*************************************************************/
/* Specify attribute list for dataset allocations            */
/*************************************************************/
ATTR PRINTDCB LRECL(133) RECFM(F B A) BLKSIZE(1330)
ATTR DEBUGDCB LRECL(80)  RECFM(F B)   BLKSIZE(3120)
ATTR UDUMPDCB LRECL(125) RECFM(V B A) BLKSIZE(1632)
ATTR EDITDCB  LRECL(79)  RECFM(F B A) BLKSIZE(4029)
/*************************************************************/
/* Datasets used by TSO                                      */
/*************************************************************/
ALLOC FI(SYSPROC) DA('QMF720.SDSQCLTE')
ALLOC FI(SYSEXEC) DA('QMF720.SDSQEXCE')
```

Figure 42. CLIST for running your program in TSO without ISPF (Part 1 of 2)

```
/*************************************************************/
/* QMF/GDDM Datasets                                         */
/*************************************************************/
ALLOC FI(ADMGGMAP) DA('QMF720.QMFMAPS')  SHR REUSE
ALLOC FI(ADMCFORM) DA('QMF720.DSQCFORM') SHR REUSE
ALLOC FI(DSQUCFRM) DA('QMF720.DSQUCFRM') SHR REUSE
ALLOC FI(ADMSYMBL) DA('ADM.GDDMSYM')  SHR REUSE
ALLOC FI(ADMGDF)   DA('ADM.GDDM.CHARTLIB') SHR REUSE
ALLOC FI(ADMDEFS)  DA('ADM.GDDM.NICKNAME') SHR REUSE
/*************************************************************/
/* Datasets used by QMF                                      */
/*************************************************************/
ALLOC FI(DSQPRINT) SYSOUT(X) USING(PRINTDCB)
ALLOC FI(DSQDEBUG) SYSOUT(X) USING(DEBUGDCB)
ALLOC FI(DSQUDUMP) SYSOUT(X) USING(UDUMPDCB)
ALLOC FI(DSQSPILL) NEW UNIT(SYSDA) SPACE(1,1) TRACKS
ALLOC FI(DSQEDIT)  NEW UNIT(SYSDA) USING(EDITDCB)
ALLOC FI(DSQPNLE)  DA('QMF720.DSQPNLE') SHR REUSE
/*************************************************************/
/* Start your program using TSO CALL command                */
/*************************************************************/
CALL sampleC
EXIT CODE(0)
```

Figure 42. CLIST for running your program in TSO without ISPF (Part 2 of 2)

Running your programs in TSO under ISPF

After your program has been compiled successfully, you can write a program similar to the following to run it:

```
PROC 0
CONTROL ASIS
/************************************************************/
/* Specify attribute list for dataset allocations           */
/************************************************************/
ATTR PRINTDCB LRECL(133) RECFM(F B A) BLKSIZE(1330)
ATTR DEBUGDCB LRECL(80)  RECFM(F B)   BLKSIZE(3120)
ATTR UDUMPDCB LRECL(125) RECFM(V B A) BLKSIZE(1632)
ATTR EDITDCB  LRECL(79)  RECFM(F B A) BLKSIZE(4029)
/************************************************************/
/* Datasets used by TSO                                     */
/************************************************************/
ALLOC FI(SYSPROC) DA('QMF720.SDSQCLTE','ISR.ISRCLIB')
ALLOC FI(SYSEXEC) DA('QMF720.SDSQEXCE')
```

Figure 43. CLIST for running DSQABFC in TSO under ISPF (Part 1 of 2)

```
/*************************************************************/
/* Datasets used by ISPF                                     */
/*************************************************************/
ALLOC FI(ISPLLIB) SHR REUSE +

      DA('QMF720.SDSQLOAD','ADM.GDDMLOAD','DSN.DSNEXIT','DSN.DSNLOAD', +
         'EDC.SEDCLINK','PLI.SIBMLINK')
ALLOC FI(ISPMLIB) SHR REUSE +
      DA('QMF720.SDSQMLBE','ISR.ISRMLIB','ISP.ISPMLIB')
ALLOC FI(ISPPLIB) SHR REUSE +
      DA('QMF720.SDSQPLBE','ISR.ISRPLIB','ISP.ISPPLIB')
ALLOC FI(ISPSLIB) SHR REUSE +
      DA('QMF720.SDSQSLBE','ISR.ISRSLIB','ISP.ISPSLIB')
ALLOC FI(ISPTLIB) SHR REUSE +
      DA('ISR.ISRTLIB','ISP.ISPTLIB')
/*************************************************************/
/* QMF/GDDM Datasets                                         */
/*************************************************************/
ALLOC FI(ADMGGMAP) DA('QMF720.QMFMAPS')  SHR REUSE
ALLOC FI(ADMCFORM) DA('QMF720.DSQCFORM') SHR REUSE
ALLOC FI(DSQUCFRM) DA('QMF720.DSQUCFRM') SHR REUSE
ALLOC FI(ADMSYMBL) DA('ADM.GDDMSYM')  SHR REUSE
ALLOC FI(ADMGDF)   DA('ADM.GDDM.CHARTLIB') SHR REUSE
ALLOC FI(ADMDEFS)  DA('ADM.GDDM.NICKNAME') SHR REUSE
/*************************************************************/
/* Datasets used by QMF                                      */
/*************************************************************/
ALLOC FI(DSQPRINT) SYSOUT(X) USING(PRINTDCB)
ALLOC FI(DSQDEBUG) SYSOUT(X) USING(DEBUGDCB)
ALLOC FI(DSQUDUMP) SYSOUT(X) USING(UDUMPDCB)
ALLOC FI(DSQSPILL) NEW UNIT(SYSDA) SPACE(1,1) TRACKS
ALLOC FI(DSQEDIT)  NEW UNIT(SYSDA) USING(EDITDCB)
ALLOC FI(DSQPNLE)  DA('QMF720.DSQPNLE') SHR REUSE
/*************************************************************/
/* Start your program as the initial ISPF dialog             */
/*************************************************************/
ISPSTART PGM(sampleC) NEWAPPL(DSQE)
EXIT CODE(4)
```

Figure 43. CLIST for running DSQABFC in TSO under ISPF (Part 2 of 2)

The EXIT CODE(4) suppresses the ISPF disposition panel.

COBOL language interface

The COBOL callable interface provided here corresponds to that provided for other SAA languages.

COBOL Language Interface

To write callable interface programs in COBOL, you must use VS COBOL II, COBOL/370, IBM COBOL for MVS and VM, or IBM COBOL for VSE. [3]

Interface communications area mapping for COBOL (DSQCOMMB)

DSQCOMMB provides DSQCOMM mapping for COBOL and is shipped with the product. Table 17 shows the values for DSQCOMMB.

Table 17. Interface communications area for DSQCOMMB

Structure Name	Data Type	Description
DSQ-RETURN-CODE	PIC 9(8)	Indicates the status of a QMF command after it has run. Its values are: **DSQ-SUCCESS** Successful execution of the request. **DSQ-WARNING** Normal completion with warnings. **DSQ-FAILURE** Command did not run correctly. **DSQ-SEVERE** Severe error; QMF session terminated.
DSQ-INSTANCE-ID	PIC 9(8)	Identifier established by QMF during execution of the START command
DSQ-COMM-LEVEL	PIC X(12)	Identifies the level of the DSQCOMM. You should set this to the value of DSQ_CURRENT_COMM_LEVEL before issuing the QMF START command.
DSQ-PRODUCT	PIC X(2)	Identifies the IBM query product in use.
DSQ-PRODUCT-RELEASE	PIC X(2)	Identifies the release level of the query product in use.
DSQ-RESERVE1	PIC X(28)	Reserved for future use
DSQ-MESSAGE-ID	PIC X(8)	Completion message ID
DSQ-Q-MESSAGE-ID	PIC X(8)	Query message ID
DSQ-START-PARM-ERROR	PIC X(8)	Parameter in error when START failed due to a parameter error
DSQ-CANCEL-IND	PIC X(1)	Contains one of two values, depending if the user canceled while a QMF command was running: • DSQ-CANCEL-YES • DSQ-CANCEL-NO
DSQ-RESERVE2	PIC X(23)	Reserved for future use

3. COBOL/370 is not supported in CICS/VSE.

Table 17. Interface communications area for DSQCOMMB (continued)

Structure Name	Data Type	Description
DSQ-RESERVE3	PIC X(156)	Reserved for future use
DSQ-MESSAGE-TEXT	PIC X(128)	Completion message text
DSQ-Q-MESSAGE-TEXT	PIC X(128)	Query message text

Function calls for COBOL

QMF provides one function call, DSQCIB, for the COBOL language. It is described in the communications macro DSQCOMMB. This function call has two formats: DSQCIB and DSQCIB extended format.

DSQCIB

This call is for QMF commands that do not require access to application program variables. Use this call for most QMF commands.

```
CALL DSQCIB USING DSQCOMM CMDLTH CMDSTR
```

The parameters have the following values:

DSQCOMM
The interface communications area

CMDLTH
Length of the command string, CMDSTR. It is an integer parameter.

CMDSTR
QMF command to run. It is an uppercase character string of the length specified by CMDLTH.

DSQCIB, extended format

This call has an extended syntax for the three QMF commands that do require access to application program variables: START and the extended formats of GET GLOBAL and SET GLOBAL.

```
DSQCIB USING
       DSQCOMM CMDLTH CMDSTR
       PNUM KLTH KWORD VLTH VALUE VTYPE
```

The parameters have the following values:

DSQCOMM
The interface communications area.

CMDLTH
Length of the command string, CMDSTR. It is an integer parameter.

CMDSTR
QMF command to run. It is an uppercase character string of the length specified by CMDLTH.

PNUM
Number of command keywords. It is an integer parameter.

KLTH
Length of each specified keyword. It is an integer parameter or an array of integer parameters.

KWORD
QMF keyword or keywords. Each is a character or structure of characters whose lengths are the same as specified by KLTH. If all the keywords have the same length, you can use an array of characters.

VLTH
Length of each value associated with the keyword. It is an integer parameter or an array of integer parameters.

VALUE
Value associated with each keyword. Its type is specified in the VTYPE parameter, and can be a character, a structure of characters, an integer parameter, or an array of integer parameters.

VTYPE
QMF data type of the value string VALUE. This type has one of two values, which are provided in the communications macro, DSQCOMMB:

- DSQ-VARIABLE-CHAR for character values
- DSQ-VARIABLE-FINT for integer values

All of the values specified in the VALUE field must have the data type specified in VTYPE.

Using ISPF LIBDEF service with COBOL

If you are using a dynamic call to the QMF interface DSQCIB and you want to use the LIBDEF function in your QMF application, change your dynamic calls to static calls. For example, change the call identifier statement

```
CALL DSQCIB USING ...
```

to its call literal form

```
CALL "DSQCIB" USING ...
```

Migration information

The DSQCOMM has changed from Version 2 Release 4 to Version 3.2.

- If you want to continue using the old DSQCOMM, you do not have to recompile your program.
- If you want to use Version 3.2 of DSQCIB, you must link-edit your Version 2 Release 4 program again.

The new DSQCOMM provides message text that is especially useful if you have an error in your START command. If you want to use the new

DSQCOMM, you need to recompile your program and initialize DSQ_COMM_LEVEL (in DSQCOMM) to DSQ_CURRENT_COMM_LEVEL. If you do *not* set this value, QMF treats your DSQCOMM as a Version 2 Release 4 level.

Note to users of CICS in MVS

The DSQCIB has changed from Version 3 Release 1 Modification 1 to Version 3 Release 2. The interface between the QMF-supplied function call and the main QMF program has changed from a CALL interface to an EXEC CICS LINK interface. The new interface provides better isolation from the user program and the QMF product. Because the interface has changed, you need to link-edit your programs again that used the callable interface.

COBOL programming example

The following program, DSQABFCO, is shipped with the QMF product. This example uses VS COBOL II.

You can look at the sample source code listing here or you can access it online.

- For OS/390, the sample program is a member of the library QMF720.SDSQSAPE.
- For VM, the sample program is on the production disk.
- For VSE, the sample program is located in the QMF sublibrary and is named DSQABFCO.Z.

The sample program for the COBOL callable interface performs the following function:

- Starts QMF
- Sets three global variables
- Runs a query called Q1
- Prints the resulting report using form F1
- Ends the QMF session

QMF does not supply query Q1 or form F1, but the sample program uses these objects.

This section also shows how to compile, link-edit, and run a COBOL program using the callable interface. QMF does not ship the REXX EXEC, JCL, or CLIST in these examples, but you can copy them from here, altering them to suit your installation.

```
*****************************************************************
*   The following is a VS COBOL II version of the query
*   callable interface *** DSQABFCO **.
*****************************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  DSQABFCO.
   DATE-COMPILED.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*************************
* Copy DSQCOMMB definition - contains query interface variables
*************************
   COPY DSQCOMMB.

* Query interface commands
 01  STARTQI      PIC X(5)  VALUE "START".
 01  SETG         PIC X(10) VALUE "SET GLOBAL".
 01  QUERY        PIC X(12) VALUE "RUN QUERY Q1".
 01  REPT         PIC X(22) VALUE "PRINT REPORT (FORM=F1 ".
 01  ENDQI        PIC X(4)  VALUE "EXIT".

* Query command length
 01  QICLTH       PIC 9(8) USAGE IS COMP-4.
* Number of variables
 01  QIPNUM       PIC 9(8) USAGE IS COMP-4.
* Keyword variable lengths
 01  QIKLTHS.
   03   KLTHS     PIC 9(8) OCCURS 10 USAGE IS COMP-4.
* Value Lengths
 01  QIVLTHS.
   03   VLTHS     PIC 9(8) OCCURS 10 USAGE IS COMP-4.
* Start Command Keyword
 01  SNAMES.
   03  SNAME1   PIC X(8) VALUE "DSQSMODE".
* Start Command Keyword Value
 01  SVALUES.
   03  SVALUE1  PIC X(11) VALUE "INTERACTIVE".
* Set GLOBAL Command Variable Names to set
 01  VNAMES.
   03  VNAME1   PIC X(7) VALUE "MYVAR01".
   03  VNAME2   PIC X(5) VALUE "SHORT".
   03  VNAME3   PIC X(7) VALUE "MYVAR03".
* Variable value parameters
 01  VVALUES.
   03   VVALS     PIC 9(8) OCCURS 10 USAGE IS COMP-4.

 01   TEMP     PIC 9(8)           USAGE IS COMP-4.
```

Figure 44. DSQABFCO, sample COBOL program (Part 1 of 2)

```
 PROCEDURE DIVISION.
*
* Start a query interface session
     MOVE DSQ-CURRENT-COMM-LEVEL TO DSQ-COMM-LEVEL.
     MOVE 5 TO QICLTH.
     MOVE 8 TO KLTHS(1).
     MOVE 11 TO VLTHS(1).
     MOVE 1 TO QIPNUM.
     CALL DSQCIB  USING DSQCOMM, QICLTH, STARTQI,
                        QIPNUM, QIKLTHS, SNAMES,
                        QIVLTHS, SVALUES, DSQ-VARIABLE-CHAR.
*
* Set numeric values into query variables using SET GLOBAL command
     MOVE 10 TO QICLTH.
     MOVE 7 TO KLTHS(1).
     MOVE 5 TO KLTHS(2).
     MOVE 7 TO KLTHS(3).
     MOVE 4 TO VLTHS(1).
     MOVE 4 TO VLTHS(2).
     MOVE 4 TO VLTHS(3).
     MOVE 20 TO VVALS(1).
     MOVE 40 TO VVALS(2).
     MOVE 84 TO VVALS(3).
     MOVE 3 TO QIPNUM.
     CALL DSQCIB  USING DSQCOMM, QICLTH, SETG,
                        QIPNUM, QIKLTHS, VNAMES,
                        QIVLTHS, VVALUES, DSQ-VARIABLE-FINT.
*
* Run a Query
     MOVE 12 TO QICLTH.
     CALL DSQCIB USING DSQCOMM, QICLTH, QUERY.
*
* Print the results of the query
     MOVE 22 TO QICLTH.
     CALL DSQCIB USING DSQCOMM, QICLTH, REPT.
*
* End the query interface session
     MOVE 4 TO QICLTH.
     CALL DSQCIB USING DSQCOMM, QICLTH, ENDQI.

     STOP RUN.
```

Figure 44. DSQABFCO, sample COBOL program (Part 2 of 2)

For CICS, the STOP RUN statement must be changed to a GOBACK statement.

DSQCOMM for COBOL

This include file is called DSQCOMMB and is shipped with QMF.

```
************************************************************     00001000
* COBOL INCLUDE FOR QUERY CALLABLE INTERFACE (MVS/VM)             00002000
************************************************************     00003000
                                                                 00004000
* STRUCTURE DECLARE FOR COMMUNICATIONS AREA                      00005000
                                                                 00006000
  01 DSQCOMM.                                                    00007000
                                                                 00008000
     03 DSQ-RETURN-CODE PIC 9(8) USAGE IS COMP.                   00009000
*                               FUNCTION RETURN CODE       *     00010000
     03 DSQ-INSTANCE-ID PIC 9(8) USAGE IS COMP.                   00011000
*                               IDENTIFIER FROM START CMD  *     00012000
     03 DSQ-COMM-LEVEL  PIC X(12).                                00013000
*                               COMMUNICATIONS LEVEL       *     00014000
     03 DSQ-PRODUCT     PIC X(2).                                 00015000
*                               QUERY PRODUCT ID           *     00016000
     03 DSQ-PRODUCT-RELEASE PIC X(2).                             00017000
*                               QUERY PRODUCT RELEASE      *     00018000
     03 DSQ-RESERVE1 PIC X(28).                                   00019000
*                               RESERVED AREA              *     00020000
     03 DSQ-MESSAGE-ID PIC X(8).                                  00021000
*                               COMPLETION MESSAGE ID      *     00022000
     03 DSQ-Q-MESSAGE-ID PIC X(8).                                00023000
*                               QUERY MESSAGE ID           *     00024000
     03 DSQ-START-PARM-ERROR PIC X(8).                            00025000
*                               START PARAMETER IN ERROR   *     00026000
     03 DSQ-CANCEL-IND PIC X(1).                                  00027000
*                               1 = COMMAND CANCELLED      *     00028000
*                               0 = COMMAND NOT CANCELLED  *     00029000
     03 DSQ-RESERVE2 PIC X(23).                                   00030000
*                               RESERVED AREA              *     00031000
     03 DSQ-RESERVE3 PIC X(156).                                  00032000
*                               RESERVED AREA              *     00033000
     03 DSQ-MESSAGE-TEXT PIC X(128).                              00034000
*                               QMF MESSAGE TEXT           *     00035000
     03 DSQ-Q-MESSAGE-TEXT PIC X(128).                            00036000
*                               QMF QUERY MESSAGE TEXT     *     00037000
*                               512 BYTES TOTAL            *     00038000
                                                                 00039000
                                                                 00040000
*  VALUES FOR DSQ-RETURN-CODE                                    00041000
                                                                 00042000
  01 DSQ-SUCCESS PIC 9(8) USAGE IS COMP VALUE 0.                 00043000
  01 DSQ-WARNING PIC 9(8) USAGE IS COMP VALUE 4.                 00044000
  01 DSQ-FAILURE PIC 9(8) USAGE IS COMP VALUE 8.                 00045000
  01 DSQ-SEVERE  PIC 9(8) USAGE IS COMP VALUE 16.                00046000
                                                                 00047000
*  VALUES FOR DSQ-INSTANCE-ID                                    00048000
                                                                 00049000
  01 DSQ-CONTINUE PIC 9(8) USAGE IS COMP VALUE 0.                00050000
```

Figure 45. DSQCOMMB, COBOL communications area (Part 1 of 2)

```
                                                                        00051000
*  VALUES FOR DSQ-COMM-LEVEL                                            00052000
                                                                        00053000
  01 DSQ-CURRENT-COMM-LEVEL PIC X(12) VALUE "DSQL>001002<".             00054000
                                                                        00055000
*  VALUES FOR DSQ-PRODUCT                                               00056000
                                                                        00057000
  01 DSQ-QRW             PIC X(2) VALUE "01".                          00058000
  01 DSQ-QMF             PIC X(2) VALUE "02".                          00059000
  01 DSQ-QM4             PIC X(2) VALUE "03".                          00060000
                                                                        00061000
*  VALUES FOR DSQ-PRODUCT-RELEASE                                       00062000
                                                                        00063000
  01 DSQ-QRW-V1R2        PIC X(2) VALUE "01".                          00064000
  01 DSQ-QRW-V1R3        PIC X(2) VALUE "02".                          00065000
  01 DSQ-QMF-V2R4        PIC X(2) VALUE "01".                          00066000
  01 DSQ-QMF-V3R1        PIC X(2) VALUE "02".                          00067000
  01 DSQ-QMF-V3R1M1      PIC X(2) VALUE "03".                          00068000
  01 DSQ-QMF-V3R2        PIC X(2) VALUE "04".                          00069000
  01 DSQ-QMF-V3R3        PIC X(2) VALUE "05".                          00070000
  01 DSQ-QMF-V6R1        PIC X(2) VALUE "06".                          00071000
  01 DSQ-QM4-V1R1        PIC X(2) VALUE "01".                          00072000
                                                                        00073000
*  VALUES FOR DSQ-CANCEL-INDE                                           00074000
                                                                        00075000
  01 DSQ-CANCEL-YES      PIC X(1) VALUE "1".                           00076000
  01 DSQ-CANCEL-NO       PIC X(1) VALUE "0".                           00077000
                                                                        00078000
*  VALUES FOR MODE                                                      00079000
                                                                        00080000
  01 DSQ-INTERACTIVE     PIC X(1) VALUE "1".                           00081000
  01 DSQ-BATCH           PIC X(1) VALUE "2".                           00082000
                                                                        00083000
*  VALUES YES AND NO                                                    00084000
                                                                        00085000
  01 DSQ-YES             PIC X(1) VALUE "1".                           00086000
  01 DSQ-NO              PIC X(1) VALUE "2".                           00087000
                                                                        00088000
*  CALLABLE INTERFACE PROGRAM NAME                                      00089000
                                                                        00090000
  01 DSQCIB            PIC X(6) VALUE "DSQCIB".                        00091000
                                                                        00092000
*  VALUES FOR VARIABLE TYPE ON CALL PARAMETER                           00093000
                                                                        00094000
  01 DSQ-VARIABLE-CHAR PIC X(4) VALUE "CHAR".                           00095000
  01 DSQ-VARIABLE-FINT PIC X(4) VALUE "FINT".                           00096000
```

Figure 45. DSQCOMMB, COBOL communications area (Part 2 of 2)

Considerations for running your COBOL callable interface program

When you translate, compile, and link-edit a program that uses the QMF callable interface, consider the following:

- Execution environment

QMF is run as an assembler subprogram in the COBOL environment. Your COBOL program must CALL the QMF interface program DSQCIB using a COBOL dynamic call.

- Quotation marks or apostrophes?

  You must use either quotes (") or apostrophes (') to delimit literals in a COBOL program. You can specify the delimiter of your choice to the CICS translation process and the COBOL compiler by specifying `QUOTE` or `APOST`. Make sure the APOST or QUOTE option in effect for the COBOL compiler matches that of the CICS translator.

  The communications area DSQCOMMB and the sample COBOL program DSQABFCO as distributed by QMF uses quotes to delimit literals. If your installation or program uses apostrophes instead, change DSQCOMMB as distributed by QMF or copy the structure to your program, changing quotes to apostrophes.

- The communications macro DSQCOMMB

  The communications area DSQCOMMB must be available to the COBOL compile step or copied into your program as a control structure.

- The interface module DSQCIB

  The QMF interface module must be available during the link-edit phase of your program.

Running your COBOL programs in CICS

After you write your program, you need to translate, compile, and link-edit it as required before you can run it. The programs listed in this section show the steps necessary to do so.

QMF does not ship the REXX EXEC, JCL, or CLIST in these examples, but you can copy them from here, altering them to suit your installation.

Translating, compiling, and link-editing for CICS in MVS

The following example shows the CICS supplied procedure DFHEBTVL, which supports COBOL. See the CICS library for details on how to translate programs for use in CICS.

```
//samCOBOL  JOB
//          EXEC PROC=DFHEBTVL
//TRN.SYSIN DD  *
*CBL      XOPTS(CICS translator options ...QUOTE COBOL2)
          .
          Your program or copy of QMF sample DSQABFCO
          .
/*
//* Provide Access to QMF Communications Macro DSQCOMMB
//COB.SYSLIB   DD DSN=QMF720.SDSQSAPE,DISP=SHR
//* Provide Access to QMF Interface Module
//LKED.QMFLOAD DD DSN=QMF720.SDSQLOAD,DISP=SHR
//LKED.SYSIN   DD *
          INCLUDE CICSLOAD(DFHECI)
          INCLUDE QMFLOAD(DSQCIB)
          ORDER DFHECI
          ENTRY samCOBOL
          MODE  AMODE(31) RMODE(ANY)
          NAME  samCOBOL(R)
/*
```

Figure 46. JCL to run the CICS translator, COBOL compiler, and linkage editor

Translating, compiling, and link-editing for CICS in VSE

The VSE job control in Figure 47 on page 178 is an example of installing a COBOL program into CICS running on VSE. See the CICS library for details on how to translate and compile your COBOL programs.

This example, shipped with QMF, is located in the QMF sublibrary and is named DSQ3CICO.Z.

```
* $$ JOB JNM=DSQ3CICO,DISP=D,CLASS=0
// JOB DSQ3CICO   Sample job to Install QMF Callable Interface (COBOL)
*  ---------------------------------------------------------------------
*  Install QMF Callable Interface Example (COBOL)
*  ---------------------------------------------------------------------
// SETPARM VOLID=volid      *-- update volid for syspch
// SETPARM START=rtrk       *-- update start track/block (syspch)
// SETPARM SIZE=ntrks       *-- update number of tracks/blocks (syspch)
*  ---------------------------------------------------------------------
// DLBL   IJSYSPH,'CICS.TRANSLAT.OUTPUT',0
// EXTENT SYSPCH,,1,0,&START,&SIZE
ASSGN SYSPCH,DISK,VOL=&VOLID,SHR
*  Library search chain must contain the QMF, CICS and COBOL sublibrary
// LIBDEF *,SEARCH=(PRD2.PROD,PRD1.BASE,PRD2.CONFIG)
// LIBDEF PHASE,CATALOG=PRD2.PROD
*  ---------------------------------------------------------------------
*  Step 1: Translate callable interface program (COBOL)
*  ---------------------------------------------------------------------
*  You may need to update or remove the SLI statement for your program.
*  ---------------------------------------------------------------------
// EXEC   DFHECP1$,SIZE=256K,PARM='XOPTS(CICS,QUOTE)'
* $$ SLI MEM=DSQABFCO.Z,S=PRD2.PROD
/*
*  ---------------------------------------------------------------------
*  Step 2: Compile callable interface program (COBOL)
*  ---------------------------------------------------------------------
CLOSE SYSPCH,00D
// DLBL   IJSYSIN,'CICS.TRANSLAT.OUTPUT',0
// EXTENT SYSIPT
ASSGN SYSIPT,DISK,VOL=&VOLID,SHR
// OPTION NODECK,CATAL
   PHASE DSQABFCO,*
         INCLUDE DFHECI
// EXEC IGYCRCTL,PARM='SZ(MAX),OBJECT,MAP,RES,NODYNAM,QUOTE,LIB,RENT'
CLOSE SYSIPT,SYSRDR
/*
*  ---------------------------------------------------------------------
*  Step 3: Link-edit callable interface program (COBOL)
*  ---------------------------------------------------------------------
// EXEC LNKEDT,PARM='AMODE=31,RMODE=ANY'
/*
/&
// JOB  RESET
ASSGN SYSIPT,SYSRDR    IF 1A93D, CLOSE SYSIPT,SYSRDR
ASSGN SYSPCH,00D       IF 1A93D, CLOSE SYSPCH,00D
/&
* $$ EOJ
```

Figure 47. Job control to run the CICS/VSE translator, COBOL compiler, and linkage editor

Compiling and running your programs under CMS in VM

The following program compiles and runs your callable interface application using the IBM COBOL compiler.

QMF does not ship the REXX EXEC in this example, but you can copy it from here, altering it to suit your installation.

```
/*********************************************************************/
/* Compile your COBOL program and run it.        */
/*********************************************************************/
TRACE off
ADDRESS CMS
/*********************************************************************/
/* Access COBOL product disk using a program, PRODUCT, that you   */
/* write.                                                            */
/*********************************************************************/
"EXEC PRODUCT COBOL"
/*********************************************************************/
/* Get QMF DSQCOMM into a macro library and set GLOBAL compile    */
/* maclibs.
/*********************************************************************/
"ERASE TEMPP MACLIB A"
"MACLIB GEN TEMPP DSQCOMMB"
Maclist = "TEMPP VSC2MAC COB2MLIB COB2PLIB DMSSP CMSLIB OSMACRO"
"GLOBAL MACLIB" Maclist
/*********************************************************************/
/* Compile the program                                               */
/*********************************************************************/
"GLOBAL TXTLIB SCEELKED"
"COBOL2 yourname (LIB RESIDENT LIST RENT DYNAM"
```

Figure 48. Program for compiling and running COBOL in CMS (Part 1 of 2)

```
/*****************************************************************/
/* Access SQL/DS and initialize database                         */
/*****************************************************************/
"EXEC PRODUCT SQLDS"
"EXEC SQLINIT DBNAME(SQLDBA)"
/*****************************************************************/
/* Access GDDM product disk                                      */
/*****************************************************************/
"EXEC PRODUCT GDDM"
/*****************************************************************/
/* Issue Filedefs for QMF product                                */
/*****************************************************************/
/* DEBUG = DDNAME FOR QMF DIAGNOSTICS OUTPUT      */
"FILEDEF DSQDEBUG PRINTER ( LRECL 80  BLKSIZE 80  RECFM FBA PERM"
/* PRINT = DDNAME FOR QMF PRINTED OUTPUT                         */
"FILEDEF DSQPRINT PRINTER ( LRECL 133 BLKSIZE 133 RECFM FBA PERM"
/* EDIT  = DDNAME FOR QMF EDIT TRANSFER FILE                     */
"FILEDEF DSQEDIT DISK QMFEDIT FILE A (PERM"
/* DSQSIDE = DDNAME FOR QMF SPILL FILE                           */
"FILEDEF DSQSPILL DISK DSQSIDE DATA A1 (PERM"
/* DSQPNLE = DDNAME FOR PANEL FILE                               */
"FILEDEF DSQPNLE DISK DSQPNLE FILE * (PERM"
"FILEDEF ISPLLIB CLEAR"
"FILEDEF ISPLLIB DISK DSQLDLIB LOADLIB *"
/*****************************************************************/
/* Provide access to QMF and COBOL program libraries             */
/*****************************************************************/
"GLOBAL LOADLIB DSQLDLIB VSC2LOAD"
"GLOBAL TXTLIB VSC2LTXT ADMRLIB ADMPLIB ADMGLIB SCEELKED"
Say "Starting to run COBOL program"
"RUN yourname"
Exit 0
```

Figure 48. Program for compiling and running COBOL in CMS (Part 2 of 2)

Running your COBOL programs in TSO

The following sections provide sample JCL to run the COBOL compiler and linkage editor and sample programs for running your complied programs in TSO either with or without ISPF.

Compiling and link-editing in TSO

The following job uses the COBOL compiler to compile your callable interface application. It then link-edits your application. Some parameters might vary from one installation to the next. See your QMF administrator for details.

```
//samCOBOL   JOB
//STEP1      EXEC PROC=IGYWCL
//* Provide Access to QMF Communications Macro DSQCOMM
//COBOL.SYSLIB   DD  DSN=QMF720.SAMPLIB,DISP=SHR
//COBOL.SYSIN    DD  *
          .
          Your program or copy of QMF sample DSQABFCO
          .
/*
//* Provide Access to QMF Interface Module
//LKED.QMFLOAD  DD DSN=QMF720.SDSQLOAD,DISP=SHR
//LKED.SYSIN    DD  *
          INCLUDE QMFLOAD(DSQCIB)
          ENTRY samCOBOL
          MODE  AMODE(31) RMODE(ANY)
          NAME  samCOBOL(R)
/*
```

Figure 49. JCL to run the COBOL compiler and linkage editor

Running your programs in TSO without ISPF

After you successfully compile your program, you can run it by writing a program similar to the following:

```
PROC 0
CONTROL ASIS
/*************************************************************/
/* Note: QMF, DB2, GDDM and COBOL load libraries must be     */
/*       allocated before running this CLIST.                */
/*       Name of QMF load library is  "QMF720.SDSQLOAD".     */
/*************************************************************/
/* Specify attribute list for dataset allocations            */
/*************************************************************/
ATTR PRINTDCB LRECL(133) RECFM(F B A) BLKSIZE(1330)
ATTR DEBUGDCB LRECL(80)  RECFM(F B)   BLKSIZE(3120)
ATTR UDUMPDCB LRECL(125) RECFM(V B A) BLKSIZE(1632)
ATTR EDITDCB  LRECL(79)  RECFM(F B A) BLKSIZE(4029)
/*************************************************************/
/* Datasets used by TSO                                      */
/*************************************************************/
ALLOC FI(SYSPROC) DA('QMF720.SDSQCLTE')
ALLOC FI(SYSEXEC) DA('QMF720.SDSQEXCE')
/*************************************************************/
/* QMF/GDDM Datasets                                         */
/*************************************************************/
ALLOC FI(ADMGGMAP) DA('QMF720.QMFMAPS')  SHR REUSE
ALLOC FI(ADMCFORM) DA('QMF720.DSQCFORM') SHR REUSE
ALLOC FI(DSQUCFRM) DA('QMF720.DSQUCFRM') SHR REUSE
ALLOC FI(ADMSYMBL) DA('ADM.GDDMSYM')  SHR REUSE
ALLOC FI(ADMGDF)   DA('ADM.GDDM.CHARTLIB') SHR REUSE
ALLOC FI(ADMDEFS)  DA('ADM.GDDM.NICKNAME') SHR REUSE
/*************************************************************/
/* Datasets used by QMF                                      */
/*************************************************************/
ALLOC FI(DSQPRINT) SYSOUT(X) USING(PRINTDCB)
ALLOC FI(DSQDEBUG) SYSOUT(X) USING(DEBUGDCB)
ALLOC FI(DSQUDUMP) SYSOUT(X) USING(UDUMPDCB)
ALLOC FI(DSQSPILL) NEW UNIT(SYSDA) SPACE(1,1) TRACKS
ALLOC FI(DSQEDIT)  NEW UNIT(SYSDA) USING(EDITDCB)
ALLOC FI(DSQPNLE)  DA('QMF720.DSQPNLE') SHR REUSE
/*************************************************************/
/* Start your program using TSO CALL command                */
/*************************************************************/
CALL samCOBOL
EXIT CODE(0)
```

Figure 50. JCL to run the COBOL compiler and linkage editor

Running your programs in TSO under ISPF

After you successfully compile your program, you can run it by writing a program similar to the following:

```
PROC 0
CONTROL ASIS
/*************************************************************/
/* Specify attribute list for dataset allocations            */
/*************************************************************/
ATTR PRINTDCB LRECL(133) RECFM(F B A) BLKSIZE(1330)
ATTR DEBUGDCB LRECL(80)  RECFM(F B)   BLKSIZE(3120)
ATTR UDUMPDCB LRECL(125) RECFM(V B A) BLKSIZE(1632)
ATTR EDITDCB  LRECL(79)  RECFM(F B A) BLKSIZE(4029)
/*************************************************************/
/* Datasets used by TSO                                      */
/*************************************************************/
ALLOC FI(SYSPROC) DA('QMF720.SDSQCLTE','ISR.ISRCLIB')
ALLOC FI(SYSEXEC) DA('QMF720.SDSQEXCE')
/*************************************************************/
/* Datasets used by ISPF                                     */
/*************************************************************/
ALLOC FI(ISPLLIB) SHR REUSE +

      DA('QMF720.SDSQLOAD','ADM.GDDMLOAD','DSN.DSNEXIT','DSN.DSNLOAD', +
         'PRDUCT.COB2LIB')
ALLOC FI(ISPMLIB) SHR REUSE +
      DA('QMF720.SDSQMLBE','ISR.ISRMLIB','ISP.ISPMLIB')
ALLOC FI(ISPPLIB) SHR REUSE +
      DA('QMF720.SDSQPLBE','ISR.ISRPLIB','ISP.ISPPLIB')
ALLOC FI(ISPSLIB) SHR REUSE +
      DA('QMF720.SDSQSLBE','ISR.ISRSLIB','ISP.ISPSLIB')
ALLOC FI(ISPTLIB) SHR REUSE +
      DA('ISR.ISRTLIB','ISP.ISPTLIB')
/*************************************************************/
/* QMF/GDDM Datasets                                         */
/*************************************************************/
ALLOC FI(ADMGGMAP) DA('QMF720.QMFMAPS')  SHR REUSE
ALLOC FI(ADMCFORM) DA('QMF720.DSQCFORM') SHR REUSE
ALLOC FI(DSQUCFRM) DA('QMF720.DSQUCFRM') SHR REUSE
ALLOC FI(ADMSYMBL) DA('ADM.GDDMSYM')  SHR REUSE
ALLOC FI(ADMGDF)   DA('ADM.GDDM.CHARTLIB') SHR REUSE
ALLOC FI(ADMDEFS)  DA('ADM.GDDM.NICKNAME') SHR REUSE
/*************************************************************/
/* Datasets used by QMF                                      */
/*************************************************************/
ALLOC FI(DSQPRINT) SYSOUT(X) USING(PRINTDCB)
ALLOC FI(DSQDEBUG) SYSOUT(X) USING(DEBUGDCB)
ALLOC FI(DSQUDUMP) SYSOUT(X) USING(UDUMPDCB)
ALLOC FI(DSQSPILL) NEW UNIT(SYSDA) SPACE(1,1) TRACKS
ALLOC FI(DSQEDIT)  NEW UNIT(SYSDA) USING(EDITDCB)
ALLOC FI(DSQPNLE)  DA('QMF720.DSQPNLE') SHR REUSE
/*************************************************************/
/* Start your program as the initial ISPF dialog             */
/*************************************************************/
ISPSTART PGM(samCOBOL) NEWAPPL(DSQE)
EXIT CODE(4)
```

Figure 51. CLIST for running your program in TSO under ISPF

The EXIT CODE(4) suppresses the showing of the ISPF disposition panel.

FORTRAN language interface

The FORTRAN callable interface corresponds to that provided for other SAA languages.

Note to CICS users
Because FORTRAN is not available under CICS, the QMF callable interface for FORTRAN does not work under CICS.

Interface communications area mapping for FORTRAN (DSQCOMMF)

DSQCOMMF provides DSQCOMM mapping for FORTRAN and is shipped with the product. Table 18 shows the information for DSQCOMMF, which you must not alter:

Table 18. Interface communications area, DSQCOMMF

Structure Name	Data Type	Description
DSQ_RETURN_CODE	INTEGER	Indicates the status of a QMF command after it has been run. Its values are: **DSQ_SUCCESS** Successful execution of the request. **DSQ_WARNING** Normal completion with warnings. **DSQ_FAILURE** Command did not run correctly. **DSQ_SEVERE** Severe error; QMF session terminated.
DSQ_INSTANCE_ID	INTEGER	Identifier established by QMF during execution of the START command
DSQ_COMM_LEVEL	CHARACTER(12)	Identifies the level of the DSQCOMM. You should set this to the value of DSQ_CURRENT_COMM_LEVEL before issuing the QMF START command.
DSQ_PRODUCT	CHARACTER(2)	Identifies the IBM query product in use.
DSQ_PRODUCT_RELEASE	CHARACTER(2)	Identifies the release level of the query product in use.
DSQ_RESERVE1	CHARACTER(28)	Reserved for future use
DSQ_MESSAGE_ID	CHARACTER(8)	Completion message ID
DSQ_Q_MESSAGE_ID	CHARACTER(8)	Query message ID
DSQ_START_PARM_ERROR	CHARACTER(8)	Parameter in error when START failed due to a parameter error

Table 18. Interface communications area, DSQCOMMF (continued)

Structure Name	Data Type	Description
DSQ_CANCEL_IND	CHARACTER(1)	Contains one of two values, depending if the user canceled while a QMF command was running: **DSQ_CANCEL_YES** CHARACTER(1) **DSQ_CANCEL_NO** CHARACTER(1)
DSQ_RESERVE2	CHARACTER(23)	Reserved for future use
DSQ_RESERVE3	CHARACTER(156)	Reserved for future use
DSQ_MESSAGE_TEXT	CHARACTER(128)	Completion message text
DSQ_Q_MESSAGE_TEXT	CHARACTER(128)	Query message text

Function calls for FORTRAN

QMF provides two function calls for the FORTRAN language: DSQCIF and DSQCIFE. Both calls are described in the communications macro DSQCOMMF.

DSQCIF

This call is for QMF commands that do not require access to application program variables. Use this call for most QMF commands.

```
 RC = DSQCIF(DSQCOMM,
+     CMDLTH,
+     CMDSTR)
```

The parameters have the following values:

DSQCOMM
The communications area

CMDLTH
Length of the command string, CMDSTR. It is an integer parameter.

CMDSTR
QMF command to run. It is an uppercase character string of the length specified by CMDLTH.

DSQCIFE

This call has an extended syntax for the three commands that require access to application program variables: START and the extended formats of GET GLOBAL and SET GLOBAL.

The syntax for this call is:

```
 RC = DSQCIFE(DSQCOMM,
+     CMDLTH,
+     CMDSTR,
+     PNUM,
+     KLTH,
+     KWORD,
+     VLTH,
+     VALUE,
+     VTYPE)
```

The parameters have the following values:

DSQCOMM
The interface communications area.

CMDLTH
Length of the command string, CMDSTR; it is an integer parameter.

CMDSTR
QMF command to run. It is an uppercase character string of the length specified by CMDLTH.

PNUM
Number of command keywords. It is an integer parameter.

KLTH
Length of each specified keyword. It is an integer parameter or parameter array.

KWORD
QMF keyword or keywords. It is a character or structure of characters whose lengths are the same as specified by KLTH. You can use an array of characters, if all of the keywords have the same length. QMF assumes that the keywords are in contiguous storage and are not separated by any special separator characters.

VLTH
Length of each value associated with the keyword, and is an integer parameter or parameter array.

VALUE
Value associated with each keyword. Its type is specified in the VTYPE parameter and can be a character, structure of characters, integer parameter, or parameter array. If you have character values, QMF assumes that the values are in contiguous storage, not separated by any special separator characters.

VTYPE
QMF data type of the value string VALUE. VTYPE can have one of two values, which are provided in the communications macro, DSQCOMMF:

- DSQ_VARIABLE_CHAR for character values.

- DSQ_VARIABLE_FINT for integer values.

All of the values specified in the VALUE field must have the data type specified in VTYPE.

FORTRAN programming example

The following program, DSQABFF, is shipped with QMF and uses VS FORTRAN.

You can look at the sample source code listing here or you can access it online. For OS/390, the sample program is a member of the library QMF720.SDSQSAPE. For VM, the sample program is on the production disk.

The sample program for the FORTRAN callable interface performs the following function:

- Starts QMF
- Sets three global variables
- Runs a query called Q1
- Prints the resulting report using form F1
- Ends the QMF session

QMF does not supply query Q1 or form F1, but the sample program uses these objects.

This section also shows how to compile, link-edit, and run a FORTRAN program using the callable interface. QMF does not ship the REXX EXEC, JCL, or CLIST in these examples, but you can copy them from here, altering them to suit your installation.

```
C*********************************************************************
C Sample Program: dsqabff
C FORTRAN Version of SAA Query Manager Callable Interface
C
C Creation Date: 11/21/89
C
C ENVIRONMENT:    API IN FORTRAN
C*********************************************************************
C
C Processing:
C         a. Start a Query Manager Session using the Callable Interface.
C
C         b. Set Global Query Manager numeric variables.
C
C         d. Run a Query Manager query using the Callable Interface.
C
C         e. Print a report using the Callable Interface.
C
C         f. Exit the Query Manager Session.
C
```

Figure 52. DSQABFF, sample FORTRAN program (Part 1 of 5)

```
C Prerequisites:1. Create the SAMPLE database.
C
C               2. Create a prompted query, Q1, which has a SELECT state
C
C               3. Create a form, F1, that displays data for query Q1.
C
C*********************************************************************

      PROGRAM DSQABFF

C*********************************************************************
C Include and declare query interface communications area
C*********************************************************************
      INCLUDE (DSQCOMMF)

C*********************************************************************
C   Query interface command lengths and commands
C*********************************************************************
      INTEGER COMMAND_LENGTH
      CHARACTER START_QUERY_INTERFACE*5,
     +          SET_GLOBAL_VARIABLES*10,
     +          RUN_QUERY*12,
     +          PRINT_REPORT*22,
     +          END_QUERY_INTERFACE*4

C*********************************************************************
C   Query command extension, number of parameters and lengths
C*********************************************************************
      INTEGER NUMBER_OF_PARAMETERS,
     +        KEYWORD_LENGTHS(10),
     +        DATA_LENGTHS(10)

C*********************************************************************
C   Variable data type constants
C*********************************************************************
      CHARACTER CHAR_DATA_TYPE*4,
     +          INT_DATA_TYPE*4

C*********************************************************************
C   Keyword parameter and value for START command
C*********************************************************************
      CHARACTER*8  START_KEYWORDS(1)
      CHARACTER*11 START_KEYWORD_VALUES(1)
```

Figure 52. DSQABFF, sample FORTRAN program (Part 2 of 5)

```
C*********************************************************************
C   Keyword parameter and values for SET command
C*********************************************************************
      CHARACTER   SET_KEYWORDS(19)
      CHARACTER   SET_KEYWORD_1*7,
     +            SET_KEYWORD_2*5,
     +            SET_KEYWORD_3*7

      EQUIVALENCE (SET_KEYWORDS( 1), SET_KEYWORD_1),
     +            (SET_KEYWORDS( 8), SET_KEYWORD_2),
     +            (SET_KEYWORDS(13), SET_KEYWORD_3)

      CHARACTER   SET_VALUES(12)
      INTEGER*4   SET_VALUE_1,
     +            SET_VALUE_2,
     +            SET_VALUE_3

      EQUIVALENCE (SET_VALUES(1), SET_VALUE_1),
     +            (SET_VALUES(5), SET_VALUE_2),
     +            (SET_VALUES(9), SET_VALUE_3)

C*********************************************************************
C Declare command length and return code variables
C*********************************************************************
      INTEGER   LEN,
     +          RC

C*********************************************************************
C Initialization
C*********************************************************************

      DATA START_QUERY_INTERFACE /'START'                /
      DATA SET_GLOBAL_VARIABLES  /'SET GLOBAL'           /
      DATA RUN_QUERY             /'RUN QUERY Q1'         /
      DATA PRINT_REPORT          /'PRINT REPORT (FORM=F1)'/
      DATA END_QUERY_INTERFACE   /'EXIT'                 /

      DATA CHAR_DATA_TYPE        /DSQ_VARIABLE_CHAR      /
      DATA INT_DATA_TYPE         /DSQ_VARIABLE_FINT      /
```

Figure 52. DSQABFF, sample FORTRAN program (Part 3 of 5)

```
C*********************************************************
C   Start Query Session
C*********************************************************

      DSQ_COMM_LEVEL = DSQ_CURRENT_COMM_LEVEL
      NUMBER_OF_PARAMETERS    = 1
      COMMAND_LENGTH          = LEN(START_QUERY_INTERFACE)
      KEYWORD_LENGTHS(1)      = LEN(START_KEYWORDS(1))
      DATA_LENGTHS(1)         = LEN(START_KEYWORD_VALUES(1))
      START_KEYWORDS(1)       = 'DSQSMODE'
      START_KEYWORD_VALUES(1) = 'INTERACTIVE'

      RC = DSQCIFE(DSQCOMM,
     +             COMMAND_LENGTH,
     +             START_QUERY_INTERFACE,
     +             NUMBER_OF_PARAMETERS,
     +             KEYWORD_LENGTHS,
     +             START_KEYWORDS,
     +             DATA_LENGTHS,
     +             START_KEYWORD_VALUES,
     +             CHAR_DATA_TYPE)

C*********************************************************************
C   Set numeric values into query using SET command
C*********************************************************************
      NUMBER_OF_PARAMETERS = 3
      COMMAND_LENGTH       = LEN(SET_GLOBAL_VARIABLES)
      SET_KEYWORD_1        = 'MYVAR01'
      SET_KEYWORD_2        = 'SHORT'
      SET_KEYWORD_3        = 'MYVAR03'
      KEYWORD_LENGTHS(1)   = LEN(SET_KEYWORD_1)
      KEYWORD_LENGTHS(2)   = LEN(SET_KEYWORD_2)
      KEYWORD_LENGTHS(3)   = LEN(SET_KEYWORD_3)
      DATA_LENGTHS(1)      = 4
      DATA_LENGTHS(2)      = 4
      DATA_LENGTHS(3)      = 4
      SET_VALUE_1          = 20
      SET_VALUE_2          = 40
      SET_VALUE_3          = 84

      RC = DSQCIFE(DSQCOMM,
     +             COMMAND_LENGTH,
     +             SET_GLOBAL_VARIABLES,
     +             NUMBER_OF_PARAMETERS,
     +             KEYWORD_LENGTHS,
     +             SET_KEYWORDS,
     +             DATA_LENGTHS,
     +             SET_VALUES,
     +             INT_DATA_TYPE)
```

Figure 52. DSQABFF, sample FORTRAN program (Part 4 of 5)

```
C*********************************************************************
C   Run a query
C*********************************************************************
      COMMAND_LENGTH = LEN(RUN_QUERY)
      RC = DSQCIF(DSQCOMM,
     +            COMMAND_LENGTH,
     +            RUN_QUERY)

C*********************************************************************
C   Print the results of the query
C*********************************************************************
      COMMAND_LENGTH = LEN(PRINT_REPORT)
      RC = DSQCIF(DSQCOMM,
     +            COMMAND_LENGTH,
     +            PRINT_REPORT)

C*********************************************************************
C   End the query interface session
C*********************************************************************
      COMMAND_LENGTH = LEN(END_QUERY_INTERFACE)
      RC = DSQCIF(DSQCOMM,
     +            COMMAND_LENGTH,
     +            END_QUERY_INTERFACE)

      END
```

Figure 52. DSQABFF, sample FORTRAN program (Part 5 of 5)

DSQCOMM for FORTRAN

This file, called DSQCOMMF, is shipped with QMF.

```
C*********************************************************************** 00001000
C       FORTRAN include file for Callable Interface (MVS/VM)            00002000
C*********************************************************************** 00003000
C     Return codes                                                      00004000
      INTEGER DSQ_SUCCESS, DSQ_WARNING, DSQ_FAILURE, DSQ_SEVERE         00005000
      PARAMETER(                                                        00006000
     +          DSQ_SUCCESS =  0,                                       00007000
     +          DSQ_WARNING =  4,                                       00008000
     +          DSQ_FAILURE =  8,                                       00009000
     +          DSQ_SEVERE  = 16)                                       00010000
                                                                        00011000
C     Communications level                                              00012000
      CHARACTER DSQ_CURRENT_COMM_LEVEL*12                               00013000
      PARAMETER(                                                        00014000
     +          DSQ_CURRENT_COMM_LEVEL = 'DSQL>001002<')                00015000
                                                                        00016000
C     Query product IDs                                                 00017000
      CHARACTER DSQ_QRW*2, DSQ_QMF*2, DSQ_QM4*2                         00018000
      PARAMETER(                                                        00019000
     +          DSQ_QRW = '01',                                         00020000
     +          DSQ_QMF = '02',                                         00021000
     +          DSQ_QM4 = '03')                                         00022000
                                                                        00023000
C     Query product release levels                                      00024000
      CHARACTER DSQ_QRW_V1R2*2,   DSQ_QRW_V1R3*2,                       00025000
     +          DSQ_QMF_V2R4*2,   DSQ_QMF_V3R1*2,                       00026000
     +          DSQ_QMF_V3R1M1*2, DSQ_QMF_V3R2*2,                       00027000
     +          DSQ_QMF_V3R3*2,   DSQ_QMF_V6R1*2,                       00028000
     +          DSQ_QM4_V1R1*2                                          00029000
      PARAMETER(                                                        00030000
     +          DSQ_QRW_V1R2   = '01',                                  00031000
     +          DSQ_QRW_V1R3   = '02',                                  00032000
     +          DSQ_QMF_V2R4   = '01',                                  00033000
     +          DSQ_QMF_V3R1   = '02',                                  00034000
     +          DSQ_QMF_V3R1M1 = '03',                                  00035000
     +          DSQ_QMF_V3R2   = '04',                                  00036000
     +          DSQ_QMF_V3R3   = '05',                                  00037000
     +          DSQ_QMF_V6R1   = '06',                                  00038000
     +          DSQ_QM4_V1R1   = '01')                                  00039000
                                                                        00040000
C     Host variable types                                               00041000
      CHARACTER DSQ_VARIABLE_CHAR*4, DSQ_VARIABLE_FINT*4                00042000
      PARAMETER(                                                        00043000
     +          DSQ_VARIABLE_CHAR = 'CHAR',                             00044000
     +          DSQ_VARIABLE_FINT = 'FINT')                             00045000
                                                                        00046000
C     Cancel indicator                                                  00047000
      CHARACTER DSQ_CANCEL_YES, DSQ_CANCEL_NO                           00048000
      PARAMETER(                                                        00049000
     +          DSQ_CANCEL_YES  = '1',                                  00050000
     +          DSQ_CANCEL_NO   = '0')                                  00051000
                                                                        00052000
      CHARACTER DSQCOMM(512)                                            00053000
```

Figure 53. DSQCOMMF, FORTRAN communications area (Part 1 of 2)

```
      INTEGER   DSQ_RETURN_CODE, DSQ_INSTANCE_ID                        00054000
      CHARACTER DSQ_COMM_LEVEL*12,                                      00055000
     +          DSQ_PRODUCT*2,                                         00056000
     +          DSQ_PRODUCT_RELEASE*2,                                 00057000
     +          DSQ_RESERVE1*28,                                       00058000
     +          DSQ_MESSAGE_ID*8,                                      00059000
     +          DSQ_Q_MESSAGE_ID*8,                                    00060000
     +          DSQ_START_PARM_ERROR*8,                                00061000
     +          DSQ_CANCEL_IND*1,                                      00062000
     +          DSQ_RESERVE2*23,                                       00063000
     +          DSQ_RESERVE3*156,                                      00064000
     +          DSQ_MESSAGE_TEXT*128,                                  00065000
     +          DSQ_Q_MESSAGE_TEXT*128                                 00066000
                                                                       00067000
      EQUIVALENCE   (DSQCOMM( 1),  DSQ_RETURN_CODE       ),            00068000
     +              (DSQCOMM( 5),  DSQ_INSTANCE_ID       ),            00069000
     +              (DSQCOMM( 9),  DSQ_COMM_LEVEL        ),            00070000
     +              (DSQCOMM(21),  DSQ_PRODUCT           ),            00071000
     +              (DSQCOMM(23),  DSQ_PRODUCT_RELEASE   ),            00072000
     +              (DSQCOMM(25),  DSQ_RESERVE1          ),            00073000
     +              (DSQCOMM(53),  DSQ_MESSAGE_ID        ),            00074000
     +              (DSQCOMM(61),  DSQ_Q_MESSAGE_ID      ),            00075000
     +              (DSQCOMM(69),  DSQ_START_PARM_ERROR  ),            00076000
     +              (DSQCOMM(77),  DSQ_CANCEL_IND        ),            00077000
     +              (DSQCOMM(78),  DSQ_RESERVE2          ),            00078000
     +              (DSQCOMM(101), DSQ_RESERVE3          ),            00079000
     +              (DSQCOMM(257), DSQ_MESSAGE_TEXT      ),            00080000
     +              (DSQCOMM(385), DSQ_Q_MESSAGE_TEXT    )             00081000
                                                                       00082000
C        Callable Interface Normal and Extended Calls                  00083000
      EXTERNAL DSQCIF                                                  00084000
      EXTERNAL DSQCIFE                                                 00085000
```

Figure 53. DSQCOMMF, FORTRAN communications area (Part 2 of 2)

Compiling and running your programs under CMS in VM

The following program compiles and runs your callable interface application using the VS FORTRAN compiler. QMF does not ship the REXX EXEC in this example, but you can copy it from here, altering it to suit your installation.

```
/*****************************************************************/
/* Compile your program and run it.       */
/*****************************************************************/
TRACE off
ADDRESS CMS

/*****************************************************************/
/* Access FORTRAN product disk using a program, PRODUCT, that you */
/* write.                                                        */
/*****************************************************************/
"EXEC PRODUCT FORTRAN"

/*****************************************************************/
/* Get QMF DSQCOMM into a macro library and set GLOBAL compile    */
/* maclibs.
/*****************************************************************/
"ERASE TEMPP MACLIB A"
"MACLIB GEN TEMPP DSQCOMMF"
Maclist = "TEMPP VSF2PLIB VSF2MLIB DMSSP CMSLIB OSMACRO"
"GLOBAL MACLIB" Maclist

/*****************************************************************/
/* Compile the program                                           */
/*****************************************************************/
'FORTVS2 yourname (RENT OPT(0) XREF'

/*****************************************************************/
/* Access SQL/DS and initialize database                         */
/*****************************************************************/
"EXEC PRODUCT SQLDS"
"EXEC SQLINIT DBNAME(SQLDBA)"

/*****************************************************************/
/* Access GDDM product disk                                      */
/*****************************************************************/
"EXEC PRODUCT GDDM"
```

Figure 54. REXX program to compile and run your program (Part 1 of 2)

```
/*****************************************************************/
/* Issue Filedefs for QMF product                                */
/*****************************************************************/
/* DEBUG = DDNAME FOR QMF DIAGNOSTICS OUTPUT       */
"FILEDEF DSQDEBUG PRINTER ( LRECL 80  BLKSIZE 80  RECFM FBA PERM"
/* PRINT = DDNAME FOR QMF PRINTED OUTPUT                         */
"FILEDEF DSQPRINT PRINTER ( LRECL 133 BLKSIZE 133 RECFM FBA PERM"
/* EDIT  = DDNAME FOR QMF EDIT TRANSFER FILE                     */
"FILEDEF DSQEDIT DISK QMFEDIT FILE A (PERM"
/* DSQSIDE = DDNAME FOR QMF SPILL FILE                           */
"FILEDEF DSQSPILL DISK DSQSIDE DATA A1 (PERM"
/* DSQPNLE = DDNAME FOR PANEL FILE                               */
"FILEDEF DSQPNLE DISK DSQPNLE FILE * (PERM"
"FILEDEF ISPLLIB CLEAR"
"FILEDEF ISPLLIB DISK DSQLDLIB LOADLIB *"

/*****************************************************************/
/* Provide access to QMF and FORTRAN program libraries           */
/*****************************************************************/
'GLOBAL LOADLIB VSF2LOAD DSQLDLIB'
'GLOBAL TXTLIB VSF2LINK VSF2FORT ADMRLIB ADMPLIB ADMGLIB'
Say "Starting to run FORTRAN program"
"RUN yourname"

Exit 0
```

Figure 54. REXX program to compile and run your program (Part 2 of 2)

You might have to modify this program to suit your installation.

Running your programs under TSO in MVS

After you write your program, you need to compile and link-edit it as required before you can run it. The programs listed in this section show the steps necessary to do so.

QMF does not ship the REXX EXEC, JCL, or CLIST in these examples, but you can copy them from here, altering them to suit your installation.

Compiling and link-editing in TSO

The following job compiles and link-edits your callable interface application using the VS FORTRAN compiler for MVS. Some parameters can vary from one installation to the next. See your QMF administrator for details.

```
//samFORT    JOB
//STEP1      EXEC PROC=VSF2CL
//* Provide Access to QMF Communications Macro DSQCOMM
//FORT.SYSLIB   DD  DSN=QMF720.SAMPLIB,DISP=SHR
//FORT.SYSIN    DD  *
          .
          Your program or copy of QMF sample DSQABFF
          .
/*
//* Provide Access to QMF Interface Module
//LKED.QMFLOAD  DD DSN=QMF720.SDSQLOAD,DISP=SHR
//LKED.SYSIN    DD  *
          INCLUDE QMFLOAD(DSQCIF)
          INCLUDE QMFLOAD(DSQCIFE)
          ENTRY samFORT
          MODE  AMODE(31) RMODE(ANY)
          NAME  samFORT(R)
/*
```

Figure 55. JCL for running the FORTRAN compiler and linkage editor

Running your programs in TSO without ISPF

The following program runs your callable interface application using the VS FORTRAN compiler. Some parameters can vary from one installation to the next. See your QMF administrator for details.

```
PROC 0
CONTROL ASIS
/*************************************************************/
/* Note: QMF, DB2, GDDM and FORTRAN load libraries must be  */
/*       allocated before running this CLIST.                */
/*       Name of QMF load library is  "QMF720.SDSQLOAD".     */
/*************************************************************/
/* Specify attribute list for dataset allocations            */
/*************************************************************/
ATTR PRINTDCB LRECL(133) RECFM(F B A) BLKSIZE(1330)
ATTR DEBUGDCB LRECL(80)  RECFM(F B)   BLKSIZE(3120)
ATTR UDUMPDCB LRECL(125) RECFM(V B A) BLKSIZE(1632)
ATTR EDITDCB  LRECL(79)  RECFM(F B A) BLKSIZE(4029)
/*************************************************************/
/* Datasets used by TSO                                      */
/*************************************************************/
ALLOC FI(SYSPROC) DA('QMF720.SDSQCLTE')
ALLOC FI(SYSEXEC) DA('QMF720.SDSQEXCE')
/*************************************************************/
/* QMF/GDDM Datasets                                         */
/*************************************************************/
ALLOC FI(ADMGGMAP) DA('QMF720.QMFMAPS')  SHR REUSE
ALLOC FI(ADMCFORM) DA('QMF720.DSQCFORM') SHR REUSE
ALLOC FI(DSQUCFRM) DA('QMF720.DSQUCFRM') SHR REUSE
ALLOC FI(ADMSYMBL) DA('ADM.GDDMSYM')  SHR REUSE
ALLOC FI(ADMGDF)   DA('ADM.GDDM.CHARTLIB') SHR REUSE
ALLOC FI(ADMDEFS)  DA('ADM.GDDM.NICKNAME') SHR REUSE
/*************************************************************/
/* Datasets used by QMF                                      */
/*************************************************************/
ALLOC FI(DSQPRINT) SYSOUT(X) USING(PRINTDCB)
ALLOC FI(DSQDEBUG) SYSOUT(X) USING(DEBUGDCB)
ALLOC FI(DSQUDUMP) SYSOUT(X) USING(UDUMPDCB)
ALLOC FI(DSQSPILL) NEW UNIT(SYSDA) SPACE(1,1) TRACKS
ALLOC FI(DSQEDIT)  NEW UNIT(SYSDA) USING(EDITDCB)
ALLOC FI(DSQPNLE)  DA('QMF720.DSQPNLE') SHR REUSE
/*************************************************************/
/* Start your program using TSO CALL command                */
/*************************************************************/
CALL samFORT
EXIT CODE(0)
```

Figure 56. CLIST for running your program in TSO without ISPF

Running in TSO under ISPF

The following program runs your callable interface application using the VS FORTRAN compiler. Some parameters can vary from one installation to the next. See your QMF administrator for details.

```
PROC 0
CONTROL ASIS
/*************************************************************/
/* Specify attribute list for dataset allocations            */
/*************************************************************/
ATTR PRINTDCB LRECL(133) RECFM(F B A) BLKSIZE(1330)
ATTR DEBUGDCB LRECL(80)  RECFM(F B)   BLKSIZE(3120)
ATTR UDUMPDCB LRECL(125) RECFM(V B A) BLKSIZE(1632)
ATTR EDITDCB  LRECL(79)  RECFM(F B A) BLKSIZE(4029)
/*************************************************************/
/* Datasets used by TSO                                      */
/*************************************************************/
ALLOC FI(SYSPROC) DA('QMF720.SDSQCLTE','ISR.ISRCLIB')
ALLOC FI(SYSEXEC) DA('QMF720.SDSQEXCE')
/*************************************************************/
/* Datasets used by ISPF                                     */
/*************************************************************/
ALLOC FI(ISPLLIB) SHR REUSE +
      DA('QMF720.SDSQLOAD','ADM.GDDMLOAD','DSN.DSNEXIT','DSN.DSNLOAD', +
         'PRDUCT.VSF2LOAD')
ALLOC FI(ISPMLIB) SHR REUSE +
      DA('QMF720.SDSQMLBE','ISR.ISRMLIB','ISP.ISPMLIB')
ALLOC FI(ISPPLIB) SHR REUSE +
      DA('QMF720.SDSQPLBE','ISR.ISRPLIB','ISP.ISPPLIB')
ALLOC FI(ISPSLIB) SHR REUSE +
      DA('QMF720.SDSQSLBE','ISR.ISRSLIB','ISP.ISPSLIB')
ALLOC FI(ISPTLIB) SHR REUSE +
      DA('ISR.ISRTLIB','ISP.ISPTLIB')
/*************************************************************/
/* QMF/GDDM Datasets                                         */
/*************************************************************/
ALLOC FI(ADMGGMAP) DA('QMF720.QMFMAPS')  SHR REUSE
ALLOC FI(ADMCFORM) DA('QMF720.DSQCFORM') SHR REUSE
ALLOC FI(DSQUCFRM) DA('QMF720.DSQUCFRM') SHR REUSE
ALLOC FI(ADMSYMBL) DA('ADM.GDDMSYM')  SHR REUSE
ALLOC FI(ADMGDF)   DA('ADM.GDDM.CHARTLIB') SHR REUSE
ALLOC FI(ADMDEFS)  DA('ADM.GDDM.NICKNAME') SHR REUSE
/*************************************************************/
/* Datasets used by QMF                                      */
/*************************************************************/
ALLOC FI(DSQPRINT) SYSOUT(X) USING(PRINTDCB)
ALLOC FI(DSQDEBUG) SYSOUT(X) USING(DEBUGDCB)
ALLOC FI(DSQUDUMP) SYSOUT(X) USING(UDUMPDCB)
ALLOC FI(DSQSPILL) NEW UNIT(SYSDA) SPACE(1,1) TRACKS
ALLOC FI(DSQEDIT)  NEW UNIT(SYSDA) USING(EDITDCB)
ALLOC FI(DSQPNLE)  DA('QMF720.DSQPNLE') SHR REUSE
/*************************************************************/
/* Start your program as the initial ISPF dialog             */
/*************************************************************/
ISPSTART PGM(samFORT) NEWAPPL(DSQE)
EXIT CODE(4)
```

Figure 57. CLIST for running your program in TSO under ISPF

The EXIT CODE(4) suppresses the showing of the ISPF disposition panel.

PL/I language interface

The PL/I callable interface corresponds to that provided for other SAA languages.

The minimum release level of PL/I required for use with QMF in CICS is PL/I Version 2. PL/I Version 2 is not supported in VSE/ESA.

Interface communications area mapping for PL/I (DSQCOMML)

DSQCOMML provides DSQCOMM mapping for PL/I and is shipped with the product. Table 19 shows the values for DSQCOMML.

Table 19. Interface communications area for DSQCOMML

Structure Name	Data Type	Description
DSQ_RETURN_CODE	FIXED BIN(31)	Indicates the status of a QMF command after it has been run. Its values are: **DSQ_SUCCESS** Successful execution of the request. **DSQ_WARNING** Normal completion with warnings. **DSQ_FAILURE** Command did not run correctly. **DSQ_SEVERE** Severe error; QMF session terminated.
DSQ_INSTANCE_ID	FIXED BIN(31)	Identifier established by QMF during execution of the START command
DSQ_COMM_LEVEL	CHAR(12)	Identifies the level of the DSQCOMM. You should set this to the value of DSQ_CURRENT_COMM_LEVEL before issuing the QMF START command.
DSQ_PRODUCT	CHAR(2)	Identifies the IBM query product in use.
DSQ_PRODUCT_RELEASE	CHAR(2)	Identifies the release level of the query product in use.
DSQ_RESERVE1	CHAR(28)	Reserved for future use
DSQ_MESSAGE_ID	CHAR(8)	Completion message ID
DSQ_Q_MESSAGE_ID	CHAR(8)	Query message ID
DSQ_START_PARM_ERROR	CHAR(8)	Parameter in error when START failed due to a parameter error
DSQ_CANCEL_IND	CHAR(1)	Contains one of two values, depending if the user canceled while a QMF command was running: • DSQ_CANCEL_YES • DSQ_CANCEL_NO

Table 19. Interface communications area for DSQCOMML (continued)

Structure Name	Data Type	Description
DSQ_RESERVE2	CHAR(23)	Reserved for future use
DSQ_RESERVE3	CHAR(156)	Reserved for future use
DSQ_MESSAGE_TEXT	CHAR(128)	Completion message text
DSQ_Q_MESSAGE_TEXT	CHAR(128)	Query message text

Function calls for PL/I

QMF provides two function calls for PL/I: DSQCIPL and DSQCIPX. Both calls are described in the communications macro DSQCOMML.

DSQCIPL syntax

This call is for QMF commands that do not require access to application program variables. Use this call for most QMF commands.

```
CALL DSQCIPL(DSQCOMM,
             CMDLTH,
             CMDSTR)
```

The parameters have the following values:

DSQCOMM
The interface communications area.

CMDLTH
Length of the command string CMDSTR.

CMDSTR
QMF command to run; it is an uppercase character string of the length specified by CMDLTH.

DSQCIPX Syntax

This call is for the three commands that do require access to application program variables: START and the extended formats of GET GLOBAL and SET GLOBAL.

The syntax for this call is:

```
CALL DSQCIPX(DSQCOMM,
             CMDLTH,
             CMDSTR,
             PNUM,
             KLTH,
             KWORD,
             VLTH,
             VALUE,
             VTYPE)
```

The parameters have the following values:

DSQCOMM
The interface communications area.

CMDLTH
Length of the command string CMDSTR. It is an integer FIXED BIN(31) parameter.

CMDSTR
QMF command to run. It is an uppercase character string of the length specified by CMDLTH.

PNUM
Number of command keywords. It is an integer FIXED BIN(31) parameter.

KLTH
Length of each specified keyword. It is an integer FIXED BIN(31) parameter or parameter array.

KWORD
QMF keyword or keywords. Each is a character or structure of characters whose lengths are the same as specified by KLTH. You can use an array of characters, if all of the keywords have the same length. QMF assumes that the keywords are in contiguous storage and are not separated by any special separator characters.

VLTH
Length of each value associated with the keyword. It is an integer FIXED BIN(31) parameter or parameter array.

VALUE
Value associated with each keyword. Its type is specified in the VTYPE parameter, and can be a character, structure of characters, integer FIXED BIN(31) parameter, or parameter array. If you have character values, QMF assumes that the values are in contiguous storage, not separated by any special separator characters.

VTYPE
QMF data type of the value string VALUE. VTYPE can have one of two values, which are provided in the communications macro, DSQCOMML:

- DSQ_VARIABLE_CHAR for character values.
- DSQ_VARIABLE_FINT for integer FIXED BIN(31) values

All of the values specified in the VALUE field must have the data type specified in VTYPE.

Migration information for users of CICS in MVS

The DSQCIPL and DSQCIPLX calls have changed from Version 3 Release 1 Modification 1 to Version 3 Release 2. The interface between the QMF-supplied function call and the main QMF program has changed from a CALL interface to an EXEC CICS LINK interface. The new interface provides

better isolation from the user program and the QMF product. Because the interface has changed, if you are migrating from Version 3 Release 1 or earlier, you need to link-edit your programs again that used the callable interface.

PL/I programming example

The following sample program, DSQABFP, is shipped with QMF and uses IBM PL/I.

You can look at the sample source code listing here or you can access it online.

- For VM, the sample program is on the production disk.
- For OS/390, the sample program is a member of the library QMF720.SDSQSAPE.
- For use with QMF in CICS, the minimum release level of PL/I required is Version 2. PL/I Version 2 is not supported in VSE/ESA.

The sample program for the PL/I callable interface performs the following function:

- Starts QMF
- Sets three global variables
- Runs a query called Q1
- Prints the resulting report using form F1
- Ends the QMF session

QMF does not supply query Q1 or form F1, but the sample program uses these objects.

This section also shows how to compile, link-edit, and run a PL/I program using the callable interface. QMF does not ship the REXX EXEC, JCL, or CLIST in these examples, but you can copy them from here, altering them to suit your installation.

```
DSQABFP: PROCEDURE OPTIONS(MAIN REENTRANT) REORDER;                    00001000
/*********************************************************************/ 00002000
/* Sample Program:  DSQABFP                                          */ 00003000
/* PL/I Version of the SAA Query Callable Interface                  */ 00004000
/*********************************************************************/ 00005000
                                                                       00006000
/*********************************************************************/ 00007000
/* Include and declare query interface communications area           */ 00008000
/*********************************************************************/ 00009000
%INCLUDE SYSLIB(DSQCOMML);                                             00010000
                                                                       00011000
/*********************************************************************/ 00012000
/* Builtin function                                                  */ 00013000
/*********************************************************************/ 00014000
DCL LENGTH BUILTIN;                                                    00015000
                                                                       00016000
/*********************************************************************/ 00017000
/* Query interface command length and commands                       */ 00018000
/*********************************************************************/ 00019000
DCL COMMAND_LENGTH         FIXED BIN(31);                              00020000
DCL START_QUERY_INTERFACE  CHAR(5)  INIT('START');                     00021000
DCL SET_GLOBAL_VARIABLES   CHAR(10) INIT('SET GLOBAL');                00022000
DCL RUN_QUERY              CHAR(12) INIT('RUN QUERY Q1');              00023000
DCL PRINT_REPORT           CHAR(22) INIT('PRINT REPORT (FORM=F1)');    00024000
DCL END_QUERY_INTERFACE    CHAR(4)  INIT('EXIT');                      00025000
                                                                       00026000
/*********************************************************************/ 00027000
/* Query command extension, number of parameters and lengths         */ 00028000
/*********************************************************************/ 00029000
DCL NUMBER_OF_PARAMETERS   FIXED BIN(31);/* number of variables     */ 00030000
DCL KEYWORD_LENGTHS(10)    FIXED BIN(31);/* lengths of keyword names*/ 00031000
DCL DATA_LENGTHS(10)       FIXED BIN(31);/* lengths of variable data*/ 00032000
                                                                       00033000
```

Figure 58. DSQABFP, sample PL/I program (Part 1 of 3)

```
/*********************************************************************/ 00034000
/* Keyword parameter and value for START command                     */ 00035000
/*********************************************************************/ 00036000
DCL START_KEYWORDS         CHAR(8)  INIT('DSQSMODE');                   00037000
DCL START_KEYWORD_VALUES   CHAR(11) INIT('INTERACTIVE');                00038000
                                                                        00039000
/*********************************************************************/ 00040000
/* Keyword parameter and value for SET command                       */ 00041000
/*********************************************************************/ 00042000
DCL 1 SET_KEYWORDS,                                                     00043000
      3  SET_KEYWORDS_1    CHAR(7) INIT('MYVAR01'),                     00044000
      3  SET_KEYWORDS_2    CHAR(5) INIT('SHORT'),                       00045000
      3  SET_KEYWORDS_3    CHAR(7) INIT('MYVAR03');                     00046000
                                                                        00047000
DCL 1 SET_VALUES,                                                       00048000
      3  SET_VALUES_1      FIXED BIN(31),                               00049000
      3  SET_VALUES_2      FIXED BIN(31),                               00050000
      3  SET_VALUES_3      FIXED BIN(31);                               00051000
                                                                        00052000
/*********************************************************************/ 00053000
/* Main program                                                      */ 00054000
/*********************************************************************/ 00055000
DSQCOMM = '';                                                           00056000
DSQ_COMM_LEVEL = DSQ_CURRENT_COMM_LEVEL;                                00057000
                                                                        00058000
/*********************************************************************/ 00059000
/* Start a query interface session                                   */ 00060000
/*********************************************************************/ 00061000
NUMBER_OF_PARAMETERS = 1;                                               00062000
COMMAND_LENGTH = LENGTH(START_QUERY_INTERFACE);                         00063000
KEYWORD_LENGTHS(1) = LENGTH(START_KEYWORDS);                            00064000
DATA_LENGTHS(1) = LENGTH(START_KEYWORD_VALUES);                         00065000
                                                                        00066000
CALL DSQCIPX(DSQCOMM,                                                   00067000
             COMMAND_LENGTH,                                            00068000
             START_QUERY_INTERFACE,                                     00069000
             NUMBER_OF_PARAMETERS,                                      00070000
             KEYWORD_LENGTHS,                                           00071000
             START_KEYWORDS,                                            00072000
             DATA_LENGTHS,                                              00073000
             START_KEYWORD_VALUES,                                      00074000
             DSQ_VARIABLE_CHAR);                                        00075000
                                                                        00076000
```

Figure 58. DSQABFP, sample PL/I program (Part 2 of 3)

```
/*****************************************************************/ 00077000
/* Set numeric values into query using SET command               */ 00078000
/*****************************************************************/ 00079000
NUMBER_OF_PARAMETERS = 3;                                          00080000
COMMAND_LENGTH = LENGTH(SET_GLOBAL_VARIABLES);                     00081000
KEYWORD_LENGTHS(1) = LENGTH(SET_KEYWORDS_1);                       00082000
KEYWORD_LENGTHS(2) = LENGTH(SET_KEYWORDS_2);                       00083000
KEYWORD_LENGTHS(3) = LENGTH(SET_KEYWORDS_3);                       00084000
DATA_LENGTHS(1) = 4;                                               00085000
DATA_LENGTHS(2) = 4;                                               00086000
DATA_LENGTHS(3) = 4;                                               00087000
SET_VALUES_1 = 20;                                                 00088000
SET_VALUES_2 = 40;                                                 00089000
SET_VALUES_3 = 84;                                                 00090000
                                                                   00091000
CALL DSQCIPX(DSQCOMM,                                              00092000
             COMMAND_LENGTH,                                       00093000
             SET_GLOBAL_VARIABLES,                                 00094000
             NUMBER_OF_PARAMETERS,                                 00095000
             KEYWORD_LENGTHS,                                      00096000
             SET_KEYWORDS,                                         00097000
             DATA_LENGTHS,                                         00098000
             SET_VALUES,                                           00099000
             DSQ_VARIABLE_FINT);                                   00100000
                                                                   00101000
/*****************************************************************/ 00102000
/* Run a Query                                                   */ 00103000
/*****************************************************************/ 00104000
COMMAND_LENGTH = LENGTH(RUN_QUERY);                                00105000
                                                                   00106000
CALL DSQCIPL(DSQCOMM,                                              00107000
            COMMAND_LENGTH,                                        00108000
            RUN_QUERY);                                            00109000
                                                                   00110000
/*****************************************************************/ 00111000
/* Print the results of the query                                */ 00112000
/*****************************************************************/ 00113000
COMMAND_LENGTH = LENGTH(PRINT_REPORT);                             00114000
                                                                   00115000
CALL DSQCIPL(DSQCOMM,                                              00116000
            COMMAND_LENGTH,                                        00117000
            PRINT_REPORT);                                         00118000
                                                                   00119000
/*****************************************************************/ 00120000
/* End the query interface session                               */ 00121000
/*****************************************************************/ 00122000
COMMAND_LENGTH = LENGTH(END_QUERY_INTERFACE);                      00123000
                                                                   00124000
CALL DSQCIPL(DSQCOMM,                                              00125000
            COMMAND_LENGTH,                                        00126000
            END_QUERY_INTERFACE);                                  00127000
                                                                   00128000
END       DSQABFP;                                                 00129000
```

Figure 58. DSQABFP, sample PL/I program (Part 3 of 3)

DSQCOMM for PL/I

```
/*****************************************************************/ 00001000
/* PL/I include for Query Callable Interface (MVS/VM)            */ 00002000
/*****************************************************************/ 00003000
                                                                    00004000
/* Structure declare for Communications Area                     */ 00005000
DCL                                                                 00006000
 1 DSQCOMM,                                                         00007000
   3 DSQ_RETURN_CODE      FIXED BIN(31),   /* function return code  */ 00008000
   3 DSQ_INSTANCE_ID      FIXED BIN(31),   /* start ID              */ 00009000
   3 DSQ_COMM_LEVEL       CHAR(12),        /* communications level  */ 00010000
   3 DSQ_PRODUCT          CHAR(2),         /* query product id      */ 00011000
   3 DSQ_PRODUCT_RELEASE  CHAR(2),         /* query product release */ 00012000
   3 DSQ_RESERVE1         CHAR(28),        /* reserved              */ 00013000
   3 DSQ_MESSAGE_ID       CHAR(8),         /* completion message ID */ 00014000
   3 DSQ_Q_MESSAGE_ID     CHAR(8),         /* query message ID      */ 00015000
   3 DSQ_START_PARM_ERROR CHAR(8),         /* start parms in error  */ 00016000
   3 DSQ_CANCEL_IND       CHAR(1),         /* cmd cancel  indicator */ 00017000
                             /* 1 = cancelled, 0 = not cancelled*/ 00018000
   3 DSQ_RESERVE2         CHAR(23),        /* reserved              */ 00019000
   3 DSQ_RESERVE3         CHAR(156),       /* reserved              */ 00020000
   3 DSQ_MESSAGE_TEXT     CHAR(128),       /* QMF command message   */ 00021000
   3 DSQ_Q_MESSAGE_TEXT   CHAR(128);       /* QMF query message     */ 00022000
                                                                    00023000
/* Return Codes                                                  */ 00024000
DCL                                                                 00025000
   DSQ_SUCCESS             FIXED BIN(31) INIT(0)  STATIC,           00026000
   DSQ_WARNING             FIXED BIN(31) INIT(4)  STATIC,           00027000
   DSQ_FAILURE             FIXED BIN(31) INIT(8)  STATIC,           00028000
   DSQ_SEVERE              FIXED BIN(31) INIT(16) STATIC;           00029000
                                                                    00030000
/* Communications Level                                          */ 00031000
DCL                                                                 00032000
   DSQ_CURRENT_COMM_LEVEL  CHAR(12) INIT('DSQL>001002<') STATIC;    00033000
                                                                    00034000
/* Query Product ID                                              */ 00035000
DCL                                                                 00036000
   DSQ_QRW                 CHAR(2) INIT('01') STATIC,               00037000
   DSQ_QMF                 CHAR(2) INIT('02') STATIC,               00038000
   DSQ_QM4                 CHAR(2) INIT('03') STATIC;               00039000
                                                                    00040000
```

Figure 59. DSQCOMML, PL/I communications area (Part 1 of 2)

```
/* Query Product Release ID                                          */ 00041000
DCL                                                                     00042000
   DSQ_QRW_V1R2            CHAR(2) INIT('01') STATIC,                   00043000
   DSQ_QRW_V1R3            CHAR(2) INIT('02') STATIC,                   00044000
   DSQ_QMF_V2R4            CHAR(2) INIT('01') STATIC,                   00045000
   DSQ_QMF_V3R1            CHAR(2) INIT('02') STATIC,                   00046000
   DSQ_QMF_V3R1M1          CHAR(2) INIT('03') STATIC,                   00047000
   DSQ_QMF_V3R2            CHAR(2) INIT('04') STATIC,                   00048000
   DSQ_QMF_V3R3            CHAR(2) INIT('05') STATIC,                   00049000
   DSQ_QMF_V6R1            CHAR(2) INIT('06') STATIC,                   00050000
   DSQ_QM4_V1R1            CHAR(2) INIT('01') STATIC;                   00051000
                                                                        00052000
/* Cancelled Indicator                                               */ 00053000
DCL                                                                     00054000
   DSQ_CANCEL_YES          CHAR(1) INIT('1') STATIC,                    00055000
   DSQ_CANCEL_NO           CHAR(1) INIT('0') STATIC;                    00056000
                                                                        00057000
/* Variable Types                                                    */ 00058000
DCL                                                                     00059000
   DSQ_VARIABLE_CHAR       CHAR(4) INIT('CHAR') STATIC,                 00060000
   DSQ_VARIABLE_FINT       CHAR(4) INIT('FINT') STATIC;                 00061000
                                                                        00062000
/* Mode                                                              */ 00063000
DCL                                                                     00064000
   DSQ_INTERACTIVE         CHAR(1) INIT('1') STATIC,                    00065000
   DSQ_BATCH               CHAR(1) INIT('2') STATIC;                    00066000
                                                                        00067000
/* Yes or No                                                         */ 00068000
DCL                                                                     00069000
   DSQ_YES                 CHAR(1) INIT('1') STATIC,                    00070000
   DSQ_NO                  CHAR(1) INIT('2') STATIC;                    00071000
                                                                        00072000
/* Query Interface Entry Point                                       */ 00073000
DCL                                                                     00074000
   DSQCIPL ENTRY (*,                     /* interface block       */ 00075000
                  FIXED BIN(31),         /* length of command     */ 00076000
                  CHAR(*))               /* command string        */ 00077000
                  EXTERNAL OPTIONS(ASSEMBLER);                          00078000
DCL                                                                     00079000
   DSQCIPX ENTRY (*,                     /* interface block       */ 00080000
                  FIXED BIN(31),         /* length of command     */ 00081000
                  CHAR(*),               /* command string        */ 00082000
                  FIXED BIN(31),         /* # of command keywords */ 00083000
                  *,                     /* length of keyword     */ 00084000
                  *,                     /* keyword string        */ 00085000
                  *,                     /* length of value       */ 00086000
                  *,                     /* value of keyword      */ 00087000
                  CHAR(4))               /* data type of value    */ 00088000
                  EXTERNAL OPTIONS(ASSEMBLER);                          00089000
```

Figure 59. DSQCOMML, PL/I communications area (Part 2 of 2)

Running your programs under CICS

After you write your program, you need to compile and run it. The examples listed in this section show the steps necessary to do so.

QMF does not ship the REXX EXEC, JCL, or CLIST in these examples, but you can copy them from here, altering them to suit your installation.

Translating, compiling, and link-editing under CICS in MVS

When you translate, compile, and link-edit a program that uses the QMF callable interface, consider the following:

- The communications area DSQCOMML must be available to the compile step or copied into your program.
- The QMF interface modules DSQCIPL and DSQCIPX must be available during the link-edit phase of your program.

The following is an example using the CICS-supplied procedure DFHEBTPL. For instructions on how to use this procedure, see your release of *CICS for VSE/ESA System Definition Guide*.

```
//samPLI    JOB
//          EXEC PROC=DFHEBTPL
//TRN.SYSIN DD  *
*PROCESS  XOPTS(CICS translator options .....)
          .
          Your program or copy of QMF sample DSQABFP
          .
/*
//* Provide Access to QMF Communications Macro DSQCOMML
//PLI.SYSLIB   DD DSN=QMF720.SDSQSAPE,DISP=SHR
//* Provide Access to QMF Interface Module
//LKED.QMFLOAD DD DSN=QMF720.SDSQLOAD,DISP=SHR
//LKED.SYSIN   DD *
          INCLUDE CICSLOAD(DFHPL1OI)
          INCLUDE CICSLOAD(DFHEPI)
          INCLUDE QMFLOAD(DSQCIPL)
          INCLUDE QMFLOAD(DSQCIPX)
          ORDER DFHPL1OI,DFHEPI
          ENTRY sampPLI
          MODE  AMODE(31) RMODE(ANY)
          NAME  sampPLI(R)
/*
```

Figure 60. JCL for running the CICS translator, PL/I compiler, and linkage editor

Translating, compiling, and link-editing under CICS in VSE

The VSE job control in Figure 61 on page 210 is an example of installing a PL/I program into CICS running on VSE. This example, provided with QMF, is located in the QMF sublibrary and is named DSQ3CIP.Z. See the *CICS for*

VSE/ESA System Definition Guide and the *PL/I VSE Programming Guide* for detailed information.

```
..* $$ JOB JNM=DSQ3CIP,DISP=D,CLASS=0
// JOB DSQ3CIP    Sample job to Install QMF Callable Interface (PL/I)
*  ---------------------------------------------------------------------
*  Install QMF Callable Interface Example (PL/I)
*  ---------------------------------------------------------------------
// SETPARM VOLID=volid      *-- update volid for syspch
// SETPARM START=rtrk       *-- update start track/block (syspch)
// SETPARM SIZE=ntrks       *-- update number of tracks/blocks (syspch)
*  ---------------------------------------------------------------------
// DLBL   IJSYSPH,'CICS.TRANSLAT.OUTPUT',0
// EXTENT SYSPCH,,1,0,&START,&SIZE
ASSGN SYSPCH,DISK,VOL=&VOLID,SHR
*  Library search chain must contain the QMF, CICS and PL/I sublibrary
// LIBDEF *,SEARCH=(PRD2.PROD,PRD1.BASE,PRD2.CONFIG)
// LIBDEF PHASE,CATALOG=PRD2.PROD
*  ---------------------------------------------------------------------
*  Step 1: Translate callable interface program (PL/I)
*  ---------------------------------------------------------------------
*  You may need to update or remove the SLI statement for your program.
*  ---------------------------------------------------------------------
// EXEC   DFHEPP1$,SIZE=256K,PARM='XOPTS(CICS)'
..* $$ SLI MEM=DSQABFP.Z,S=PRD2.PROD
/*
*  ---------------------------------------------------------------------
*  Step 2: Compile callable interface program (PL/I)
*  ---------------------------------------------------------------------
CLOSE SYSPCH,00D
// DLBL   IJSYSIN,'CICS.TRANSLAT.OUTPUT',0
// EXTENT SYSIPT
ASSGN SYSIPT,DISK,VOL=&VOLID,SHR
// OPTION NODECK,CATAL
   PHASE DSQABFP,*
         INCLUDE DFHPL1I
// EXEC PLIOPT
CLOSE SYSIPT,SYSRDR
/*
```

Figure 61. Sample JCL for VSE (Part 1 of 2)

```
*  ------------------------------------------------------------------
*  Step 3: Link-edit callable interface program (PL/I)
*  ------------------------------------------------------------------
// EXEC LNKEDT,PARM='AMODE=31,RMODE=ANY'
/*
/&
// JOB  RESET
ASSGN SYSIPT,SYSRDR    IF 1A93D, CLOSE SYSIPT,SYSRDR
ASSGN SYSPCH,00D       IF 1A93D, CLOSE SYSPCH,00D
/&
..* $$ EOJ
```

Figure 61. Sample JCL for VSE (Part 2 of 2)

Compiling and running your programs under CMS in VM

The following program compiles and runs your callable interface application using the PL/I compiler.

QMF does not ship the REXX EXEC in this example, but you can copy it from here, altering it to suit your installation.

```
/*********************************************************************/
/* Compile QMF PL/I program and run it.          */
/*********************************************************************/
TRACE off
ADDRESS CMS

/*********************************************************************/
/* Access PL/I product disk using a program, PRODUCT, that you    */
/* write.                                                          */
/*********************************************************************/
"EXEC PRODUCT PLIV"

/*********************************************************************/
/* Get QMF DSQCOMM into a macro library and set GLOBAL compile    */
/* maclibs.                                                        */
/*********************************************************************/
"ERASE TEMPP MACLIB A"
"MACLIB GEN TEMPP DSQCOMML"
Maclist = "TEMPP PLICOMP DMSSP CMSLIB OSMACRO"
"GLOBAL MACLIB" Maclist

/*********************************************************************/
/* Compile the program                                             */
/*********************************************************************/
POPTS = '(INC SOURCE LIST LMSG M NAG NC(E) NIS NOESD NSTG OPT(2)'
'PLIOPT' yourname popts

/*********************************************************************/
/* Access SQL/DS and initialize database                           */
/*********************************************************************/
"EXEC PRODUCT SQLDS"
"EXEC SQLINIT DBNAME(SQLDBA)"

/*********************************************************************/
/* Access GDDM product disk                                        */
/*********************************************************************/
"EXEC PRODUCT GDDM"
```

Figure 62. REXX program to compile and run your program (Part 1 of 2)

```
/*****************************************************************/
/* Issue Filedefs for QMF product                                */
/*****************************************************************/
"FILEDEF ISPLLIB CLEAR"
/* DEBUG = DDNAME FOR QMF DIAGNOSTICS OUTPUT       */
"FILEDEF DSQDEBUG PRINTER ( LRECL 80  BLKSIZE 80  RECFM FBA PERM"
/* PRINT = DDNAME FOR QMF PRINTED OUTPUT                        */
"FILEDEF DSQPRINT PRINTER ( LRECL 133 BLKSIZE 133 RECFM FBA PERM"
/* EDIT  = DDNAME FOR QMF EDIT TRANSFER FILE                    */
"FILEDEF DSQEDIT DISK QMFEDIT FILE A (PERM"
/* DSQSIDE = DDNAME FOR QMF SPILL FILE                          */
"FILEDEF DSQSPILL DISK DSQSIDE DATA A1 (PERM"
/* DSQPNLE = DDNAME FOR PANEL FILE                              */
"FILEDEF DSQPNLE DISK DSQPNLE FILE * (PERM"
"FILEDEF ISPLLIB CLEAR"
"FILEDEF ISPLLIB DISK DSQLDLIB LOADLIB *"

/*****************************************************************/
/* Provide access to QMF and PL/I program libraries              */
/*****************************************************************/
'GLOBAL MACLIB TEMPP'
'GLOBAL LOADLIB DSQLDLIB PLILIB'
'GLOBAL TXTLIB PLILIB IBMLIB ADMRLIB ADMPLIB ADMGLIB'


Say "Starting to run PL/I program"
"RUN yourname"

Exit 0
```

Figure 62. REXX program to compile and run your program (Part 2 of 2)

You might have to modify this program to suit your installation.

Compiling and link-editing in TSO

The following job uses the PL/I compiler to compile your callable interface application and then link-edits the application. Some parameters can vary from one installation to the next. See your QMF administrator for details.

```
//samPLI     JOB
//STEP1      EXEC IEL1CL
//* Provide Access to QMF Communications Macro DSQCOMML
//PLI.SYSLIB   DD  DSN=QMF720.SAMPLIB,DISP=SHR
//PLI.SYSIN    DD  *
          .
          Your program or copy of QMF sample DSQABFP
          .
/*
//* Provide Access to QMF Interface Module
//LKED.QMFLOAD  DD DSN=QMF720.SDSQLOAD,DISP=SHR
//LKED.SYSIN    DD  *
          INCLUDE QMFLOAD(DSQCIPL)
          INCLUDE QMFLOAD(DSQCIPX)
          ENTRY sampPLI
          MODE  AMODE(31) RMODE(ANY)
          NAME  sampPLI(R)
/*
```

Figure 63. JCL to run the PL/I compiler and linkage editor

Running in TSO without ISPF

After you compile your program for the TSO environment, the following CLIST runs your program:

```
PROC 0
CONTROL ASIS
/*************************************************************/
/* Note: QMF, DB2, GDDM and PL/I load libraries must be      */
/*       allocated before running this CLIST.                */
/*       Name of QMF load library is  "QMF720.SDSQLOAD".     */
/*************************************************************/
/* Specify attribute list for dataset allocations            */
/*************************************************************/
ATTR PRINTDCB LRECL(133) RECFM(F B A) BLKSIZE(1330)
ATTR DEBUGDCB LRECL(80)  RECFM(F B)   BLKSIZE(3120)
ATTR UDUMPDCB LRECL(125) RECFM(V B A) BLKSIZE(1632)
ATTR EDITDCB  LRECL(79)  RECFM(F B A) BLKSIZE(4029)
/*************************************************************/
/* Datasets used by TSO                                      */
/*************************************************************/
ALLOC FI(SYSPROC) DA('QMF720.SDSQCLTE')
ALLOC FI(SYSEXEC) DA('QMF720.SDSQEXCE')
/*************************************************************/
/* QMF/GDDM Datasets                                         */
/*************************************************************/
ALLOC FI(ADMGGMAP) DA('QMF720.QMFMAPS')  SHR REUSE
ALLOC FI(ADMCFORM) DA('QMF720.DSQCFORM') SHR REUSE
ALLOC FI(DSQUCFRM) DA('QMF720.DSQUCFRM') SHR REUSE
ALLOC FI(ADMSYMBL) DA('ADM.GDDMSYM')  SHR REUSE
ALLOC FI(ADMGDF)   DA('ADM.GDDM.CHARTLIB') SHR REUSE
ALLOC FI(ADMDEFS)  DA('ADM.GDDM.NICKNAME') SHR REUSE
/*************************************************************/
/* Datasets used by QMF                                      */
/*************************************************************/
ALLOC FI(DSQPRINT) SYSOUT(X) USING(PRINTDCB)
ALLOC FI(DSQDEBUG) SYSOUT(X) USING(DEBUGDCB)
ALLOC FI(DSQUDUMP) SYSOUT(X) USING(UDUMPDCB)
ALLOC FI(DSQSPILL) NEW UNIT(SYSDA) SPACE(1,1) TRACKS
ALLOC FI(DSQEDIT)  NEW UNIT(SYSDA) USING(EDITDCB)
ALLOC FI(DSQPNLE)  DA('QMF720.DSQPNLE') SHR REUSE
/*************************************************************/
/* Start your program using TSO CALL command                */
/*************************************************************/
CALL sampPLI
EXIT CODE(0)
```

Figure 64. CLIST to run your program in TSO without ISPF

Running in TSO under ISPF

After you compile your program for the TSO environment, the following CLIST runs your program:

```
PROC 0
CONTROL ASIS
/*************************************************************/
/* Specify attribute list for dataset allocations            */
/*************************************************************/
ATTR PRINTDCB LRECL(133) RECFM(F B A) BLKSIZE(1330)
ATTR DEBUGDCB LRECL(80)  RECFM(F B)   BLKSIZE(3120)
ATTR UDUMPDCB LRECL(125) RECFM(V B A) BLKSIZE(1632)
ATTR EDITDCB  LRECL(79)  RECFM(F B A) BLKSIZE(4029)
/*************************************************************/
/* Datasets used by TSO                                      */
/*************************************************************/
ALLOC FI(SYSPROC) DA('QMF720.SDSQCLTE','ISR.ISRCLIB')
ALLOC FI(SYSEXEC) DA('QMF720.SDSQEXCE')
/*************************************************************/
/* Datasets used by ISPF                                     */
/*************************************************************/
ALLOC FI(ISPLLIB) SHR REUSE +
      DA('QMF720.SDSQLOAD','ADM.GDDMLOAD','DSN.DSNEXIT','DSN.DSNLOAD', +
         'PLI.PLILINK','PLI.SIBMLINK')
ALLOC FI(ISPMLIB) SHR REUSE +
      DA('QMF720.SDSQMLBE','ISR.ISRMLIB','ISP.ISPMLIB')
ALLOC FI(ISPPLIB) SHR REUSE +
      DA('QMF720.SDSQPLBE','ISR.ISRPLIB','ISP.ISPPLIB')
ALLOC FI(ISPSLIB) SHR REUSE +
      DA('QMF720.SDSQSLBE','ISR.ISRSLIB','ISP.ISPSLIB')
ALLOC FI(ISPTLIB) SHR REUSE +
      DA('ISR.ISRTLIB','ISP.ISPTLIB')
/*************************************************************/
/* QMF/GDDM Datasets                                         */
/*************************************************************/
ALLOC FI(ADMGGMAP) DA('QMF720.QMFMAPS')  SHR REUSE
ALLOC FI(ADMCFORM) DA('QMF720.DSQCFORM') SHR REUSE
ALLOC FI(DSQUCFRM) DA('QMF720.DSQUCFRM') SHR REUSE
ALLOC FI(ADMSYMBL) DA('ADM.GDDMSYM')  SHR REUSE
ALLOC FI(ADMGDF)   DA('ADM.GDDM.CHARTLIB') SHR REUSE
ALLOC FI(ADMDEFS)  DA('ADM.GDDM.NICKNAME') SHR REUSE
/*************************************************************/
/* Datasets used by QMF                                      */
/*************************************************************/
ALLOC FI(DSQPRINT) SYSOUT(X) USING(PRINTDCB)
ALLOC FI(DSQDEBUG) SYSOUT(X) USING(DEBUGDCB)
ALLOC FI(DSQUDUMP) SYSOUT(X) USING(UDUMPDCB)
ALLOC FI(DSQSPILL) NEW UNIT(SYSDA) SPACE(1,1) TRACKS
ALLOC FI(DSQEDIT)  NEW UNIT(SYSDA) USING(EDITDCB)
ALLOC FI(DSQPNLE)  DA('QMF720.DSQPNLE') SHR REUSE
/*************************************************************/
/* Start your program as the initial ISPF dialog             */
/*************************************************************/
ISPSTART PGM(sampPLI) NEWAPPL(DSQE)
EXIT CODE(4)
```

Figure 65. CLIST for running your program in TSO under ISPF

The EXIT CODE(4) suppresses the ISPF disposition panel.

REXX language interface

The REXX callable interface provided here corresponds to that provided for other SAA languages.

Note to CICS users
Because REXX is not available under QMF CICS, the QMF callable interface for REXX does not work under CICS.

REXX is an interpretive language; you do not need to compile it. However, programs written using compiled REXX or other compiled languages have better performance than the same programs written using interpretive REXX. A REXX compiler is available for REXX programs, but not for procedures with logic.

Under TSO, you can reduce the resources required to use REXX services when you use procedures with logic and certain Form functions (calculations, defined columns, and conditions) by invoking QMF using the REXX callable interface. All of these functions use REXX.

For example, less resources are required to perform PRINT REPORT or BOTTOM on the REPORT panel if the QMF session is initiated using the REXX callable interface. The reduction of resource consumption can be substantial and is most noticeable when running QMF under TSO/E.

The REXX language always operates in a *command environment* that determines how and where the command is processed. If you write a REXX program that issues QMF commands, you can use the QMF command environment through the ADDRESS QRW command. For more information, see Chapter 5, "ADDRESS QRW: Using the QMF Command Environment" on page 37.

Interface communications variables for REXX

The communications variables consist of the following REXX variables. They are set after the completion of every call.

Table 20 on page 218 shows the interface communication variables, which must *not* be altered by the calling program:

Table 20. Interface communications variables for REXX

Structure Name	Description
dsq_return_code	Integer that indicates the status of SAA Query. Possible values are: **dsq_success** Successful processing of the request. **dsq_warning** Normal completion with warnings. **dsq_failure** Command did not process correctly. **dsq_severe** Severe error; SAA Query session has ended. Because the SAA Query session has ended, additional calls to SAA Query cannot be made using this instance ID. The value of `dsq_return_code` is also placed in the REXX variable *rc*.
dsq_instance_id	Identifier that is established by SAA Query during processing of the START command.
dsq_product	Query manager product in use. Possible values are: **dsq_qrw** OS/2 Query Manager **dsq_qmf** QMF **dsq_qm4** OS/400 Query Management

Table 20. Interface communications variables for REXX (continued)

Structure Name	Description
dsq_product_release	Release level of the query product in use. Possible values are: • OS/2: **dsq_qrw_v1r2** Version 1 Release 2 **dsq_qrw_v1r3** Version 1 Release 3 • OS/400: **dsq_qm4_v1r4** Version 1 Release 1 • QMF: **dsq_qmf_v2r4** QMF Version 2 Release 4 **dsq_qmf_v3r1** QMF Version 3 Release 1 **dsq_qmf_v3r1m1** QMF Version 3 Release 1 Modification 1 **dsq_qmf_v3r2** QMF Version 3 Release 2 **dsq_qmf_v3r3** QMF Version 3 Release 3 **dsq_qmf_v6r1** QMF Version 6 **dsq_qmf_v7r2** QMF Version 7 Release 2
dsq_message_id	Completion message ID.
dsq_q_message_id	Query message ID.
dsq_start_parm_error	Parameter in error when START failed due to a parameter error.
dsq_cancel_ind	Command cancel indicator; indicates whether the user had canceled command processing while QMF was running a command. Possible values are: **dsq_cancel_yes** The user canceled the command **dsq_cancel_no** The user did not cancel the command

Table 20. Interface communications variables for REXX (continued)

Structure Name	Description
dsq_message_text	Completion message text.
dsq_q_message_text	Query message text.

Function call for REXX

The callable interface is accessed by using normal REXX function calls. QMF provides an external subroutine called DSQCIX, which is used to run all SAA Query commands.

DSQCIX Linear Syntax

```
call DSQCIX cmd parmlist
```

- *cmd* is a QMF command written as an uppercase character string.
- *parmlist* is a list of parameter and value pairs, as shown in the following diagram:

The entire command, including the *parmlist*, should be passed to QMF as a single REXX variable written as a character string. This string must be enclosed in quotation marks (' ') or (" "). When using REXX variables as part of the command string, don't enclose the argument. For example:

```
CALL DSQCIX "RUN QUERY NAME (&ECN="REXAUG",CONFIRM=YES)"
```

parmname
Name of a parameter.

value
Value that is to be associated with the parameter name specified by parmname.

Examples::

```
call DSQCIX "RUN QUERY Q1"
call DSQCIX "PRINT REPORT (FORM=F1"
call DSQCIX "EXIT"
```

In the *parmlist*, the same results occur whether the following elements are present or not:

Comma (.) between parameters
A space produces the same result

Closing parenthesis ())
Not required

Equal sign (=) between *parmname* **and** *value*
A space produces the same result

Each of the following would produce the same result.

```
call dsqcix "SET GLOBAL (abc=17, def=26"
call dsqcix "SET GLOBAL ( abc=17 def=26"
call dsqcix "SET GLOBAL ( abc=17 , def=26)"
call dsqcix "SET GLOBAL (abc 17 def=26)"
```

REXX programming example

The following program, DSQABFX, is shipped with QMF.

You can look at the sample source code listing here or you can access it online. For MVS, the sample program is a member of the library QMF710.SDSQEXCE; for VM, the sample program is on the production disk. REXX is not available in QMF CICS.

The sample program for the REXX callable interface performs the following function:

- Starts QMF
- Sets three global variables
- Runs a query called Q1
- Prints the resulting report using form F1
- Ends the QMF session

QMF does not supply query Q1 or form F1, but the sample program uses these objects.

```
/*REXX***************************************************************/
/* Sample Program:  DSQABFX                                        */
/* REXX Version of the SAA Query Callable Interface                */
/*********************************************************************/


/*********************************************************************/
/* Start a query interface session                                 */
/*********************************************************************/

call dsqcix "START (DSQSMODE=INTERACTIVE"
say dsq_message_id dsq_message_text
if dsq_return_code = dsq_severe then exit dsq_return_code

/*********************************************************************/
/* Set numeric values into query using SET command                 */
/*********************************************************************/

call dsqcix "SET GLOBAL (MYVAR01=20,SHORT=40,MYVAR03=84"
say dsq_message_id dsq_message_text
if dsq_return_code = dsq_severe then exit dsq_return_code

/*********************************************************************/
/* Run a Query                                                     */
/*********************************************************************/

call dsqcix "RUN QUERY Q1"
say dsq_message_id dsq_message_text
if dsq_return_code = dsq_severe then exit dsq_return_code

/*********************************************************************/
/* Print the results of the query                                  */
/*********************************************************************/

call dsqcix "PRINT REPORT (FORM=F1)"
say dsq_message_id dsq_message_text
if dsq_return_code = dsq_severe then exit dsq_return_code

/*********************************************************************/
/* End the query interface session                                 */
/*********************************************************************/

call dsqcix "EXIT"
say dsq_message_id dsq_message_text
exit dsq_return_code
```

Figure 66. DSQABFX, a sample REXX program

Running your programs under CMS in VM

The following program runs your callable interface application using the REXX CALL interface.

You might have to modify this program to suit your installation.

```
/*************************************************************** */
/* Access SQL/DS and initialize database                        */
/*************************************************************** */
"EXEC PRODUCT SQLDS"                                            */
"EXEC SQLINIT DBNAME(SQLDBA)"                                   */
                                                                */
/*************************************************************** */
/* Access GDDM product disk                                     */
/*************************************************************** */
"EXEC PRODUCT GDDM"                                             */
                                                                */
/*************************************************************** */
/* Issue Filedefs for QMF product                               */
/*************************************************************** */
 /* DEBUG = DDNAME FOR QMF DIAGNOSTICS OUTPUT                   */
"FILEDEF DSQDEBUG PRINTER ( LRECL 80  BLKSIZE 80  RECFM FBA PERM"
 /* PRINT = DDNAME FOR QMF PRINTED OUTPUT                       */
"FILEDEF DSQPRINT PRINTER ( LRECL 121 BLKSIZE 121 RECFM FBA PERM"
 /* EDIT  = DDNAME FOR QMF EDIT TRANSFER FILE                   */
"FILEDEF DSQEDIT DISK QMFEDIT FILE A (PERM"
 /* DSQSIDE = DDNAME FOR QMF SPILL FILE                         */
"FILEDEF DSQSPILL DISK DSQSIDE DATA A1 (PERM"
 /* DSQPNLE = DDNAME FOR PANEL FILE                             */
"FILEDEF DSQPNLE DISK DSQPNLE FILE * (PERM"
"FILEDEF ISPLLIB CLEAR"
"FILEDEF ISPLLIB DISK DSQLDLIB LOADLIB *"

/*****************************************************************/
/* Provide access to QMF and GDDM program libraries              */
/*****************************************************************/
"GLOBAL LOADLIB DSQLDLIB "
"GLOBAL TXTLIB ADMRLIB ADMPLIB ADMGLIB"

/* The beginning of your REXX program ..... */
.
.
.
.
/* The end of your REXX program .....       */
```

Figure 67. REXX program to run your program in CMS

Running your programs under TSO in MVS

You can run your REXX program by writing a program similar to the following:

```
 /*****************************************************************/
 /* Issue TSO Allocates for QMF Product                           */
 /*****************************************************************/
 Address TSO

 "ATTR PRINTDCB LRECL(133) RECFM(F B A) BLKSIZE(1330)"
 "ATTR DEBUGDCB LRECL(80) RECFM(F B) BLKSIZE(3120)"
 "ATTR UDUMPDCB LRECL(125) RECFM(V B A) BLKSIZE(1632)"
 "ATTR EDITDCB LRECL(79) RECFM(F B A) BLKSIZE(4029)"
 "ALLOC FI(SYSPROC) SHR REUSE ",
  "DA('QMF720.DSQCLSTE,'",
    "'DSN.DSNCLIST')"
 "ALLOC FI(SYSEXEC) SHR REUSE ",
  "DA('QMF720.SDSQEXCE')"
 "ALLOC FI(ISPLLIB) SHR REUSE ",
  "DA('QMF720.SDSQLOAD,'",
    "'ADM.GDDM.GDDMLOAD,'",
    "'DSN.DSNLOAD')"
"ALLOC FI(DSQPNLE) DA('QMF710.DSQPNLE') SHR REUSE"
"ALLOC FI(DSQPRINT) SYSOUT USING(PRINTDCB)"
"ALLOC FI(SYSPRT) SYSOUT(X) LRECL(132) RECFM(FBA) BLKSIZE(132)"
"ALLOC FI(DSQDEBUG) SYSOUT(X) USING(DEBUGDCB)"
"ALLOC FI(DSQUDUMP) SYSOUT(X) USING(UDUMPDCB)"
"ALLOC FI(DSQSPILL) NEW UNIT(SYSDA) SPACE(1,1) TRACKS"
"ALLOC DDNAME(DSQEDIT) UNIT(SYSDA) NEW USING(EDITDCB)"
"ALLOC FI(ADMDEFS)  DA('ADM.GDDM.NICKNAME') SHR REUSE"
"ALLOC FI(ADMGGMAP) DA('QMF720.DSQMAPE') SHR REUSE"
"ALLOC FI(ADMCFORM) DA('QMF720.DSQCHART') SHR REUSE"
"ALLOC FI(DSQUCFRM) DA('QMF720.DSQUCFRM') SHR REUSE"
"ALLOC FI(ADMGDF)   DA('GDDM.ADMGDF') SHR REUSE"
"ALLOC FI(ADMSYMBL) DA('ADM.GDDM.GDDMSYM') SHR REUSE"

/* The beginning of your REXX program .....    */
.
.
.
/* The end of your REXX program ........       */
```

Figure 68. REXX program to run your program in TSO

A REXX example of using an INTERACT l'QMF720oop

Normally, when your callable interface program issues an INTERACT command and the user issues the END command, QMF immediately returns control to your program. However, interactive QMF allows the user to issue the END command to return to the QMF Home panel. Issuing the END command a second time ends the QMF session.

By adding the following logic to your program, you can make the END command in an interactive session started by the INTERACT command from a callable interface program behave similarly to the way END behaves in interactive QMF.

This program uses dsq_message_id to determine how to proceed. These values can change from one release to the next.

This program is *not* distributed with QMF.

```
/*REXX************************************************************/
/* Sample Program: Using INTERACT loop                            */
/*******************************************************************/
/*******************************************************************/
/* Start an interactive QMF session                               */
/*******************************************************************/
trace error

parms = "START (DSQSMODE=INTERACTIVE"
call dsqcix parms
if dsq_return_code = dsq_severe then exit dsq_return_code
/*******************************************************************/
/* SET GLOBAL to show panel IDs                                   */
/*******************************************************************/
call dsqcix "SET GLOBAL (DSQDC_SHOW_PANID=1"
if dsq_return_code = dsq_severe then exit dsq_return_code
/*******************************************************************/
/* Issue message                                                  */
/*******************************************************************/
call dsqcix "MESSAGE (TEXT='Ok, You may enter a command.')"
if dsq_return_code = dsq_severe then exit dsq_return_code
/*******************************************************************/
/* INTERACT loop                                                  */
/*******************************************************************/
Continue = "yes"
Do while continue = "yes"
  call DSQCIX "INTERACT"
  Select
    When (dsq_return_code = dsq_severe) Then  /* Severe Error */
          Continue = "no"
    When (dsq_message_id = "DSQ21869")  Then  /* END from HOME panel */
          Continue = "no"
    When (dsq_message_id = "DSQ90557")  Then  /* User issued EXIT    */
          Continue = "no"
    Otherwise nop                             /* OK continue session */
  End
End
/*******************************************************************/
/* End the session                                                */
/*******************************************************************/
if dsq_message_id <> "DSQ90557" then   /* EXIT not issued         */
  call dsqcix "EXIT"                   /* Issue EXIT              */

exit dsq_return_code
```

Figure 69. REXX program that uses an INTERACT loop

Appendix B. Export/Import Formats

This chapter describes the QMF format for data, and lists the table and field numbers for each encoded format object:

- Form
- Prompted query
- Report

For explanations and examples of these lists, see Chapter 9, "Importing and Exporting QMF Objects" on page 77.

QMF format for data

The data file you export using the `EXPORT` command (DATAFORMAT=QMF) consists of two parts: header records, which describe the data in the records, and the data records, which contain the data.

Header records

The record length of an external data file is the length of a row of the data, as described under "Data records" on page 229. The header records that precede the data records are also split into this length. Table 21 shows the information contained in the header records.

Table 21. Header record information

Byte Position	Information and Type
1-8	QMF object format level (8 characters of data). A QMF object format level specifies how many times the format has been updated for a particular object. Because the form object has been altered three times since QMF 1.0, the object format level for a form exported from QMF 3.2 is 4. Because the data object format has not been altered, it is still at object format level 1.
9-10	Number of header records (halfword signed integer).
11-12	Number of data columns (halfword signed integer).
13-30, 37-54, ...	Column name (18 characters of data).
31-32, 55-56, ...	Data type (halfword signed integer). Data type codes are shown in Table 22 on page 228.

Table 21. Header record information (continued)

Byte Position	Information and Type
33-34, 57-58, ...	Column width (halfword signed integer). For most data types this is the width of the column in bytes. Exceptions are: • In DECIMAL columns, the first byte of the halfword represents the precision, and the second byte represents the scale. • In GRAPHIC and VARGRAPHIC columns, this value reflects the width of double-byte characters. • In FLOAT columns, this value is either 4, indicating single precision floating point, or 8, indicating double precision floating point.
35, 59, ...	Nulls allowed: Y if nulls are allowed; N if they are not allowed (1 character of data).
36, 60, ...	Unused byte.

Bytes 11-12 indicate the number of columns; this means that the information in bytes 13 through 36 is repeated for each column in the header records. Each column requires 24 bytes in the header record.

The data type codes are shown in Table 22.

Table 22. Data type codes

Code in Hexadecimal	Code in Decimal	Data Type	Meaning
X'180'	384	DATE	Date
X'184'	388	TIME	Time
X'188'	392	TIMESTAMP	Time stamp
X'1C0'	448	VARCHAR	Varying character
X'1C4'	452	CHAR	Fixed character
X'1D0'	464	VARGRAPHIC	Varying graphic
X'1D4'	468	GRAPHIC	Fixed graphic
X'1E0'	480	FLOAT	Floating point
X'1E4'	484	DECIMAL	Decimal
X'1F0'	496	INTEGER	Integer
X'1F4'	500	SMALLINT	Small integer

Date, time, and time stamp data is always exported in ISO format.

For more details of the format of data types, refer to *DB2 UDB for OS390 SQL Reference*.

Data records

Data records are of fixed-length format and contain the data to be exported. The maximum length a data record can be is 7,000 bytes. The length of a data record is the sum of the widths of the data types that comprise the record. Use the following table to calculate the widths of each data type.

Important: You cannot export a table with a VARCHAR column whose maximum allowable length is over 254.

Table 23. Data widths in encoded format data records. Calculate the width of a particular data type by adding the number of bytes in each column.

Data type	Null Indicator	Length Field	SO/SI	Data
Character	2			Length in header (LIH)
Date	2			LIH
Floating point	2			8
Integer	2			LIH
Small integer	2			LIH
Time	2			LIH
Time stamp	2			LIH
Decimal	2			(Precision + 2) // 2
Graphic	2		2	(LIH × 2)
Variable character	2	2		LIH
Variable graphic	2	2	2	(2 × LIH)

Note: The LIH is the width given in the header record for that column.

Every data record has two bytes of indicator information, which can have the following values and corresponding meaning:

Value	Meaning
X'0000'	The column contains valid data.
X'FFFF'	The column contains a null value. Any data in the column is meaningless.
X'FFFE'	

Table and field numbers for the prompted query object

The following table contains prompted query table and field numbers for T records that describe each table in the prompted query exported format. The information in the DESCRIPTION column uniquely identifies specific fields in the prompted query base panel.

Table definitions (field number 1110) are always exported. Join conditions (field number 1510) are always exported if more than one table is selected.

To import a prompted query file, the file must have a H record followed by the tables T record. No tables need to be specified. If no tables are specified, an empty query is imported. Join conditions are not required unless more than one table is selected.

Table 24. Table and field numbers for exported prompted query object

Record type	Table number	Field number	Field Description
T	1110	-	Table definitions table
		1112	-- Table ID (valid table IDs are A-Z, and #,$,@)
		1113	-- Table name
T	1150	-	Join conditions table
		1152	-- Column 1 name
		1153	-- Column 2 name
T	1210	-	Columns table
		1212	-- Column type: • C=column • E=expression • S=summary function with expression • F=summary function with only a column
		1213	-- Column name, expression, or summary function
T	1310	-	Row selection conditions
		1312	-- Entry type: • 1 - left of operator • 2 - operator • 3 - right of operator • 4 - connector

Table 24. Table and field numbers for exported prompted query object (continued)

Record type	Table number	Field number	Field Description
		1313	-- For entry type '1', identifies column type: • C=column • E=expression • S=summary function • F=summary function (column name only specified)
			-- For entry type '2', identifies the verb: • IS for 'is' (Default) • ISN for 'is not'
			-- For entry type '3', (not used)
			-- For entry type '4', identifies a connector: • O for 'or' • A for 'and' (Default)
		1314	-- For entry type '1' this field is: • Column name, expression, or summary function
			-- For entry type '2', identifies the operator: • EQ for 'equal to' • LT for 'less than' • LE for 'less than or equal to' • GT for 'greater than' • GE for 'greater than or equal to' • BT for 'between' • SW for 'starting with' • EW for 'ending with' • CT for 'containing' • NL for NULL
			-- For entry type '3', identifies a value
			-- For entry type '4', (not used)
T	1410	-	Sort conditions table
		1412	-- Sort direction: • A for 'ascending' • D for 'descending'
		1413	-- Column
V		1501	Duplicate rows treatment: • K for 'keep' • D for 'discard'

The meaning of values for fields 1313 and 1314 depends on the sequence number indicated in field number 1312 in the 1310 table.

Table and field numbers for the form object

Table 25 lists the table numbers for T records and field numbers for V records for the form object. Each number corresponds to a particular part of the form.

Field 3080, a V record, acts as a "trigger" for the break panels that follow it. This record appears once for every break panel in your form. The value of the field reflects the number of the break panel that the fields following field 3080 describe.

Table 25. Table and field numbers for exported form object

Table or field number	Record type	Description	Form panel
1110	T	Column headings table	FORM.COLUMNS
1112	R	Column data type [4]	FORM.COLUMNS
1113	R	Column heading	FORM.COLUMNS
1114	R	Column usage code	FORM.COLUMNS
1115	R	Column indentation	FORM.COLUMNS
1116	R	Column width	FORM.COLUMNS
1117	R	Column edit code	FORM.COLUMNS
1118	R	Column sequence	FORM.COLUMNS
1119	R	Column heading alignment [5]	FORM.COLUMNS
1120	R	Column data alignment [5]	FORM.COLUMNS
1121	R	Column definition [5]	FORM.COLUMNS
1122	R	Pass nulls on column definition [5]	FORM.COLUMNS
1180	T	Summary calculations table	FORM.CALC
1182	R	Calculation identification number	FORM.CALC
1183	R	Summary calculation expression	FORM.CALC
1184	R	Summary calculation width	FORM.CALC
1185	R	Summary calculation edit code	FORM.CALC
1186	R	Pass nulls on calculation [5]	FORM.CALC
1201	V	Blank lines before heading	FORM.PAGE
1202	V	Blank lines after heading	FORM.PAGE
1210	T	Page heading table	FORM.PAGE

4. QMF does not display this field on the form panel.

5. This field is new for Version 3.

Table 25. Table and field numbers for exported form object (continued)

Table or field number	Record type	Description	Form panel
1212	R	Page heading line number	FORM.PAGE
1213	R	Page heading alignment	FORM.PAGE
1214	R	Page heading text	FORM.PAGE
1301	V	Blank lines before footing	FORM.PAGE
1302	V	Blank lines after footing	FORM.PAGE
1310	T	Page foot table	FORM.PAGE
1312	R	Page footing line number	FORM.PAGE
1313	R	Page footing alignment	FORM.PAGE
1314	R	Page footing text	FORM.PAGE
1401	V	New page for final text	FORM.FINAL
1402	V	Final summary line number	FORM.FINAL
1403	V	Blank lines before final text	FORM.FINAL
1410	T	Final text table	FORM.FINAL
1412	R	Final text line number	FORM.FINAL
1413	R	Final text alignment	FORM.FINAL
1414	R	Final text	FORM.FINAL
1501	V	Detail line spacing	FORM.OPTIONS
1502	V	Outlining for break columns	FORM.OPTIONS
1503	V	Default break text	FORM.OPTIONS
1504	V	Function name in column heading for grouping	FORM.OPTIONS
1505	V	Column-wrapped lines kept on a page	FORM.OPTIONS
1506	V	Across-summary column	FORM.OPTIONS
1507	V	Separators for column heading	FORM.OPTIONS
1508	V	Separators for break summary	FORM.OPTIONS
1509	V	Separators for across heading	FORM.OPTIONS
1510	V	Separators for final summary	FORM.OPTIONS
1511	V	Width of wrapped report lines	FORM.OPTIONS
1512	V	Page re-numbering at breaks	FORM.OPTIONS
1513	V	Width of break or final text	FORM.OPTIONS
1514	V	Column re-ordering	FORM.OPTION
1515	V	Fixed columns	FORM.OPTIONS
2790	V	Detail variation number	FORM.DETAIL
2791	V	Detail variation selection	FORM.DETAIL
2805	V	Include column heading	FORM.DETAIL

Table 25. Table and field numbers for exported form object (continued)

Table or field number	Record type	Description	Form panel
2810	T	Detail heading table	FORM.DETAIL
2812	R	Detail heading text line	FORM.DETAIL
2813	R	Detail heading alignment	FORM.DETAIL
2814	R	Detail heading text	FORM.DETAIL
2901	V	New page for detail text	FORM.DETAIL
2902	V	Line number of column data	FORM.DETAIL
2904	V	Number of lines to skip after detail text	FORM.DETAIL
2906	V	Repeat detail heading	FORM.DETAIL
2907	V	Number of detail text lines to keep together	FORM.DETAIL
2910	T	Detail text table	FORM.DETAIL
2912	R	Detail text line number	FORM.DETAIL
2913	R	Detail text alignment	FORM.DETAIL
2914	R	Detail text	FORM.DETAIL
3080	V	Break panel number [5]	FORM.BREAKn
3101	V	New page for break heading [5]	FORM.BREAKn
3102	V	Repeat break heading [5]	FORM.BREAKn
3103	V	Number of lines to skip before break heading [5]	FORM.BREAKn
3104	V	Number of lines to skip after break heading [5]	FORM.BREAKn
3110	T	Break heading text table [5]	FORM.BREAKn
3112	R	Break heading line number [5]	FORM.BREAKn
3113	R	Break heading alignment [5]	FORM.BREAKn
3114	R	Break heading text [5]	FORM.BREAKn
3201	V	New page for break text [5]	FORM.BREAKn
3202	V	Break text summary line [5]	FORM.BREAKn
3203	V	Number of lines to skip before break text [5]	FORM.BREAKn
3204	V	Number of lines to skip after break text [5]	FORM.BREAKn
3210	T	Break text table [5]	FORM.BREAKn
3212	R	Break text line [5]	FORM.BREAKn
3213	R	Break text alignment [5]	FORM.BREAKn
3214	R	Break text [5]	FORM.BREAKn
3310	T	Conditions table [5]	FORM.CONDITIONS
3312	R	Condition identification number [5]	FORM.CONDITIONS

Table 25. Table and field numbers for exported form object (continued)

Table or field number	Record type	Description	Form panel
3313	R	Conditional expression [5]	FORM.CONDITIONS
3314	R	Pass nulls on conditions panel [5]	FORM.CONDITIONS

Table 26 shows fields that are valid for objects that were created before Version 3 Release 1. QMF accepts these fields on input, but does not create them on output. There is a unique set of field numbers for each break panel.

Table 26. Field numbers for exported form object, before QMF 3.1

Table or field number	Record type	Description	Form panel
1601	V	BREAK1: New page for heading	FORM.BREAK1
1602	V	BREAK1: Repeat column headings	FORM.BREAK1
1603	V	BREAK1: Blank lines before heading	FORM.BREAK1
1604	V	BREAK1: Blank lines after heading	FORM.BREAK1
1610	T	BREAK1: Heading table	FORM.BREAK1
1612	R	BREAK1: Heading lines	FORM.BREAK1
1612	R	BREAK1: Heading alignment	FORM.BREAK1
1614	R	BREAK1: Heading text	FORM.BREAK1
1701	V	BREAK1: New page for footing	FORM.BREAK1
1702	V	BREAK1: Repeat column footings	FORM.BREAK1
1703	V	BREAK1: Blank lines before footing	FORM.BREAK1
1704	V	BREAK1: Blank lines after footing	FORM.BREAK1
1710	T	BREAK1: Footing table	FORM.BREAK1
1712	R	BREAK1: Footing lines	FORM.BREAK1
1713	R	BREAK1: Footing alignment	FORM.BREAK1
1714	R	BREAK1: Footing text	FORM.BREAK1
1801	V	BREAK2: New page for heading	FORM.BREAK2
1802	V	BREAK2: Repeat column headings	FORM.BREAK2
1803	V	BREAK2: Blank lines before heading	FORM.BREAK2
1804	V	BREAK2: Blank lines after heading	FORM.BREAK2
1810	T	BREAK2: Heading table	FORM.BREAK2
1812	R	BREAK2: Heading lines	FORM.BREAK2
1813	R	BREAK2: Heading alignment	FORM.BREAK2

Table 26. Field numbers for exported form object, before QMF 3.1 (continued)

Table or field number	Record type	Description	Form panel
1814	R	BREAK2: Heading text	FORM.BREAK2
1901	V	BREAK2: New page for footing	FORM.BREAK2
1902	V	BREAK2: Repeat column footings	FORM.BREAK2
1903	V	BREAK2: Blank lines before footing	FORM.BREAK2
1904	V	BREAK2: Blank lines after footing	FORM.BREAK2
1910	T	BREAK2: Footing table	FORM.BREAK2
1912	R	BREAK2: Footing lines	FORM.BREAK2
1913	R	BREAK2: Footing alignment	FORM.BREAK2
1914	R	BREAK2: Footing text	FORM.BREAK2
2001	V	BREAK3: New page for heading	FORM.BREAK3
2002	V	BREAK3: Repeat column headings	FORM.BREAK3
2003	V	BREAK3: Blank lines before heading	FORM.BREAK3
2004	V	BREAK3: Blank lines after heading	FORM.BREAK3
2010	T	BREAK3: Heading table	FORM.BREAK3
2012	R	BREAK3: Heading lines	FORM.BREAK3
2013	V	BREAK3: Heading alignment	FORM.BREAK3
2014	R	BREAK3: Heading text	FORM.BREAK3
2101	V	BREAK3: New page for footing	FORM.BREAK3
2102	V	BREAK3: Repeat column footings	FORM.BREAK3
2103	V	BREAK3: Blank lines before footing	FORM.BREAK3
2104	V	BREAK3: Blank lines after footing	FORM.BREAK3
2110	T	BREAK3: Footing table	FORM.BREAK3
2112	R	BREAK3: Footing lines	FORM.BREAK3
2113	R	BREAK3: Footing alignment	FORM.BREAK3
2114	R	BREAK3: Footing text	FORM.BREAK3
2201	V	BREAK4: New page for heading	FORM.BREAK4
2202	V	BREAK4: Repeat column headings	FORM.BREAK4
2203	V	BREAK4: Blank lines before heading	FORM.BREAK4
2204	V	BREAK4: Blank lines after heading	FORM.BREAK4
2210	T	BREAK4: Heading table	FORM.BREAK4

Table 26. Field numbers for exported form object, before QMF 3.1 (continued)

Table or field number	Record type	Description	Form panel
2212	R	BREAK4: Heading lines	FORM.BREAK4
2213	R	BREAK4: Heading alignment	FORM.BREAK4
2214	R	BREAK4: Heading text	FORM.BREAK4
2301	V	BREAK4: New page for footing	FORM.BREAK4
2301	V	BREAK4: Repeat column footings	FORM.BREAK4
2303	V	BREAK4: Blank lines before footing	FROM.BREAK4
2304	V	BREAK4: Blank lines after footing	FORM.BREAK4
2310	T	BREAK4: Footing table	FORM.BREAK4
2312	R	BREAK4: Footing lines	FORM.BREAK4
2313	R	BREAK4: Footing alignment	FORM.BREAK4
2314	R	BREAK4: Footing text	FORM.BREAK4
2401	V	BREAK5: New page for heading	FORM.BREAK5
2402	V	BREAK5: Repeat column headings	FORM.BREAK5
2403	V	BREAK5: Blank lines before heading	FORM.BREAK5
2404	V	BREAK5: Blank lines after heading	FORM.BREAK5
2410	T	BREAK5: Heading table	FORM.BREAK5
2412	R	BREAK5: Heading lines	FORM.BREAK5
2413	R	BREAK5: Heading alignment	FORM.BREAK5
2414	R	BREAK5: Heading text	FORM.BREAK5
2501	V	BREAK5: New page for footing	FORM.BREAK5
2502	V	BREAK5: Repeat column footings	FORM.BREAK5
2503	V	BREAK5: Blank lines before footing	FORM.BREAK5
2504	V	BREAK5: Blank lines after footing	FORM.BREAK5
2510	T	BREAK5: Footing table	FORM.BREAK5
2512	R	BREAK5: Footing lines	FORM.BREAK5
2513	R	BREAK5: Footing alignment	FORM.BREAK5
2514	R	BREAK5: Footing text	FORM.BREAK5
2601	V	BREAK6: New page for heading	FORM.BREAK6
2602	V	BREAK6: Repeat column headings	FORM.BREAK6
2603	V	BREAK6: Blank lines before heading	FORM.BREAK6

Table 26. Field numbers for exported form object, before QMF 3.1 (continued)

Table or field number	Record type	Description	Form panel
2604	V	BREAK6: Blank lines after heading	FORM.BREAK6
2610	T	BREAK6: Heading table	FORM.BREAK6
2612	R	BREAK6: Heading lines	FORM.BREAK6
2613	R	BREAK6: Heading alignment	FORM.BREAK6
2614	R	BREAK6: Heading text	FORM.BREAK6
2701	V	BREAK6: New page for footing	FORM.BREAK6
2702	V	BREAK6: Repeat column footings	FORM.BREAK6
2703	V	BREAK6: Blank lines before footing	FORM.BREAK6
2704	V	BREAK6: Blank lines after footing	FORM.BREAK6
2710	T	BREAK6: Footing table	FORM.BREAK6
2712	R	BREAK6: Footing lines	FORM.BREAK6
2713	R	BREAK6: Footing alignment	FORM.BREAK6
2714	R	BREAK6: Footing text	FORM.BREAK6

Table and field numbers for the report object

The following figure shows the table numbers for T records and field numbers for V records:

Table 27. General reports. Table and field numbers for exported report object

Table or field number	Record type	Description
1001	V	Profile DECIMAL option
1002	V	Length of L record control area + fixed area
1010	T	Formatted report table
		For each formatted data column in the report:
1012	T	For all usage codes except OMIT
1013	T	Edit code by which data is formatted
1014	T	Starting position for field containing formatted data (including indent area)
1015	T	Starting position for field containing formatted data (excluding indent area)
1016	T	Ending position for field containing formatted data

Table 27. General reports. Table and field numbers for exported report object (continued)

Table or field number	Record type	Description
1017	T	Number of relative physical report line within logical report line in which formatted column value appears

See Note 2 after Table 28 for the meaning of fields 1014, 1015, and 1016 when the report is an across-style report.

Table 28. Across reports. Field numbers for exported report object.

Field number	Record type	Description
2001	V	Edit code by which across value is formatted
2002	V	Number of data lines per across group
2003	V	Indicates whether the across summary column exists
2010	T	Across report table
		For each across value:
2012	T	Starting position for formatted across value. (The across value appears in the column heading lines)
2013	T	Ending position for formatted across value
2014	T	Starting position for the set of report columns associated with this across value, including preceding indent area

Notes:

1. Position 1 of the report line immediately follows the L record fixed area.
2. For aggregated columns in an across report, the fields 1014, 1015, and 1016 describe the relative starting and ending positions of the field within an across value's set of aggregated columns. (Refer to field 2014 in the Table 28.)
3. R records for text lines in each report heading (PAGE or BREAK) or footing (PAGE, BREAK, or FINAL) are only written up to and including the last line that contains modifications to the form defaults.

   At least one R record is written for each heading or footing even when the fields for a given heading or footing all have their original values.
4. Continuation records are written for the report object when the maximum record length would otherwise be exceeded.

HTML tags used in QMF reports

Table 29 briefly describes the HTML tag sets that QMF uses to format a report for display on the world wide web. Each of these tag sets consists of a start tag and an end tag. The end tags begin with a forward slash (/), and all tags are enclosed in angle brackets (

). For a full description of these tags, see your HTML 3.0 documentation.

Table 29. HTML 3.0 Tags used in HTML Reports

Tag set	Description
<HTML></HTML>	Defines the file as an HTML document.
<HEAD></HEAD>	These tags mark the boundaries of the document header.
<TITLE></TITLE>	QMF inserts the word "Report" between these tags. Content between these tags is included in the HTML document title. Placement of the title is browser and platform dependent. These tags are placed within the header.
<BODY></BODY>	These tags follow the header and contain the body of the document. Report output is placed in the body of the document.
<PRE></PRE>	Content between these tags is displayed as is. No HTML formatting is performed between them. QMF places report output between these tags in the body of the HTML document.

Appendix C. Integrated Exchange Format (IXF)

When you use the EXPORT command to export a DATA or TABLE object using the DATAFORMAT=IXF option, the file, data set, or CICS data queue is exported in the Integrated Exchange Format (IXF). QMF supports a subset of IXF, which is described in this section. See *Data Extract: Reference* for a description of the complete Integration Exchange Format.

The IXF format is especially useful if you want to create tables outside the QMF environment and import them. To do this, set OUTPUTMODE to CHARACTER.

In QMF an exported IXF file, data set, or CICS data queue consists of the following records:

- Header record (H)
- Table record (T)
- Column record (C)
- Data record (D)

The exported file, data set, or CICS data queue consists of one H record, followed by one T record. The T record contains a count of how many C records follow the T record. There is a C record for each column in the table. D records follow C records. There is a D record for each row in the table. The arrangement of records in an exported file, data set, or CICS data queue looks like Figure 70.

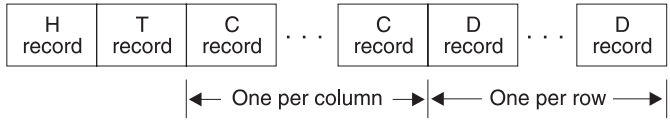


Figure 70. Arrangement of records in an exported data file, data set, or CICS data queue (IXF format)

Note: The Database manager PC/IXF file format is not identical to the System/370™ IXF format. IXF formatted data cannot be transported between PC and System/370 platforms.

The following sections describe the format of each of these records. The values shown in parentheses are the values QMF supplies when data is exported.

Header record (H)

A header record (which is mandatory) is the first record in the file, data set, or CICS data queue. It is a 42-byte record containing character data. The format of the H record is as follows:

Byte Position	Information and Type
01	Header record indicator (H)
02-04	file, data set, or CICS data queue identifier (IXF)
05-08	IXF version (0000)
09-14	Originating product name (QMF)
15-20	Originating product release level (V7R1M0)
21-28	Date the file, data set, or CICS data queue was created; in the form YYYYMMDD
29-34	Time the file, data set, or CICS data queue was created; in the form HHMMSS
35-39	The number of records preceding the first D (Data) record in the file, data set, or CICS data queue. This is a 5-digit numeric value expressed in character form.
40	DBCS indicator. Tells whether DBCS data is a possibility; Y or N.
41-42	Reserved

Table record (T)

A table record follows the header record. Each IXF file, data set, or CICS data queue must have a T record. A table record contains table and data information concerning the file, data set, or CICS data queue being exported. The format of a T record is as follows:

Byte Position	Information and Type
01	Table record indicator (T)
02-03	Data name length (18)
04-21	Name of the table from which data is retrieved; left-justified, padded with blanks to the right. The entire 18-byte field is blank if the table does not have a name.
22-29	Data name qualifier. Name of the owner of the database table from which the data is retrieved. The 8-byte field is blank if the table does not have an owner.
30-41	Data source (database)

Byte Position	Information and Type
42	Convention used to describe data: C for columnar data
43	Data format: C for character (OUTPUTMODE=CHARACTER) M for machine (OUTPUTMODE=BINARY)
44	Data location: I for internal
45-49	Count of column (C) records. A numeric value in character form specifying the number of C records before the first data (D) record.
50-51	Reserved
52-81	Blanks

Column record (C)

A column record describes the data characteristics of the column. There is a column record for each column in the table. The format of the column record follows:

Byte Position	Information and Type
01	Column record indicator (C)
02–03	Column name length
04–21	Column name, as obtained from the database or generated by QMF (in the case where the column did not originally have a name). The name is left-justified, and padded with blanks to the right if necessary.
22	Indicator that tells if nulls are allowed; (Y or N)
23	Column selected indicator (Y)
24	Key column indicator (Y)
25	Data class (R)
26–28	Data type (see Table 12 on page 108 for data type codes)
29–33	Code page (00000)
34–38	Reserved
39–43	Column data length; a numeric value in character form. If data type is DECIMAL, the first 3 bytes represent data precision, and the next 2 bytes represent the scale. If data type is INTEGER, SMALLINT, DATE, TIME, or TIMESTAMP, this field is blank (length is inherent in data type).

Byte Position	Information and Type
44–49	Starting position of column data. A value (in character form) reflecting the offset of the data for a column from the start of the data record. If the column allows nulls, this field points to the null indicator. If the column does not allow nulls, it points to the data itself. Whether or not the column allows nulls, space for the null indicator is always present in the record. The starting position is based from the first byte that contains data. Therefore, the first five bytes of the data (D) record are not included in any consideration of starting position. The first data position is position 1, not position 0.
50–79	Column label information, if available, otherwise blanks
80–81	Two bytes of zeros in character form (00)

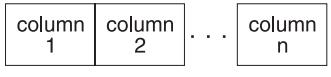
Data record (D)

There is a data record for each row in the table. The format of the data record follows:

Byte Position	Information and Type
01	Data record indicator (D)
02–04	Reserved
05	Blank
06–end of record	Row data in binary or character form, depending on whether byte 43 of the table record is M (machine) or C (character). Byte 6 represents the start (position 1) of row data for the first column.

Column data format

Data in D records for *n* columns is placed side by side:



For each column, the data consists of a null indicator followed by the data itself. If nulls are allowed (byte 22 of C record = Y), then bytes 44–49 of each C record points to the null indicator that precedes the data for that column. If byte 22 = N (nulls not allowed), then bytes 44–49 points to the data itself. However, in the latter case, space for the null indicator is left in the data record. The first position in bytes 44–49 is represented by a value of 1, which points to byte 6 of a D record (bytes 1 through 5 are ignored).

The representation of the null indicator depends on what is specified for OUTPUTMODE: character or binary. OUTPUTMODE is reflected in byte 43 of the T record: C for character or M for machine (binary). When data format is character, one byte is used for the null indicator:

- – (dash) indicates data is null
- (blank) indicates data is not null

See Figure 71 on page 252 for an illustration of two D records showing data that is null in one case and not null in the other.

When the data format is binary, two bytes are used for the null indicator:

- X'FFFF' indicates data is null
- X'0000' indicates data is not null

Figure 72 on page 254 which shows two D records, illustrating both null and non-null data indicators.

Format of column data by data type

Table 30 shows the length and format of column data in D records for each data type for both character and binary formats. In the table, IXFCLENG refers to the contents of bytes 39–43 of a C record (length of column data).

Table 30. Format of IXF column data by data type

Data Type Code	Data Type	Data Length Information Character Format	Data Length Information Binary Format
384	DATE	The value in IXFCLENG is not significant. The length (10 bytes) is inherent in the data type. The format is: `yyyy-mm-dd` where yyyy represents the year, mm the month, and dd the day. yyyy, mm, and dd must be numeric characters. Leading zeroes *cannot* be omitted. The allowable range for yyyy is 0001–9999; for mm, 01–12. The dd range depends on the month. Examples:	Same as character format

Table 30. Format of IXF column data by data type (continued)

Data Type Code	Data Type	Data Length Information Character Format	Data Length Information Binary Format

388	TIME	The value in IXFCLENG is not significant. The length (8 bytes) is inherent in the data type.
		
		The format is:
		`hh.mm.ss`
		
		where hh represents the hour in 24-hour format, mm is minutes, and ss is seconds. hh, mm, and ss must all be numeric characters. Leading zeroes can *not* be omitted. Allowable ranges are:
		• 00 - 23 for hh
		• 00 - 59 for mm
		• 00 - 59 for ss
		
		The special value 24.00.00 for midnight is valid. Examples:
		`10.37.42 is 10:37:42 AM`
		`08.00.00 is 8 AM exactly`
		`23.30.00 is 11:30 PM`
			Same as character format

Table 30. Format of IXF column data by data type (continued)

Data Type Code	Data Type	Data Length Information Character Format	Data Length Information Binary Format

392	TIMESTAMP	The value in IXFCLENG is not significant. The length (26 bytes) is inherent in the data type.
		
		The format is:
		`yyyy-mm-dd-hh`
		`.mm.ss.nnnnnn`
		
		where yyyy is the year, the first mm is the month, dd is the day, hh is hour in 24 hour format, the second mm is minutes, ss is seconds, and nnnnnn is microseconds. Valid ranges for year, month, day, hour, minutes, and seconds are the same as the DATE and TIME data types. nnnnnn can be 000000–999999. Examples:
		`1997-12-31-23`
		`.59.59.999999`
		`(the last microsecond`
		`in 1997)`
		`1998-01-01-00`
		`.00.00.000000`
		`(the first microsecond`
		`in 1998)`
		
		24.00.00.000000 is valid for the time portion of a time stamp.
			Same as character format

448	VARCHAR	IXFCLENG is the maximum length of character string. Data length consists of N bytes indicated by IXFCLENG preceded by a 5-byte character count field. (The allowable range for N is 0-254 and for the count field it is 0-N). The number of characters indicated by the count field are valid; the rest are meaningless. Example:
		`If IXFCLENG=00010`
		`Data format is:`
		`00005JONESxxxxx`
		
		where each x is a blank character (X'40').
			IXFCLENG is the maximum length of character string. Data length consists of N bytes indicated by IXFCLENG preceded by a 2-byte binary count field. (The allowable range for N is 1-254 and for the count field 0-N). The number of characters indicated by the count field are valid; the rest are meaningless. Example:
			`If IXFCLENG=00010`
			`Data format is:`
			`nnJONESxxxxx`
			
			where nn=X'0005' and each x is a blank character (X'40').

Table 30. Format of IXF column data by data type (continued)

Data Type Code	Data Type	Data Length Information Character Format	Data Length Information Binary Format

452	CHAR	IXFCLENG is length of character string. Data length is indicated by N bytes of IXFCLENG. (Allowable range for N is 1-254). Example:
		`If IXFCLENG=00005`
		`Data format is: JONES`
		
		where JONES is the 5-byte character string pointed to by bytes 44-49 of the C record.
			Same as character format

456	LONG VARCHAR	Same as VARCHAR, except that allowable range for N is 0-32767	Same as character format
464	VARGRAPHIC	IXFCLENG is the maximum number of double-byte characters (2×N bytes). Data length consists of a 5-byte character count field, plus twice the number of bytes indicated by IXFCLENG, plus 2 (for shift characters). The number of 2-byte characters in the count field are valid plus a shift-out (X'0E') immediately preceding the data, and a shift-in (X'0F') immediately following the data. The rest can be meaningless. (Allowable range for N is 1-127 and for the count field 0-N.) Example: If `IXFCLENG = 00006` data format is: `00003oZZYYXXixxxxxx` where o is shift-out, i is shift-in, and each x is a blank character (X'40').	Data length consists of a 2-byte binary count field followed by twice the number of bytes indicated by IXFCLENG. The allowable range for IXFCLENG is 1-127, and for the count field 0-IXFCLENG. The number of 2-byte characters in the count field are valid. There are *no* surrounding shift-out and shift-in characters. The rest can be meaningless. Example: If `IXFCLENG = 00008` data format is: `nnZZYYXXWWxxxxxxxx` where nn=X'0004' and each x is a blank character (X'40').

Table 30. Format of IXF column data by data type (continued)

Data Type Code	Data Type	Data Length Information Character Format	Data Length Information Binary Format

468	GRAPHIC	IXFCLENG is the number of double byte characters (2*N bytes). Data length is 2*N bytes plus a shift out (X'0E') immediately preceding the data, and a shift-in (X'0F') immediately following the data. Example:
		`If IXFCLENG=00005`
		`Data format is:`
		`oZZYYXXWWVVi`
		
		where o is shift-out and i is shift-in.
			Same as character format except that there are no surrounding shift-in and shift-out characters in the data string. Example:
			`If IXFCLENG=00005`
			`Data format is:`
			`ZZYYXXWWVV`

472	LONG VARGRAPHIC	Same as VARGRAPHIC, except that the allowable range for N is 0-16383.	Same as character format.

480	FLOAT	The value in IXFCLENG is 8. The length and format of data is inherent in the data type:
		
		Data consists of a 23-byte character value arranged as follows:
		• 1 character for sign
		• 18 characters for mantissa (17 digits and a decimal point)
		• The character E
		• 3-character signed exponent
		
		Examples:
		`-1.2345678901234567E+14`
		`+6.2345678901234567E-01`
		`0.0000000000000000E+00`
			The value in IXFCLENG is 8. The length and format of data is inherent in the data type:
			
			Data consists of an 8-byte floating point value in standard IBM S/370 format for long floating point.

Table 30. Format of IXF column data by data type (continued)

Data Type Code	Data Type	Data Length Information Character Format	Data Length Information Binary Format

484	DECIMAL	Bytes 39-43 of the C record represent the precision P (first 3 bytes) and scale S (next 2 bytes) of the number. Allowable range for P is 0- 15. S can be any value less than or equal to P.
		
		Data is formatted as a P+2 byte character value (or P+1 bytes if S=0), right-justified, with the first byte reserved for a sign, and a decimal point (position implied by S) present only if S is not equal to zero. Examples:
		`If P=005, S=00;`
		`Data format is: 12345`
		`If P=006, S=02;`
		`Data format is: +2345.10`
		`If P=004, S=03;`
		`Data format is: -8.515`
			Bytes 39-43 of the C record represent the precision P (first 3 bytes) and scale S (next 2 bytes) of the number. Allowable range for P is 0-15. S can be any value less than or equal to P.
			
			Data consists of a (P+2)/2 byte packed decimal value in standard IBM S/370 packed decimal format, with S of the P digits interpreted as following the implied decimal point. Examples:
			`If P=005, S=00;`
			`Data format is:`
			`X'12345C'`
			`If P=006, S=02;`
			`Data format is:`
			`X'0234510D'`

496	INTEGER	The value in IXFCLENG is not significant. The length and format of data is inherent in the data type.
		
		Data consists of an 11-byte character value, right-justified, with the first character reserved for a sign. Examples:
		`0000000013`
		`+1187642200`
		`-0033588727`
			The value in IXFCLENG is not significant. The length and format of data is inherent in the data type.
			
			Data consists of a 4-byte binary value.

500	SMALLINT	The value in IXFCLENG is not significant. The length and format of data is inherent in the data type.
		
		Data consists of a 6-byte character value, right-justified, with the first character reserved for a sign. Examples:
		`00023`
		`+00763`
		`-21311`
			The value in IXFCLENG is not significant. The length and format of data is inherent in the data type.
			
			Data consists of a 2-byte binary value.

Examples of IXF

Assume the same table shown in the example on page 80 (which was exported using the QMF format) is now exported using the IXF format (with OUTPUTMODE=CHARACTER):

```
ID  NAME     COMM
___ _______  ______
10  SANDERS    -
20  PERNAL   612.45
```

The exported file, data set, or CICS data queue consists of a total of seven records; an H record, a T record, three C records, and two D records as shown:

```
HIXF0000QMF   V7R1M019981204095600000005N
T18                          database         CCI00003
C02ID                NYNR50000000          000002                      00
C04NAME              YYNR44800000     00009000008                      00
C04COMM              YYNR48400000     00702000023                      00
D      00010 00007SANDERSxx-
D      00020 00006PERNALxxx  00612.45
```

Unprintable binary characters are shown as x's. Figure 71 on page 252 gives more detailed information about these records.

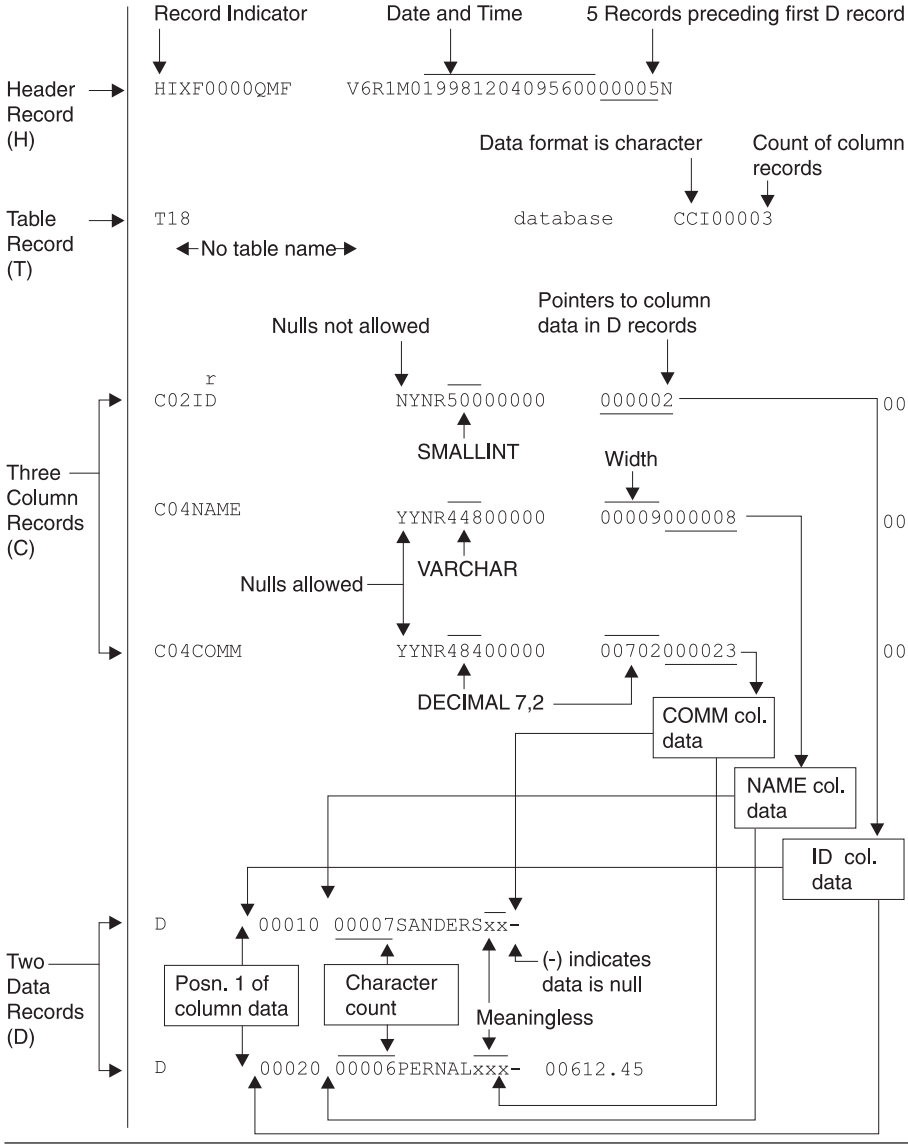


Figure 71. Format of sample IXF records (OUTPUTMODE=CHARACTER)

Now suppose the same table is exported using the IXF format but with OUTPUTMODE=BINARY. As in the previous example, the exported file, data set, or CICS data queue consists of seven records which are shown here:

```
HIXF0000QMF   V7R1M019981204095650000005N
T18                          database        CMI00003
C02ID               NYNR50000000          000003                    00
```

```
C04NAME              YYNR44800000     00009000005                    00
C04COMM              YYNR48400000     00702000018                    00
D    xxxxxxxxSANDERSxxxxxxxxx
D    xxxxxxxxPERNALxxxxxxxxx
```

With the exception of bytes 44 through 49 (starting position of column data), the information in the H, T, and C records is essentially the same. The data in the D records, however, differs significantly. Figure 72 on page 254 contains more information about the records of the exported file, data set, or CICS data queue.

```
HIXF0000QMF     V6R1M0199812040956500000 5N
                                         Data format is binary
T18                  database     CMI00003
                                         Pointer to start of
                                         data for ID column
C02ID          NYNR50000000          000003                    00
                                         Pointer to start of
                                         data for NAME column
C04NAME        YYNR44800000     00009000005                    00
                                         Pointer to start of
                                         data for COMM column
C04COMM        YYNR48400000     00702000018                    00

D        xxxxxxxxSANDERSxxxxxxxxx

D        xxxxxxxxPERNALxxxxxxxxx
```

The two data (D) records are shown below in hexadecimal notation with the various fields explained:

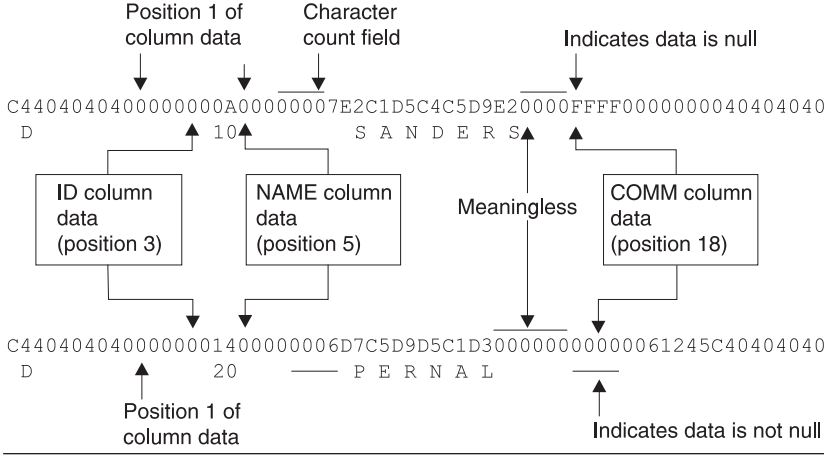


Figure 72. Format of sample IXF records (OUTPUTMODE=BINARY)

Appendix D. Product Interface Macros

The macros identified in this appendix are provided by QMF as General-Use Programming Interfaces for customers.

Warning: Do not use any QMF macros as programming interfaces other than those identified in this appendix.

Product interface macro

- DSQQMF*n*

  where *n* is the NLF identifier. For English, this identifier is E.

Callable interface macros

- **Assembler**
  - **DSQCIA**
  - **DSQCOMMA**
- **C/370**
  - **DSQCIC**
  - **DSQCICE**
  - **DSQCOMMC**
- **COBOL**
  - **DSQCIB**
  - **DSQCOMMB**
- **FORTRAN**
  - **DSQCIF**
  - **DSQCIFE**
  - **DSQCOMMF**
- **PL/I**
  - **DSQCIPL**
  - **DSQCIPX**
  - **DSQCOMML**
- **REXX**
  - **DSQCIX**

Command interface macro

- DSQCCI

QMF governor exit interface macros

- DXEGOVA
- DXEXCBA

QMF user edit exit macro

- DXEECS

Appendix E. QMF Global Variable Tables

QMF provides many variables for use in your applications. In Version 3, QMF introduced the current naming convention for the callable interface. The corresponding command interface variable names are still valid.

The callable interface global variable names can be up to 18 characters long. Callable interface users can use either the old (eight character) names or the new (18 character) names; however, using the new names is recommended. Command interface users *must* use the old names.

The new naming convention is **DSQcc_xxxxxxxxxxx**

cc Can be any one of the following category identifiers:

- **AP** Profile-related state information
- **AO** Other (not profile-related) state information
- **CM** Information about the message produced by the previous command
- **CP** Information about the Table Editor
- **DC** Controls how QMF displays information on the screen
- **EC** Controls how QMF executes commands and procedures
- **QC** Variables produced by a CONVERT QUERY option
- **QM** RUN QUERY error message information
- **QW** Variables unique to QMF for Windows

_ An underscore character

xxxxxxxxxxx
A descriptive name up to 12 characters long

Beginning with Version 3.3, QMF provides a special procedure named Q.SYSTEM_INI that allows you to customize global variables at initialization. See the QMF *Installing and Managing* book for your operating system for more information.

DSQ Global Variables for Profile-Related State Information

None of these global variables can be modified by the SET GLOBAL command.

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQAP_CASE	DSQAPCAS	01	CASE parameter. Values can be: **1** for UPPER **2** for MIXED **3** for STRING
DSQAP_CONFIRM	DSQAPRMP	01	CONFIRM parameter. Values can be: **0** for NO **1** for YES
DSQAP_DECIMAL	DSQAPDEC	01	DECIMAL parameter. Values can be: **1** for PERIOD **2** for COMMA **3** for FRENCH
DSQAP_LENGTH	DSQAPLEN	18	LENGTH parameter. Its value is that of the parameter. ('1' through '999' or 'CONT')
DSQAP_PFKEY_TABLE	DSQAPPFK	31	Name of the function keys table
DSQAP_PRINTER	DSQAPPRT	08	PRINTER parameter. Values can be: • A nickname for a GDDM printer. • Blanks for the printer associated with DSQPRINT.
DSQAP_QUERY_LANG	DSQAPLNG	01	LANGUAGE parameter. Values can be: **1** for SQL **2** for QBE **3** for PROMPTED
DSQAP_QUERY_MODEL	DSQAMODP	01	MODEL parameter. Value can be '1' for RELATIONAL
DSQAP_RESOURC_GRP	DSQAPGRP	16	RESOURCE GROUP parameter.
DSQAP_SPACE	DSQAPSPC	50	SPACE parameter. Its value is that of the parameter.
DSQAP_SYNONYM_TBL	DSQAPSYN	31	SYNONYMS parameter.
DSQAP_TRACE	DSQAPTRC	18	TRACE parameter. Values can be: **ALL** (maximum tracing) **NONE** (minimum tracing) Specifications for individual QMF components (Example: A2L2C1)
DSQAP_WIDTH	DSQAPWID	18	WIDTH parameter. Its value is that of the parameter. ('22' through '999')

DSQ Global Variables for State Information Not Related to the Profile

None of these global variables can be modified by the SET GLOBAL command.

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQAO_APPL_TRACE	DSQATRAC	01	Application trace level. Values can be: **0** for level A0 **1** for level A1 **2** for level A2
DSQAO_ATTENTION	DSQCATTN	01	User attention flag.
DSQAO_BATCH	DSQABATC	01	Batch or interactive mode. Value will be: **1** for an interactive session. **2** for a batch-mode session.
DSQAO_CONNECT_ID	DSQAAUTH	08	The user ID used to connect to the database. (This is the user ID under which work is done.)
DSQAO_CONNECT_LOC	none	18	The location name of the database to which the user is currently connected. The name is 18 characters (padded to the right with blanks, if necessary).
DSQAO_CURSOR_OPEN	DSQACRSR	01	Database cursor status. Values can be: **1** if the cursor is open. **2** if the cursor is closed.
DSQAO_DB_MANAGER	DSQADBMG	01	Database manager. Values can be: **1** for DB2 for VM/ESA or VSE/ESA **2** for DB2 for MVS/ESA **3** for workstation database servers
DSQAO_DBCS	DSQADBCS	01	DBCS support status. Values can be: **1** for DBCS support. **2** for no DBCS support.

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQAO_FORM_PANEL	DSQASUBP	02	Current form panel. Values can be: **1** for FORM.MAIN **2** for FORM.COLUMN **3** for FORM.PAGE **4** for FORM.FINAL **5** for FORM.BREAK1 **6** for FORM.BREAK2 **7** for FORM.BREAK3 **8** for FORM.BREAK4 **9** for FORM.BREAK5 **10** for FORM.BREAK6 **11** for FORM.OPTIONS **12** for FORM.CALC **13** for FORM.DETAIL **14** for FORM.CONDITIONS A blank value means the form does not exist in QMF temporary storage.
DSQAO_INTERACT	DSQAIACT	01	Setting of interact flag. Values can be: **0** for no interactive execution. **1** for interactive execution allowed.
DSQAO_LOCAL_DB2	none	18	The location name of the local DB2 database. This is the location name for the subsystem named in the variable DSQAO_SUBSYS_ID. In a remote unit of work environment, DSQ_LOCAL_DB2 is the name of the application requester. The name is 16 characters (padded to the right with blanks, if necessary). This field is blank if QMF is running in the VM or VSE environment.
DSQAO_LOCATION	DSQAITLO	16	Location name of the current object, if any. This value is applicable only if a three-part name was used.
DSQAO_NLF_LANG	DSQALANG	01	National language of user. For the English language environment, this is 'E'.

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQAO_NUM_FETCHED	DSQAROWS	16	Fetched data rows. Contains '0' when the DATA object is empty.
DSQAO_OBJ_NAME	DSQAITMN	18	The name of the table (contained in a report), query, procedure, or form shown on the currently displayed panel. If the current panel does not display an object, or if the displayed object has no name, this variable contains blanks.
DSQAO_OBJ_OWNER	DSQAITMO	08	The owner of the table (contained in a report), query, procedure, or form shown on the currently displayed panel. If the current panel does not display an object, or if the displayed object has no owner, this variable contains blanks.
DSQAO_PANEL_TYPE	DSQAITEM	01	Type of current panel. Values can be: **1** for HOME **2** for QUERY **3** for REPORT **4** for FORM **5** for PROC **6** for PROFILE **7** for CHART **8** for LIST **9** for Table Editor **A** for GLOBALS
DSQAO_QMF_RELEASE	DSQAREVN	02	Numeric release number of QMF. For QMF Version 7, this is '12'.
DSQAO_QMF_VER_RLS	DSQAQMF	10	Version and release of QMF. • For QMF Version 7 • this is 'QMF V7'.
DSQAO_QRY_SUBTYPE	DSQASUBI	01	Query subtype. Values can be: **1** for a subtype of SQL **2** for a subtype of QBE **3** for a subtype of PROMPTED Blank means the current panel is not QUERY.
DSQAO_QUERY_MODEL	DSQAMODL	01	Model of current query. Value can be '1' for RELATIONAL

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQAO_SAME_CMD	DSQACMDM	01	Values can be: **0** if the two commands aren't the same. **1** if the two commands are the same.
DSQAO_SUBSYS_ID	none	04	If QMF is running in TSO, this is the ID of the local DB2 subsystem to which QMF is attached. If you specify a value for the DSQSUBS program parameter from CMS or CICS, this global variable contains that value. This happens because the parameter is tolerated and the value is not processed; that is, the value is placed in the global variable field and nothing is done with it. This logic permits the same EXEC to be used in multiple environments.
DSQAO_SYSTEM_ID	DSQASYST	01	Current operating system. Values can be: **1** for VM/SP **2** for MVS/SP **3** for MVS/XA or MVS/ESA **4** for VM/XA or VM/ESA **5** for CICS
DSQAO_TERMINATE	DSQCSESC	01	QMF termination flag. Values can be: **0** if the session was not marked. **1** if the session was marked.
DSQAO_VARIATION	DSQAVARN	02	Form panel variation number. Blank means FORM.DETAIL is not the current panel.

DSQ Global Variables Associated with CICS

Of the variables in this table, only DSQAP_CICS_PQNAME and DSQAP_CICS_PQTYPE can be modified by the SET GLOBAL command.

When the queue type is TD, the maximum length of the corresponding queue name is 4. For example, if DSQAO_CICS_SQTYPE is TD, the maximum length of DSQAO_CICS_SQNAME is 4.

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQAP_CICS_PQNAME	none	08	Names the CICS data queue to contain the QMF print.
DSQAP_CICS_PQTYPE	none	02	Type of CICS storage used to contain the QMF print. **TS** writes the QMF print to a CICS temporary storage queue on an "auxiliary" storage device. This is the default. **TD** writes the QMF print to a CICS transient data queue.
DSQAO_CICS_SQNAME	none	08	Names the CICS data queue to be used as the spill file.
DSQAO_CICS_SQTYPE	none	02	Type of CICS storage used to contain the QMF spill file. **TS** writes the QMF spill file to a CICS temporary storage queue on an "auxiliary" storage device. This is the default. **TD** writes the QMF spill file to a CICS transient data queue.
DSQAO_CICS_TQNAME	none	08	Names the CICS data queue to contain the QMF trace.
DSQAO_CICS_TQTYPE	none	02	Type of CICS storage used to contain the QMF trace. **TS** writes the QMF trace to a CICS temporary storage queue on an "auxiliary" storage device. **TD** writes the QMF trace to a CICS transient data queue. This is the default.

DSQ Global Variables Related to a Message Produced by the Previous Command

None of these global variables can be modified by the SET GLOBAL command.

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQCM_MESSAGE	DSQCIMSG	80	Message text
DSQCM_MSG_HELP	DSQCIMID	08	ID of message help panel
DSQCM_MSG_NUMBER	DSQCIMNO	08	Message number

QMF Global Variables

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQCM_SUB_TXT_*nn*	DSQCIM*nn*	20	Substitution value *nn*
DSQCM_SUBST_VARS	DSQCIM00	04	Number of substitution variables in the message

DSQ Global Variables Associated with Table Editor

All of these global variables can be modified by the SET GLOBAL command.

If the CONFIRM option of the EDIT TABLE command is NO, the Table Editor suppresses the display of all confirmation panels. If the CONFIRM option is YES, the Table Editor determines which categories of confirmation are enabled by checking the values of the global variables shown in this table.

The Table Editor defaults depend on the SAVE keyword from the EDIT TABLE command:

- When SAVE=IMMEDIATE, the default for each category is to enable.
- When SAVE=END, the default for the DELETE, MODIFY, and END/CANCEL categories is to enable; the default for the ADD and CHANGE categories is to disable.

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQCP_TEADD	none	01	Displays a confirmation panel after an ADD subcommand. Values can be: **0** panel is disabled. **1** panel is enabled. **2** panel is enabled or disabled depending on the Table Editor defaults. This is the default.
DSQCP_TECHG	none	01	Displays a confirmation panel after a CHANGE subcommand. Values can be: **0** panel is disabled. **1** panel is enabled. **2** panel is enabled or disabled depending on the Table Editor defaults. This is the default.

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQCP_TEEND	none	01	Displays a confirmation panel when the user issues an END subcommand or a CANCEL subcommand to terminate a Table Editor subsession. The panel can appear in several variations, depending on whether or not END or CANCEL was issued, whether modifications were made to the database, and whether the screen contained modified data when END or CANCEL was issued. Values can be: **0** panel is disabled. **1** panel is enabled. **2** panel is enabled or disabled depending on the Table Editor defaults. This is the default.
DSQCP_TEDEL	none	01	Displays a confirmation panel after a DELETE subcommand. Values can be: **0** panel is disabled. **1** panel is enabled. **2** panel is enabled or disabled depending on the Table Editor defaults. This is the default.
DSQCP_TEDFLT	none	01	The reserved character used to indicate the default value for a column in the Table Editor. Initially set to a plus sign (+) character.
DSQCP_TEDFLT_DBCS	none	04	The reserved DBCS character used to indicate the default value for a graphic string column in the Table Editor. The value must be a four-byte, mixed string, composed of one DBCS character, preceded by the shift-out character, and followed by the shift-in character. Initially set to a DBCS plus sign (+) character. Note that this global variable is used only in a DBCS environment.

QMF Global Variables

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQCP_TEMOD	none	01	Displays a confirmation panel when displayed data is modified and a PREVIOUS, CLEAR, SHOW CHANGE, SHOW SEARCH, REFRESH, or NEXT subcommand is issued. The resulting panel includes the name of the subcommand as part of the panel text. Values can be: **0** panel is disabled. **1** panel is enabled. **2** panel is enabled or disabled depending on the Table Editor defaults.
DSQCP_TENULL	none	01	The reserved character used to indicate the null value for a column in the Table Editor. Initially set to a hyphen (-) character.
DSQCP_TENULL_DBCS	none	04	The reserved DBCS character used to indicate the null value (or, in the context of search criteria, to indicate ignore) for a graphic string column in the Table Editor. The value must be a four-byte, mixed string, composed of one DBCS character, preceded by the shift-out character, and followed by the shift-in character. Initially set to a DBCS hyphen (-) character. Note that this global variable is used only in a DBCS environment.

DSQ Global Variables That Control How Information is Displayed on the Screen

All of these global variables can be modified by the SET GLOBAL command.

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQDC_COST_EST	none	01	Optionally suppress database cost estimate. Values can be: **0** = no—Do not display the cost estimate. **1** = yes—Display the cost estimate. This is the default.

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQDC_CURRENCY	none	18	The currency symbol used when the DC edit code is specified. The value can be a string with a length from 1 to 18 bytes. For English, the default is the euro currency symbol. The default varies for other languages. In a DBCS environment, this value can be a mixed string of SBCS and DBCS characters. The total length of the mixed string, including the shift-out and shift-in characters, cannot exceed 18 bytes.
DSQDC_DISPLAY_RPT	DSQADPAN	01	Display report after RUN QUERY. Values can be: **0** if you don't want QMF to display the resulting report from a RUN query command. This is the default if QMF is started interactively with DSQQMFE or in BATCH mode. Changing this variable when QMF is started in BATCH mode will not cause any QMF screen to display. **1** if you want QMF to automatically display the report. This is the default if QMF is started with the callable interface. This can be overridden with the DSQADPAN program parameter on the START command. This global variable is for applications only. It has no effect when the RUN QUERY command is entered on the command line.

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQDC_LIST_ORDER	none	02	Sets the default sort order for objects in a list of database objects. Values for the first character can be: **1** The list will use the default order **2** The list will be sorted by object owner. **3** The list will be sorted by object name. **4** The list will be sorted by object type. **5** The list will be sorted by date modified. **6** The list will be sorted by date last used. Values for the second character can be: **A** The list will be sorted in ascending order. **D** The list will be sorted in descending order. This variable applies only to objects that are listed as a result of the LIST command. It does not apply to lists produced in other contexts, such as from a Display prompt panel, and it does not apply to lists of tables.

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQDC_SCROLL_AMT	none	04	Sets the scroll amount for QMF panels. Values can be: **Csr** Sets scroll amount to cursor. Depending on whether the user scrolls backward, forward, left, or right, QMF scrolls the line or column where the cursor is positioned to the bottom, top, far left, or far right of the scrollable area. **Half** Sets scroll amount to half the scrollable area. **Page** Sets scroll amount to a full page. This is the default. **n** Sets scroll amount to *n* number of lines or columns. *n* can be any number from 1 to 9999.
DSQDC_SHOW_PANID	DSQCPDSP	01	Display panel IDs on CUA-like panels. Values can be: **0** Suppress panel identifiers. This is the default. **1** Display panel identifiers.

DSQ Global Variables That Control How Commands and Procedures Are Executed

All of these global variables can be modified by the SET GLOBAL command.

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQEC_ALIASES	none	31	View for retrieving lists of table and view aliases when the user requests a list of tables from a DB2 for MVS/ESA location or if the current server is DB2 for MVS/ESA, or a workstation database server.
DSQEC_COLS_LDB2	none	31	View for retrieving column information for a table at the current location, if that location is DB2.
DSQEC_COLS_RDB2	none	31	View for retrieving column information for a table at a remote DB2 location (if it is not the current location).

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQEC_COLS_SQL	none	31	View for retrieving column information for a table in a DB2 for VM/ESA or VSE/ESA database.
DSQEC_FORM_LANG	none	01	Establishes the default NLF language in a saved or exported form. Values can be: **0** The form will use the presiding NLF language. **1** The form will use English. This is the default.
DSQEC_ISOLATION	none	01	Default Query isolation level. Values can be: **0** Isolation level UR, Uncommitted Read. **1** Isolation level CS, Cursor Stability. This is the default. **Attention:** Setting the value to '0' can introduce non-existent data into a QMF report. Do not set the value to '0' if your QMF reports must be free of non-existent data. **Limited support:** For QMF 7.1 the use of the value '0' is only effective with the following database servers (those supporting the SQL WITH clause): • DB2 for MVS V4 or higher • DB2 for VM/VSE V4 or higher
DSQEC_NLFCMD_LANG	none	01	Set expected NLF language for commands. Values can be: **0** Commands must be in the presiding NLF language. This is the default. **1** Commands must be in English.

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQEC_RERUN_IPROC	none	01	Rerun invocation procedure after the END command. Values can be: **0** Suppress rerun of invocation procedure after the END command. **1** Rerun the invocation procedure after the END command. This is the default. If you start QMF with an invocation procedure, then set this variable to '0', QMF terminates instead of rerunning the procedure.
DSQEC_RESET_RPT	none	31	Determines whether or not QMF prompts the user when an incomplete DATA object in temporary storage appears to be affecting performance. Possible values are: **0** Reset Report Prompt Panel is not displayed and QMF completes the running report. This is the default value. **1** Reset Report Prompt Panel is displayed. This panel prompts the user to complete or reset the currently running report before starting the new command. **2** Reset Report Prompt Panel is not displayed and QMF resets the currently running report.
DSQEC_SHARE	none	31	Specifies the default value for the SHARE parameter. The possible values are: **0** Do not share data with other users. **1** Do share data with other users.
DSQEC_TABS_LDB2	none	31	View for retrieving lists of tables and views at the current server, if it is DB2 for MVS/ESA, or a workstation database server.
DSQEC_TABS_RDB2	none	31	View for retrieving lists of tables and views at remote DB2 subsystems.

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQEC_TABS_SQL	none	31	View for retrieving lists of tables and views for a DB2 for VM/ESA or VSE/ESA database.

DSQ Global Variables That Show Results of CONVERT QUERY

None of these global variables can be modified by the SET GLOBAL command.

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQQC_LENGTH_*nnn*	DSQCL*nnn*	05	Length of converted result *nnn*
DSQQC_QRY_COUNT	DSQCQCNT	03	Number of queries in converted result. Value must always be '1' unless the original query is a QBE I. or U. query.
DSQQC_QRY_LANG	DSQCQLNG	01	Language of converted query. Values can be: **1** for SQL **2** for QBE **3** for prompted
DSQQC_QRY_TYPE	DSQCQTYP	not specified	First word in converted results
DSQQC_RESULT_*nnn*	DSQCQ*nnn*	not specified	Converted result *nnn*

DSQ Global Variables That Show RUN QUERY Error Message Information

None of these global variables can be modified by the SET GLOBAL command.

Callable Interface Variable Name	Command Interface Variable Name	Length	Description
DSQQM_MESSAGE	DSQCIQMG	80	Text of query message
DSQQM_MSG_HELP	DSQCIQID	08	ID of message help panel
DSQQM_MSG_NUMBER	DSQCIQNO	08	Message number
DSQQM_SQL_RC	DSQCISQL	16	The SQLCODE from the last command or query.
DSQQM_SQL_STATE	none	05	The SQLSTATE associated with the SQLCODE in DSQQM_SQL_RC, if SQLSTATE is returned by the database manager.
DSQQM_SUB_TXT_*nn*	DSQCIQ*nn*	20	Substitution value *nn*
DSQQM_SUBST_VARS	DSQCIQ00	04	Number of substitution variables

Appendix F. 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
San Jose, CA 95141-1003
U.S.A.

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

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

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

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

Lotus and 1-2-3 are trademarks of Lotus Development Corporation in the Unites States, other counrties, 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.

Glossary of Terms and Acronyms

This glossary defines terms as they are used throughout the QMF library. If you do not find the term you are looking for, refer to the index in this book, or to the *IBM Dictionary of Computing*.

abend. The abnormal termination of a task.

ABENDx. The keyword for an abend problem.

Advanced Peer-to-Peer Networking. A distributed network and session control architecture that allows networked computers to communicate dynamically as equals. Compare with Advanced Program-to-Program Communication (APPC). An implementation of the SNA synchronous data link control LU 6.2 protocol that allows interconnected systems to communicate and share the processing of programs.

Advanced Program-to-Program Communication (APPC). An implementation of the SNA synchronous data link control LU 6.2 protocol that allows interconnected systems to communicate and share the processing of programs.

aggregation function. Any of a group of functions that summarizes data in a column. They are requested with these usage codes on the form panels: AVERAGE, CALC, COUNT, FIRST, LAST, MAXIMUM, MINIMUM, STDEV, SUM, CSUM, PCT, CPCT, TPCT, TCPCT.

aggregation variable. An aggregation function that is placed in a report using either the FORM.BREAK, FORM.CALC, FORM.DETAIL, or FORM.FINAL panels. Its value appears as part of the break footing, detail block text, or final text when the report is produced.

alias. In DB2 UDB for OS/390, an alternate name that can be used in SQL statements to refer to a table or view in the same or a remote DB2 UDB for OS/390 subsystem. In OS/2, an alternate name used to identify a object, a database, or a network resource such as an LU. In QMF, a locally defined name used to access a QMF table or view stored on a local or remote DB2 UDB for OS/390 subsystem.

APAR. Authorized Program Analysis Report.

APPC. Advanced Program-to-Program Communication

application. A program written by QMF users that extends the capabilities of QMF without modifying the QMF licensed program. Started from a QMF session by issuing a RUN command for a QMF procedure, an installation-defined command, or a CMS or TSO command that invokes an EXEC or CLIST, respectively.

application requester. (1) A facility that accepts a database request from an application process and passes it to an application server. (2) In DRDA, the source of a request to a remote relational database management system.

The application requester is the DBMS code that handles the QMF end of the distributed connection. The local DB2 UDB for OS/390 subsystem to which QMF attaches is known as the application requester for QMF, because DB2 UDB for OS/390's application requester is installed within the local database

manager. Therefore, an entire DB2 UDB for OS/390 subsystem (including data) is associated with the application requester, but the SQL statements are processed at the current location. This subsystem is called the "local DB2 UDB for OS/390".

With DB2 for VM and VSE the application requester runs in the same virtual machine as QMF; that is, no database is inherently associated with the DB2 for VM and VSE application requester.

application server. The target of a request from an application requester. (1) The local or remote database manager to which the application process is connected. The application server executes at the system containing the desired data. (2) In DRDA, the target of a request from an application requester. With DB2 UDB for OS/390, the application server is part of a full DB2 UDB for OS/390 subsystem.

With DB2 for VM and VSE, the application server is part of a DB2 for VM and VSE database machine.

application-support command. A QMF command that can be used within an application program to exchange information between the application program and QMF. These commands include INTERACT, MESSAGE, STATE, and QMF.

area separator. The barrier that separates the fixed area of a displayed report from the remainder of the report.

argument. An independent variable.

base QMF environment. The English-language environment of QMF, established when QMF is installed. Any other language environment is established after installation.

batch QMF session. A QMF session running in the background. Begins when a specified QMF procedure is invoked and ends when the procedure ends. During a background QMF session, no user interaction and panel display interaction are allowed.

bind. In DRDA, the process by which the SQL statements in an application program are made known to a database management system over application support protocol (and database support protocol) flows. During a bind, output from a precompiler or preprocessor is converted to a control structure called a package. In addition, access paths to the referenced data are selected and some authorization checking is performed. (Optionally in DB2 UDB for OS/390, the output may be an application plan.)

built-in function. Generic term for scalar function or column function. Can also be "function."

calculation variable. CALCid is a special variable for forms that contains a user-defined calculated value. CALCid is defined on the FORM.CALC panel.

callable interface. A programming interface that provides access to QMF services. An application can access these services even when the application is running outside of a QMF session. Contrast with command interface.

chart. A graphic display of information in a report.

CICS. Customer Information Control System.

client. A functional unit that receives shared services from a server.

CMS. Conversational Monitor System.

column. A vertical set of tabular data. It has a particular data type (for example, character or numeric) and a name. The values in a column all have the same data characteristics.

column function. An operation that is applied once to all values in a column, returns a single value as a result, and is expressed in the form of a function name followed by one or more arguments enclosed in parentheses.

column heading. An alternative to the column name that a user can specify on a form. Not saved in the database, as are the column name and label.

column label. An alternative descriptor for a column of data that is saved in the database. When used, column labels appear by default on the form, but they can be changed by users.

column wrapping. Formatting values in a report so that they occupy several lines within a column. Often used when a column contains values whose length exceeds the column width.

command interface. An interface for running QMF commands. The QMF commands can only be issued from within an active QMF session. Contrast with callable interface.

command synonym. The verb or verb/object part of an installation-defined command. Users enter this for the command, followed by whatever other information is needed.

command synonym table. A table each of whose rows describes an installation-defined command. Each user can be assigned one of these tables.

commit. The process that makes a data change permanent. When a commit occurs, data locks are freed enabling other applications to reference the just-committed data. See also “rollback”.

concatenation. The combination of two strings into a single string by appending the second to the first.

connectivity. The enabling of different systems to communicate with each other. For example, connectivity between a DB2 UDB for OS/390 application requester and a DB2 for VM and VSE application server enables a DB2 UDB for OS/390 user to request data from a DB2 for VM and VSE database.

conversation. A logical connection between two programs over an LU 6.2 session that allows them to communicate with each other while processing a transaction.

correlation name. An alias for a table name, specified in the FROM clause of a SELECT query. When concatenated with a column name, it identifies the table to which the column belongs.

CP. The Control Program for VM.

CSECT. Control section.

current location. The application server to which the QMF session is currently connected. Except for connection-type statements, such as CONNECT (which are handled by the application requester), this server processes all the SQL statements. When initializing QMF, the current location is indicated by the DSQSDBNM startup program parameter. (If that parameter is not specified, the local DB2 UDB for OS/390 subsystem

current object. An object in temporary storage currently displayed. Contrast with saved object.

Customer Information Control System (CICS). An IBM licensed program that enables transactions entered at remote terminals to be processed concurrently by user-written application programs. It includes facilities for building, using, and maintaining databases.

DATA. An object in temporary storage that contains the information returned by a retrieval query. Information represented by alphanumeric characters contained in tables and formatted in reports.

database. A collection of data with a given structure for accepting, storing, and providing on demand data for multiple users. In DB2 UDB for OS/390, a created object that contains table spaces and index spaces. In DB2 for VM and VSE, a collection of tables, indexes, and supporting information (such as control information and data recovery information) maintained by the system. In OS/2, a collection of information, such as tables, views, and indexes.

database administrator. The person who controls the content of and access to a database.

database management system. A computer-based system for defining, creating, manipulating, controlling, managing, and using databases. The database management system also has transaction management and data recovery facilities to protect data integrity.

database manager. A program used to create and maintain a database and to communicate with programs requiring access to the database.

database server. (1) In DRDA, the target of a request received from an application server (2) In OS/2, a workstations that provides database services for its local database to database clients.

date. Designates a day, month, and year (a three-part value).

date/time default formats. Date and time formats specified by a database manager installation option. They can be the EUR, ISO, JIS, USA, or LOC (LOCAL) formats.

date/time data. The data in a table column with a DATE, TIME, or TIMESTAMP data type.

DB2 UDB for OS/390. DB2 Universal Database for OS/390 (an IBM relational database management system).

DB2 for AIX. DATABASE2 for AIX. The database manager for QMF's relational data.

DBCS. Double-byte character set.

DBMS. Database management system.

default form. The form created by QMF when a query is run. The default form is not created if a saved form is run with the query.

destination control table (DCT). In CICS, a table containing a definition for each transient data queue.

detail block text. The text in the body of the report associated with a particular row of data.

detail heading text. The text in the heading of a report. Whether or not headings will be printed is specified in FORM.DETAIL.

dialog panel. A panel that overlays part of a Prompted Query primary panel and extends the dialog that helps build a query.

distributed data. Data that is stored in more than one system in a network, and is available to remote users and application programs.

distributed database. A database that appears to users as a logical whole, locally accessible, but is comprised of databases in multiple locations.

distributed relational database. A distributed database where all data is stored according to the relational model.

Distributed Relational Database Architecture. A connection protocol for distributed relational database processing that is used by IBM and vendor relational database products.

distributed unit of work. A method of accessing distributed relational data in which users or applications can, within a single unit of work, submit SQL statements to multiple relational database management systems, but no more than one RDBMS per SQL statement.

DB2 UDB for OS/390 introduced a limited form of distributed unit of work support in its V2R2 called system-directed access, which QMF supports.

DOC. The keyword for a document problem.

double-byte character. An entity that requires two character bytes.

double-byte character set (DBCS). A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols that can be represented by 256 code points, require double-byte character sets. Because each character requires two bytes, the typing, display, and printing of DBCS characters requires hardware and programs that support DBCS. Contrast with single-byte character set.

DRDA. Distributed Relational Database Architecture.

duration. An amount of time expressed as a number followed by one of seven keywords: YEARS, MONTHS, DAYS, HOURS, MINUTES, SECONDS, MICROSECONDS.

EBCDIC. Extended Binary-Coded Decimal Interchange Code.

echo area. The part of the Prompted Query primary panel in which a prompted query is built.

EUR (European) format. A format that represents date and time values as follows:

- Date: dd.mm.yyyy
- Time: hh.mm.ss

extended syntax. QMF command syntax that is used by the QMF callable interface; this syntax defines variables that are stored in the storage acquired by the callable interface application and shared with QMF

example element. A symbol for a value to be used in a calculation or a condition in a QBE query.

example table. The framework of a QBE query.

fixed area. That part of a report that contains fixed columns.

fixed columns. The columns of a report that remain in place when the user scrolls horizontally. On multiple-page, printed reports, these columns are repeated on the left side of each page.

form. An object that contains the specifications for printing or displaying a report or chart. A form in temporary storage has the name of FORM.

function key table. A table containing function key definitions for one or more QMF panels, along with text describing the keys. Each user can be assigned one of these tables.

gateway. A functional unit that connects two computer networks of different network architectures. A gateway connects networks or systems of different architectures, as opposed to a bridge, which connects networks or systems with the same or similar architectures.

GDDM. Graphical Data Display Manager.

global variable. A variable that, once set, can be used for an entire QMF session. A global variable can be used in a procedure, query, or form. Contrast with run-time variable.

Graphical Data Display Manager. A group of routines that allows pictures to be defined and displayed procedurally through function routines that correspond to graphic primitives.

grouped row. A row of data in a QBE target or example table that is summarized either by a G. or a built-in function.

HELP. Additional information about an error message, a QMF panel, or a QMF command and its options.

host. A mainframe or mid-size processor that provides services in a network to a workstation.

HTML. Hypertext Markup Language. A standardized markup language for documents displayed on the World Wide Web.

ICU. Interactive Chart Utility.

INCORROUT. The keyword for incorrect output.

index. A collection of data about the locations of records in a table, allowing rapid access to a record with a given key.

initial procedure. A QMF procedure specified by the DSQSRUN parameter on the QMF start command which is executed immediately after QMF is invoked.

initialization program. A program that sets QMF program parameters. This program is specified by DSQSCMD in the callable interface. The default program for interactive QMF is DSQSCMD*n*, where *n* is the qualifier for the presiding language ('E' for English).

installation-defined command. A command created by an installation. QMF will process it as one of its own commands or as a combination of its commands.

installation-defined format. Date and time formats, also referred to as LOCAL formats, that are defined (or built) by the installation.

interactive execution. Execution of a QMF command in which any dialog that should take place between the user and QMF during the command's execution actually does take place.

interactive session. Any QMF session in which the user and QMF can interact. Could be started by another interactive session by using the QMF INTERACT command.

interactive switch. A conceptual switch which, when on, enables an application program to run QMF commands interactively.

invocation CLIST or EXEC. A program that invokes (starts) QMF.

ISO (International Standards Organization) format. A format that represents date and time values as follows:

- Date: yyyy-mm-dd
- Time: hh.mm.ss

ISPF. Interactive System Productivity Facility.

IXF. Integration Exchange Format: A protocol for transferring tabular data among various software products.

JCL. Job control language for OS/390.

job control. In VSE, a program called into storage to prepare each job or job step to be run. Some of its functions are to assign I/O devices to symbolic names, set switches for program use, log (or print) job control statements, and fetch the first phase of each job step.

JIS (Japanese Industrial Standard) format. A format that represents date and time values as follows:

- Date: yyyy-mm-dd
- Time: hh:mm:ss

join. A relational operation that allows retrieval of data from two or more tables based on matching columns that contain values of the same data type.

keyword parameter. An element of a QMF command consisting of a keyword and an assigned value.

like. Pertaining to two or more similar or identical IBM operating environments. For example, like distribution is distribution between two DB2 UDB for OS/390's with compatible server attribute levels. Contrast with "unlike".

literal. In programming languages, a lexical unit that directly represents a value. A character string whose value is given by the characters themselves.

linear procedure. Any procedure *not* beginning with a REXX comment. A linear procedure can contain QMF commands, comments, blank lines, RUN commands, and substitution variables. See also "procedure with logic."

linear syntax. QMF command syntax that is entered in one statement of a program or procedure, or that can be entered on the QMF command line.

line wrapping. Formatting table rows in a report so they occupy several lines. The row of column names and each row of column values are split into as many lines as are required by the line length of the report.

local. Pertaining to the relational database, data, or file that resides in the user's processor. See also "local DB2 UDB for OS/390", and contrast with *remote*.

local area network (LAN). (1) Two or more processors connected for local resource sharing (2) A network within a limited geographic area, such as a single office building, warehouse, or campus.

local data. Data that is maintained by the subsystem that is attempting to access the data. Contrast with remote data.

local DB2 UDB for OS/390. With DB2 UDB for OS/390, the application requester is part of a DB2 UDB for OS/390 subsystem that is running in the same MVS system as QMF. Therefore, an entire DB2 UDB for OS/390 subsystem (including data) is associated with the application requester, but the SQL statements are processed at the current location. This subsystem is where the QMF plan is bound.

When QMF runs in TSO, this subsystem is specified using DSQSSUBS startup program parameter. When QMF runs in CICS, this subsystem is identified in the Resource Control Table (RCT). The local DB2 UDB for OS/390 is the subsystem ID of the DB2 UDB for OS/390 that was started in the CICS region.

location. A specific relational database management system in a distributed relational database system. Each DB2 UDB for OS/390 subsystem is considered to be a location.

logical unit (LU). A port through which an end user accesses the SNA network to communicate with another end user and through which the end user accesses the functions provided by system services control points.

Logical Unit type 6.2 (LU 6.2). The SNA logical unit type that supports general communication between programs in a distributed processing environment.

LU. Logical unit.

LU 6.2. Logical Unit type 6.2.

LOOP. The keyword for an endless-loop problem.

MSGx. The keyword for a message problem.

Multiple Virtual Storage. Implies the MVS/ESA product

MVS/ESA. Multiple Virtual Storage/Enterprise System Architecture (IBM operating system).

NCP. Network Control Program.

Network Control Program (NCP). An IBM licensed program that provides communication controller support for single-domain, multiple-domain, and interconnected network capability.

NLF. National Language Feature. Any of several optional features available with QMF that lets the user select a language other than US English.

NLS. National Language Support.

node. In SNA, an end point of a link or a junction common to two or more links in a network. Nodes can be distributed to host processors, communication controllers, cluster controllers, or terminals. Nodes can vary in routing and other functional capabilities.

null. A special value used when there is no value for a given column in a row. *Null* is not the same as zero.

null value. See *null*.

object. A QMF query, form, procedure, profile, report, chart, data, or table. The report, chart, and data objects exist only in temporary storage; they cannot be saved in a database. The table object exists only in a database.

object name. A character string that identifies an object owned by a QMF user. The character string can be a maximum of 18 bytes long and must begin with an alphabetic character. The term "object name" does not include the "owner name" prefix. Users can access other user's objects only if authorized.

object panel. A QMF panel that can appear online after the execution of one QMF command and before the execution of another. Such panels include the home, report, and chart panels, and all the panels that display a QMF object. They do not include the list, help, prompt, and status panels.

online execution. The execution of a command from an object panel or by pressing a function key.

owner name. The authorization id of the user who creates a given object.

package. The control structure produced when the SQL statements in an application program are bound to a relational database management system. The database management system uses the control structure to process SQL statements encountered during statement execution.

panel. A particular arrangement of information, grouped together for presentation in a window. A panel can contain informational text, entry fields, options the user can choose from, or a mixture of these.

parameter. An element of a QMF command. This term is used generically in QMF documentation to reference a *keyword parameter* or a *positional parameter*.

partner logical unit. In SNA, the remote system in a session.

PERFM. The keyword for a performance problem.

permanent storage. The database where all tables and QMF objects are stored.

plan. A form of package where the SQL statements of several programs are collected together during bind to create a plan.

positional parameter. An element of a QMF command that must be placed in a certain position within the command.

primary panel. The main Prompted Query panel containing your query.

primary QMF session. An interactive session begun from outside QMF Within this session, other sessions can be started by using the INTERACT command.

procedure. An object that contains QMF commands. It can be run with a single RUN command. A procedure in temporary storage has the name of PROC. See also “linear procedure” and “procedure with logic.”

procedure termination switch. A conceptual switch that a QMF MESSAGE command can turn on. While on, every QMF procedure to which control returns terminates immediately.

procedure with logic. Any QMF procedure beginning with a REXX comment. In a procedure with logic, you can perform conditional logic, make calculations, build strings, and pass commands back to the host environment. See also “linear procedure.”

profile. An object that contains information about the characteristics of the user’s session. A stored profile is a profile that has been saved in permanent storage. A profile in temporary storage has the name PROFILE. There can be only one profile for each user.

prompt panel. A panel that is displayed after an incomplete or incorrect QMF command has been issued.

Prompted Query. A query built in accordance with the user’s responses to a set of dialog panels.

protocol. The rules governing the functions of a communication system that must be followed if communication is to be achieved.

PSW. Program status word.

PTF. Program temporary fix.

QBE (Query-By-Example). A language used to write queries graphically. For more information see *Using QMF*

QMF administrative authority. At minimum, insert or delete priviledge for the Q.PROFILES control table.

QMF administrator. A QMF user with QMF administrative authority.

QMF command. Refers to any command that is part of the QMF language. Does **not** include installation-defined commands.

QMF session. All interactions between the user and QMF from the time the user invokes QMF until the EXIT command is issued.

qualifier. When referring to a QMF object, the part of the name that identifies the owner. When referring to a TSO data set, any part of the name that is separated from the rest of the name by periods. For example, ‘TCK’, ‘XYZ’, and ‘QUERY’ are all qualifiers in the data set name ‘TCK.XYZ.QUERY’.

query. An SQL or QBE statement, or a statement built from prompting, that performs data inquiries or manipulations. A saved query is an SQL query, QBE query, or Prompted Query that has been saved in a database. A query in temporary storage, has the name QUERY.

RDBMS. Relational database management system

relational database. A database perceived by its users as a collection of tables.

relational database management system (RDBMS). A computer-based system for defining, creating, manipulating, controlling, managing, and using relational databases.

remote. Pertaining to a relational DBMS other than the local relational DBMS.

remote data. Data that is maintained by a subsystem other than the subsystem that is attempting to access the data. Contrast with local data.

remote data access. Methods of retrieving data from remote locations. The two remote data access functions used by QMF are *remote unit of work* and DB2 UDB for OS/390-only distributed unit of work, which is called *system-directed access*.

remote unit of work. (1) The form of SQL distributed processing where the application is on a system different from the relational database and a single application server services all remote unit of work requests within a single logical unit of work. (2) A unit of work that allows for the remote preparation and execution of SQL statements.

report. The formatted data produced when a query is issued to retrieve data or a DISPLAY command is entered for a table or view.

REXX. Restructured extended executor.

rollback. The process that removes uncommitted database changes made by one application or user. When a rollback occurs, locks are freed and the state of the resource being changed is returned to its state at the last commit, rollback, or initiation. See also *commit*.

row. A horizontal set of tabular data.

row operator area. The leftmost column of a QBE target or example table.

run-time variable. A variable in a procedure or query whose value is specified by the user when the procedure or query is run. The value of a run-time variable is only available in the current procedure or query. Contrast with global variable.

sample tables. The tables that are shipped with QMF. Data in the sample tables is used to help new QMF users learn the product.

saved object. An object that has been saved in the database. Contrast with current object.

SBCS. Single-byte character set.

scalar. A value in a column or the value of a literal or an expression involving other scalars.

scalar function. An operation that produces a single value from another value and is expressed in the form of a function name followed by a list of arguments enclosed in parentheses.

screen. The physical surface of a display device upon which information is presented to the user.

scrollable area. The view of a displayed object that can be moved up, down, left, and right.

server. A functional unit that provides shared services to workstations over a network.

session. All interactions between the user and QMF from the time the user logs on until the user logs off.

single-byte character. A character whose internal representation consists of one byte. The letters of the Latin alphabet are examples of single-byte characters.

SNA. Systems Network Architecture.

SNAP dump. A dynamic dump of the contents of one or more storage areas that QMF generates during an abend.

sort priority. A specification in a retrieval query that causes the sorted values in one retrieved column to determine the sorting of values in another retrieved column.

SQL. Structured Query Language.

SQLCA. Structured Query Language Communication Area.

SSF. Software Support Facility. An IBM online database that allows for storage and retrieval of information about all current APARs and PTFs.

stored object. An object that has been saved in permanent storage. Contrast with current object.

string. A set of consecutive items of a similar type; for example, a character string.

Structured Query Language. A language used to communicate with DB2 UDB for OS/390 and DB2 for VSE or VM. Used to write queries in descriptive phrases.

subquery. A complete SQL query that appears in a WHERE or HAVING clause of another query (the main query or a higher-level subquery).

substitution variable. (1) A variable in a procedure or query whose value is specified either by a global variable or by a run-time variable. (2) A variable in a form whose value is specified by a global variable.

substring. The part of a string whose beginning and length are specified in the SUBSTR function.

System Log (SYSLOG). A data set or file in which job-related information, operational data, descriptions of unusual occurrences, commands, and messages to and from the operator may be stored.

Systems Network Architecture. The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through and controlling the configuration and operation of networks.

table. A named collection of data under the control of the relational database manager. A table consists of a fixed number of rows and columns.

Table Editor. The QMF interactive editor that lets authorized users make changes to a database without having to write a query.

table name area. The leftmost column of a QBE example table.

tabular data. The data in columns. The content and the form of the data is specified on FORM.MAIN and FORM.COLUMNS.

target table. An empty table in which example elements are used to combine columns, combine rows, or include constant values in a report.

temporary storage. An area where the query, form, procedure, profile, report, chart, and data objects in current use are stored. All but the data object can be displayed.

temporary storage queue. In CICS, a temporary storage area used for transfer of objects between QMF and an application or a system service.

time. Designates a time of day in hours and minutes and possibly seconds (a two- or three-part value).

thread. The DB2 UDB for OS/390 structure that describes an application's connection, traces its progress, provides resource function processing capability, and delimits its accessibility to DB2 UDB for OS/390 resources and services. Most DB2 UDB for OS/390 functions execute under a thread structure.

three-part name. A fully-qualified name of a table or view, consisting of a location name, owner ID, and object name. When supported by the application server (that is, DB2 UDB for OS/390), a three-part name can be used in an SQL statement to retrieve or update the specified table or view at the specified location.

timestamp. A date and a time, and possibly a number of microseconds (a six- or seven-part value).

TP. Transaction Program

TPN. Transaction program name

transaction. The work that occurs between 'Begin Unit of Work' and 'Commit' or 'Rollback'.

transaction program. A program that processes transactions in an SNA network. There are two kinds of transactions programs: application transaction programs and service transaction programs.

transaction program name. The name by which each program participating in an LU 6.2 conversation is known. Normally, the initiator of a connection identifies the name of the program it wants to connect to at the other LU. When used in conjunction with an LU name, it identifies a specific transaction program in the network.

transient data queue. In CICS, a storage area, whose name is defined in the Destination Control Table (DCT), where objects are stored for subsequent internal or external processing.

TSO. Time Sharing Option.

two-phase commit. A protocol used in distributed unit of work to ensure that participating relational database management systems commit or roll back a unit of work consistently.

unit of work. (1) A recoverable sequence of operations within an application process. At any time, an application process is a single unit of work, but the life of an application process may involve many units of work as a result of commit or rollback operations. (2) In DRDA, a sequence of SQL commands that the database manager treats as a single entity. The database manager ensures the consistency of data by verifying that either all the data changes made during a unit of work are performed or none of them are performed.

unlike. Refers to two or more different IBM operating environments. For example, unlike distribution is distribution between DB2 for VM and VSE and DB2 UDB for OS/390. Contrast with *like*.

unnamed column. An empty column added to an example table. Like a target table, it is used to combine columns, combine rows, or include constant values in a report.

USA (United States of America) format. A format that represents date and time values as follows:

- Date: mm/dd/yyyy
- Time: hh:mm xM

value. A data element with an assigned row and column in a table.

variation. A data formatting definition specified on a FORM.DETAIL panel that conditionally can be used to format a report or part of a report.

view. An alternative representation of data from one or more tables. It can include all or some of the columns contained in the table or tables on which it is defined. (2) The entity or entities that define the scope of the data to be searched for a query.

Virtual Storage Extended. An operating system that is an extension of Disk Operating System/ Virtual Storage. A VSE consists of (1) VSE/Advanced Functions support and (2) any IBM-supplied and user-written programs that are required to meet the data processing needs of a user. VSE and the hardware it controls form a complete computing system.

VM. Virtual Machine (IBM operating system). The generic term for the VM/ESA environment.

VSE. Virtual Storage Extended (IBM operating system). The generic term for the VSE/ESA environment.

WAIT. The keyword for an endless-wait-state problem.

window. A rectangular portion of the screen in which all or a portion of a panel is displayed. A window can be smaller than or equal to the size of the screen.

Workstation Database Server. The IBM family of DRDA database products on the UNIX and Intel platforms (such as DB2 Universal Database (UDB), DB2 Common Server, DB2 Parallel Edition, and DataJoiner.)

wrapping. See “column wrapping” and “line wrapping”.

Bibliography

The following lists do not include all the books for a particular library. To get copies of any of these books, or to get more information about a particular library, contact your IBM representative.

For a list of QMF publications, see "The QMF Library" on page v.

APPC Publications

- *Communicating with APPC and CPI-C: A Technical Overview*
- *Networking with APPC: An Overview*

CICS Publications

CICS Transaction Server for OS390

- *CICS/OS390 User's Handbook*
- *CICS/OS390 Application Programmers Reference*
- *CICS/OS390 Application Programming Guide*
- *CICS/OS390 DB2 Guide*
- *CICS/OS390 Resource Definition (Macro)*
- *CICS/OS390 Resource Definition (Online)*
- *CICS/OS390 Problem Determination Guide*
- *CICS/OS390 System Definition Guide*
- *CICS/OS390 Intercommunication Guide*
- *CICS/OS390 Performance Tuning Handbook*

CICS for VSE

- *CICS for VSE/ESA User's Handbook*
- *CICS for VSE/ESA Application Programmer's Reference*
- *CICS for VSE/ESA Application Programming Guide*
- *CICS for VSE/ESA Resource Definition (Macro)*
- *CICS for VSE/ESA Resource Definition (Online)*
- *CICS for VSE/ESA Problem Determination Guide*
- *CICS/OS390 System Definition Guide*
- *CICS for VSE/ESA Intercommunication Guide*
- *CICS for VSE/ESA Performance Tuning Handbook*

Bibliography

COBOL Publications

- *VS COBOL II Application Programming Guide for VSE*
- *COBOL/VSE Language Reference*
- *COBOL/VSE Programming Guide*

DATABASE 2 Publications

DB2 UDB for OS390

- *DB2 UDB for OS390 Installation Guide*
- *DB2 UDB for OS390 Administration Guide*
- *DB2 UDB for OS390 SQL Reference*
- *DB2 UDB for OS390 Command Reference*
- *DB2 UDB for OS390 Application Programming and SQL Guide*
- *DB2 UDB for OS390 Message and Codes*
- *DB2 UDB for OS390 Uitility Guide and Reference*
- *DB2 UDB for OS390 Call Level Interface Guide and Reference*
- *DB2 UDB for OS390 Reference for Remote DRDA Requesters and Servers*

DB2 for VSE & VM

- *DB2 Server for VM Installation Guide*
- *DB2 Server for VSE Installation Guide*
- *DB2 Server for VSE & VM Database Administration*
- *DB2 Server for VM System Administration*
- *DB2 Server for VSE System Administration*
- *DB2 Server for VSE & VM Operation*
- *DB2 Server for VSE & VM SQL Reference*
- *DB2 Server for VSE & VM Application Programming*
- *DB2 Server for VSE & VM Interactive SQL Guide and Reference*
- *DB2 Server for VSE & VM Database Services Utility*
- *DB2 Server for VM Message and Codes*
- *DB2 Server for VSE Message and Codes*
- *DB2 Server for VSE & VM Diagnostic Guide and Reference*
- *DB2 Server for VSE & VM Performance Tuning Handbook*

DB2 for AS/400

- *DB2 for AS/400 SQL Reference*
- *DB2 for AS/400 SQL Programming*

Parallel Edition

- *DB2 Parallel Edition Administration Guide and Reference*

DB2 Universal Database

- *DB2 Universal Database Command Reference*
- *DB2 Universal Database SQL Reference*
- *DB2 Universal Database Message Reference*

DataJoiner

- *DataJoiner Application Programming and SQL Reference Supplement*

DCF Publications

- *DCF and DLF General Information*

DRDA Publications

- *DRDA Every Manager's Guide*
- *DRDA Connectivity Guide*

DXT Publications

- *DXT Guide to Dialogs*
- *Data Extract: Planning and Administration Guide for Dialogs*
- *Data Extract: Users Guide*
- *Learning to Use DXT*

Graphical Data Display Manager (GDDM) Publications

- *GDDM General Information*
- *GDDM Base Programming Reference*
- *GDDM Base Programming Guide*
- *GDDM Guide for Users*
- *GDDM Installation and System Management for VSE*
- *GDDM Messages*

HLASM Publications

- *IBM High-Level Assembler Programmer's Guide for OS/390, VM and VSE*
- *IBM High-Level Assembler Language Reference for OS/390, VM and VSE*

ISPF/PDF Publications

OS/390

- *Interactive System Productivity Facility for OS/390 Installation and Customization*
- *Interactive System Productivity Facility for OS/390 Dialog Management Guide*
- *Interactive System Productivity Facility for OS/390 Dialog Management Services and Examples*

VM

- *ISPF for VM Dialog Management Services and Examples*

OS/390 Publications

Utilities

- *OS/390 Administration: Utilities*
- *OS/390 Extended Architecture Utilities*

JCL

- *OS/390 Extended Architecture JCL Reference*
- *OS/390 Extended Architecture JCL User's Guide*
- *OS/390 JCL Reference*
- *OS/390 JCL Users Guide*

Pageable Link Pack Area (PLPA)

- *OS/390 Extended Architecture Initialization and Tuning*
- *OS/390 SPL: Initialization and Tuning*

VSAM

- *OS/390 VSAM Administration Guide*
- *OS/390 VSAM Catalog Administration Access Method Services*

TSO

- *OS/390 TSO Primer*
- *OS/390 User's Guide*

SMP/E

- *OS/390 System Modification Program Extended Messages and Codes*
- *OS/390 System Modification Program Extended Primer*
- *OS/390 System Modification Program Extended Reference*
- *OS/390 System Modification Program Extended User's Guide*

PL/I Publications

- *PL/I VSE Language Reference*
- *PL/I VSE Programming Guide*

REXX Publications

OS/390 environment

- *IBM Compiler and Library for REXX/370: Users Guide and Reference*
- *TSO Extensions REXX/MVS Reference*

VM environment

- *Procedures Language VM/REXX Reference*
- *Procedures Language VM/REXX User's Guide*

ServiceLink Publications

- *ServiceLink User's Guide*

VM Publications

- *Virtual Machine Planning Guide and Reference*
- *Virtual Machine CMS Command and Macro Reference*

VSE Publications

- *VSE Planning Guide*
- *VSE Guide to System Functions*
- *VSE System Utilities*
- *VSE Guide for Solving Problems*

Bibliography

Index

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

V

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

Query Management Facility™
Developing QMF Applications
Version 7 Release 2

Publication No. SC27-0718-01

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

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

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

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

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

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

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SC27-0718-01

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

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

POSTAGE WILL BE PAID BY ADDRESSEE

IBM CORPORATION
Department HHX/H3
555 Bailey Ave.
San Jose, CA
U.S.A.
95161-9023

Fold and Tape **Please do not staple** Fold and Tape

SC27-0718-01

Cut or Fold
Along Line

Program Number: 5675-DB2
5697-F42

Printed in U.S.A.

SC27-0718-01

Spine information:

QMF Developing QMF Applications Version 7 Release 2