Db2 Query Management Facility
Version 12 Release 1

Developing Db2 QMF applications
Db2 Query Management Facility
Version 12 Release 1

Developing Db2 QMF applications

IBM
Note

Before using this information and the product it supports, be sure to read the general information under “Notices” at the end of this information.

This edition applies to Version 12 Release 1 of IBM Db2 Query Management Facility (QMF) Classic Edition and Enterprise Edition, which are features of IBM Db2 12 for z/OS (5650-DB2) and IBM Db2 11 for z/OS (5615-DB2). It also applies to Version 12 Release 1 of IBM Db2 QMF for z/OS (5697-QM2), which is a stand-alone IBM Db2 for z/OS tool. This information applies to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 1982, 2016.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
Contents

About this information ........................................ v
What you should know before you begin .................. v
Service updates and support information .................. vi
Highlighting conventions ................................... vi
How to read syntax diagrams ................................ vi
How to send your comments ................................ vii

Chapter 1. QMF application development overview ........................................ 1
What is application development in QMF? ............... 1
How can users use your application? ...................... 1
Interacting primarily with the application .............. 1
Starting the application from a QMF session .......... 2
What QMF application development tools are available? .......... 3
QMF procedures ............................................. 3
Application programming interfaces to QMF .......... 3
Conventions for National Language Feature information ........................................ 4

Chapter 2. Procedures as applications .................. 7
Initial procedures ......................................... 7
QMF CONNECT within a procedure ...................... 8
Substitution variables in procedures .................... 9
Specifying values on the RUN command .......... 9
Specifying values on the RUN command prompt panel ........................................ 10
REXX variables in procedures with logic .......... 11
Passing arguments to a procedure with logic .... 11
REXX error-handling statements in procedures with logic ........................................ 12
Branching to error-handling subroutines .......... 12
Messages with the REXX EXIT statement .......... 12
Calling REXX programs from a procedure with logic ........................................ 13
Calling REXX programs without substitution variables ........................................ 13
Calling REXX programs that contain substitution variables ........................................ 14

Chapter 3. The callable interface and QMF applications ........................................ 17
What is the callable interface? .......................... 17
Considerations for using the QMF callable interface ........................................ 17
The interface communications area (DSQCOMM) .......... 18
Return codes ............................................. 20
Commands for using the callable interface .......... 20
Starting QMF from an application .................. 21
Running your callable interface application ........ 21
The callable interface in QMF ......................... 22
Error handling ........................................... 22
Running callable interface programs under CICS .......... 22

Chapter 4. Issuing QMF commands from an ISPF dialog ........................................ 25
Writing a program that uses the command interface: an example .................. 25
Invoking the command interface ..................... 26
The END command ........................................ 26
Variables in the command interface .............. 26
Command interface return codes .................... 27

Chapter 5. ADDRESS QWF and the QMF command environment ........................................ 29

Chapter 6. Writing QMF applications that use ISPF services ........................................ 31
Starting and running QMF from an ISPF application ........ 31
Running queries that contain variables .......... 32
Starting a program that uses ISPF services from within QMF .................. 32
ISPF services in a procedure with logic .......... 32
The EDIT command with ISPF ......................... 33
ISPF and debugging applications .......... 34

Chapter 7. Writing bilingual applications ........................................ 35
Comparing the English and NLF environments .......... 35
Creating objects for use in bilingual applications .... 37
The command language variable .................. 37
Initial procedures in bilingual applications .......... 38
English-only commands ................................ 39
Multilingual environments ................. 39
Creating translatable applications .......... 39

Chapter 8. QMF commands in applications ........................................ 41
Commands designed for applications .............. 41
CONNECT ............................................. 41
END ................................................. 42
EXIT ................................................ 42
GET GLOBAL (extended syntax) ..................... 43
INTERACT ......................................... 43
MESSAGE ........................................... 46
SET GLOBAL (extended syntax) ..................... 48
START ............................................. 50
TRACE .............................................. 57
Commands you can use in a RUN QUERY report minisession .......... 62
Command synonyms .................................. 65

Chapter 9. Exporting and importing objects ........................................ 67
What you can do with an exported UNIX file, TSO data set, or CICS data queue .......... 67
Exporting versus saving data ......................... 68

© Copyright IBM Corp. 1982, 2016
Chapter 10. Debugging your QMF applications ......... 135
Debugging your callable interface applications ......... 135
The L option for tracing .................................. 135
The A option for tracing .................................. 135
Turning the tracing off .................................... 136
Allocating the QMF trace data output ................. 136
The QMF MESSAGE command for tracing ............ 137
Errors on the START or other QMF commands ....... 137

Chapter 11. Programming language specifications for using the callable interface .......... 139
Assembler language interface ......................... 139
Interface communications area mapping for Assembler (DSQCOMMA) ................. 139
Function calls for Assembler language ............... 140
Assembler programming example ..................... 141
DSQCOMM for Assembler ................................ 148
Running your Assembler programs in CICS .......... 149
Running your Assembler programs in TSO .......... 150
C language interface .................................. 152
Interface communications area mapping for C language (DSQCOMMC) ................. 152
Function calls for the C language ..................... 153
C language programming example .................... 155
DSQCOMM for C ......................................... 158
Running your C programs in CICS .................... 159
Running your C programs in TSO ...................... 160
COBOL language interface ............................ 162
Interface communications area mapping for COBOL (DSQCOMMB) .................. 162
Function calls for COBOL .............................. 163
The ISPF LIBDEF service with COBOL ................. 164
COBOL programming example .......................... 165
DSQCOMM for COBOL .................................. 166
Considerations for running your COBOL callable interface program .............. 168
Running your COBOL programs in CICS ............... 169
Running your COBOL programs in TSO ............... 169
Fortran language interface ............................ 171
Interface communications area mapping for Fortran (DSQCOMMF) .................. 172
Function calls for Fortran .............................. 173
Fortran programming example ......................... 174
DSQCOMM for Fortran .................................. 177
Running your Fortran programs ....................... 179
PL/I language interface ............................... 181
Interface communications area mapping for PL/I (DSQCOMML) .................... 181
Function calls for PL/I ................................. 182
PL/I programming example ............................ 184
DSQCOMM for PL/I ...................................... 186
Running your programs under CICS ................. 187
Running your programs under TSO ................. 188
REXX language interface ............................ 190
Interface communications variables for REXX ........................................ 191
Function call for REXX ................................ 192
REXX programming example .......................... 193
Running your REXX programs ......................... 194
A REXX example of using an INTERACT loop .................. 194

Appendix A. Product interface macros 197
Appendix B. QMF global variables 199
Naming convention for QMF global variables .......... 199
Setting and displaying values for global variables .... 199
Global variables for state information not related to the profile ...................... 200
Global variables for profile-related state information ........................................ 205
Global variables associated with CICS ................ 206
Global variables related to a message produced by the most recent command .... 207
Global variables associated with the Table Editor ............................................. 207
Global variables that control various displays ................................................. 210
Global variables that control how commands and procedures are executed ........ 218
Global variables that store results of CONVERT QUERY .................................... 233
Global variables that show RUN QUERY error message information .................. 234
Global variables that store panel input values ........ 234

Notices ............................................. 245
Programming interface information ................ 246
Trademarks ........................................... 246

Glossary of terms and acronyms ................. 249
Index ............................................. 263
About this information

IBM® Db2® Query Management Facility for TSO and CICS® is a tightly integrated, powerful, and reliable tool that offers query and reporting functions that help you access and present data from any of the following relational databases:

- Db2 for z/OS®
- Db2 for Linux, UNIX, and Windows
- DB2® for iSeries
- DB2 Server for VSE and VM

This information is written for application programmers responsible for developing applications that make use of QMF™ functions. These topics help you to:

- Make application programming design decisions
- Choose between different programming techniques
- Understand how to use the QMF command and callable interfaces
- Write bilingual applications

Specific programming examples are provided for Assembler, C, FORTRAN, COBOL, PL/I, and REXX.

What you should know before you begin

You should be familiar with the components that make up your specific environment, as well as some concepts and terms, before you begin application programming for QMF.

Products

To develop applications for QMF, you may need to be familiar with some or all of the following products, depending on your environment and your business needs:

- The z/OS operating system.
- Db2, the database manager for QMF.
- Time Sharing Option(TSO), which is an environment that supports Db2 QMF and its related products.
- Interactive System Productivity Facility (ISPF), a dialog manager for Db2 QMF.
- Customer Information Control System(CICS), a general-purpose data communication and online transaction processing system. CICS provides the interface between Db2 QMF and z/OS.
- The base Graphical Data Display Manager (GDDM) product, which is required to display panels and create charts. You can also use GDDM to provide printing services from QMF.
- Assembler, C, COBOL, FORTRAN, PL/I or REXX, which you might use to create callable interface applications for QMF.

Concepts

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 information assumes you already know how to write queries and
procedures, format reports, and modify the database.

**Related information:**

[The IBM Publications Center](#)

Search for publications that explain the products.

---

### Service updates and support information

To find service updates and support information, including software fix packs, PTFs, Frequently Asked Questions (FAQs), technical notes, troubleshooting information, and downloads, refer to the following Web page:

[IBM Software Support website](#)

---

### Highlighting conventions

This information uses the following highlighting conventions:

- **Boldface** type indicates commands or user interface controls such as names of fields, folders, icons, or menu choices.
- **Monospace** type indicates examples of text that you enter exactly as shown.
- **Italic** indicates the titles of other publications or emphasis on significant terms. It is also used to indicate variables that you should replace with a value.

---

### How to read syntax diagrams

The following rules apply to the syntax diagrams that are used in this information:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line. The following conventions are used:
  - The ```>>>``` symbol indicates the beginning of a syntax diagram.
  - The ```-->``` symbol indicates that the syntax diagram is continued on the next line.
  - The ```-->``` symbol indicates that a syntax diagram is continued from the previous line.
  - The ```<-->``` symbol indicates the end of a syntax diagram.
- Required items appear on the horizontal line (the main path).

```---required_item---```

- Optional items appear below the main path.

```---required_item---optional_item---```

If an optional item appears above the main path, that item has no effect on the execution of the syntax element and is used only for readability.

```---required_item---optional_item---```

- If you can choose from two or more items, they appear vertically, in a stack. If you *must* choose one of the items, one item of the stack appears on the main path.
If choosing one of the items is optional, the entire stack appears below the main path.

If one of the items is the default, it appears above the main path, and the remaining choices are shown below.

- An arrow returning to the left, above the main line, indicates an item that can be repeated.

If the repeat arrow contains a comma, you must separate repeated items with a comma.

A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Keywords, and their minimum abbreviations if applicable, appear in upper case. They must be spelled exactly as shown. Variables appear in all lowercase italic letters (for example, column-name). They represent user-supplied names or values.
- Separate keywords and parameters by at least one space if no intervening punctuation is shown in the diagram.
- Enter punctuation marks, parentheses, arithmetic operators, and other symbols exactly as shown in the diagram.
- Footnotes are shown by a number in parentheses; for example, (1).

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or any other documentation, use either of the following options:

- Use the online reader comment form, which is located at: [http://www.ibm.com/software/data/rcf](http://www.ibm.com/software/data/rcf)
- Send your comments by e-mail to comments@us.ibm.com. Be sure to include the name of the book, the part number of the book, the version of your product,
and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).
Chapter 1. QMF application development overview

Application development refers to the process of creating a procedure or application in QMF.

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 and tables
- Display or print reports or charts
- Enable the user to enter or change data in the database
- Enable the user to make global changes to several objects at once

You can also write applications that provide helpful functions to your users in QMF. For example, write a command that prints QMF reports at a remote location, or a function key that generates a chart of weekly sales results.

What is application development in QMF?

The word application can have many meanings. In QMF, an application is a procedure or program that issues QMF commands and uses QMF services to accomplish a specific business task.

Application development includes:

- Understanding the problem that your procedure or application needs to solve
- Designing the procedure or application
- Writing the code, associated messages, and help panels

How can users use your application?

There are two major types of QMF applications: applications in which the users interact with the application, and applications in which the application can be started from within QMF.

Interacting primarily with the application

If your application is intended for users who are unfamiliar with QMF, you probably want them to interact primarily with your application.

You might not want your users to know that QMF is active. In this case, your application uses QMF services, but runs outside of QMF. Your program issues QMF commands only as needed.

Suppose that you write an application that uses QMF services. This application provides the user with a menu-driven interface. In the following example, your application controls QMF. Your user interacts only with your user interface and is not aware that QMF is active.

If the user selects option 1, the application executes a QMF procedure that runs a query and prints the resulting report.
Starting the application from a QMF session

If your users are familiar with QMF, you might want your 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.

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

You expect your users to run your application from within the QMF environment. The users can use the command line to issue a QMF command synonym called SEND_TO (which you create). Alternatively, you can assign the application to a function key that automatically runs your application.

After generating a report, the user can send the report to Smith by entering the customized QMF command SEND_TO SMITH on the QMF command line.
What QMF application development tools are available?

You can write applications that use QMF procedures and application programming interfaces to QMF.

You can create command synonyms to invoke your procedures and applications. A command synonym is a command that runs a QMF, TSO, or CICS command. You create a command synonym by entering the command and its definition into a command synonyms table. During initialization, QMF loads the command synonyms table that is specified in the QMF profile of the user who started QMF.

QMF procedures

QMF procedures are QMF objects that run within QMF and issue QMF commands. QMF procedures can be run interactively or non-interactively. You can run a procedure non-interactively by starting QMF in batch mode. If you are using QMF for TSO, you can also start QMF as a Db2 for z/OS stored procedure.

QMF procedures can execute any QMF commands available at your site. 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 that helps 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, including system commands.

QMF provides a system initialization procedure and other methods that enable you to run commands and set global variables before the user sees the QMF home panel.

**Related concepts:**

Chapter 2, “Procedures as applications,” on page 7

You can write applications entirely as procedures. If you are using QMF in the CICS environment, you can write **linear procedures**, which are procedures that include only QMF commands or synonyms that issue QMF commands. If you are using QMF in the TSO environment, you can write **procedures with logic** in addition to linear procedures. Procedures with logic can include REXX statements and functions in addition to QMF commands and command synonyms.

**Related reference:**

“Global variables that control how commands and procedures are executed” on page 218

DSQEC global variables control how commands and procedures are executed. All of these global variables can be modified by the SET GLOBAL command.

Application programming interfaces to QMF

There are two application programming interfaces to QMF: Callable interface and command interface.

**Callable interface**

You can use the QMF callable interface to create an application that is invoked outside of QMF. That application then starts a QMF session and sends commands to QMF for execution.
The callable interface is a programming interface that provides access to QMF services. The callable interface is available for the programming languages and environments shown in the following table.

Table 1. Callable interface support

<table>
<thead>
<tr>
<th>Programming Language</th>
<th>CICS</th>
<th>TSO</th>
<th>Native z/OS batch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembler</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>C</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>COBOL</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Fortran</td>
<td></td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>PL/1</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>REXX</td>
<td></td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

QMF supports all versions of these programming languages that are supported by Db2 for z/OS.

Command interface (TSO only)

You can create applications that submit commands to QMF from an ISPF dialog by using the QMF command interface. QMF communicates with the ISPF dialog through the ISPF variable pool through this command interface. QMF must be started before the ISPF application is started.

The command interface is available only where ISPF is available; it is not available in CICS. You can write a command interface application in any programming language that is supported by ISPF.

Related concepts:

- Chapter 3, “The callable interface and QMF applications,” on page 17
  Programming languages can use the QMF callable interface to run QMF commands.
- Chapter 4, “Issuing QMF commands from an ISPF dialog,” on page 25
  You can issue QMF commands from an ISPF dialog that is running under QMF by using the QMF command interface.

Related information:

- Programming languages and methods for developing application programs
  See the information about programming languages that are supported by Db2 for z/OS.

Conventions for National Language Feature information

Db2 QMF is available in several different languages, each of which is provided by a National Language Feature (NLF).

Use NLFs to allow users to enter QMF commands, view help, and complete QMF tasks in languages other than English. NLFs are installed as separate features of Db2 QMF.

All tasks described in this information can be completed for the base QMF product (English language) and for any NLF. The procedures for both the base and NLF sessions are the same; however, any special considerations for NLF users are identified.
Some names of programs or data sets shown in this information have the variable \( n \) in them, indicating that this character of the name can vary. Replace the variable \( n \) with the one-character national language identifier (NLID) in the following table that matches the language feature that you are using. The table also shows the names by which QMF recognizes each language.

<table>
<thead>
<tr>
<th>National Language Feature</th>
<th>Identifier (NLID)</th>
<th>Name that QMF uses for this NLF</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>E</td>
<td>ENGLISH</td>
</tr>
<tr>
<td>Uppercase English</td>
<td>U</td>
<td>UPPERCASE</td>
</tr>
<tr>
<td>Canadian French</td>
<td>C</td>
<td>FRANCAIS CANADIEN</td>
</tr>
<tr>
<td>Danish</td>
<td>Q</td>
<td>DANSK</td>
</tr>
<tr>
<td>French</td>
<td>F</td>
<td>FRANCAIS</td>
</tr>
<tr>
<td>German</td>
<td>D</td>
<td>DEUTSCH</td>
</tr>
<tr>
<td>Italian</td>
<td>I</td>
<td>ITALIANO</td>
</tr>
<tr>
<td>Japanese Kanji</td>
<td>K</td>
<td>NIHONGO</td>
</tr>
<tr>
<td>Korean Hangeul</td>
<td>H</td>
<td>HANGEUL</td>
</tr>
<tr>
<td>Brazil Portuguese</td>
<td>P</td>
<td>PORTUGUES</td>
</tr>
<tr>
<td>Spanish</td>
<td>S</td>
<td>ESPANOL</td>
</tr>
<tr>
<td>Swedish</td>
<td>V</td>
<td>SVENSKA</td>
</tr>
<tr>
<td>Swiss French</td>
<td>Y</td>
<td>FRANCAIS (SUISSE)</td>
</tr>
<tr>
<td>Swiss German</td>
<td>Z</td>
<td>DEUTSCH (SCHWEIZ)</td>
</tr>
</tbody>
</table>

The Uppercase English feature uses the English language, but converts all text to uppercase characters. The uppercase characters allow users who work with Katakana display devices to use the product and get English online help and messages.
Chapter 2. Procedures as applications

You can write applications entirely as procedures. If you are using QMF in the CICS environment, you can write linear procedures, which are procedures that include only QMF commands or synonyms that issue QMF commands. If you are using QMF in the TSO environment, you can write procedures with logic in addition to linear procedures. Procedures with logic can include REXX statements and functions in addition to QMF commands and command synonyms.

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

Related concepts:
“ISPF services in a procedure with logic” on page 32

You must transfer from the QMF program dialog to an ISPF command dialog to run ISPF commands from a QMF procedure with logic that is running under ISPF.

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 and understand how initial procedures behave in specific situations.

You can use the DSQSRUN parameter:
- With the DSQQMFn command when QMF is started interactively (where n is a one-character national language identifier that matches the language feature that you are using).
- With the QMF START command when QMF is started through the callable interface.

In TSO, ISPF, and nativez/OS batch, applications can also set program parameters by using a REXX program. The program is specified by the DSQSCMD parameter of the QMF START command. Because QMF for CICS does not support REXX, you must specify all program parameters on the START command by using DSQSMODE=I. This value for DSQSMODE specifies interactive operation, in CICS. The default mode from the callable interface is B (for batch operation).

Considerations for writing initial procedures

Consider the following points when you write and use an initial procedure:
- By default, QMF reruns the initial procedure whenever the user issues the END command in an interactive session of QMF started by DSQQMFn. (The variable n is a one-character national language identifier that matches the language feature that you are using). The DSQEC_RERUN_IPROC global variable specifies whether the initial procedure is rerun. The default value of this variable is 1 to rerun the procedure; a setting of 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.
- When you write initial procedures to use in an interactive QMF session, avoid using the home panel as the current panel at the end of the procedure. QMF
does not interactively display a panel at the end of the procedure in this case. 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, consider one of these options:

- 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).

The number of ampersands (&) you must use before the name of the substitution variables in initial procedures can vary depending on your environment. For example, you can specify DSQSRUN as:

\[ \text{DSQSRUN=INITPROC (VAR1 = value)} \]

The number of ampersands that you need to specify with VAR1 depends on whether QMF is running under CICS, TSO, or native z/OS batch. The number varies if ISPF is present and if the program that starts 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).

When you use the QMF CONNECT command in initial procedures, you must disconnect from the remote location. In other words, code the application to reconnect to your original location before you can code an END command to invoke your initial procedure again. If you are still connected to the remote location, you receive an error. Disconnecting is also required with QMF CONNECT issued from the command line during an interactive session that is set up by an initial procedure.

**Related reference:**

- “Conventions for National Language Feature information” on page 4
- Db2 QMF is available in several different languages, each of which is provided by a National Language Feature (NLF).
- “START command keywords” on page 51

You can specify keywords on the START command.

**QMF CONNECT within a procedure**

To connect to another user ID or to a remote Db2 database to use remote-unit-of-work support, issue the QMF CONNECT command. 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 circumstances:

- 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).
• Commands that affect the database (SQL statements, QMF queries, or EDIT TABLE updates) run at the current location.

• If the procedure contains system-specific commands (CICS 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 is running, your procedure cannot run successfully.

• Any data sets or data queues used in a system-specific command must exist on the system where QMF is running (the local system).

• If your site uses TSO and takes advantage of RACF® support for mixed-case passwords, ensure that the CASE option of your QMF profile is set to MIXED. Otherwise, QMF converts all input to uppercase, causing the CONNECT command to fail. When CASE=MIXED, ensure that you tell QMF application users to enter all input in uppercase, because QMF recognizes commands only in uppercase.

• If the procedure is passed as a parameter on a CALL statement that starts QMF for TSO by using the stored procedure interface, the procedure cannot access remote databases. Any commands in the procedure that attempt to access a remote database must be removed or commented out before the procedure is run with this interface.

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. A substitution variable is always preceded by an ampersand (&). You can assign a value to a substitution variable in these ways:

• Setting global variables with the SET GLOBAL command
• Specifying values on the RUN command
• Specifying values on the RUN command prompt panel.

Related reference:

“SET GLOBAL (extended syntax)” on page 48

To create your own global variables and use them in QMF commands as substitution variables, issue the SET GLOBAL command. You can also use the SET GLOBAL command to set values for QMF predefined global variables, which start with "DSQ."

Specifying values on the RUN command

You can assign a value to a substitution command by using the RUN command.

If the procedure is a linear procedure, assign the variable value on the RUN PROC command as follows:

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

If the procedure is a procedure with logic, assign the variable value on the RUN PROC command as follows:

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

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

In this example:
• The variable value VACATION is surrounded by single quotation marks 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 that contains 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**

When you run a query or procedure with a substitution variable, you can assign a value on the RUN command or through a global variable. However, if the variable in the query or procedure does not have a value, QMF presents a RUN command prompt panel. You can then 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 that contains the variables. To prompt the user one time, you can run a separate procedure that prompts for variables.

For example, you want to be prompted once for the substitution variables LASTNAME and DEPT_NUM. These variables occur on two different lines in your procedure with logic:

```sql
/* This procedure runs two queries, displaying the report after each */
/* query has run. */
"RUN QUERY REG_QUERY (&&LASTNAME=&LASTNAME";
"INTERACT"
"RUN QUERY REG2_QUERY (&&DEPT_NUM=&DEPT_NUM";
```

**Figure 3. Prompting for variable values in a procedure with logic**

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

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

**Important:** All procedures with logic must begin with at least one comment line.

In this command, PROMPT_ME is a procedure with logic like the following, which contains a comment line and no instructions:

```sql
/* This procedure is a dummy procedure that provides prompting */
```
The complete procedure includes the RUN PROC command for the PROMPT_ME procedure that prompts for variables:

```sql
/* This procedure runs two queries, displaying the report after each */
/* query 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 the SET GLOBAL command to prompt for all the values in your procedure at the same time, as in the following example:

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

**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 use REXX variables in a procedure with logic in these ways:
- 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

**Passing arguments to a procedure with logic**

For procedures with logic, QMF provides an ARG option on the RUN PROC command. Use this option to 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 example shown here:

```
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
```

*Figure 5. Passing variable values using the ARG option of the RUN PROC command*

The RUN command for this procedure is as follows:

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

In this command, *query_name* and *form_name* are REXX variable names that describe the parameters that are passed to the procedure with logic. Use these variables, which contain the object names for a query and a form, to reference the parameters that were passed to the procedure with logic.
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. You can also 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 instruction
  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 example:

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

Messages with the REXX EXIT statement

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

This table shows which message you see, based on the conditions.

<table>
<thead>
<tr>
<th>Return code from the last QMF command</th>
<th>Procedure return code</th>
<th>Examples of messages at the completion of the procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>OK, your procedure was run</td>
</tr>
<tr>
<td>0</td>
<td>Nonzero</td>
<td>The return code from your procedure was 8</td>
</tr>
<tr>
<td>Nonzero</td>
<td>0</td>
<td>The error message provided by QMF</td>
</tr>
<tr>
<td>Nonzero</td>
<td>Nonzero</td>
<td>The error message provided by QMF</td>
</tr>
</tbody>
</table>

A QMF 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 instruction. For example:

```
:  "MESSAGE (TEXT='"dsq_message_text"')"
exit dsq_return_code
```

The variables dsq_message_text and dsq_return_code are REXX variables provided by QMF. You can use the MESSAGE command and the dsq_message_text variable to store and display a message later, as shown in this example:

```
/* 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;
```

Related reference:

["Interface communications variables for REXX" on page 191](#)

The interface communications variables consist of several REXX variables. They are set after the completion of every call and must not be altered by the calling program.

["MESSAGE" on page 46](#)

When you create applications, you often want to send specific messages to your users about the information displayed for them or their next action. You can write your own messages and display them on QMF panels through the MESSAGE command. In ISPF, you can also choose to have QMF display the message help for an ISPF error message.

---

**Calling REXX programs from a procedure with logic**

You use different methods to invoke your REXX program when you use substitution variables and when you do not.

**Calling REXX programs without substitution variables**

If your REXX program does not contain an embedded RUN command that includes substitution variables, invoke your program by using the ADDRESS or CALL instruction. You can also call the program as a function.

Use one of these commands to invoke your program:

- The ADDRESS instruction
  
  This instruction establishes a command environment. For example, if your program is named PANDA, and you want to call it from within the TSO environment, the command is:
  
  ```
  ADDRESS TSO "PANDA"
  ```

- The CALL instruction
  
  This instruction invokes a program. For example, for the program named PANDA, the command is:
  
  ```
  CALL PANDA
  ```
• A function

You also can call the program PANDA as a function:

```
answer = PANDA()
```

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

**Related concepts:**

[Chapter 5, “ADDRESS QRW and the QMF command environment,” on page 29](#)

When QMF is started in TSO, ISPF, or native z/OS, QMF creates a REXX command environment called QRW. When you are executing a REXX program, you can set the default command environment to QRW by issuing the REXX ADDRESS command ADDRESS QRW. With ADDRESS QRW, QMF remains the default command environment until you issue another ADDRESS command.

### 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 the TSO *program_name* command.

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.

For example, consider the following 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:

```
"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 your 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 TSO command to call the application.
  
  When you call the application, QMF does not process any substitution variables it encounters. In the preceding RUN QUERY 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. You can also set the DSEQC_USERGXLV_SAV global variable to save global variable values from one session to another.
Related reference:

“Global variables that control how commands and procedures are executed” on page 218

DSQEC global variables control how commands and procedures are executed. All of these global variables can be modified by the SET GLOBAL command.
Chapter 3. The callable interface and QMF applications

Programming languages can use the QMF callable interface to run QMF commands.

Related reference:
Chapter 11, “Programming language specifications for using the callable interface,” on page 139

The QMF application programming interface is available for several programming languages.

What is 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 start communication between the program and QMF. This communication is made by issuing a call to a QMF 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 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 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 routine.

Considerations for using the QMF callable interface

The flow of control between your application and QMF using the callable interface is subject to certain rules.

Keep in mind the following points as you write application programs to be used with 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 presiding NLF language. Your commands must also be
written in uppercase or converted to uppercase by QMF. Commands are converted to uppercase by QMF if the CASE option in your QMF profile is set to UPPER. You set the presiding language when you start QMF, by providing a value for the DSQALANG parameter on the START command. This value is recorded in the DSQEC_NLFCMD_LANG global variable.

- The maximum length of the passed commands is 2,048 bytes for REXX programs and 32,768 bytes for all other languages.

This diagram shows how the application passes commands through the callable interface to QMF.

```
<table>
<thead>
<tr>
<th>Application session</th>
<th>Callable interface</th>
<th>QMF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Establish communication</td>
<td>START</td>
<td>QMF session starts</td>
</tr>
<tr>
<td>with QMF</td>
<td>---------------------------------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DSQL_RETURN_CODE=0</td>
<td></td>
</tr>
<tr>
<td>2. Issue a QMF command</td>
<td>QMF Command</td>
<td>QMF runs the command</td>
</tr>
<tr>
<td></td>
<td>---------------------------------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DSQL_RETURN_CODE=0</td>
<td></td>
</tr>
<tr>
<td>3. Get status information</td>
<td>GET GLOBAL</td>
<td>QMF retrieves status information</td>
</tr>
<tr>
<td></td>
<td>---------------------------------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DSQL_RETURN_CODE=0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Global variables are stored in Application storage</td>
<td></td>
</tr>
<tr>
<td>4. Terminate communication</td>
<td>EXIT</td>
<td>QMF session</td>
</tr>
<tr>
<td>with QMF</td>
<td>---------------------------------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DSQL_RETURN_CODE=0</td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 6. How an application uses the QMF callable interface to communicate with QMF

Related reference:

“START command keywords” on page 51

You can specify keywords on the START command.

### The interface communications area (DSQCOMM)

QMF provides an interface communications area for each supported programming language. This area contains definitions of return and reason codes and definitions of the function calls to QMF.

The interface communications area defines storage for the interface communications variables. The variables stored in this area are accessible to both QMF and the callable interface application. However, allow only QMF to alter the values. Ensure that the application program treats 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
• From a single CICS transaction

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

These 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 exception 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. With this setting, 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 programming language has a unique 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 variables within the interface communications area contain the information shown in this table:

**Table 4. DSQCOMM fields that must not be altered**

<table>
<thead>
<tr>
<th>Information provided by the variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return code</td>
<td>Indicates the status of QMF processing after QMF processes a command</td>
</tr>
<tr>
<td>Instance identifier</td>
<td>Identifies the instance of QMF that was started by the START command</td>
</tr>
<tr>
<td>Completion message ID</td>
<td>Contains the message ID of the message that QMF displays. This field is set at the completion of every QMF command. It contains the message QMF displays at the end of a command.</td>
</tr>
<tr>
<td>Query message ID</td>
<td>Contains the message ID of the message that is displayed on the query panel upon completion of a RUN QUERY command. This field is set when an error occurs while a query is running. It contains the message that QMF displays within the query object at the end of a command.</td>
</tr>
<tr>
<td>START command parameter in error</td>
<td>Contains the name of the parameter in error when the START command fails because of a parameter error</td>
</tr>
<tr>
<td>Cancel indicator</td>
<td>Indicates whether the user canceled processing while QMF was running the command</td>
</tr>
<tr>
<td>Completion message</td>
<td>Contains the completion message that QMF displays</td>
</tr>
</tbody>
</table>
Table 4. **DSQCOMM fields that must not be altered** (continued)

<table>
<thead>
<tr>
<th>Information provided by the variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query message</td>
<td>Contains the query message text of the message that is displayed on the query panel upon completion of a RUN QUERY command. For example, if you run a query object with an error, QMF displays a message that describes the error that prevented the query from running. The query message field then contains this error message text.</td>
</tr>
</tbody>
</table>

**Related reference:**

Chapter 11, “Programming language specifications for using the callable interface,” on page 139

The QMF application programming interface is available for several programming languages.

**Return codes**

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

The values of return codes can be different on other systems. If you want your applications to be portable across systems, the applications must reference the values of these codes by the variable names. The names of the return-code variables within the interface communications area are documented with the programming language specification.

This table shows the possible return codes for callable interface conditions.

```
Table 5. Callable interface return codes

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful execution</td>
</tr>
<tr>
<td>4</td>
<td>QMF session marked for termination by an EXIT or END command</td>
</tr>
<tr>
<td>8</td>
<td>Execution failed, but the error did not mark the session for termination</td>
</tr>
<tr>
<td>16</td>
<td>Severe error: session marked for termination</td>
</tr>
</tbody>
</table>
```

**Related reference:**

Chapter 11, “Programming language specifications for using the callable interface,” on page 139

The QMF application programming interface is available for several programming languages.

**Commands for using the callable interface**

You can use the callable interface to issue any QMF command that you would use in a procedure. However, some commands have special syntax for the callable interface: START, GET GLOBAL, SET GLOBAL, and TRACE.

The START and TRACE commands work only in the callable interface. To use the GET GLOBAL and SET GLOBAL commands in a callable interface application written in a language other than REXX, use the extended syntax for these commands.
For examples of the START and SET GLOBAL commands in a programming language, see the specification for that language.

**Related concepts:**
- Chapter 8, “QMF commands in applications,” on page 41
  Certain commands are designed to be used in applications, and you can create your own command synonyms.

**Related reference:**
- “GET GLOBAL (extended syntax)” on page 43
  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 for the GET GLOBAL command.
- “SET GLOBAL (extended syntax)” on page 48
  To create your own global variables and use them in QMF commands as substitution variables, issue the SET GLOBAL command. You can also use the SET GLOBAL command to set values for QMF predefined global variables, which start with "DSQ."

**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 in your application. You can have only one QMF session at a time.

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

DSQ50720 QMF already active; secondary session not permitted.

If your START command results in a non-severe error (a return code of 4 or 8), QMF starts, and a session is established. In this case, you must issue a QMF EXIT command to stop QMF. Inspect the contents of the interface communications area or the QMF trace data output for the cause of the error.

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

**Related reference:**
- “START” on page 50
  When you start QMF through the callable interface, you need to use the START command.

**Running your callable interface application**

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

For information about setting up your environment and compiling and running your callable interface application, see the coding sample in the appropriate language specification.

**Related reference:**
- Chapter 3. The callable interface and QMF applications
The QMF application programming interface is available for several programming languages.

**The callable interface in QMF**

If you need to modify a QMF object from a user program, you can use the callable interface from within QMF. For example, you can export or import objects through the callable interface during an interactive QMF session. You can use the callable interface from within QMF by using the TSO command to call the application. You can run any valid QMF command from the application.

**Restriction:** You cannot use the callable interface from within QMF while in the CICS environment.

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**

At the completion of every QMF command, the DSQCOMM communications area contains message text in the dsq_message_text variable and a return code in the dsq_return_code variable.

The return code is assigned one of the following values:

- **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

The variables and fields in each DSQCOMM area are documented with the programming language specifications.

**Related reference:**

Chapter 11, “Programming language specifications for using the callable interface,” on page 139

The QMF application programming interface is available for several programming languages.

**Running callable interface programs under CICS**

To run programs that use the QMF callable interface, install them on CICS with your normal method of installing CICS programs.

In addition to the normal CICS requirements, the following considerations apply to all QMF callable interface programs that run 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 large conversational program.

QMF is an assembler-language program that contains CICS commands. It can be link-edited with other assembler-language programs or with programs supported by the callable application programming interface. When you call QMF using a high-level language, the program must be link-edited first. In addition, the resource definition online (RDO) program definition must specify that high-level language. Each high-level program has specific CICS considerations and restrictions.

In CICS, if you want to override any of the default QMF program parameters, you must specify the override values as parameters on the START command. For example, the default mode of operation from the callable interface is batch mode. To run an interactive QMF session you must issue the START command with DSQSMODE=I.

CICS region considerations
The user program that contains 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
The CICS transaction that invokes your program must be described to Db2.

Related concepts:

“Application programming interfaces to QMF” on page 3
There are two application programming interfaces to QMF: Callable interface and command interface.
Chapter 4. Issuing QMF commands from an ISPF dialog

You can issue QMF commands from an ISPF dialog that is running under QMF by using the QMF command interface.

Restriction: 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.

Using the QMF command interface, QMF communicates with the dialog through the ISPF variable pool, as shown in this diagram.

![Diagram](image)

Figure 7. QMF command interface application interacting with QMF

To use the command interface effectively, you also need to understand ISPF services and variable pools.

Writing a program that uses the command interface: an example

In this example you write a program to display an ISPF panel that prompts the user to specify a query name, runs the specified query, and displays a report.

Procedure

1. Write your command interface REXX program:
   a. Display the ISPF panel with the DISPLAY services: In this example, the panel name is QRYNAME
      ADDRESS ISPEXEC "DISPLAY PANEL(QRYNAME)"
   b. Run a QMF query based on user input from the previous DISPLAY service. The ISPF variable QNAME contains the name of the QMF query
      ADDRESS ISPEXEC "SELECT PGM(DSQCCI) PARM(RUN QUERY" QNAME ")"
   c. Display the result of the query:
      ADDRESS ISPEXEC "SELECT PGM(DSQCCI) PARM(INTERACT)"
2. Call your program with the TSO command from the QMF command line. For example, if your program is named GETINFO, use this command:

   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.

To start the command interface, first start ISPF. Then start QMF using the ISPF SELECT service to call the QMF command interface (DSQCCI). You can pass QMF commands by using the PARM option of the ISPF SELECT PGM command.

After you start the command interface, you can pass QMF commands you want to run by using the PARM parameter of the ISPF SELECT PGM 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 convert the commands from lowercase to uppercase. If you specify your QMF commands in lowercase, QMF does not recognize them.

If you want to use QMF command prompts from your ISPF application, you can issue the INTERACT command. Follow the INTERACT command with the QMF command for which you need a prompt to open and end the statement with a question mark. For example, to display the RUN QUERY prompt panel, issue the following command:

   SELECT PGM(DSQCCI) PARM(INTERACT RUN QUERY ?)

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

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

**The END command**

The END command terminates the DSQCCI program and returns control to the calling application when issued while the command interface is running. The QMF session remains active.

QMF sets the global variable DSQCSESC to mark the session for termination if it encounters an EXIT command or a severe error during a command interface invocation. When the program that called DSQCCI ends and returns control to QMF, the QMF session terminates.

**Variables in the command interface**

The STATE command provides the current value for each variable provided by QMF. The STATE command can be used only in the command interface. You can place the QMF variables in the ISPF variable pool through the VPUT command.

**Related reference:**

26  Developing Db2 QMF applications
Appendix B, “QMF global variables,” on page 199

QMF provides many global variables that help you control aspects of your QMF session, QMF commands, and panel display. The global variables also help you control behavior of QMF functions in procedures and applications.

Command interface return codes

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

Return codes are displayed 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. The example shows how to run a query and test for an error with the REXX variable RC:

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

You can place code for handling errors in program modules and 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. Along with the code, the command interface returns the values of the QMF command message variables in the ISPF shared pool of the application.

Table 6. Return codes 0 through 16

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful execution</td>
</tr>
<tr>
<td>4</td>
<td>QMF session marked for termination by an EXIT or END command</td>
</tr>
<tr>
<td>8</td>
<td>Execution failed, but the error did not cause the session to be marked for termination</td>
</tr>
<tr>
<td>16</td>
<td>Severe error: session marked for termination</td>
</tr>
</tbody>
</table>

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 application.

Return codes of 20 or higher

These codes usually reflect some failure in the command interface (DSQCCI). The failure prevents the interface from copying a variable into the application shared
pool. As a result, the QMF variables might be invalid or they might not be set. The same result might be true of the STATE variables if your program uses the STATE command. (A variable is set if it is copied into the application shared pool.)

These return codes usually indicate more serious errors than return codes in the 0 through 16 range. Some return codes might require the services of IBM Software Support.

In this table, shared variables refers to the QMF variables (and the STATE variables, if the current command is the STATE command). Some codes indicate that the command was run but the shared variables were not set. QMF ran the STATE command properly, but the command interface failed to set the updated shared QMF and STATE variables. The reason for the failure is described in the explanation of the error code.

Table 7. Return codes of 20 or more

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>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 were not set.</td>
</tr>
<tr>
<td>24</td>
<td>An error occurred in an ISPF VCOPY command. The command passed to the command interface was run. The shared variables were not set.</td>
</tr>
<tr>
<td>32</td>
<td>An error occurred in an ISPF VREPLACE command. The command passed to the command interface was run. The shared variables were not set.</td>
</tr>
<tr>
<td>36</td>
<td>An error occurred in an ISPF VPUT command. The command passed to the command interface was run. The shared variables were not set.</td>
</tr>
<tr>
<td>40</td>
<td>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 were not set.</td>
</tr>
<tr>
<td>44</td>
<td>An error occurred in an ISPF VPUT command. This code applies only to the execution of the STATE command. The QMF variables were set, but the STATE variables were not.</td>
</tr>
<tr>
<td>60</td>
<td>An invalid call was made to the command interface. The command passed to the command interface was not run. The shared variables were not set.</td>
</tr>
<tr>
<td>64</td>
<td>This error is issued when DSQCCI is run and ISPF is not active. For example, the user called DSQCCI without using an ISPF SELECT PGM command.</td>
</tr>
<tr>
<td>100</td>
<td>This error occurs when an application tries to issue a QMF command when QMF is not active. Start QMF before you begin your application. The command passed to the command interface was not run. The shared variables were not set.</td>
</tr>
<tr>
<td>104</td>
<td>The anchor was not located. The command passed to the command interface was not run. The shared variables were set but are not valid.</td>
</tr>
</tbody>
</table>
Chapter 5. ADDRESS QRW and the QMF command environment

When QMF is started in TSO, ISPF, or native z/OS, QMF creates a REXX command environment called QRW. When you are executing a REXX program, you can set the default command environment to QRW by issuing the REXX ADDRESS command ADDRESS QRW. With ADDRESS QRW, QMF remains the default command environment until you issue another ADDRESS command.

Restriction: ADDRESS QRW is not supported in the CICS environment.

You can also direct a single command to be executed by the QRW environment by issuing the REXX ADDRESS command followed by the QMF command:

ADDRESS QRW qmf_command

In this situation, QMF is the command environment only for the command that follows the ADDRESS QRW statement.

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

© Copyright IBM Corp. 1982, 2016
Chapter 6. Writing QMF applications that use ISPF services

You can bypass the QMF panels by writing applications that have their own user interfaces. You can use either the callable interface or the command interface to write applications that use ISPF.

Restriction: ISPF does not run in the CICS environment, so ISPF services are not available under CICS.

Related concepts:
- Chapter 3, “The callable interface and QMF applications,” on page 17
  Programming languages can use the QMF callable interface to run QMF commands.
- Chapter 4, “Issuing QMF commands from an ISPF dialog,” on page 25
  You can issue QMF commands from an ISPF dialog that is running under QMF by using the QMF command interface.

Starting and running QMF from an ISPF application

When you start and run QMF using the callable interface from an ISPF application, you must follow certain rules.

When you write a callable interface application that uses ISPF, you need to ensure that you follow these requirements:

- The callable interface application must match the language of your ISPF dialog.
  For example, if your ISPF dialog is a PL/I program, you must use the QMF callable interface for PL/I to write your application.
- You must use the correct national language identifier.
  You must start your ISPF application with an ID of DSQn, 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. The ID also ensures that the ISPF environment remains intact even after QMF is started. For example, this statement begins a PL/I program MYPROG that starts QMF using the callable interface START command:
    ```sql
    SELECT PGM(MYPROG) NEWAPPL(DSQn)
    ```
- Use the GET GLOBAL or SET GLOBAL commands in your application instead of the STATE command to set and retrieve variable values.
  The STATE command works only for variables that contain state information. The GET GLOBAL and SET GLOBAL commands work for all the QMF global variables. However, you cannot set global variables that are read-only.

Related reference:
- “Conventions for National Language Feature information” on page 4
  Db2 QMF is available in several different languages, each of which is provided by a National Language Feature (NLF).
- “START command keywords” on page 51
  You can specify keywords on the START command.
- “Global variables for state information not related to the profile” on page 200
  DSQAO global variables contain status information or settings of parameters or flags. None of these global variables can be modified by the SET GLOBAL command.

© Copyright IBM Corp. 1982, 2016
Running queries that contain variables

Applications that use ISPF services 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 are displayed as ISPF dialog variables. After giving these variables the appropriate values, your program starts certain ISPF file-tailoring services. The result is a sequential file that contains the query.
  The program can then import the query into QMF temporary storage and have QMF run it. The required 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 that uses the ISPF dialog service.

- 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 use REXX to write PDF macros.

- Create a query that uses an ISPF dialog.
  Your program can use ISPF display services to display a screen and create a file based on input from the user. This file, which then contains an SQL query, is then imported into QMF and run.

Starting a program that uses ISPF services from within QMF

If you want to start your ISPF program from within QMF, you must call the program from a linear procedure or a procedure with logic.

To call your program, use the ISPF SELECT PGM service by including the following command in your procedure:

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

Use the CMD keyword to indicate to ISPF that you are running your program as an ISPF dialog function. The syntax for this command is:

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

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

In this statement, `cmdname` is the name of your callable interface command.

ISPF services in a procedure with logic

You must transfer from the QMF program dialog to an ISPF command dialog to run ISPF commands from a QMF procedure with logic that is running under ISPF.
To set the correct ISPF environment and run a program that contains your ISPF commands, use the following ISPF SELECT CMD statement with the CMD keyword:

ADDRESS ISPEXEC "SELECT CMD(userprogram)"

In this statement, *userprogram* is the program that contains your ISPF commands.

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

ADDRESS ISPEXEC "SELECT CMD(DIALOG)"

You can also use a QMF TSO command to run your program that contains ISPF commands (for example, TSO DIALOG). In this case, QMF issues the ISPF SELECT CMD statement for you.

When running QMF under ISPF, a procedure with logic that starts a program that requires ISPF services must use the ISPF SELECT CMD environment. For example, suppose that you are running QMF under ISPF and your procedure with logic issues the Db2 command DSN. Because the DSN command uses ISPF services, use one of the following commands to issue the DSN command:

ADDRESS ISPEXEC "SELECT CMD(DSN)"

or

ADDRESS ISPEXEC "SELECT CMD(DSNEXEC)"

In the second statement, DSNEXEC is a program that contains the ADDRESS TSO DSN statement.

---

**The EDIT command with ISPF**

When you run your QMF application under ISPF, you can edit your QMF SQL query or procedure by using the EDIT QUERY command or the EDIT PROC command.

If you issue the QMF EDIT command from a PROC panel or QUERY panel, you do not need to specify the PROC or QUERY object types. EDIT assumes these values when you invoke it from the respective panels. By default, the QMF EDIT command places your procedure or query in a PDF editor session. QMF starts the PDF editor by 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 the default editor, use the EDIT QUERY and EDIT PROC commands as follows:

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

In these statements, *name* can be either of the following values:

- An editor available to you
- The name of a REXX program that specifies an application ID other than the default. The default application ID is DSQ__n, where *n* is the national language identifier for the NLF you are using)

Use an application ID different from the QMF default application ID if you want to have function keys different from the keys that 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 DSQnEDRT PROFILE data set member (where n is the appropriate NLF character). The NLF must be installed before you issue the command.

**Related reference:**
- “Conventions for National Language Feature information” on page 4

Db2 QMF is available in several different languages, each of which is provided by a National Language Feature (NLF).

---

### ISPF and debugging applications

The QMF trace facility can help you trace QMF activity at various levels of detail. To help you more effectively debug applications that use ISPF you can also use the ISPF log service and PDF dialog test service. These services complement the QMF trace facility.

#### The ISPF log service

You can 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)
```

In this statement, *message-id* is the identification code for the message that is to be retrieved from the message library and written to the log.

#### The PDF dialog test service

If your site has PDF, you can use the PDF dialog test service to log ISPF application service calls to the ISPF log file. Additionally, you can use the log option of the PDF dialog test service 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. For example, you can debug interactively. You can run all or portions of your application, examine the results, change your application, and rerun it. You can also use dialog test services to accomplish these goals:

- 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

To create, change, and delete trace definitions, use the trace (TRACES) option of the dialog test service. Also use this option to 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 the dialog test services to browse the ISPF log, or examine the printed output when you exit ISPF.

**Related concepts:**
- Chapter 10, “Debugging your QMF applications,” on page 135

In addition to error handling, QMF provides debugging facilities for your callable interface applications.
Chapter 7. Writing bilingual applications

Many business applications need to run in several different national languages. You can write one English application and run it in any national language that QMF supports.

Each national language that QMF supports is called a National Language Feature (NLF). An NLF provides a user with a QMF session that is tailored to a specific national language.

QMF provides bilingual support for commands and forms. You can run English QMF commands and display English forms in any NLF.

Related reference:
“Conventions for National Language Feature information” on page 4

Db2 QMF is available in several different languages, each of which is provided by a National Language Feature (NLF).

Comparing the English and NLF environments

Although aspects of the QMF session environment are the same no matter which National Language Features (NLF) is in use, there are some differences. When no NLFs are installed, the only available QMF session environment is the English-language environment.

Environmental similarities

These similarities are the most important ones in the QMF session environments no matter which NLF is in use:

Capabilities
In general, you can do anything in an NLF session that you can do in an English-language session. You can create objects in temporary storage and save them in the database, format and print reports, and issue SQL commands. You can also run Prompted Query, SQL, and QBE queries, and QMF procedures. The difference between environments lies not in what you can do, but in how you need to enter your input and what languages are displayed.

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

Usage codes for forms
The codes are identical; they are not translated.

System commands
TSO and CICS commands can still be issued from QMF through the TSO or CICS command. These commands are unaffected by translation: enter TSO or CICS followed by the command to be run, and enter the command exactly as you would if you were running it outside of QMF.
Environmental differences

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

The QMF command language
Every NLF has a complete set of translated QMF verbs and keywords. These translated verbs and keywords must appear in your QMF commands when you are operating in the language environment of an NLF. For a particular NLF, these words might be translated. For example, suppose that in the German NLF the verb DISPLAY and the keyword PROC are translated into ANZEIGEN and PROZEDUR. During a German NLF session, you can issue the command ANZEIGEN PROZEDUR but you cannot issue the command 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 translated QMF messages and panels. Like the verbs and keywords for QMF commands, these messages and panels might or might not be translated. In most cases, they are translated. Within the panels and messages, the fixed portions of text can be translated. Information that can vary within each panel or message, such as query names, is not translated.

Allowable panel input
Many QMF panels that require user input restrict the range of some entries to a small set of keywords, which are translated. Examples of these panels include prompt panels and form panels. YES and NO responses in English, for example, are JA and NEIN in German.

Profile parameter values
In a multilingual environment, users have a separate profile for each available NLF they can use for a QMF session. For each of these profiles, the parameters are the same and have the same meanings, but the parameter names are translated. Certain parameter values are also translated.

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 valid values are GROSS, KETTE, and GEMISCHT.

Exported and saved form objects
Use the SAVE, EXPORT, and IMPORT commands to specify the language in which you want form objects to be saved. You can save them in English or in the presiding language of your current session.

Sample tables and queries
IBM supplies translated versions of the English sample tables and queries, except for the Swedish and Uppercase features. Sample tables are not provided for these features.
Creating objects for use in bilingual 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 create or save bilingual applications in English depends on the specific object:

Queries
You can create prompted and QBE queries in any language supported by the QMF NLFs or you can create SQL queries in English.

Forms
Always create forms in the presiding language. Save them by either using the default language on the SAVE command (ENGLISH) or use the presiding language.

The global variable DSQEC_FORM_LANG controls the language in which a form is saved, imported, or exported. The default value is 1 for English. A value of zero specifies that the forms are to be saved, imported, and exported in the presiding session language.

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

Analytics
You can create analytics objects in either English or the presiding language.

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

```
SAUVER FORM SEMAINE_F EN WEEKLY_F (LANGUE=ANGLAIS)
```

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

Related reference:
“Conventions for National Language Feature information” on page 4

Db2 QMF is available in several different languages, each of which is provided by a National Language Feature (NLF).

The command language variable

To use English commands in an NLF session, set the presiding language variable, DSQEC_NLFCMD_LANG, to English. Use this variable to switch between English and the presiding language of the NLF session.

For example, suppose that your application is a procedure named WEEKLY_P. The commands shown here demonstrate how to switch between English and the presiding NLF language.

```
"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:

1. This line of the procedure saves the presiding language value in a variable.
The GET GLOBAL command saves the value for the presiding language in a variable called CURR_LANG.

2 This line of the procedure sets the presiding language to the language for which the application was written.

The WEEKLY_P application in this example was written with English commands. For this reason, the SET GLOBAL command sets the presiding language to English by setting the DSQEC_NLFCMD_LANG variable to 1.

3 This line of the procedure runs the application.

After the QMF session is set to English, the application in the example can be run. User commands 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 data set type, data queue type, or path name is also in the presiding language.

The QMF profile in effect for the session is the user's profile under the NLF that was set when the application started. The QMF profile of the presiding language is not the profile in effect. For example, a user who runs QMF in both English and German has both English and German QMF profiles. If the user starts a QMF session under the German NLF, the options in the German QMF profile are in effect. Then the user sets the DSQEC_NLFCMD_LANG variable to English to run a procedure written with English commands. In this case, the options in the German QMF profile remain in effect throughout the session.

4 This line of the procedure returns to the presiding language.

After the application ends, reset the command language variable to its original value as shown in the example.

**Initial procedures in bilingual applications**

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, a user who runs QMF in English 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 here:
  
  ```
  /* QMF commands in the presiding language */
  "SET GLOBAL (DSQEC_NLFCMD_LANG=0"
  
  
  "GET GLOBAL (CURR_LANG=DSQEC_NLFCMD_LANG"
  "SET GLOBAL (DSQEC_NLFCMD_LANG=CURR_LANG"
  ``

- 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.
English-only commands

For most QMF commands, you must change the presiding language variable before you can run the command in English. However, some commands must be issued in English even when the presiding language variable is not set to English.

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

```plaintext
proceed_text = 'Continue...'
"RUN WEEKLY\_0" /* Use the English RUN command */
"SET GLOBAL\"OSQEC\_NLF\_CMD\_LANG\=0\" /* switch back to French */
"MESSAGE \{TEXT=\"proceed\_text\"\} /* message in French */
"INTERACT" /* show the report with message */
```

*Figure 8. Using the MESSAGE and INTERACT commands to display messages*

The following English commands work in any NLF:

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

Multilingual environments

When one or more NLFs are installed, a multilingual environment is created. With the appropriate authorization, you can choose a 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 cannot switch languages during a QMF session, you can switch the command language variable. End the current session and start another to obtain the appropriate language environment.

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. Use variables to run the same program in several NLFs.

These variables can include:

- The verbs, object names, and options of a QMF command
- User-defined panel names
  - If you create your own ISPF panels for your application, you need a set of translated panels for each language in which the application is run. Give these panels unique names and make them available to the application users. The application can then use variables for the panel names.
- User-defined message identifiers
  - If you create your own ISPF panels, you also create messages to be issued by ISPF. These messages panels have unique IDs and you can use variables to refer
to them. Translate the message text into the appropriate NLF languages. The application can use variables for the message names.
Chapter 8. QMF commands in applications

Certain commands are designed to be used in applications, and you can create your own command synonyms.

Commands designed for applications

Any command that is valid on the QMF command line in a particular environment is valid in an application. However, certain commands are specially designed to be used in applications.

Commands that are designed for applications can be used in both callable and command interface applications, with two exceptions. You can use the START and TRACE commands only with the callable interface.

CONNECT

Use the QMF CONNECT command to access data and objects on a remote server. When you connect to the remote system, this system becomes the current location.

When you write applications, you can issue the CONNECT command from:
• The callable interface
• The command interface
• Within a procedure (linear or with logic)
  However you cannot issue the CONNECT command if the procedure is a parameter on the CALL statement that starts QMF for TSO as a Db2 for z/OS stored procedure. Connectivity with remote servers is not supported when QMF is started as a stored procedure.

Certain aspects of your applications can be affected when you use the QMF CONNECT command to access a remote server. 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.
• When QMF starts, the program can issue a QMF CONNECT command to connect to a remote server. Any subsequent QMF commands or SQL statements that affect database objects are run at the remote server.
• Different types of commands behave differently with remote unit of work. When your applications use remote unit of work, all system-specific and most QMF commands run at the system where QMF is running (the local system).
  However, when a QMF command does either of the following, the commands affect the database at the remote server:
  – Sends SQL commands to the database
  – Uses or alters QMF objects and data stored in the database
• If your site uses TSO and takes advantage of RACF support for mixed-case passwords, ensure that the CASE option of your QMF profile is set to MIXED. Otherwise, QMF converts all input to uppercase, causing the CONNECT command to fail because the case of the password is incorrect. When CASE=MIXED, ensure that you tell QMF application users to enter all input in uppercase, because QMF recognizes commands only in uppercase.
The following example statements show how a REXX callable interface program can use the QMF CONNECT command. The program connects to a remote server, performs a predefined task, and exits QMF:

1. The following statement starts a QMF session:
   
   ```sql
   CALL DSQCIX "START"
   ```

2. The following statement connects to the remote Db2 database (DALLAS):
   
   ```sql
   CALL DSQCIX "CONNECT TO DALLAS"
   ```

3. This statement runs a procedure with logic (EARNINGS) that queries the remote server for data, formats the data, and prints the report:
   
   ```sql
   CALL DSQCIX "RUN PROC EARNINGS"
   ```
   
   The procedure EARNINGS contains the following logic:
   
   ```sql
   : 
   "RUN QUERY EARNQ (FORM=EARNF"
   "PRINT REPORT"
   ```

4. This statement ends the QMF session:
   
   ```sql
   CALL DSQCIX "EXIT"
   ```

**END**

You can use the END command to set the QMF home panel as the current panel.

For example, a QMF report is the current QMF panel. Issuing the END command from a callable interface or command interface program sets the QMF home panel as the current screen. When the QMF home panel is the current screen, issuing the END command has no effect on the QMF 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.

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 command when the application no longer needs 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 that runs 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.

**Related reference:**

(Global variables for state information not related to the profile” on page 200)

DSQAO global variables contain status information or settings of parameters or flags. None of these global variables can be modified by the SET GLOBAL command.
GET GLOBAL (extended syntax)

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 for the GET GLOBAL command.

```
GET Global—(— Variable definitions ——–)
```

**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. An 18-character area padded with trailing blanks is allowed.

- **varnames**
  A list of names of the QMF variables.
  Do not specify trailing blanks in global variable names; QMF deletes trailing blanks.

- **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 the value stored in QMF, QMF truncates on the right and returns a truncated value.
  - If the value length you supply is greater than the value stored in QMF, QMF returns a value padded with trailing blanks.

- **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

You can use the INTERACT command to display the current QMF panel and allow users to interact with QMF at different points in your application. The INTERACT command has two forms: session and command.
When the user issues the END command from a QMF panel, QMF returns control to the application. When the user issues the EXIT command from a QMF panel, the QMF session is marked for termination and QMF returns control to the application.

**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 after 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 after 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**

This example shows a procedure that requires only one step to produce a report.

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

*Figure 9. A simple procedure without the INTERACT command*

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

You write a procedure that involves several steps and you want to see the intermediate results of the procedure. For example, you want to see the intermediate results of a procedure that runs more than one query. Use the INTERACT command at the points in the procedure where you want to see the results of a command. In this case, insert an INTERACT command immediately following the first RUN QUERY command in the following procedure:

```sql
/* 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 10. Using INTERACT in a procedure*

When you run this procedure from the home panel, QMF displays the REPORT panel that contains your formatted data. Next, enter the END command from the REPORT panel. The procedure runs the second query and displays the final report. If you omit the INTERACT command, QMF displays only the final report without showing the result of the first query.
Suppressing the display of reports

If you run a query in a QMF callable interface application, QMF displays the resulting report. However, you can tell QMF not to automatically display the 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 the RUN QUERY command is issued from the QMF command line.

Ending a session started by the INTERACT command

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, the user modifies the current form object in the new interactive session. In this case, the current form object in the old session contains these modifications when the new session ends.

You can make your application display the QMF home panel after the user issues an END command from a QMF object panel. Add the logic of an INTERACT loop.

Related reference:
“A REXX example of using an INTERACT loop” on page 194

You can make the END command in an interactive session behave similarly to the way END behaves in interactive QMF.

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 DSQSMODE=I on the START command.

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

\[
\text{INTERACT command}
\]

In this statement, \textit{command} is the command that you want to run interactively. 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:

\[
\text{INTERACT RUN QUERY ABC ?}
\]

If interactive execution is not allowed, the command form of INTERACT has no effect on the command it precedes. An interactive session is not allowed in a QMF batch session or when QMF for TSO is started as a Db2 for z/OS stored procedure.

You can check whether interactive execution is allowed in the current session by examining a global variable named DSQAO_INTERACT. A value of 1 for the DSQAO_INTERACT global variable means that INTERACT is allowed.

Related reference:
MESSAGE

When you create applications, you often want to send specific messages to your users about the information displayed for them or their next action. You can write your own messages and display them on QMF panels through the MESSAGE command. In ISPF, you can also choose to have QMF display the message help for an ISPF error message.

Syntax

The MESSAGE command syntax is as follows:

```
MESSAGE [number] (Help=helppanel | Stopproc=Yes|NO | Text=value)
```

**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.

You cannot modify a QMF panel to be displayed if its definition is in DSQPNLE.

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

In ISPF, if you 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 reference a help-panel indicator, then the MESSAGE command does not provide message help. Instead, the QMF help for the object panel is displayed on the user's screen when the user requests help.

**STOPPROC**

Use *Stopproc* to suppress the execution of linear procedures by setting the procedure termination switch. 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 affects only linear procedures.
While this switch is on, any QMF procedure that receives 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 back on. When the switch is on, it stays on until one of the following happens:

- Another QMF command is issued. This command can be any QMF command except a MESSAGE command with the option to turn on the switch.
- 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 procedure termination switch is on by examining the DSQCM_MESSAGE variable. If the termination option is in effect, this variable contains the message for the MESSAGE command that turned on the termination switch.

**TEXT**

Use the TEXT option 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 quotation marks, parentheses, and double quotation marks. When the delimiters are double quotation marks, the quotation marks are displayed as part of the message. The maximum length for a message value is 360 single-byte characters. How much of the message can be displayed is determined by the width of the display device you are using. A value longer than 78 characters is truncated to contain only the first 78 characters. QMF does not fold the text to uppercase; however, ISPF might fold the text to uppercase if MESSAGE is issued through the command interface.

If your message contains quotation marks, you need to double the quotation marks 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, regardless of the value of the CASE setting in the user's QMF profile.

Suppose that you want to write an application by using a procedure that runs two queries and displays two reports. When QMF displays the first report, you want to tell users how to view the second report. You can write a linear procedure like the one shown here, which includes on the REPORT panel a message defined by the MESSAGE command. To have your message be displayed on the REPORT panel, place the MESSAGE command immediately before the INTERACT command, as shown here:

```plaintext
::
  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 11. Example of using the MESSAGE command*

If you are using a procedure with logic, you can use a REXX variable in place of the text string you specify for the MESSAGE command, as shown here. When you use REXX variables, you must use double quotation marks around the variable...
name in the message text string.

```rxx
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 12. Using REXX variables with the MESSAGE command in a procedure

This message spans multiple lines with REXX continuation characters.

/* QMF REXX PROCEDURE */
MSGTEXT="You entered a data value incompatible with ",
"the column data type; check the data type of the ",
"column and try again."
"MESSAGE(TEXT=("MSGTEXT"))"
EXIT

Examples

Here are some examples of how to issue the MESSAGE command under various conditions.

• Example of issuing the MESSAGE command from a QMF linear procedure:
  This message spans multiple lines by using continuation characters for linear procedures:
  ```rxx
  MESSAGE(TEXT='You entered a data value incompatible with "+the column data type; check the data type of the "+column and try again.'
  "MESSAGE(TEXT=("MSGTEXT"))"
  EXIT
  ```

• Examples of using MESSAGE commands with ISPF:
  The following are examples of how you can use the MESSAGE command if you are building an application that uses ISPF:
  ```rxx
  - 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 switch is turned on, which suppresses the execution of QMF linear procedures in the application.
  ```

SET GLOBAL (extended syntax)

To create your own global variables and use them in QMF commands as substitution variables, issue the SET GLOBAL command. You can also use the SET GLOBAL command to set values for QMF predefined global variables, which start with "DSQ."

Restriction: You cannot use the SET GLOBAL command to set global variables that are defined as read-only.

Syntax of SET GLOBAL (extended syntax)

You can use the extended syntax of the SET GLOBAL command to change the values of variables in callable interface languages other than REXX. Examples of other languages include Assembler, C, COBOL, Fortran, and PL/I.
The variable name can be up to 18 characters long for variables used with callable interface applications. If the variable is to be used as a substitution variable, the name can be up to 17 characters long. The maximum length of the command, including the command syntax, is 2,000 bytes.

The syntax of the command is as follows:

```
SET GLOBAL (Variable definitions)
```

**Variable definitions:**

```
number of varnames, varname lengths, varnames, value lengths, values, value type
```

- **number of varnames**
  - The number of variables requested.

- **varname lengths**
  - A list of lengths for each variable name specified.
  
  Ensure that the length of the global variable name is equal to the actual length of the global name in your program. 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. 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.
  
  QMF uses the value from your program, 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.

Examples of how to use the extended syntax of the SET GLOBAL command are documented with the programming language specifications.

**Related reference:**

Chapter 11, “Programming language specifications for using the callable interface,” on page 139

The QMF application programming interface is available for several programming languages.

**Guidelines for defining and using global variables**

Global variable names are subject to certain rules.

When you are defining and using global variable names, keep the following rules in mind:

- On the SET GLOBAL command, variable names are not preceded with an ampersand as they are on the RUN and CONVERT commands.
If you create a global variable with the same name as a form variable or aggregation variable, QMF does not use the global variable in the form. QMF uses the form variable (or aggregation variable) value in the form rather than the global variable value.

- The QMF form does not recognize global variables with question marks in the names.
- Global variable names are limited to 18 characters unless the variable is to be used as a substitution variable. Substitution variable names are limited to 17 characters.
- A global variable name can contain numeric characters, but the first character of a global variable name cannot be numeric.
- Global variable names 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:
  - * ( ) - + \| : ; " ' < > / , . , = &
- QMF strips the trailing blanks from global variable names.
- By default, a global variable value is retained until you reset it or end the QMF session. However, you can set the DSQEC_USERGLV_SAV global variable to save global variable values from one session to another.

Related reference:

- “Global variables that control how commands and procedures are executed” on page 218
- DSQEC global variables control how commands and procedures are executed. All of these global variables can be modified by the SET GLOBAL command.
- Appendix B, “QMF global variables,” on page 199
- QMF provides many global variables that help you control aspects of your QMF session, QMF commands, and panel display. The global variables also help you control behavior of QMF functions in procedures and applications.

START

When you start QMF through the callable interface, you need to use the START command.

General syntax

The syntax of the START command depends on which programming language you are using for your callable interface application.

Only one QMF session can be active at a time. When you start QMF from an application, issue a START command to test whether QMF is started.

Examples of the syntax for each programming language are documented with the programming language specifications.

The following is the general syntax for the START command:

```
START (Keyword definitions)
```
Keyword definitions:

- number of keywords
- keyword lengths
- keywords
- value lengths
- values
- value type

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.

**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 value. The data type must be character for the START command.

Related concepts:

**“Starting QMF from an application” on page 21**

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 in your application. You can have only one QMF session at a time.

Related reference:

Chapter 11, “Programming language specifications for using the callable interface,” on page 139

The QMF application programming interface is available for several programming languages.

**“START command keywords”**

You can specify keywords on the START command.

**START command keywords**

You can specify keywords on the START command.

These keywords are available on the START command:

- DSQADPAN
- DSQALANG
- DSQSBSTG
- DSQSCMD (TSO only)
- DSQSDBCS
- DSQSDBNM
- DSQSBQN (CICS only)
- DSQSBQT (CICS only)
- DSQSDBUG
- DSQSIROW
- DSQSMODE
- DSQSMRFI
- DSQSPILL
- DSQSPLAN (TSO only)
- DSQSPRID (TSO only)
- DSQSPTYP (TSO only)
- DSQSRSTG (TSO only)
- DSQSRUN
- DSQSSPQN (CICS only)
- DSQSSUBS (TSO only)

These keywords are described in the following table.

You can specify START command keywords with the following conventions:

- You can specify any keyword on the START command. In TSO, you can also specify any keyword in the REXX program named by the DSQSCMD keyword. Because QMF for CICS does not support REXX, you must specify all keywords on the START command.
- If your application or the initial procedure (specified by the DSQSRUN keyword) specifies keywords that are not supported in a particular environment, those keywords are ignored. With this convention, you can compile a single program to run in multiple QMF environments without changing the environment-specific keywords.
- 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 shown in this table.

### Table 8. START command keywords, descriptions, and default values

<table>
<thead>
<tr>
<th>START command keywords</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQADPAN</td>
<td>Meant for use only with the callable interface, this START command parameter 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 not display the report.</td>
<td>1 (display report)</td>
</tr>
</tbody>
</table>
Table 8. START command keywords, descriptions, and default values (continued)

<table>
<thead>
<tr>
<th>START command keywords</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
</table>
| DSQALANG               | Determines the presiding national language for the session you are starting. You can specify this parameter in your applications so that users can enter or specify QMF commands in a national language. The value for this parameter is a one-character national language identifier, shown here. If you want to enter English commands when the presiding language is a language other than English, you can use QMF bilingual support. The national language feature for the language you specify must be installed.  
  - C - Canadian French  
  - D - German  
  - E - English  
  - F - French  
  - H - Korean (Hangeul)  
  - I - Italian  
  - K - Japanese (Kanji)  
  - P - Brazilian Portuguese  
  - Q - Danish  
  - S - Spanish  
  - U - Uppercase English  
  - V - Swedish  
  - Y - Swiss French  
  - Z - Swiss German | E (English)               |
Table 8. START command keywords, descriptions, and default values (continued)

<table>
<thead>
<tr>
<th>START command keywords</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQSBSTG</td>
<td>Specifies the maximum amount of virtual storage per user that is to be used to generate QMF reports. Valid values are: 0 - 2147483647 (Specifies storage in bytes. 0KB - 2097152KB (Specifies storage in KB. 0MB - 2048MB (Specifies storage in MB. 0GB - 2GB (Specifies storage in GB. 1% - 100% (Specifies a percent of available storage. Percentages are valid in TSO only. If you are using CICS, you can specify a fixed amount of storage in bytes, KB, MB, or GB. If you are using TSO, by default QMF determines the maximum amount of storage that is to be used for generating QMF reports. If you set the DSQSBSTG parameter to a valid non-zero value, the amount of storage that you specify for the parameter is used instead. If you are using TSO, you can specify the value for DSQSBSTG as a fixed amount of storage or as a percentage of the available virtual storage. If you set the DSQSBSTG parameter to 0 and set the DSQSRSTG parameter to a non-zero value, the DSQSRSTG parameter is used to determine the maximum amount of storage for reports. If you specify valid non-zero values for both DSQSBSTG and DSQSRSTG, the value of DSQSBSTG is used. If you set the DSQSBSTG value to less than the minimum amount of storage that is required to produce a report, QMF automatically allocates the minimum amount of storage required. In TSO, the minimum amount of storage is 15,000 bytes. Storage availability is reassessed throughout the QMF session. Restrictions: • Percent values must include the percent sign (%) and contain no spaces. • Values in GB, MB, or KB format cannot contain spaces (for example, you cannot enter 2 GB). • Values in GB, MB, or KB format cannot contain characters after the units (for example, you cannot enter 2GBxyz). • GB, MB, or KB entered with no preceding number is handled as a value of 0.</td>
<td></td>
</tr>
</tbody>
</table>

In CICS: 500000
In TSO: 0
Table 8. START command keywords, descriptions, and default values  (continued)

<table>
<thead>
<tr>
<th>START command keywords</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQSCMD (TSO only)</td>
<td>This keyword specifies the REXX program that sets the QMF program parameters in QMF for TSO. When QMF receives the START command from a callable interface application, TSO calls the REXX program specified by this keyword. This REXX program provides values for the QMF program parameters unless you specified their values directly on the START command. The default program provided with TSO is DSQSCMDE, which provides default English program parameter values for all keywords shown in this table. A value of NULL for a particular parameter indicates that TSO uses the default value for that parameter. If you are using an NLF, you can change the default program name to DSQSCMDn. The n variable is the national language identifier (NLID) for the language you are using. Though not shown in this table, the DSQSDBLG parameter is also set by default in the DSQSCMDE program provided with TSO. This parameter is set when you start QMF for TSO as a Db2 for z/OS stored procedure. It cannot be externally set outside the context of the stored procedure interface.</td>
</tr>
<tr>
<td>DSQSCMDE</td>
<td>DSQSCMDE</td>
</tr>
<tr>
<td>DSQSDBCS</td>
<td>Determines whether QMF allows double-byte characters when the display device does not support the double-byte character set (DBCS). Values are YES or NO. Set the value to YES when you intend to print DBCS data from a non-DBCS display device or run a QMF batch job that prints DBCS data. Otherwise, set the value to NO.</td>
</tr>
<tr>
<td>DSQSDBNM</td>
<td>NULL</td>
</tr>
<tr>
<td>DSQSDBQM (CICS only)</td>
<td>Specifies the remote server 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).</td>
</tr>
<tr>
<td>DSQSDBQT (CICS only)</td>
<td>Specifies the type of CICS storage to be used for QMF trace data. The name must conform to CICS name specifications for the type of CICS queue specified by DSQSDBQT.</td>
</tr>
<tr>
<td>DSQSDBQT</td>
<td>DSQD</td>
</tr>
<tr>
<td>DSQSDBUG</td>
<td>Specifies whether product activity is traced during QMF initialization. The values are:</td>
</tr>
<tr>
<td></td>
<td>ALL       Specifies the most detailed QMF tracing.</td>
</tr>
<tr>
<td></td>
<td>NONE      Specifies no QMF tracing. When you start QMF in batch mode, all messages and commands are traced (equivalent to an L2 level of tracing) regardless of how you set DSQSDBUG.</td>
</tr>
<tr>
<td></td>
<td>NONE</td>
</tr>
</tbody>
</table>

Chapter 8. QMF commands in applications  55
<table>
<thead>
<tr>
<th>START command keywords</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQSIROW</td>
<td>Indicates the number of rows QMF fetches before displaying the first screen of data for a RUN QUERY, IMPORT DATA, or DISPLAY command.</td>
<td>100</td>
</tr>
<tr>
<td>DSQSMODE</td>
<td>Specifies the mode in which to start QMF.</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>Specifies interactive mode.</td>
<td>B (batch)</td>
</tr>
<tr>
<td>B</td>
<td>Specifies batch mode.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>When the value of DSQSMODE is B, panel display is inhibited so that QMF can run as a background job.</td>
<td></td>
</tr>
<tr>
<td>DSQSMRFI</td>
<td>Specifies whether the QMF session that you are starting uses Db2 multirow fetch and insert. Db2 multirow fetch and insert increases performance for many QMF commands (such as DISPLAY TABLE, EXPORT DATA or EXPORT TABLE, IMPORT TABLE, PRINT REPORT or PRINT TABLE, RUN QUERY or RUN PROC) when these commands retrieve data. Db2 multirow fetch and insert also increases performance for commands such as SAVE DATA, DPRE, and BOTTOM or FORWARD during report navigation. This parameter sets the DSQAO_DSQSMRFI global variable.</td>
<td></td>
</tr>
<tr>
<td>YES</td>
<td>Specifies that QMF uses Db2 multirow fetch and insert.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>When MR is set to YES and you use a QMF command that includes a three-part name, the servers must run Db2 for z/OS. Both the requester where the command is initiated and the server to which the command is directed must run at this version level. Commands with three-part names cannot be directed to DB2 for VSE and VM servers.</td>
<td></td>
</tr>
<tr>
<td>NO</td>
<td>Specifies that QMF will not use Db2 multirow fetch and insert capabilities.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>When retrieving XML or LOB data, QMF uses single-row fetch, regardless of the DSQSMRFI parameter setting.</td>
<td></td>
</tr>
<tr>
<td>DSQSPILL</td>
<td>Specifies whether QMF uses spill storage when extra storage for reports is needed. Possible values are YES or NO.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If you specify a value of YES for DSQSPILL and are using QMF for TSO, ensure that the DSQSPTYP parameter is set to accommodate the type of spill storage you use.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If you are using CICS, see the explanation of the DSQSSPQN keyword for how to name the temporary storage queue that holds spill data.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For CICS: NO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For TSO: YES</td>
<td></td>
</tr>
<tr>
<td>DSQSPLEN (TSO only)</td>
<td>Specifies the Db2 application plan ID assigned to QMF.</td>
<td>QMF12</td>
</tr>
<tr>
<td>DSQSPRID (TSO only)</td>
<td>Specifies whether to use the TSO logon ID or the primary database authorization ID to select the appropriate row from Q.PROFILES and to qualify Q.ERROR_LOG entries. Allowable values are PRIMEID or TSOID.</td>
<td>PRIMEID</td>
</tr>
</tbody>
</table>
Table 8. START command keywords, descriptions, and default values (continued)

<table>
<thead>
<tr>
<th>START command keywords</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQSPTYP (TSO only)</td>
<td>When the DSQSPI II parameter is set to YES, specifies the type of storage to use for spill data. A value of FILE specifies a file; a value of 64BIT specifies extended storage. When extended storage is specified, QMF acquires storage on each call to the extended storage manager in the amount specified in the DSQEC_EXTND_STG global variable.</td>
<td>FILE</td>
</tr>
<tr>
<td>DSQSRSTG (TSO only)</td>
<td>Dynamically allocates virtual storage available for reports. You can alternatively use the DSQSBSTG keyword to set the maximum amount of storage as a fixed amount or a percentage of the available storage. If DSQSBSTG is set to a non-zero value, QMF ignores the DSQSRSTG value.</td>
<td>0</td>
</tr>
<tr>
<td>DSQSRUN</td>
<td>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 values to customize the user's session.</td>
<td>NULL</td>
</tr>
<tr>
<td>DSQSSPQN (CICS only)</td>
<td>Specifies the name of the CICS temporary storage queue that is used for QMF spill data. When the program parameter DSQSPI II has a value of YES, this spill area is used to contain report data.</td>
<td>DSQSvid, where id is the CICS terminal ID</td>
</tr>
<tr>
<td>DSQSSUBS (TSO only)</td>
<td>Specifies the ID of the Db2 database in which QMF is started. The database ID you specify on this keyword must be configured as an application requester.</td>
<td>DSN</td>
</tr>
</tbody>
</table>

Related concepts:
- [Chapter 7, “Writing bilingual applications,” on page 35](#)

Many business applications need to run in several different national languages. You can write one English application and run it in any national language that QMF supports.

Related reference:
- [“Conventions for National Language Feature information” on page 4](#)
 Db2 QMF is available in several different languages, each of which is provided by a National Language Feature (NLF).
- [“Global variables that control how commands and procedures are executed” on page 218](#)
 DSQEC global variables control how commands and procedures are executed. All of these global variables can be modified by the SET GLOBAL command.

**TRACE**

You can use the TRACE command to add trace information from callable interface applications to the QMF trace data output. This command can be used in Assembler, C, COBOL, Fortran, and PL/I applications. It cannot be issued from the QMF command line.

The TRACE command syntax is as follows:

```
TRACE (Trace area definitions)
```
Trace area definitions:

- number of trace areas
- trace title lengths
- trace title addresses
- trace area lengths
- trace area addresses
- value type

number of trace areas
The number of trace area definitions that you are using in your TRACE command. This number must be in the range of 1 through 10.

trace title lengths
A list of lengths for each trace title that is specified.

trace title addresses
A list of addresses that point to the trace titles that are to be used for each trace area. A trace title can be up to 40-characters long. Trailing blanks are removed. When the first trace title is SNAPREGS, all other trace titles and trace area addresses are ignored, and QMF register values are written to the QMF trace.

trace area lengths
A list that contains the number of bytes of storage that are to be displayed starting at the corresponding trace area address. Trace area lengths must be contained in FIXED(31) integer values.

trace area addresses
A list of addresses that are to be displayed in the trace output. The number of bytes of storage that are displayed at each trace area address is determined by the trace area length. When the first trace title is “SNAPREGS,” all trace area addresses and trace area lengths are ignored. When the trace area address is 0, the trace area length is also considered to be 0.

value type
Must be value “FINT.” “FINT” is a constant value that is provided in the interface communications area (DSQCOMM) for each programming language. The constant is a name that is similar to DSQ_VARIABLE_FINT. Check the interface communications area for your programming language to confirm the variable value.

The TRACE command writes trace area definitions to the QMF trace data output, regardless of QMF trace settings. If you want to write trace output only if QMF trace settings are active, use the DSQAO_APPL_TRACE or DSQAP_TRACE QMF global variables.

PL/I coding example for TRACE
You can use the TRACE command in a PL/I application to write trace information for the application to the QMF trace output.

The following coding example first verifies that the user is tracing QMF application activity by checking that the global variable DSQAO_APPL_TRACE is not set to ‘0’. If the application trace is on, then the TRACE command is issued. The TRACE command specifies three trace area definitions to write to the QMF trace output.

This example is not included in the DSQABFP file that is distributed with QMF.

```
DSQABFP: PROCEDURE OPTIONS(MAIN REENTRANT) REORDER;
/******************DSQABFP: PL/I Version of the QMF Callable Interface***********/
/* Sample Program: DSQABFP */
/* PL/I Version of the QMF Callable Interface */
```
/**

/* Include and declare query interface communications area */
#include SYSLIB(DSQCOMML);

/* Builtin function */
DCL LENGTH BUILTIN;

/* Query interface command length and commands */
DCL COMMAND_LENGTH FIXED BIN(31);
DCL START_QUERY_INTERFACE CHAR(5) INIT('START');
DCL GET_GLOBAL_VARIABLES CHAR(10) INIT('GET GLOBAL');
DCL RUN_QUERY CHAR(12) INIT('RUN QUERY Q1');
DCL PRINT_REPORT CHAR(22) INIT('PRINT REPORT (FORM=F1)');
DCL TRACE_COMMAND CHAR(5) INIT('TRACE');
DCL END_QUERY_INTERFACE CHAR(4) INIT('EXIT');

/* Query command extension, number of parameters and lengths */
DCL NUMBER_OF_PARAMETERS FIXED BIN(31); /* number of variables */
DCL KEYWORD_LENGTHS(10) FIXED BIN(31); /* lengths of keyword names */
DCL DATA_LENGTHS(10) FIXED BIN(31); /* lengths of variable data */

/* Trace command parameters */
DCL AREA_DESCRIPTION(10) CHAR(40);
DCL AREA_DESCRIPTION_LENGTH(10) FIXED BIN(31);
DCL AREA_PTR(10) PTR;
DCL AREA_LENGTH(10) FIXED BIN(31); /* Length of area at */
                     /* AREA_PTR to be */
                     /* displayed. */

/* Keyword parameter and value for START command */
DCL 1 START_KEYWORDS,
     3 START_KEYWORDS_1 CHAR(8) INIT('DSQSMODE'),
     3 START_KEYWORDS_2 CHAR(8) INIT('DSQSDBUG'),
     3 START_KEYWORDS_3 CHAR(8) INIT('DSQSSUBS'),
     3 START_KEYWORDS_4 CHAR(8) INIT('DSQSPLAN');
DCL 1 START_KEYWORD_VALUES,
     3 START_KEYWORD_VALUES_1 CHAR(11) INIT('BATCH'),
     3 START_KEYWORD_VALUES_2 CHAR(3) INIT('ALL'),
     3 START_KEYWORD_VALUES_3 CHAR(4) INIT('DSNA'),
     3 START_KEYWORD_VALUES_4 CHAR(6) INIT('QMFDEV');

/* Keyword parameter and value for SET command */
DCL 1 SET_KEYWORDS,
     3 SET_KEYWORDS_1 CHAR(7) INIT('MYVAR01'),
     3 SET_KEYWORDS_2 CHAR(5) INIT('SHORT'),
     3 SET_KEYWORDS_3 CHAR(7) INIT('MYVAR03');

DCL 1 SET_VALUES,
     3 SET_VALUES_1 FIXED BIN(31),
     3 SET_VALUES_2 FIXED BIN(31),
     3 SET_VALUES_3 FIXED BIN(31);
/* Keyword parameter and value for GET command */

DCL 1 GET_TRACE_KEYWORDS,
    3 GET_TRACE_KEYWORDS_1 CHAR(16) INIT('DSQAO_APPL_TRACE');

DCL 1 GET_TRACE_VALUE,
    3 GET_TRACE_VALUE_1 CHAR(1);
DATA_LENGTHS(2) = 4;
DATA_LENGTHS(3) = 4;
SET_VALUES_1 = 20;
SET_VALUES_2 = 40;
SET_VALUES_3 = 4;

CALL DSQCIPX(DSQCOMM,
          COMMAND_LENGTH,
          SET_GLOBAL_VARIABLES,
          NUMBER_OF_PARAMETERS,
          KEYWORD_LENGTHS,
          SET_KEYWORDS,
          DATA_LENGTHS,
          SET_VALUES,
          DSQ_VARIABLE_FINT);

//*************************************************************************/
/**
 Run a Query */
/*****************************************************************************/
COMMAND_LENGTH = LENGTH(RUN_QUERY);

CALL DSQCIP(DSQCOMM,
           COMMAND_LENGTH,
           RUN_QUERY);

//*************************************************************************/
/**
 Trace command */
/*****************************************************************************/
IF GET_TRACE_VALUE_1 ^= '0' THEN DO;
  NUMBER_OF_PARAMETERS = 3;
  COMMAND_LENGTH = LENGTH(TRACE_COMMAND);
  AREA_DESCRIPTION(1) = 'DSQAO_APPL_TRACE:';
  AREA_DESCRIPTION_LENGTH(1) = LENGTH(AREA_DESCRIPTION(1));
  AREA_PTR(1) = ADDR(GET_TRACE_VALUE_1);
  AREA_LENGTH(1) = LENGTH(GET_TRACE_VALUE_1);
  AREA_DESCRIPTION(2) = 'DSQ_COMM_LEVEL:';
  AREA_DESCRIPTION_LENGTH(2) = LENGTH(AREA_DESCRIPTION(2));
  AREA_PTR(2) = ADDR(DSQ_COMM_LEVEL);
  AREA_LENGTH(2) = LENGTH(DSQ_COMM_LEVEL);
  AREA_DESCRIPTION(3) = 'DSQ_CURRENT_COMM_LEVEL:';
  AREA_DESCRIPTION_LENGTH(3) = LENGTH(AREA_DESCRIPTION(3));
  AREA_PTR(3) = ADDR(DSQ_CURRENT_COMM_LEVEL);
  AREA_LENGTH(3) = LENGTH(DSQ_CURRENT_COMM_LEVEL);

CALL DSQCIPX(DSQCOMM,
             COMMAND_LENGTH,
             TRACE_COMMAND,
             NUMBER_OF_PARAMETERS,
             AREA_DESCRIPTION_LENGTH,
             AREA_DESCRIPTION,
             AREA_LENGTH,
             AREA_PTR,
             DSQ_VARIABLE_FINT);
END;

//*************************************************************************/
/**
 Print the results of the query */
/*****************************************************************************/
COMMAND_LENGTH = LENGTH(PRINT_REPORT);

CALL DSQCIP(DSQCOMM,
           COMMAND_LENGTH,
           PRINT_REPORT);

//*************************************************************************/
/**
 End the query interface session */
****************************************************************************/
COMMAND_LENGTH = LENGTH(END_QUERY_INTERFACE);

CALL DSQCPL(DSQCOMM,
            COMMAND_LENGTH,
            END_QUERY_INTERFACE);

END DSQABFP;

When the program detects that the user has tracing set on, the following trace output is written to the QMF trace output:

```
--------------------------------------------------------------------
DSQDTRC : TRACE COMMAND OUTPUT (14534)
TRACE_AREA_NUMBER
341033B8: 00000001 *.... *
TRACE_AREA_TITLE
341036A4: C4E2D8C1 D6D0C1D7 D7D36DE3 D9C1C3C5 *DSQAO_APPL_TRACE*
341036B4: 7A *: *
TRACE_AREA_CONTENTS
340D0A14: F2 *2 *
--------------------------------------------------------------------
DSQDTRC : TRACE COMMAND OUTPUT (14535)
TRACE_AREA_NUMBER
341033B8: 00000002 *.... *
TRACE_AREA_TITLE
341036A4: C4E2D8D6 C3DB6D4D4 60D3C5E5 C5D5737A *DSQ_COMM_LEVEL: *
TRACE_AREA_CONTENTS
340D0500: C4E2D8D3 6DF0F0F1 F0F0F24C *DSQL>001002< *
--------------------------------------------------------------------
DSQDTRC : TRACE COMMAND OUTPUT (14536)
TRACE_AREA_NUMBER
341033B8: 00000003 *.... *
TRACE_AREA_TITLE
341036A4: C4E2D8D6 C3E4D9D9 C5D5E36D C3D6D4D4 *DSQ_CURRENT_COMM*
341036B4: 60D3C5E5 C5D5737A *LEVEL: *
TRACE_AREA_CONTENTS
33F00C50: C4E2D8D3 6DF0F0F1 F0F0F24C *DSQL>001002< *
--------------------------------------------------------------------
```

### Commands you can use in a RUN QUERY report minisession

Some commands you use in QMF applications force the display of a report while the application is running. This environment is called a report minisession. 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.

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 ends.

The QMF global variable DSQDC_DISPLAY_RPT determines whether QMF starts a report minisession. This situation is because DSQDC_DISPLAY_RPT determines whether QMF displays a report after running a query. Set this variable to 1 to display the report and 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 DSQQMF (either interactively or in batch mode) the default value of this global variable is 0. The variable n here represents the national language identifier.

If you run a procedure or an application that runs a query, QMF starts a report minisession. QMF displays the report that results from the query in this minisession.

If your procedure or application does not run a query, or if you run a query from the SQL Query panel, QMF does not start a report minisession.

If you do not want QMF to start a report minisession, take one of the following actions:

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

From a report minisession, you can issue the following commands and synonyms for those commands. Restrictions are shown in parentheses.

- BACKWARD
- BOTTOM
- CANCEL (when pop-up window is active)
- CICS
- DISPLAY REPORT
- DISPLAY CHART
- END
- FORWARD
- GET GLOBAL
- HELP
- INTERACT
- ISPF
- LEFT
- MESSAGE
- PRINT REPORT
- PRINT CHART
- QMF
- RETRIEVE
- RIGHT
- SAVE DATA
- SET PROFILE
- SET GLOBAL
- SHOW REPORT
- SHOW CHART
- SWITCH (when online help is active)
- TOP
- TSO

The following are commands that are not valid in a minisession:

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

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

Related reference:

“Conventions for National Language Feature information” on page 4

Db2 QMF is available in several different languages, each of which is provided by a National Language Feature (NLF).
To create your own global variables and use them in QMF commands as substitution variables, issue the SET GLOBAL command. You can also use the SET GLOBAL command to set values for QMF predefined global variables, which start with "DSQ."

Command synonyms

You can create command synonyms, which are commands that resemble QMF commands and can perform a variety of functions.

Command synonyms give you flexibility and they are useful for users. For example, command synonyms can perform the function of a command or start an application.
Chapter 9. Exporting and importing objects

You can write applications that issue QMF EXPORT and IMPORT commands to place objects outside of the QMF environment.

Your applications can export tables and the following QMF objects:

- DATA
- QUERY
- PROC
- FORM
- REPORT
- CHART

When you export objects except reports or objects that are in CSV format, QMF converts the object to an externalized format. QMF stores the externalized format of the object in a UNIX file (in the case of data or tables only), a TSO data set, or a CICS data queue. The externalized format of QMF objects is a powerful element of QMF application development.

The IMPORT command reads the externalized format and places the object either in QMF temporary storage or in the database. The location depends on how you issue the command.

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

The QMF EXPORT and IMPORT commands on data objects are useful in several situations.

For example, you can use the IMPORT and EXPORT commands to accomplish these goals:

- Provide query results to your application
  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 by using the appropriate format for the object. For example, in the case of data or tables, when you import the UNIX file into QMF, a new QMF object is created. A new object is also created when you import a TSO data set or CICS data queue that contains the object.
  You cannot import reports, charts, or CSV files into QMF. For reports and charts, you can instead import the data and the forms that were used to create them.

- Make QMF objects available to other environments or products.
  **Attention:** Use caution when transferring exported objects between systems with different CCSIDs or character sets, such as between EBCDIC and ASCII systems, or between different NLF environments. Transferring the objects between systems in this way might render them unusable.
If you need to import a prompted or QBE query into a program other than QMF, you must first use the CONVERT QUERY command. The CONVERT QUERY command converts the query to an SQL query that you can export and use in other products.

You can transfer QMF objects between QMF under TSO, ISPF, or native z/OS batch. You can also transfer QMF objects under CICS by using CICS extrapartition transient data queues.

- 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 by 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 the EXPORT DATA and SAVE DATA commands is in where and how the object is stored.

This difference in how objects are stored affects what you can do with the results:

- Exporting a data object produces a UNIX file, TSO data set, or CICS data queue. You can read, modify, or print each item sequentially through QMF application programs or other external applications.
- The SAVE DATA command produces a database table. Actions that use saved data must be taken through the database.

### Exporting data objects and database tables

You can export data and table objects in the QMF, Integrated Exchange Format (IXF), XML, or comma-separated value (CSV) format. The QMF format is the default.

When you run a query, QMF displays the result in a report. The raw data for the report is stored in a temporary storage area as a data object. Relational tables and views that are stored in the database are referred to as table objects.

You can export data and table objects to storage areas external to QMF. The exported formats of a table in temporary storage (DATA) and a table stored in the database (TABLE) are identical. An object that is exported as data can be imported as a table, and vice versa, unless the data is in CSV format.

You can create your own tables outside of QMF. Use the QMF, IXF, or XML format and import the contents of the UNIX file, TSO data set, or CICS data queue that contains the table. Include the required fields and add your own data as appropriate. Then import the UNIX file, TSO data set, or CICS data queue into QMF as a table object.

Related concepts:

“Rules and information for exporting and importing data objects and tables” on page 93

QMF exports data and table objects to temporary storage and has rules for how it allocates that storage. QMF also has specific ways for handling import errors.
Exporting data or tables in QMF format

The data file that you export by using the EXPORT command with the DATAFORMAT=QMF clause consists of two parts: header records, which describe the data in the records, and the data records, which contain the data.

Header records

Header records describe the exported data in the data file.

The record length of an external data file is the length of a row of the data, as described in the data record. The header records that precede the data records are the same length as the data records. If the header information exceeds the length of the data record, multiple header records are written.

Two formats are used for header records. One is used for short column names, and the other is used for long column names. The following tables show the information that is contained in each format of the header records.

Table 9. Header record information for short column names

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Information and type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>QMF object format level (8 characters of data)</td>
</tr>
<tr>
<td></td>
<td>These byte positions say REL 1.0 when all the column names are short names and the DSQDC_SHORT_EXPT global variable is set to 1.</td>
</tr>
<tr>
<td>9-10</td>
<td>Number of header records (halfword signed integer)</td>
</tr>
<tr>
<td>11-12</td>
<td>Number of data columns (halfword signed integer)</td>
</tr>
<tr>
<td>13-30</td>
<td>Column name</td>
</tr>
<tr>
<td></td>
<td>The maximum column width is 18 bytes.</td>
</tr>
<tr>
<td>31-32</td>
<td>Data type (halfword signed integer)</td>
</tr>
<tr>
<td></td>
<td>Data type codes are shown in Table 11 on page 70. This field stores the hexadecimal equivalent of the decimal codes shown in the table for each data type.</td>
</tr>
<tr>
<td>33-34</td>
<td>Column width (halfword signed integer); for most data types this width is the width of the column in bytes, with the following exceptions:</td>
</tr>
<tr>
<td></td>
<td>• In DECIMAL columns, the first byte of the halfword represents the precision, and the second byte represents the scale.</td>
</tr>
<tr>
<td></td>
<td>• In GRAPHIC and VARGRAPHIC columns, this value reflects the width of double-byte characters.</td>
</tr>
<tr>
<td></td>
<td>• In FLOAT columns, this value is either 4, indicating single-precision floating point, or 8, indicating double-precision floating point.</td>
</tr>
<tr>
<td></td>
<td>• In DECFLOAT columns, this value is 8 for long-format values and 16 for extended-format values.</td>
</tr>
<tr>
<td>35</td>
<td>Nulls allowed: Y if nulls are allowed; N if they are not allowed (1 character of data)</td>
</tr>
<tr>
<td>36</td>
<td>Unused byte</td>
</tr>
</tbody>
</table>

The block that is described by bytes 13-36 repeats for as many columns as there are in the data.
Table 10. Header record information for long column names

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Information and type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>QMF object format level (8 characters of data)</td>
</tr>
<tr>
<td></td>
<td>These byte positions say REL 3.0 when the DSQDC_SHORT_EXPT global variable is set to 0.</td>
</tr>
<tr>
<td>9-10</td>
<td>Number of header records (halfword signed integer)</td>
</tr>
<tr>
<td>11-12</td>
<td>Number of data columns (halfword signed integer)</td>
</tr>
<tr>
<td>13-42</td>
<td>Column name</td>
</tr>
<tr>
<td></td>
<td>The default maximum name length is 30 bytes. However, you can use the DSQDC_SHORT_EXPT global variable to set a maximum column width of 18 bytes before you export the data. In that case, the header record format for short column names is used.</td>
</tr>
<tr>
<td>43-44</td>
<td>Data type (halfword signed integer)</td>
</tr>
<tr>
<td></td>
<td>Data type codes are shown in Table 11. This field stores the hexadecimal equivalent of the decimal codes shown in the table for each data type.</td>
</tr>
<tr>
<td>45-46</td>
<td>Column width (halfword signed integer); for most data types this width is the width of the column in bytes, with the following exceptions:</td>
</tr>
<tr>
<td></td>
<td>• In DECIMAL columns, the first byte of the halfword represents the precision, and the second byte represents the scale.</td>
</tr>
<tr>
<td></td>
<td>• In GRAPHIC and VARGRAPHIC columns, this value reflects the width of double-byte characters.</td>
</tr>
<tr>
<td></td>
<td>• In FLOAT columns, this value is either 4, indicating single-precision floating point, or 8, indicating double-precision floating point.</td>
</tr>
<tr>
<td></td>
<td>• In DECFLOAT columns, this value is 8 for long-format values and 16 for extended-format values.</td>
</tr>
<tr>
<td></td>
<td>The default maximum column width is 30 bytes. However, you can use the DSQDC_SHORT_EXPT global variable to set a maximum column width of 18 bytes before you export the data. In that case, the header record format for short column names is used.</td>
</tr>
<tr>
<td>47</td>
<td>Nulls allowed: Y if nulls are allowed; N if they are not allowed (1 character of data)</td>
</tr>
<tr>
<td>48</td>
<td>Unused byte</td>
</tr>
</tbody>
</table>

The block that is described by bytes 13-48 repeats for as many columns as there are in the data.

The data type codes are shown in this table. The hexadecimal codes shown in the first column are used to indicate each data type shown in the third column.

Table 11. Data type codes

<table>
<thead>
<tr>
<th>Code in hexadecimal</th>
<th>Code in decimal</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'180'</td>
<td>384</td>
<td>DATE</td>
<td>Date</td>
</tr>
<tr>
<td>X'184'</td>
<td>388</td>
<td>TIME</td>
<td>Time</td>
</tr>
<tr>
<td>X'188'</td>
<td>392</td>
<td>TIMESTAMP</td>
<td>Timestamp</td>
</tr>
<tr>
<td>X'1C0'</td>
<td>448</td>
<td>VARCHAR</td>
<td>Varying-length character</td>
</tr>
</tbody>
</table>
### Table 11. Data type codes (continued)

<table>
<thead>
<tr>
<th>Code in hexadecimal</th>
<th>Code in decimal</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'1C4'</td>
<td>452</td>
<td>CHAR</td>
<td>Fixed-length character</td>
</tr>
<tr>
<td>X'1D0'</td>
<td>464</td>
<td>VARGRAPHIC</td>
<td>Varying-length graphic</td>
</tr>
<tr>
<td>X'1D4'</td>
<td>468</td>
<td>GRAPHIC</td>
<td>Fixed-length graphic</td>
</tr>
<tr>
<td>X'1E0'</td>
<td>480</td>
<td>FLOAT</td>
<td>Floating point</td>
</tr>
<tr>
<td>X'1E4'</td>
<td>484</td>
<td>DECIMAL</td>
<td>Decimal</td>
</tr>
<tr>
<td>X'1EC'</td>
<td>492</td>
<td>BIGINT</td>
<td>Big integer</td>
</tr>
<tr>
<td>X'1F0'</td>
<td>496</td>
<td>INTEGER</td>
<td>Integer</td>
</tr>
<tr>
<td>X'1F4'</td>
<td>500</td>
<td>SMALLINT</td>
<td>Small integer</td>
</tr>
<tr>
<td>X'38C'</td>
<td>908</td>
<td>VARBINARY</td>
<td>Varying-length binary</td>
</tr>
<tr>
<td>X'390'</td>
<td>912</td>
<td>BINARY</td>
<td>Fixed-length binary</td>
</tr>
<tr>
<td>X'3E4'</td>
<td>996</td>
<td>DECFLOAT(16) and DECFLOAT(34)</td>
<td>Long-format decimal floating point and extended-format decimal floating point</td>
</tr>
<tr>
<td>X'990'</td>
<td>2448</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>Timestamp with time zone</td>
</tr>
</tbody>
</table>

Columns containing DATE, TIME, TIMESTAMP, or TIMESTAMP WITH TIME ZONE data types are always exported in ISO format.

### Data records

Data records are in fixed block (FB) format and contain the data to be exported.

The maximum allowable length of a data record is 7,000 bytes. The length of a data record is the sum of the widths of the data types that are included in the record. Use the following table to calculate the widths of each data type.

**Table 12. Widths of data records.** Calculate the width of a particular data record by adding the number of bytes in each column.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Null indicator</th>
<th>Length field</th>
<th>SO/SI</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 for long-format values; 16 for extended-format values</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 for long-format values; 16 for extended-format values</td>
<td>2</td>
<td></td>
<td></td>
<td>Length in header (LIH)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LIH</td>
</tr>
</tbody>
</table>
Table 12. Widths of data records (continued). Calculate the width of a particular data record by adding the number of bytes in each column.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Null indicator</th>
<th>Length field</th>
<th>SO/SI</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating point</td>
<td>2</td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>8 for long-format values; 16 for extended-format values</td>
<td>2</td>
<td></td>
<td></td>
<td>8 for long-format values; 16 for extended-format values</td>
</tr>
<tr>
<td>Decimal floating point</td>
<td>2</td>
<td></td>
<td></td>
<td>8 for long-format values; 16 for extended-format values</td>
</tr>
<tr>
<td>Big integer</td>
<td>2</td>
<td></td>
<td></td>
<td>LIH</td>
</tr>
<tr>
<td>8 for long-format values; 16 for extended-format values</td>
<td>2</td>
<td></td>
<td></td>
<td>LIH</td>
</tr>
<tr>
<td>Integer</td>
<td>2</td>
<td></td>
<td></td>
<td>LIH</td>
</tr>
<tr>
<td>8 for long-format values; 16 for extended-format values</td>
<td>2</td>
<td></td>
<td></td>
<td>LIH</td>
</tr>
<tr>
<td>Small integer</td>
<td>2</td>
<td></td>
<td></td>
<td>LIH</td>
</tr>
<tr>
<td>8 for long-format values; 16 for extended-format values</td>
<td>2</td>
<td></td>
<td></td>
<td>LIH</td>
</tr>
<tr>
<td>Time</td>
<td>2</td>
<td></td>
<td></td>
<td>LIH</td>
</tr>
<tr>
<td>8 for long-format values; 16 for extended-format values</td>
<td>2</td>
<td></td>
<td></td>
<td>LIH</td>
</tr>
<tr>
<td>Timestamp</td>
<td>2</td>
<td></td>
<td>LIH</td>
<td>LIH</td>
</tr>
<tr>
<td>8 for long-format values; 16 for extended-format values</td>
<td>2</td>
<td></td>
<td>LIH</td>
<td>LIH</td>
</tr>
<tr>
<td>Timestamp with time zone</td>
<td>2</td>
<td>2</td>
<td>LIH</td>
<td>LIH</td>
</tr>
<tr>
<td>Decimal</td>
<td>2</td>
<td></td>
<td></td>
<td>(Precision + 2) // 2</td>
</tr>
<tr>
<td>Graphic</td>
<td>2</td>
<td>2</td>
<td>LIH</td>
<td>(LIH × 2)</td>
</tr>
<tr>
<td>Variable character</td>
<td>2</td>
<td>2</td>
<td>LIH</td>
<td>(LIH × 2)</td>
</tr>
<tr>
<td>Variable graphic</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>(2 × LIH)</td>
</tr>
<tr>
<td>Binary</td>
<td>2</td>
<td></td>
<td>LIH</td>
<td>LIH</td>
</tr>
<tr>
<td>Variable binary</td>
<td>2</td>
<td>2</td>
<td>LIH</td>
<td>LIH</td>
</tr>
</tbody>
</table>

**Important:** The LIH is the width given in the header record for that column.

Every data record has 2 bytes of indicator information, which can have the values and meanings shown in this table:
Table 13. Hex values that show the validity of data records

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0000'</td>
<td>The column contains valid data.</td>
</tr>
<tr>
<td>X'FFFF' or</td>
<td>The column contains a null value; any data in the column is meaningless.</td>
</tr>
<tr>
<td>X'FFE'</td>
<td></td>
</tr>
</tbody>
</table>

Interpreting a data object in QMF format

You can calculate the length of the header record when you have the length of the data records.

For example, suppose that you export the following data from the Q.STAFF table:

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

In this example, each data record is 23 bytes long. Table 10 on page 70 shows that the first 12 bytes contain level and number information.

Calculate the widths of each column as shown in this table:

Table 14. Calculating column widths

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

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:

\[(12 + (24 \times 3) = 84)\]

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

This sample header shows the header from the report and its hexadecimal representation. 40 is the hexadecimal code for a blank character. The reversed-type numbers are associated with notes that follow the sample.
The next sample shows the data from the report and the hexadecimal representation of that data. For information about what the byte positions mean, see Table 10 on page 70.

**Figure 13. Sample header records for an exported data object in QMF format.** 40 is the hexadecimal code for a blank character.

The next sample shows the data from the report and the hexadecimal representation of that data. For information about what the byte positions mean, see Table 10 on page 70.

**Figure 14. Sample data records for an exported data object in QMF format**

1 REL 3.0
   Object format level: 3.0
   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 the first column of the first data record: 10

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

10 X'FFFF'
   Indicator information: column contains a null value
Exporting data or tables in IXF format

When you use the EXPORT command to export a data object or table with the DATAFORMAT=IXF option, the data is exported in the Integrated Exchange Format (IXF). QMF supports a subset of IXF.

The TSO data set or CICS data queue that contains the exported data or table consists of the following records:

- Header record (H)
- Table record (T)
- Column records (C)
- Data records (D)

The exported 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 the records is displayed in the following graphic:

![Figure 15. Arrangement of records in an exported data set or CICS data queue (IXF format)](image)

**Header record (H)**

A header record (which is mandatory) in IXF format is the first record in the data set or CICS data queue.

The header record is a 42-byte record that contains character data. The format of the H record is shown in this table:

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Information and type</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Header record indicator (H)</td>
</tr>
<tr>
<td>02-04</td>
<td>TSO data set or CICS data queue identifier</td>
</tr>
<tr>
<td>05-08</td>
<td>IXF version; the version can be one of the following types:</td>
</tr>
<tr>
<td></td>
<td>• 0000, which supports data or tables that contain short column names (18 or fewer characters) and no TIMESTAMP WITH TIME ZONE columns</td>
</tr>
<tr>
<td></td>
<td>• 0001, which supports data or tables that contain at least one long column name (19 characters or more) and no TIMESTAMP WITH TIME ZONE columns</td>
</tr>
<tr>
<td></td>
<td>• 0002, which supports data or tables that contain short column names (18 or fewer characters) and at least one TIMESTAMP WITH TIME ZONE column</td>
</tr>
<tr>
<td></td>
<td>• 0003, which supports data or tables that contain at least one long column name (19 characters or more) and at least one TIMESTAMP WITH TIME ZONE column</td>
</tr>
</tbody>
</table>
Table 15. Parts of a header record in an IXF data set or data queue that contain an exported data object or database table (continued)

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Information and type</th>
</tr>
</thead>
<tbody>
<tr>
<td>09-14</td>
<td>Originating product name (QMF)</td>
</tr>
<tr>
<td>15-20</td>
<td>Originating product release level (VBR2M0)</td>
</tr>
<tr>
<td>21-28</td>
<td>Date that the data set or CICS data queue was created, in the form YYYYMMDD</td>
</tr>
<tr>
<td>29-34</td>
<td>Time that the data set or CICS data queue was created, in the form HHMMSS</td>
</tr>
<tr>
<td>35-39</td>
<td>The number of records that precede the first D (data) record in the data set or CICS data queue; this value is a five-digit numeric value expressed in character form</td>
</tr>
<tr>
<td>40</td>
<td>DBCS indicator that tells whether DBCS data is a possibility; Y or N</td>
</tr>
<tr>
<td>41-42</td>
<td>Blanks</td>
</tr>
</tbody>
</table>

Table record (T)

A table record in IXF format follows the header record. Each data set or data queue that contains an object in IXF format must have a T record. A table record contains table and data information about the object that was exported with the EXPORT TABLE or EXPORT DATA command.

Names of tables that are exported in IXF format are truncated at 18 characters and owner names are truncated at 8 characters. If you run a query and export the resulting DATA object, the table record contains a blank owner and name.

The format of a T record is shown in this table:

Table 16. Parts of a table record in an IXF data set or data queue that contain an exported data object or database table

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Information and type</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Table record indicator (T)</td>
</tr>
<tr>
<td>02-03</td>
<td>Data name length (18)</td>
</tr>
<tr>
<td>04-21</td>
<td>Name of the table from which data is retrieved; left-aligned, padded with blanks to the right</td>
</tr>
<tr>
<td></td>
<td>The entire 18-byte field is blank if the table does not have a name.</td>
</tr>
<tr>
<td>22-29</td>
<td>Data name qualifier; name of the owner of the database table from which the data is retrieved</td>
</tr>
<tr>
<td></td>
<td>The 8-byte field is blank if the table does not have an owner.</td>
</tr>
<tr>
<td>30-41</td>
<td>Data source (database)</td>
</tr>
<tr>
<td>42</td>
<td>Convention used to describe data: C for columnar data</td>
</tr>
<tr>
<td>43</td>
<td>Data format: C for character (OUTPUTMODE=CHARACTER); M for machine (OUTPUTMODE=BINARY)</td>
</tr>
<tr>
<td>44</td>
<td>Data location: I for internal</td>
</tr>
</tbody>
</table>
Table 16. Parts of a table record in an IXF data set or data queue that contain an exported data object or database table (continued)

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Information and type</th>
</tr>
</thead>
<tbody>
<tr>
<td>45-49</td>
<td>Count of column (C) records: a numeric value in character form that specifies the number of C records before the first data (D) record</td>
</tr>
<tr>
<td>50-51</td>
<td>Reserved</td>
</tr>
<tr>
<td>52-81</td>
<td>Blanks</td>
</tr>
</tbody>
</table>

Column records (C)

A column record in IXF format describes the data characteristics of the column. There is a column record for each column in the table.

When a column name longer than 18 characters exists, the column name field in the column record must be increased from 18 to 30 characters. The IXF version number that is used in the header record depends not only on whether there is a column name longer than 18 characters, but also whether the data contains a TIMESTAMP WITH TIME ZONE column. Table 15 on page 75 shows the IXF version numbers used in each case.

The following table shows the format of a column record for data or tables that contain no TIMESTAMP WITH TIME ZONE columns (IXF version numbers 0000 or 0001). For information about data or tables that contain one or more TIMESTAMP WITH TIME ZONE columns (IXF version numbers 0002 or 0003), see Table 18 on page 78.

Table 17. IXF format with no TIMESTAMP WITH TIME ZONE columns (IXF version numbers 0000 or 0001)

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Information and type</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Column record indicator (C)</td>
</tr>
<tr>
<td>02-03</td>
<td>Column name length</td>
</tr>
<tr>
<td>04-21</td>
<td>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-aligned, and padded with blanks to the right if necessary.</td>
</tr>
<tr>
<td>22 or 34</td>
<td>Indicator that tells if nulls are allowed; Y or N</td>
</tr>
<tr>
<td>23 or 35</td>
<td>Column-selected indicator (Y)</td>
</tr>
<tr>
<td>24 or 36</td>
<td>Key column indicator (Y)</td>
</tr>
<tr>
<td>25 or 37</td>
<td>Data class (R)</td>
</tr>
<tr>
<td>26-28 or 38-40</td>
<td>Data type (For data type codes, see Table 20 on page 81)</td>
</tr>
<tr>
<td>29-33 or 41-45</td>
<td>Code page</td>
</tr>
<tr>
<td>34-38 or 46-50</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
Table 17. IXF format with no TIMESTAMP WITH TIME ZONE columns (IXF version numbers 0000 or 0001) (continued)

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Information and type</th>
</tr>
</thead>
<tbody>
<tr>
<td>39-43 or 51-55</td>
<td>Column data length; a decimal value in character form</td>
</tr>
<tr>
<td></td>
<td>If the data type is DECIMAL, the first 3 bytes represent data precision and the next 2 bytes represent the scale. If the data type is BIGINT, INTEGER, or SMALLINT, this field is blank because the length is inherent in the data type.</td>
</tr>
<tr>
<td>44-49 or 56-61</td>
<td>Starting position of column data; a decimal value in character form</td>
</tr>
<tr>
<td></td>
<td>This value reflects the offset of the data for a column from the start of the data record.</td>
</tr>
<tr>
<td></td>
<td>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 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 for starting position of the actual data. (The first data position is position 1, not position 0.)</td>
</tr>
<tr>
<td>50-79 or 62-91</td>
<td>Column label information, if available (if not available, these byte positions contain blanks)</td>
</tr>
<tr>
<td>78</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

If the data or table contains one or more TIMESTAMP WITH TIME ZONE columns, the format of the column record is as follows:

Table 18. IXF format with one or more TIMESTAMP WITH TIME ZONE columns (IXF version numbers 0002 or 0003)

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Information and type</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Column record indicator (C)</td>
</tr>
<tr>
<td>02-03</td>
<td>Column name length</td>
</tr>
<tr>
<td>04-21</td>
<td>Column name, as obtained from the database or generated by QMF (in the case where the column did not originally have a name)</td>
</tr>
<tr>
<td></td>
<td>The name is left-aligned, and padded with blanks to the right if necessary.</td>
</tr>
<tr>
<td>22 or 34</td>
<td>Indicator that tells if nulls are allowed; Y or N</td>
</tr>
<tr>
<td>23 or 35</td>
<td>Column-selected indicator (Y)</td>
</tr>
<tr>
<td>24 or 36</td>
<td>Key column indicator (Y)</td>
</tr>
<tr>
<td>25 or 37</td>
<td>Data class (R)</td>
</tr>
<tr>
<td>26-29 or 38-41</td>
<td>Data type (see Table 20 on page 81 for data type codes)</td>
</tr>
<tr>
<td>30-34 or 42-46</td>
<td>Code page</td>
</tr>
<tr>
<td>34-38 or 47-50</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
Table 18. IXF format with one or more TIMESTAMP WITH TIME ZONE columns (IXF version numbers 0002 or 0003) (continued)

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Information and type</th>
</tr>
</thead>
<tbody>
<tr>
<td>39-43 or 51-55</td>
<td>Column data length; a decimal value in character form</td>
</tr>
<tr>
<td></td>
<td>If the data type is DECIMAL, the first 3 bytes represent data precision and the next 2 bytes represent the scale. If the data type is BIGINT, INTEGER, or SMALLINT, this field is blank because the length is inherent in the data type.</td>
</tr>
<tr>
<td>44-49 or 56-61</td>
<td>Starting position of column data; a decimal value in character form</td>
</tr>
<tr>
<td></td>
<td>This value reflects the offset of the data for a column from the start of the data record.</td>
</tr>
<tr>
<td></td>
<td>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 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 for starting position of the actual data. (The first data position is position 1, not position 0.)</td>
</tr>
<tr>
<td>50-79 or 62-91</td>
<td>Column label information, if available (if not available, these byte positions contain blanks)</td>
</tr>
<tr>
<td>80-81 or 92-93</td>
<td>Two bytes of zeros in character form (00)</td>
</tr>
</tbody>
</table>

Data records (D)

Data records in IXF format are in variable block (VB) format. There is a data record for each row in the table.

This table shows the format of a data record:

Table 19. Format of a data record in an IXF data set or data queue that contains an exported data object or table

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Information and type</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Data record indicator (D)</td>
</tr>
<tr>
<td>02-04</td>
<td>Reserved</td>
</tr>
<tr>
<td>05</td>
<td>Blank</td>
</tr>
<tr>
<td>06-end of record</td>
<td>Row data in binary or character form, depending on whether byte 43 of the table record is M (machine) or C (character)</td>
</tr>
<tr>
<td></td>
<td>Byte 6 represents the start (position 1) of row data for the first column.</td>
</tr>
</tbody>
</table>

Column data format

Data in D records for \( n \) columns is placed side by side, as shown in this figure.
For each column, the data consists of a null indicator followed by the data itself. If nulls are allowed (byte 22 or 34 of the C record has a value of Y), then bytes 44-49 or 56-61 of each C record point to the null indicator that precedes the data for that column. If nulls are not allowed (byte 22 or 34 of the C record has a value of N), then bytes 44-49 or 56-61 point 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 or 56-61 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 the data format is character, 1 byte is used for the null indicator:
- A dash (-) indicates that the data is null
- A blank indicates that the data is not null

When the data format is binary, 2 bytes are used for the null indicator:
- X'FFFF' indicates that the data is null
- X'0000' indicates that the data is not null

For more information about the null indicator, see the examples that help you interpret an object in IXF format, below.

**Format of column data by data type**

The length and format of IXF column data in D records can differ depending on the OUTPUTMODE.

This table shows the length and format of column data in D records for each data type for both character and binary export formats. In the table, IXFCLENG refers to the contents of bytes 39-43 or 51-55 of a C record (length of column data).
Table 20. Format of IXF column data by data type

<table>
<thead>
<tr>
<th>Data type code</th>
<th>Data type</th>
<th>Data length information (when OUTPUTMODE = CHARACTER)</th>
<th>Data length information (when OUTPUTMODE = BINAR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>384</td>
<td>DATE</td>
<td>The value in IXFCLENG is not significant for this data type. 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 zeros cannot be omitted. The allowable range for ( yyyy ) is 0001-9999; for ( mm ) it is 01-12. The ( dd ) range depends on the month. For example, the following value specifies a date of February 28, 2002: 2002-02-28</td>
<td>Same as character format</td>
</tr>
<tr>
<td>388</td>
<td>TIME</td>
<td>The value in IXFCLENG is not significant for this data type. 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 zeros cannot 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</td>
<td>Same as character format</td>
</tr>
</tbody>
</table>
Table 20. Format of IXF column data by data type (continued)

<table>
<thead>
<tr>
<th>Data type code</th>
<th>Data type</th>
<th>Data length information (when OUTPUTMODE = CHARACTER)</th>
<th>Data length information (when OUTPUTMODE = BINARY)</th>
</tr>
</thead>
</table>
| 392            | TIMESTAMP       | The length of TIMESTAMP(0) is 19; the length of TIMESTAMP(n) is 20+n, where n is a number from 1 to 12. For example, a column defined as TIMESTAMP(12) has a length of 32.  
The format is:  
\[ yyyy-mm-dd-hh.mm.ss.nnnnnnnnnnn \]  
where \( yyyy \) is the year, \( mm \) is the month, \( dd \) is the day, \( hh \) is hour in 24-hour format, \( mm \) is minutes, \( ss \) is seconds, and \( nnnnnnnnnnn \) is fractional seconds. Valid ranges for year, month, day, hour, minutes, and seconds are the same as the DATE and TIME data types.  
Examples:  
\[ 2010-12-31-23.59.59.999999999999 \] (the last fractional second in 2010)  
\[ 2010-01-01-00.00.00.000000000001 \] (the first fractional second in 2010)  
\[ 24.00.00.000000000000 \] is valid for the time portion of a timestamp. | Same as character format |
| 448            | VARCHAR         | IXFCLENG is the maximum length of the 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 0-32704 and for the count field it is 0-\( n \).) The number of characters indicated by the count field are valid; the rest are meaningless. For example, if IXFCLENG=00010, the data takes this format:  
\[ 00005JONESxxxxx \]  
In this format, each \( x \) is a blank character (X’40’). | IXFCLENG is the maximum length of the character string. The data length consists of \( n \) bytes indicated by IXFCLENG preceded by a 2-byte binary count field. (The allowable range for \( n \) is 1-32704 and for the count field it is 0-\( n \).) The number of characters indicated by the count field are valid; the rest are meaningless. For example, if IXFCLENG=00010, the data format is as follows:  
\[ nnJONESxxxxx \]  
In this format, \( nn=x'0005' \) and each \( x \) is a blank character (X’40’). |
### Table 20. Format of IXF column data by data type (continued)

<table>
<thead>
<tr>
<th>Data type code</th>
<th>Data type</th>
<th>Data length information (when OUTPUTMODE = CHARACTER)</th>
<th>Data length information (when OUTPUTMODE = BINARY)</th>
</tr>
</thead>
</table>
| 452            | CHAR           | IXFCLENG is the length of the character string. Data length is indicated by $n$ bytes of IXFCLENG. (The allowable range for $n$ is 1-254). For example, if IXFCLENG=00005, the data takes this format: JONES  
In this format, JONES is the 5-byte character string pointed to by bytes 44-49 or 56-61 of the C record. | Same as character format                                              |
| 464            | VARGRAPHIC     | IXFCLENG is the maximum number of double-byte characters ($2\times n$ 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. (The allowable range for $n$ is 1-16352 and for the count field the allowable range is 0-$n$.) For example, if IXFCLENG = 00006, the data takes this format: 00003aZZYXXixxxxxx  
In this format, the letter 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 1-16352, and for the count field it is 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. For example, if IXFCLENG = 00008, the data takes this format: nnZZYXXixxxxxx  
In this format, $nn$=X'0004' and each x is a blank character (X'40'). |                                                                 |
| 468            | GRAPHIC        | IXFCLENG is the number of double-byte characters ($2\times n$ bytes). Data length is $2\times n$ bytes plus a shift out (X'0E') immediately preceding the data, and a shift-in (X'0F') immediately following the data. For example, if IXFCLENG=00005, the data takes this format: 0ZZYYXXWWVVi  
In this format, the letter 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  
For example, if IXFCLENG=00005, the data format is as follows: ZZYYXXWWVV |
Table 20. Format of IXF column data by data type (continued)

<table>
<thead>
<tr>
<th>Data type code</th>
<th>Data type</th>
<th>Data length information (when OUTPUTMODE = CHARACTER)</th>
<th>Data length information (when OUTPUTMODE = BINARY)</th>
</tr>
</thead>
<tbody>
<tr>
<td>480</td>
<td>FLOAT</td>
<td>The value in IXFCLENG is 4 for single-precision values and 8 for double-precision values. The length and format of the data is determined by the precision of the value. Single-precision values consist of up to 14 characters in the following format: \texttt{sn.\ldots nnnnnnnnnnE\ldots} In this format: \begin{itemize} \item \texttt{s} is an optional sign (a plus, a minus, or, in the case of the first position, a blank if the number is positive). \item \texttt{n} represents the digits of the significand, which can be up to 8 digits. A decimal point must be in the second position of the significand. \item \texttt{E} signifies the beginning of the exponent. \item \texttt{e} represents the digits of the exponent, which can be one or two digits. \end{itemize} The value must be in the range +/-5.4E-79 to +/-7.2E+75. Examples: \begin{itemize} \item -1.2345679E+07 \item 6.2345679E-01 \item 0.0000000E+00 \end{itemize} Double-precision values consist of up to 23 characters in the following format: \texttt{sn.\ldots nnnnnnnnnnnnnnnnE\ldots} In this format: \begin{itemize} \item \texttt{s} is an optional sign (a plus, a minus, or, in the case of the first position, a blank if the number is positive). \item \texttt{n} represents the digits of the significand, which can be up to 18 digits. A decimal point must be in the second position of the significand. \item \texttt{E} signifies the beginning of the exponent. \item \texttt{e} represents the digits of the exponent, which can be 1 or 2 digits. \end{itemize} The value must be in the range +/-5.4E-79 to +/-7.2E+75. Examples: \begin{itemize} \item -1.2345678901234568E+14 \item 6.2345678901234568E-01 \item 0.0000000000000000E+00 \end{itemize}</td>
<td></td>
</tr>
<tr>
<td>Data type code</td>
<td>Data type</td>
<td>Data length information (when OUTPUTMODE = CHARACTER)</td>
<td>Data length information (when OUTPUTMODE = BINARY)</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
<td>--------------------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>484</td>
<td>DECIMAL</td>
<td>Bytes 39-43 or 51-55 of the C record represent the precision, or P (first 3 bytes), and scale, or S (next 2 bytes), of the number. The 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-aligned, with the first byte reserved for a sign, and a decimal point (the position of which is implied by S) present only if S is not equal to zero. For example, if P=005 and S=00, the data takes the following format: 12345 If P=006 and S=02, the data takes the following format: +2345.10 If P=004 and S=03, the data takes the following format: -8.515</td>
<td>Bytes 39-43 or 51-55 of the C record represent the precision, or P (first 3 bytes), and scale, or S (next 2 bytes), of the number. The allowable range for P is 0-15. S can be any value less than or equal to P. The data consists of a (P+2)/2-byte decimal value in packed decimal format. The last byte indicates the sign of the value. For example, if P=005 and S=00, the data format is as follows: X'12345' If P=006 and S=02, the data format is as follows: X'02345100'</td>
</tr>
<tr>
<td>492</td>
<td>BIGINT</td>
<td>The value in IXFCLENG is not significant for this data type. The length and format of the data is inherent in the data type. The data consists of a 20-byte character value, right-aligned, with the first character reserved for a sign. Examples: 0000000000000000033 +9223372036854775807 -9223372036854775808</td>
<td>The value in IXFCLENG is not significant. The length and format of the data is inherent in the data type. The data consists of an 8-byte binary value.</td>
</tr>
<tr>
<td>496</td>
<td>INTEGER</td>
<td>The value in IXFCLENG is not significant for this data type. The length and format of the data is inherent in the data type. The data consists of an 11-byte character value, right-aligned, with the first character reserved for a sign. Examples: 0000000013 +1187642200 -0033588727</td>
<td>The value in IXFCLENG is not significant. The length and format of the data is inherent in the data type. The data consists of a 4-byte binary value.</td>
</tr>
</tbody>
</table>
Table 20. Format of IXF column data by data type (continued)

<table>
<thead>
<tr>
<th>Data type code</th>
<th>Data type</th>
<th>Data length information (when OUTPUTMODE = CHARACTER)</th>
<th>Data length information (when OUTPUTMODE = BINARY)</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>SMALLINT</td>
<td>The value in IXFCLENG is not significant for this data type. The length and format of the data is inherent in the data type. The data consists of a 6-byte character value, right-aligned, with the first character reserved for a sign. Examples: 00023 +00763 -21311</td>
<td>The value in IXFCLENG is not significant. The length and format of the data is inherent in the data type. The data consists of a 2-byte binary value.</td>
</tr>
<tr>
<td>908</td>
<td>VARBINARY</td>
<td>Not applicable</td>
<td>Same as VARCHAR, except that: • IXFCLENG is the maximum length (number of bytes) of the binary string. • The allowable range for ( n ) is 0-32704.</td>
</tr>
<tr>
<td>912</td>
<td>BINARY</td>
<td>Not applicable</td>
<td>Same as CHAR, except that: • IXFCLENG is the length (number of bytes) of the binary string sequence. • The allowable range for ( n ) is 1-255.</td>
</tr>
<tr>
<td>Data type code</td>
<td>Data type</td>
<td>Data length information (when OUTPUTMODE = CHARACTER)</td>
<td>Data length information (when OUTPUTMODE = BINARY)</td>
</tr>
<tr>
<td>---------------</td>
<td>------------</td>
<td>-------------------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>996</td>
<td>DECFLOAT</td>
<td>The value in IXFCLENG is 16 for long-format values and 34 for extended-format values. The length and format of the data is determined by the precision of the value. Long-format values consist of up to 23 characters in the following format: sn.nnnnnnnnnnnnnnEeee In this format: • s is an optional sign (a plus, a minus, or, in the case of the first position, a blank if the number is positive). • n represents the digits of the significand, which can be up to 16 digits. A decimal point must be in the second position of the significand. • E signifies the beginning of the exponent. • e represents the digits of the exponent, which can be up to 3 digits. The value must be in the range +/-1.0E-398 to +/-1.0E+384. Examples: -1.234567890123456E+003 1.234567890123456E-015 0.000000000000000E+000</td>
<td>The value in IXFCLENG is 8 for long-format values and 16 for extended-format values. The data consists of an 8-byte decimal floating-point value for long format and a 16-byte decimal floating-point value for extended format. Extended-format values consist of up to 42 characters in the following format: sn.nnnnnnnnnnnnnnnnnnnnnnnnnnnnEeee In this format: • s is an optional sign (a plus, a minus, or, in the case of the first position, a blank if the number is positive). • n represents the digits of the significand, which can be up to 34 digits. A decimal point must be in the second position of the significand. • E signifies the beginning of the exponent. • e represents the digits of the exponent, which can be up to 4 digits. The value must be in the range +/-1.0E-6176 to +/-1.0E+6144. Examples: -1.234567890123456789012345678901234E+0033 4.321098765432109876543210987654321E-0001</td>
</tr>
</tbody>
</table>
Table 20. Format of IXF column data by data type (continued)

<table>
<thead>
<tr>
<th>Data type code</th>
<th>Data type</th>
<th>Data length information (when OUTPUTMODE = CHARACTER)</th>
<th>Data length information (when OUTPUTMODE = BINARY)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2448</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>The length is 147 for TIMESTAMP(0) WITH TIME ZONE and 148 + n for TIMESTAMP(n) WITH TIME ZONE, where n = 1-12.</td>
<td>Same as character format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The format is:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>yyyy-mm-dd-hh-mm-ss-nnnnnnnnnnnnzmth:tm</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>where yyyy-mm-dd-hh-mm-ss specifies the timestamp in the same way as for TIMESTAMP data and:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• nnnnnnnnnnnn specifies a 0-12 digit number that represents the number of fractional seconds.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• z is a plus (+) or minus (-) sign that indicates the time zone offset relative to Coordinated Universal Time UTC,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>formerly known as Greenwich Mean Time (GMT).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• th is a two-digit value that represents the time zone hours.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• tm is a two-digit value that represents the time zone minutes.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The valid range for the time zone portion of the format is from -24:00 to +24:00. To specify UTC, you can either</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>specify a time zone of -0:00 or +0:00 or replace the time zone offset and its sign with an uppercase Z.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>For example, 2010-09-30-13.08.36.123456654321-08:00 indicates a time of 1:08 P.M. and 36.123456654321 seconds on</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>September 30, 2010, in San Jose, California, in the United States. The timestamp 2010-09-30-13.08.123456654321Z indicates</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>a time of 1:08 P.M. and 36.123456654321 seconds wherever UTC is in effect.</td>
<td></td>
</tr>
</tbody>
</table>

Interpreting an object exported in IXF format

The following example helps you interpret data that is exported in IXF format.

Assume that the table shown in the example of a data object in QMF format is now exported with the IXF format (with OUTPUTMODE=CHARACTER). The table to be exported is as follows:

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>COMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>SANDERS</td>
<td>-</td>
</tr>
<tr>
<td>20</td>
<td>PERNAL</td>
<td>612.45</td>
</tr>
</tbody>
</table>

The exported 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 here:

<table>
<thead>
<tr>
<th>HIXF0000QMF</th>
<th>VAR1M020101204095600000005N</th>
<th>database</th>
<th>CC100003</th>
</tr>
</thead>
<tbody>
<tr>
<td>T18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C02ID</td>
<td>NYNR5000000000</td>
<td>000002</td>
<td>00</td>
</tr>
<tr>
<td>C04NAME</td>
<td>YNNR4480000000</td>
<td>000009000008</td>
<td>00</td>
</tr>
<tr>
<td>C04COMM</td>
<td>YNNR484400000000</td>
<td>007020000023</td>
<td>00</td>
</tr>
<tr>
<td>D</td>
<td>00010</td>
<td>000007SANDERSxx</td>
<td>-</td>
</tr>
<tr>
<td>D</td>
<td>00020</td>
<td>000006PERNALxxx</td>
<td>00612.45</td>
</tr>
</tbody>
</table>
Unprintable binary characters are shown as \( x \) characters. This figure gives more detailed information about these records.

Figure 17. Format of sample IXF records (OUTPUTMODE=CHARACTER)

Now suppose that the same table is exported with the IXF format, but with OUTPUTMODE=BINARY. The exported data set or CICS data queue consists of seven records which are shown in the table:

<table>
<thead>
<tr>
<th>Record Indicator</th>
<th>Date and Time</th>
<th>H Records preceding first D record</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIXF0000QMF</td>
<td>VAR1M020120409565000005N</td>
<td>Data format is character</td>
</tr>
</tbody>
</table>

| T18 database CCI00003 |

<table>
<thead>
<tr>
<th>C02ID</th>
<th>NYNR500000000000000002</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SMALLINT</td>
<td>00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C04NAME</th>
<th>YYNR448000000000000008</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VARCHAR</td>
<td>00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C04COMM</th>
<th>YYNR4840000000000000023</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DECIMAL 7,2</td>
<td>00</td>
</tr>
</tbody>
</table>

Zero padding used

<table>
<thead>
<tr>
<th>Posn. 1 of column data</th>
<th>Character count</th>
<th>Meaningless</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000700000000007000010</td>
<td>SANDERSxxxxxxx-</td>
<td>(-) indicates data is null</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Posn. 2 of column data</th>
<th>Character count</th>
<th>Meaningless</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000000000000000020</td>
<td>PERNALxxxxxx-</td>
<td>Meanings</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID col. data</th>
<th>NAME col. data</th>
</tr>
</thead>
<tbody>
<tr>
<td>00612.45</td>
<td>000009000008</td>
</tr>
</tbody>
</table>

Except for bytes 44-49 or 56-61 (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. The following figure contains more information.
about the records of the exported data set or CICS data queue.

Exporting data or tables in XML format

If your data or table contains an XML column or LOB data, you must use the DATAFORMAT=XML clause on the EXPORT DATA or EXPORT TABLE command. This format can also be used when the data or table to be exported does not contain an XML column or LOB data.

Restriction: QMF supports operations with XML data only when you are connected to a database release that supports the XML data type.

When you export data or tables in XML format, the data is exported to the UNIX file, TSO data set, or CICS data queue that you specify in the command. QMF uses the XML 1.0 specification (fourth edition) when importing and exporting data.
QMF uses z/OS XML parse services and z/OS Unicode conversion services when processing XML data for export or import, so these services must be configured and active.

All tags shown in the XML examples must be present before you import the XML column data. The tags must be present in the contents of the file, data set, or CICS data queue because QMF uses these tags to parse the file. When QMF encounters the <extensions> tag at the end of the file, the cursor is closed and the import is finished. Modifying or deleting this tag results in an infinite read of the data.

The data is exported as an XML document in Unicode UTF-8 format with a CCSID of 1208. The exported XML data set or CICS data queue consists of header records, records that define the result set, metadata records for each column in the data or table, and data records for each row in the exported data or table.

**Header records**

The header records in the exported XML file contain the version of XML that is used, the encoding scheme, and the style sheet that is used to format the exported XML document.

The following example shows the type of information that is included in the header records of an exported file:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!-- ?xml-stylesheet type="text/xsl" href="qmf.xslt" ? -->
```

QMF provides a style sheet that you can use to format your exported XML data. The default style sheet (with the default name qmf.xslt) is supplied as member DSQ1STSH of the QMF samples data set, QMF1210.SDSQSAP (where n is a national language identifier). Copy this default style sheet to the location of the exported file, then open the XML document to have it formatted to these specifications. If you use a different name for the style sheet, change the header in the exported file to refer to the new style sheet name.

**Records that define the result set**

The result set definition contains a namespace definition and a schema definition for the QMF schema file that is used with the XML file.

This example shows the records for the result set definition in a sample exported XML file that contains seven columns:

```xml
<ResultSet>
<Metadata>
<SourceDescription />
<ColumnsAmount>7</ColumnsAmount>

......Definitions for each column go here.
</Metadata>
<Data>

......Data for each row goes here.
</Data>
</ResultSet>
<Extensions />
</DataSet>
```
Metadata records

The column metadata in the exported XML file consists of the number of columns, column names, column labels (if applicable), data types, data lengths, whether the data is null, and the format.

An example of the metadata for a column called "ID" is shown here. The exported XML file contains one column-description block for each column.

```xml
<ColumnDescription id="1">
  <Name>ID</Name>
  <Label>ID</Label>
  <Type>smallint</Type>
  <Width>2</Width>
  <Nullable>false</Nullable>
  <Format>plain</Format>
</ColumnDescription>
```

Data records

The exported XML file contains one row-definition block for each row of exported data. Data records are in variable block spanned (VBS) format.

A `<cell>` tag identifies each column in the row by number, as shown here for the first row of the Q.STAFF sample table:

```xml
<Row id="0">
  <Cell id="1">10</Cell>
  <Cell id="2">SANDERS</Cell>
  <Cell id="3">20</Cell>
  <Cell id="4">MGR</Cell>
  <Cell id="5">7</Cell>
  <Cell id="6">99999.99</Cell>
  <Cell id="7" null="1" />
</Row>
```

When you use the DATAFORMAT=XML clause on the EXPORT DATA or EXPORT TABLE command and the data contains a column defined with the XML data type, QMF wraps the XML data in CDATA tags. The CDATA tags prevent the parser from trying to process the XML data. Here is an example of how XML data appears in an exported file.

```xml
<Data>
  <Row id="0">
    <Cell id="1">Murphy</Cell>
    <Cell id="2">1234</Cell>
    <Cell id="3">[CDATA[xml version="1.0" encoding="utf-8"]]</Cell>
  </Row>
</Data>
```

How QMF validates the XML

An XML schema document describes the structure of an XML document and defines parameters for the validity of elements and attributes within the XML document.

A default schema file is provided with QMF as member DSQ1SCM of the QMF samples data set, which is QMF1210.SDSQSAPn (where n is a national language identifier). Copy this member to the directory where the file that contains the XML document is stored. Name the schema document qmf_data.xsd, which is the name for the default schema document under QMF for Workstation and QMF for WebSphere®. You can modify the default schema file according to your business
needs for formatting XML data. If you use a name other than qmf_data.xsd or use a different schema file, change the name in any files that reference the schema document.

Exporting data or tables in CSV format
You can export data or tables in CSV (comma-separated value) format. You can then download the exported data to your workstation where the data in CSV format can be used with applications such as Microsoft Excel.

When you export data or tables in CSV format, you can use the HEADER option to specify whether to export column headings along with the data. The default is to include the column headings. If you export column headings, the value of the DSQDC_COL_LABELS global variable controls whether column labels or column names are exported. The default of DSQDC_COL_LABELS is 1, which means that column labels are exported.

Exported data is formatted as rows of column values that are separated by the column separator. If HEADER=YES is specified, one row of column names separated by a column separator precedes the rows of column values. The column separator value is determined by the user's Q.PROFILES.DECIMAL value.

Data that is exported in CSV format is encoded in the local Db2 for z/OS SBCS encoding scheme.

Rules and information for exporting and importing data objects and tables
QMF exports data and table objects to temporary storage and has rules for how it allocates that storage. QMF also has specific ways for handling import errors.

Allocation of UNIX files, TSO data sets, or CICS data queues
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 UNIX file, TSO data set, or CICS data queue remains open and allocated to QMF. QMF reads records when the user scrolls through the data.

This connection is maintained until the data object is replaced or reset, or QMF reads all the records. Then the UNIX file, TSO data set, or CICS data queue is closed and is no longer considered allocated to QMF. An application should not attempt to delete or alter a UNIX file, TSO data set, or CICS data queue allocated to QMF with an IMPORT DATA command. The application needs either to use another data source or empty the QMF temporary storage area for the data object (by using a RESET DATA command). Then the application can try to alter or delete the file, data set, or data queue it is reading.

During the execution of the IMPORT command, QMF does not lock the UNIX file, TSO data set or CICS data queue while it is being read. It does not take steps to prevent the file, data set, or queue from being altered while it is being read. If the file, data set, or queue is erased or altered before QMF finishes reading it, the results are unpredictable and can cause a system error.

An incomplete data prompt can occur when there is not enough temporary storage to retrieve the entire object to be exported. If you need extra storage, you can use the DSQSPILL parameter to specify the use of spill storage. If you are using QMF
under TSO, you can specify the DSQSPTYP parameter in addition to the DSQSPILL parameter to use extended storage for spilling data.

**Export errors**
After QMF imports data from a UNIX file, TSO data set, or CICS data queue, QMF displays the REPORT panel and a confirmation message.

If the file, data set, or data queue contains format errors, QMF does not display the REPORT panel. If there are format errors, 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.

---

**Exporting forms, reports, and prompted queries**

The form and prompted query objects are exported and imported in an encoded format that represents the object in a tabular structure. Reports are also exported in an encoded format; however, they cannot be imported.

The encoded format helps you manipulate individual parts of an object more easily. The following conditions apply 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 zeros, if necessary).
- A 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.

**Related reference:**

"Size specifications for externalized objects“ on page 130

External tables and objects have both record size and record format specifications that vary by the type of object.

---

**General format of the exported file**

The encoded format of a form, report, or prompted query consists of fixed-format header records and variable-format records that the object is comprised of.

**Header records**

Most records of exported forms, reports, or prompted queries have a variable format. However, header records have a fixed format, even though the data set or data queue that contains the records can be of variable format.

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 data set or data queue. It describes the characteristics of the object.

A header record contains the information described in this table. An asterisk indicates that the field is required for import.
Table 21. Header record information

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Information and type</th>
</tr>
</thead>
<tbody>
<tr>
<td>01*</td>
<td>Header record indicator (H)</td>
</tr>
<tr>
<td>02</td>
<td>Blank</td>
</tr>
<tr>
<td>03-05*</td>
<td>Product identifier (QMF)</td>
</tr>
<tr>
<td>06</td>
<td>Blank</td>
</tr>
<tr>
<td>07-08</td>
<td>QMF release level in which the form, report, or prompted query was exported; this number is 19 for QMF Version 12 Release 1</td>
</tr>
<tr>
<td>09</td>
<td>Blank</td>
</tr>
</tbody>
</table>
| 10*           | Type of object:  
|               | • F for form  
|               | • R for report  
|               | • T for prompted query |
| 11            | Blank |
| 12-13*        | QMF object level:  
|               | • 01 for report  
|               | • 04 for form  
|               | • 01 for prompted queries that do not contain a time period specification  
|               | • 02 for prompted queries in which at least one of the tables in the query has a time period specification  
|               | The object level denotes a change in the format of an object. Each time the format is changed in a QMF release, its object level is also changed. The object level increases only when the change in the format might create an error in your application. |
| 14            | Blank |
| 15*           | Data format of the object ("E" for encoded format used to export form, report, and prompted query 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 the object was exported (E for English) |
| 22            | Blank |
| 23*           | You can create a form, report, or prompted query outside of QMF in the appropriate format and import it into QMF. Code an R in this byte position if you want QMF to replace the object in temporary storage with the object you are importing. |
| 24            | Blank |
| 25-26         | Length of control area at the beginning of each record:  
|               | • 01 for form  
|               | • 02 for report  
|               | • 01 for prompted query |
| 27            | Blank |
| 28-29         | Length of integer length fields specified in V and T records (03) |
Table 21. Header record information (continued)

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Information and type</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>Blank</td>
</tr>
<tr>
<td>31-38</td>
<td>Date stamp in the format (yy/mm/dd)</td>
</tr>
<tr>
<td>39</td>
<td>Blank</td>
</tr>
<tr>
<td>40-44</td>
<td>Timestamp in the format (hh:mm)</td>
</tr>
<tr>
<td>45</td>
<td>Blank</td>
</tr>
<tr>
<td>51</td>
<td>Blank</td>
</tr>
</tbody>
</table>

**Related reference:**

- “Exporting a form” on page 103
  The form object contains all the information specified in all the QMF form panels. When you export a form, QMF converts to the encoded format any form panels whose values deviate from the default values.

- “Exporting a prompted query” on page 120
  An exported prompted query object contains the information displayed in the echo area of the Prompted Query primary panel.

- “Exporting a standard report” on page 113
  When QMF displays a report, you see the result of interaction between the form and the data object 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.

- “Conventions for National Language Feature information” on page 4
  Db2 QMF is available in several different languages, each of which is provided by a National Language Feature (NLF).

**Records of the exported object**

Except for header records, which are fixed-format records, all records of exported forms, reports, and prompted queries are variable-format records. Variable-format records are accepted on input.

Variable-format records have the general form shown in this figure:

![Figure 19. General form of variable-format records](image)

The contents of the control area are shown here:

Table 22. General form of variable-format records

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Record identifier (H, V, T, R, E, *, L, C)</td>
</tr>
<tr>
<td>02</td>
<td>Blank (sometimes omitted; see specific type of variable-format record)</td>
</tr>
</tbody>
</table>

The record data area is a variable-length area that contains information about that specific record. Fields in this area are separated by a delimiter (a blank character is used in these examples).
Data value records (V):

Value records of exported forms, reports, or prompted queries 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 length of the field
- The value of the field

The control area for V records is shown in this table:

*Table 23. Control area for V records*

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Value record identifier (V)</td>
</tr>
<tr>
<td>02</td>
<td>Blank (used only for reports; omitted for forms and prompted queries)</td>
</tr>
</tbody>
</table>

The record data area for V records is shown in this table:

*Table 24. Record data area for V records*

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

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

Data table description records (T):

In the encoded format, most data in an object appears in tables. These tables 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.
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 that your application program refers to the contents of tables of an exported form, report, or prompted query by using the encoding 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 control area for T records is shown in this table:

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Table record identifier (T)</td>
</tr>
<tr>
<td>02</td>
<td>Blank (used only for reports; omitted for forms and prompted queries)</td>
</tr>
</tbody>
</table>

The record data area for T records is shown in this table. The byte positions in the table are offsets that follow the end of the control area, the length of which is indicated in the header record.

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Blank</td>
</tr>
<tr>
<td>02-05</td>
<td>Table number (1001-9999)</td>
</tr>
<tr>
<td>06</td>
<td>Blank</td>
</tr>
<tr>
<td>07-09</td>
<td>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.</td>
</tr>
<tr>
<td>10</td>
<td>Blank</td>
</tr>
<tr>
<td>11-13</td>
<td>The number of columns in the record (000-999)</td>
</tr>
<tr>
<td>14</td>
<td>Blank</td>
</tr>
<tr>
<td>15-18, 24-27, ...</td>
<td>The field number for this column (repeating field)</td>
</tr>
<tr>
<td>19, 28, ...</td>
<td>Blank (repeating field)</td>
</tr>
<tr>
<td>20-22, 29-31, ...</td>
<td>The length of the data values in this column (repeating field)</td>
</tr>
</tbody>
</table>

Bytes 11-13 (number of columns) indicate how many field number/data value length pairs follow. So, the information from byte 15 on is repeated for each column.

Keep the following information about T records in mind as you export and import objects:

- 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.
• 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.

• The number of field number/length pairs is limited to the number of columns in the table, and their order is arbitrary.

• Columns with a length of zero are set to their default values when the object in the temporary storage area is updated and a warning is issued. Columns not included in this table are also handled in this way. However, with prompted queries, a default is supplied when possible. Otherwise, an error occurs.

• 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 of exported forms, reports, or prompted queries 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 control area for R records is shown in this table:

Table 27. Control area for R records

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Row record identifier (R)</td>
</tr>
<tr>
<td>02</td>
<td>Blank (used only for reports; omitted for forms and prompted queries)</td>
</tr>
</tbody>
</table>

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:

_value..._value...

In this format, value... represents the data value for this row and column and _ is the delimiter.

Keep in mind the following information as you work with R records:

• An R record must immediately follow another R record or a T record.

• The number of data values must match the description in the associated T record.

• 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. In other words, the row and column 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 of an exported form, report, or prompted query 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 that follow the E record are ignored. If an E record is not included with the file that is imported, QMF assumes that an end-of-file implies the end of the object.

**Application data record (·)**:

Application data records of exported forms, reports, or prompted queries allow application programs to associate their own data with an object in the external file. Application programs frequently use these records as comment records to further describe the object in the file.

The information that follows the asterisk is 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 upon export. However, you can use these records in the data set or CICS data queue you create. The contents of an application data record are shown in this table:

*Table 28. Contents of an application data record*

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Application data record identifier (·)</td>
</tr>
<tr>
<td>02-end of record</td>
<td>Data</td>
</tr>
</tbody>
</table>

Here is an example of an application data record that appears in an exported form:

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

**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, and 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 format of an L record is shown in the following figure.

*Figure 20. Format of an L record*

The control area for an L record is shown in this table:

*Table 29. Control area for an L record*

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Value record identifier (L)</td>
</tr>
</tbody>
</table>
Table 29. Control area for an L record (continued)

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>02</td>
<td>Continuation indicator</td>
</tr>
<tr>
<td></td>
<td>Indicates whether the current record is continued to a data continuation record:</td>
</tr>
<tr>
<td></td>
<td>• C for continued</td>
</tr>
<tr>
<td></td>
<td>A C record immediately follows an L record marked with a continuation character in byte 2 of the control area.</td>
</tr>
<tr>
<td></td>
<td>• 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</td>
</tr>
<tr>
<td></td>
<td>When D is specified for the continuation indicator in the control area, the current record is too long to fit into a single physical record. 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 any DBCS data that is continued.</td>
</tr>
<tr>
<td></td>
<td>• Blank if not continued</td>
</tr>
</tbody>
</table>

The record data area for an L record is shown in this table. Bytes 6-13 are 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).

Table 30. Record data area for an L record

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Blank</td>
</tr>
<tr>
<td>02-04</td>
<td>Report part indicator:</td>
</tr>
<tr>
<td></td>
<td>110 = Page heading</td>
</tr>
<tr>
<td></td>
<td>120 = Page footing</td>
</tr>
<tr>
<td></td>
<td>13n = Break heading</td>
</tr>
<tr>
<td></td>
<td>(n is the break number, 1-6)</td>
</tr>
<tr>
<td></td>
<td>15n = Break footing</td>
</tr>
<tr>
<td></td>
<td>(n is the break number, 1-6)</td>
</tr>
<tr>
<td></td>
<td>170 = Column heading</td>
</tr>
<tr>
<td></td>
<td>171 = Detail heading</td>
</tr>
<tr>
<td></td>
<td>180 = Detail line</td>
</tr>
<tr>
<td></td>
<td>181 = Group summary line</td>
</tr>
<tr>
<td></td>
<td>190 = Final footing</td>
</tr>
<tr>
<td>05</td>
<td>Blank</td>
</tr>
<tr>
<td>06</td>
<td>1</td>
</tr>
<tr>
<td>07</td>
<td>Data</td>
</tr>
<tr>
<td>08</td>
<td>Text</td>
</tr>
<tr>
<td>09</td>
<td>Separator</td>
</tr>
</tbody>
</table>
Table 30. Record data area for an L record (continued)

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Column wrap</td>
</tr>
<tr>
<td></td>
<td>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 L format record means that the column data or wrapped line is continued on a following L format record.</td>
</tr>
<tr>
<td>11</td>
<td>Line wrap</td>
</tr>
<tr>
<td></td>
<td>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 L format record means that the column data or wrapped line is continued on a following L format record.</td>
</tr>
<tr>
<td>12</td>
<td>Second data line (across reports only)</td>
</tr>
<tr>
<td></td>
<td>Across reports that contain 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.</td>
</tr>
<tr>
<td>13</td>
<td>Reserved</td>
</tr>
<tr>
<td>14</td>
<td>Blank</td>
</tr>
</tbody>
</table>

The following example shows an L record for a break footing line in a report that contains text and data:

L 151 111000000 DEPARTMENT TOTALS 93,659.45

Data continuation records (C):

A C record in an exported form, report, or prompted query is used to continue a value or set of values across more than one record. It immediately follows the record that is continued. The format of a C record corresponds to the format of the original record that is continued. QMF uses C records to continue L records only.

The control area for a C record is shown in this table:

Table 31. Control area for a C record

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Value record identifier (C)</td>
</tr>
</tbody>
</table>
Table 31. Control area for a C record (continued)

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Description</th>
</tr>
</thead>
</table>
| 02            | Continuation indicator. Indicates whether the current record is continued to another C record:  
  - **C** for continued  
    A C record immediately follows an L record marked with a continuation character in byte 2 of the control area.  
  - **D** for continued with DBCS delimiters $0$ and $1$ inserted at the end of the current record and the beginning of the data portion of the next record  
    When D is specified for the continuation indicator in the control area, the current record is too long to fit into a single physical record. In the process of splitting up the record, $0$ (shift out) and $1$ (shift in) characters were added to the current and next records to preserve the integrity of any DBCS that is continued.  
  - Blank if not continued |

The record data area for a C record is shown in this table. The byte positions shown are offset from the end of the control area, the length of which is indicated in the header record.

Table 32. Record data area for a C record

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Blank</td>
</tr>
<tr>
<td>02-end</td>
<td>Value or set of values that are continued</td>
</tr>
</tbody>
</table>

Exporting a form

The form object contains all the information specified in all the QMF form panels. When you export a form, QMF converts to the encoded format any form panels whose values deviate from the default values.

The following panels are in the encoded format only if you modified the panel:  
- **FORM.BREAK**, where $n = 1$-$6$  
- **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 to see example export results

You can create a default form by running any query that creates an empty report, such as the query shown here:
SELECT JOB
FROM Q.STAFF
WHERE NAME='NO_NAME'

When QMF displays the report, enter EXPORT FORM TO DEFAULT (including the QUEUETYPE=xx parameter in CICS).

**How the exported form looks**

Your data set or CICS data queue named DEFAULT contains the information shown in this sample format of an exported form:

```
H QMF 19 F 04 E V W E R 01 03 13/01/15 16:20
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
  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
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
```
You can import your default data set or CICS data queue every time you log on by issuing the command IMPORT FORM FROM DEFAULT (including the QUEUETYPE=xx parameter in CICS) in your initial procedure.

Interpreting the header record in the exported data set or queue

The following example shows a header record for a QMF form:

```
H QMF 19 F 04 E V W E R 01 03 13/01/15 16:20
```

This table explains the example.

<table>
<thead>
<tr>
<th>Value from example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>H QMF 19 F</td>
<td>This record is a QMF form header record for Version 12.1.</td>
</tr>
<tr>
<td>04</td>
<td>The structure of the form is at object level 4.</td>
</tr>
<tr>
<td>E</td>
<td>The format is encoded (the format used for exported forms, reports, and prompted queries).</td>
</tr>
<tr>
<td>V</td>
<td>The exported form does not contain any errors or warnings.</td>
</tr>
<tr>
<td>W</td>
<td>The file contains the entire form.</td>
</tr>
<tr>
<td>E</td>
<td>The national language in use when the object was exported is English.</td>
</tr>
<tr>
<td>R</td>
<td>When importing, the object in temporary storage is replaced.</td>
</tr>
<tr>
<td>01</td>
<td>The length of the control area is 1 byte.</td>
</tr>
<tr>
<td>03</td>
<td>The length of integer length fields is 3 bytes.</td>
</tr>
<tr>
<td>13/01/15</td>
<td>The date stamp specifies January 15, 2013.</td>
</tr>
<tr>
<td>16:20</td>
<td>The timestamp specifies a time of 4:20 PM.</td>
</tr>
</tbody>
</table>

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

Interpreting the records of the exported form

"How the exported form looks" on page 104 shows an example of an exported form. The exported form contains V, T, and R records whose associated codes have special meanings to help you interpret the exported result. This table explains each field and code in the exported 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 that follow field 3080 describe.
Table 34. Table and field numbers for an exported FORM object

<table>
<thead>
<tr>
<th>Table or field number</th>
<th>Record type</th>
<th>Description</th>
<th>Form panel</th>
</tr>
</thead>
<tbody>
<tr>
<td>1110</td>
<td>T</td>
<td>Column headings table</td>
<td>FORM.COLUMNS</td>
</tr>
<tr>
<td>1112</td>
<td>R</td>
<td>Column data type</td>
<td>FORM.COLUMNS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The column data type is not displayed on the form panels but is associated</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>with the form in its external format.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The column data type is not required when a form is imported. If it is</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>missing during import, CICS provides default data type information from the</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>edit codes. (For more information, see “Importing a form object” on page</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>109.)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>During export, the column data type QMF provides is based on the specified</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>edit code. For edit codes U, V, M, or invalid edit codes, QMF specifies the</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>data type keyword UNKNOWN. <a href="#">Table 35 on page 108</a> shows the data type</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>keywords that QMF uses.</td>
<td></td>
</tr>
<tr>
<td>1113</td>
<td>R</td>
<td>Column heading</td>
<td>FORM.COLUMNS</td>
</tr>
<tr>
<td>1114</td>
<td>R</td>
<td>Column usage code</td>
<td>FORM.COLUMNS</td>
</tr>
<tr>
<td>1115</td>
<td>R</td>
<td>Column indentation</td>
<td>FORM.COLUMNS</td>
</tr>
<tr>
<td>1116</td>
<td>R</td>
<td>Column width</td>
<td>FORM.COLUMNS</td>
</tr>
<tr>
<td>1117</td>
<td>R</td>
<td>Column edit code</td>
<td>FORM.COLUMNS</td>
</tr>
<tr>
<td>1118</td>
<td>R</td>
<td>Column sequence</td>
<td>FORM.COLUMNS</td>
</tr>
<tr>
<td>1119</td>
<td>R</td>
<td>Column heading alignment</td>
<td>FORM.COLUMNS</td>
</tr>
<tr>
<td>1120</td>
<td>R</td>
<td>Column data alignment</td>
<td>FORM.COLUMNS</td>
</tr>
<tr>
<td>1121</td>
<td>R</td>
<td>Column definition</td>
<td>FORM.COLUMNS</td>
</tr>
<tr>
<td>1122</td>
<td>R</td>
<td>Pass nulls on column definition</td>
<td>FORM.COLUMNS</td>
</tr>
<tr>
<td>1180</td>
<td>T</td>
<td>Summary calculations table</td>
<td>FORM.CALC</td>
</tr>
<tr>
<td>1182</td>
<td>R</td>
<td>Calculation identification number</td>
<td>FORM.CALC</td>
</tr>
<tr>
<td>1183</td>
<td>R</td>
<td>Summary calculation expression</td>
<td>FORM.CALC</td>
</tr>
<tr>
<td>1184</td>
<td>R</td>
<td>Summary calculation width</td>
<td>FORM.CALC</td>
</tr>
<tr>
<td>1185</td>
<td>R</td>
<td>Summary calculation edit code</td>
<td>FORM.CALC</td>
</tr>
<tr>
<td>1186</td>
<td>R</td>
<td>Pass nulls on calculation</td>
<td>FORM.CALC</td>
</tr>
<tr>
<td>1201</td>
<td>V</td>
<td>Blank lines before heading</td>
<td>FORM.PAGE</td>
</tr>
<tr>
<td>1202</td>
<td>V</td>
<td>Blank lines after heading</td>
<td>FORM.PAGE</td>
</tr>
<tr>
<td>1210</td>
<td>T</td>
<td>Page heading table</td>
<td>FORM.PAGE</td>
</tr>
<tr>
<td>1212</td>
<td>R</td>
<td>Page heading line number</td>
<td>FORM.PAGE</td>
</tr>
<tr>
<td>1213</td>
<td>R</td>
<td>Page heading alignment</td>
<td>FORM.PAGE</td>
</tr>
</tbody>
</table>
### Table 34. Table and field numbers for an exported FORM object (continued)

<table>
<thead>
<tr>
<th>Table or field number</th>
<th>Record type</th>
<th>Description</th>
<th>Form panel</th>
</tr>
</thead>
<tbody>
<tr>
<td>1214</td>
<td>R</td>
<td>Page heading text</td>
<td>FORM.PAGE</td>
</tr>
<tr>
<td>1301</td>
<td>V</td>
<td>Blank lines before footing</td>
<td>FORM.PAGE</td>
</tr>
<tr>
<td>1302</td>
<td>V</td>
<td>Blank lines after footing</td>
<td>FORM.PAGE</td>
</tr>
<tr>
<td>1310</td>
<td>T</td>
<td>Page footing table</td>
<td>FORM.PAGE</td>
</tr>
<tr>
<td>1312</td>
<td>R</td>
<td>Page footing line number</td>
<td>FORM.PAGE</td>
</tr>
<tr>
<td>1313</td>
<td>R</td>
<td>Page footing alignment</td>
<td>FORM.PAGE</td>
</tr>
<tr>
<td>1314</td>
<td>R</td>
<td>Page footing text</td>
<td>FORM.PAGE</td>
</tr>
<tr>
<td>1401</td>
<td>V</td>
<td>New page for final text</td>
<td>FORM.FINAL</td>
</tr>
<tr>
<td>1402</td>
<td>V</td>
<td>Final summary line number</td>
<td>FORM.FINAL</td>
</tr>
<tr>
<td>1403</td>
<td>V</td>
<td>Blank lines before final text</td>
<td>FORM.FINAL</td>
</tr>
<tr>
<td>1410</td>
<td>T</td>
<td>Final text table</td>
<td>FORM.FINAL</td>
</tr>
<tr>
<td>1412</td>
<td>R</td>
<td>Final text line number</td>
<td>FORM.FINAL</td>
</tr>
<tr>
<td>1413</td>
<td>R</td>
<td>Final text alignment</td>
<td>FORM.FINAL</td>
</tr>
<tr>
<td>1414</td>
<td>R</td>
<td>Final text</td>
<td>FORM.FINAL</td>
</tr>
<tr>
<td>1501</td>
<td>V</td>
<td>Detail line spacing</td>
<td>FORM.OPTIONS</td>
</tr>
<tr>
<td>1502</td>
<td>V</td>
<td>Outlining for break columns</td>
<td>FORM.OPTIONS</td>
</tr>
<tr>
<td>1503</td>
<td>V</td>
<td>Default break text</td>
<td>FORM.OPTIONS</td>
</tr>
<tr>
<td>1504</td>
<td>V</td>
<td>Function name in column heading for grouping</td>
<td>FORM.OPTIONS</td>
</tr>
<tr>
<td>1505</td>
<td>V</td>
<td>Column-wrapped lines kept on a page</td>
<td>FORM.OPTIONS</td>
</tr>
<tr>
<td>1506</td>
<td>V</td>
<td>Across-summary column</td>
<td>FORM.OPTIONS</td>
</tr>
<tr>
<td>1507</td>
<td>V</td>
<td>Separators for column heading</td>
<td>FORM.OPTIONS</td>
</tr>
<tr>
<td>1508</td>
<td>V</td>
<td>Separators for break summary</td>
<td>FORM.OPTIONS</td>
</tr>
<tr>
<td>1509</td>
<td>V</td>
<td>Separators for across heading</td>
<td>FORM.OPTIONS</td>
</tr>
<tr>
<td>1510</td>
<td>V</td>
<td>Separators for final summary</td>
<td>FORM.OPTIONS</td>
</tr>
<tr>
<td>1511</td>
<td>V</td>
<td>Width of wrapped report lines</td>
<td>FORM.OPTIONS</td>
</tr>
<tr>
<td>1512</td>
<td>V</td>
<td>Page renumbering at breaks</td>
<td>FORM.OPTIONS</td>
</tr>
<tr>
<td>1513</td>
<td>V</td>
<td>Width of break or final text</td>
<td>FORM.OPTIONS</td>
</tr>
<tr>
<td>1514</td>
<td>V</td>
<td>Column reordering</td>
<td>FORM.OPTIONS</td>
</tr>
<tr>
<td>1515</td>
<td>V</td>
<td>Fixed columns</td>
<td>FORM.OPTIONS</td>
</tr>
<tr>
<td>2790</td>
<td>V</td>
<td>Detail variation number</td>
<td>FORM.DETAIL</td>
</tr>
<tr>
<td>2791</td>
<td>V</td>
<td>Detail variation selection</td>
<td>FORM.DETAIL</td>
</tr>
<tr>
<td>2805</td>
<td>V</td>
<td>Include column heading</td>
<td>FORM.DETAIL</td>
</tr>
<tr>
<td>2810</td>
<td>T</td>
<td>Detail heading table</td>
<td>FORM.DETAIL</td>
</tr>
<tr>
<td>2812</td>
<td>R</td>
<td>Detail heading text line</td>
<td>FORM.DETAIL</td>
</tr>
<tr>
<td>2813</td>
<td>R</td>
<td>Detail heading alignment</td>
<td>FORM.DETAIL</td>
</tr>
<tr>
<td>2814</td>
<td>R</td>
<td>Detail heading text</td>
<td>FORM.DETAIL</td>
</tr>
<tr>
<td>2901</td>
<td>V</td>
<td>New page for detail text</td>
<td>FORM.DETAIL</td>
</tr>
<tr>
<td>2902</td>
<td>V</td>
<td>Line number of column data</td>
<td>FORM.DETAIL</td>
</tr>
</tbody>
</table>
Table 34. Table and field numbers for an exported FORM object (continued)

<table>
<thead>
<tr>
<th>Table or field number</th>
<th>Record type</th>
<th>Description</th>
<th>Form panel</th>
</tr>
</thead>
<tbody>
<tr>
<td>2904</td>
<td>V</td>
<td>Number of lines to skip after detail text</td>
<td>FORM.DETAIL</td>
</tr>
<tr>
<td>2906</td>
<td>V</td>
<td>Repeat detail heading</td>
<td>FORM.DETAIL</td>
</tr>
<tr>
<td>2907</td>
<td>V</td>
<td>Number of detail text lines to keep together</td>
<td>FORM.DETAIL</td>
</tr>
<tr>
<td>2910</td>
<td>T</td>
<td>Detail text table</td>
<td>FORM.DETAIL</td>
</tr>
<tr>
<td>2912</td>
<td>R</td>
<td>Detail text line number</td>
<td>FORM.DETAIL</td>
</tr>
<tr>
<td>2913</td>
<td>R</td>
<td>Detail text alignment</td>
<td>FORM.DETAIL</td>
</tr>
<tr>
<td>2914</td>
<td>R</td>
<td>Detail text</td>
<td>FORM.DETAIL</td>
</tr>
<tr>
<td>3080</td>
<td>V</td>
<td>Break panel number</td>
<td>FORM.BREAKn</td>
</tr>
<tr>
<td>3101</td>
<td>V</td>
<td>New page for break heading</td>
<td>FORM.BREAKn</td>
</tr>
<tr>
<td>3102</td>
<td>V</td>
<td>Repeat break heading</td>
<td>FORM.BREAKn</td>
</tr>
<tr>
<td>3103</td>
<td>V</td>
<td>Number of lines to skip before break heading</td>
<td>FORM.BREAKn</td>
</tr>
<tr>
<td>3104</td>
<td>V</td>
<td>Number of lines to skip after break heading</td>
<td>FORM.BREAKn</td>
</tr>
<tr>
<td>3110</td>
<td>T</td>
<td>Break heading text table</td>
<td>FORM.BREAKn</td>
</tr>
<tr>
<td>3112</td>
<td>R</td>
<td>Break heading line number</td>
<td>FORM.BREAKn</td>
</tr>
<tr>
<td>3113</td>
<td>R</td>
<td>Break heading alignment</td>
<td>FORM.BREAKn</td>
</tr>
<tr>
<td>3114</td>
<td>R</td>
<td>Break heading text</td>
<td>FORM.BREAKn</td>
</tr>
<tr>
<td>3201</td>
<td>V</td>
<td>New page for break text</td>
<td>FORM.BREAKn</td>
</tr>
<tr>
<td>3202</td>
<td>V</td>
<td>Break text summary line</td>
<td>FORM.BREAKn</td>
</tr>
<tr>
<td>3203</td>
<td>V</td>
<td>Number of lines to skip before break text</td>
<td>FORM.BREAKn</td>
</tr>
<tr>
<td>3204</td>
<td>V</td>
<td>Number of lines to skip after break text</td>
<td>FORM.BREAKn</td>
</tr>
<tr>
<td>3210</td>
<td>T</td>
<td>Break text table</td>
<td>FORM.BREAKn</td>
</tr>
<tr>
<td>3212</td>
<td>R</td>
<td>Break text line</td>
<td>FORM.BREAKn</td>
</tr>
<tr>
<td>3213</td>
<td>R</td>
<td>Break text alignment</td>
<td>FORM.BREAKn</td>
</tr>
<tr>
<td>3214</td>
<td>R</td>
<td>Break text</td>
<td>FORM.BREAKn</td>
</tr>
<tr>
<td>3310</td>
<td>T</td>
<td>Conditions table</td>
<td>FORM.CONDITIONS</td>
</tr>
<tr>
<td>3312</td>
<td>R</td>
<td>Condition identification number</td>
<td>FORM.CONDITIONS</td>
</tr>
<tr>
<td>3313</td>
<td>R</td>
<td>Conditional expression</td>
<td>FORM.CONDITIONS</td>
</tr>
<tr>
<td>3314</td>
<td>R</td>
<td>Pass nulls on conditions panel</td>
<td>FORM.CONDITIONS</td>
</tr>
</tbody>
</table>

The following table shows the data-type keywords QMF generates for the edit codes specified on the form. In this table, $x$ represents the number of decimal places to be displayed, where $x$ is an integer from 0 to 99.

Table 35. Data type keywords generated for edit codes specified on the QMF form panels

<table>
<thead>
<tr>
<th>Edit code specified</th>
<th>Data type keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>C, CW, CT, CD$x$</td>
<td>CHAR</td>
</tr>
</tbody>
</table>
Table 35. Data type keywords generated for edit codes specified on the QMF form panels (continued)

<table>
<thead>
<tr>
<th>Edit code specified</th>
<th>Data type keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>B, BW, X, XW</td>
<td>BINAR</td>
</tr>
<tr>
<td>G, GW</td>
<td>GRAPHIC</td>
</tr>
<tr>
<td>E, D, I, J, K, L, P, EZ, DZ, IZ, JZ, KZ, LZ, PZ, DZC, Dx, Ix, Jx, Kx, Lx, Px</td>
<td>NUMERIC</td>
</tr>
<tr>
<td>Any edit code that starts with the characters TD</td>
<td>DATE</td>
</tr>
<tr>
<td>Any edit code that starts with the characters TT</td>
<td>TIME</td>
</tr>
<tr>
<td>TSI</td>
<td>TIMEST</td>
</tr>
<tr>
<td>TSZ</td>
<td>TSTMPTZ</td>
</tr>
<tr>
<td>M</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>U, V</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>Invalid edit codes entered</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

When you export a form, QMF exports only those variation panels with values that were changed from the default. Therefore, the total number of variations in the external form can be fewer than what is 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.

**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, character, graphic, or unknown data types) in the R records for the COLUMNS table
  If your site supports date/time data types, data type values DATE, TIME, TIMEST, and TSTMPTZ 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. The T record must also include a numeric count of the number of rows in the encoded format (an * row count is not allowed).

If the entire COLUMNS table is 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 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 and the national language identifier in the H record is an E, QMF translates the reserved words. QMF translates the reserved words into your current session’s language. Examples of reserved words are values in the USAGE column in FORM.COLUMNS

- 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.

This table contains information about values for the field that contains the data type of the column. In addition to the data type values shown here, there is an UNKNOWN data type keyword that QMF uses in response to a U, V, or invalid edit code.

<table>
<thead>
<tr>
<th>Data type as it appears in externalized form</th>
<th>Code in decimal</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>384</td>
<td>DATE</td>
<td>Date</td>
</tr>
<tr>
<td>TIME</td>
<td>388</td>
<td>TIME</td>
<td>Time</td>
</tr>
<tr>
<td>TIMEST</td>
<td>392</td>
<td>TIMESTAMP</td>
<td>Timestamp</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>496</td>
<td>INTEGER</td>
<td>Integer</td>
</tr>
<tr>
<td>500</td>
<td>SMALLINT</td>
<td>Small integer</td>
<td></td>
</tr>
<tr>
<td>492</td>
<td>BIGINT</td>
<td>Big integer</td>
<td></td>
</tr>
<tr>
<td>484</td>
<td>DECIMAL</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>480</td>
<td>FLOAT</td>
<td>Floating point</td>
<td></td>
</tr>
<tr>
<td>996</td>
<td>DECFLOAT</td>
<td>Decimal floating point</td>
<td></td>
</tr>
<tr>
<td>CHAR</td>
<td>448</td>
<td>VARCHAR</td>
<td>Varying-length character</td>
</tr>
<tr>
<td>452</td>
<td>CHAR</td>
<td>Fixed-length character</td>
<td></td>
</tr>
<tr>
<td>456</td>
<td>LONG VARCHAR</td>
<td>Long varying-length character</td>
<td></td>
</tr>
<tr>
<td>904</td>
<td>ROWID</td>
<td>Row identifier</td>
<td></td>
</tr>
<tr>
<td>GRAPHIC</td>
<td>464</td>
<td>VARGRAPHIC</td>
<td>Varying-length graphic</td>
</tr>
<tr>
<td>468</td>
<td>GRAPHIC</td>
<td>Fixed-length graphic</td>
<td></td>
</tr>
<tr>
<td>472</td>
<td>LONG VARGRAPHIC</td>
<td>Long varying-length graphic</td>
<td></td>
</tr>
</tbody>
</table>
Table 36. Values for the field that contains the data type of the column (continued)

<table>
<thead>
<tr>
<th>Data type as it appears in externalized form</th>
<th>Code in decimal</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY</td>
<td>908 912</td>
<td>VARBINARY BINARY</td>
<td>Varying-length binary Fixed-length binary</td>
</tr>
<tr>
<td>TSTMPSTZ</td>
<td>2448</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>Timestamp with time zone</td>
</tr>
</tbody>
</table>

- Detecting errors during import
  If QMF detects an error in the format of the form file during import, the import function ends. QMF issues a message describing the error and its location in the file.
  If an error is encountered in the header record and a form 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
    - The specified length field does not match the length of the data 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 as follows:
- Fix one problem at a time.
- Set the TRACE option of the profile to L2 (by using the command SET PROFILE (TRACE=L2) and run the IMPORT FORM command. The L2 tracing option traces messages and commands at the highest level of detail. This option allows you to see all the message text related to the IMPORT command.

The following command displays the message text associated with a particular message code:
```
HELP DSQnmmm
```

where `nmmm` is a 5-character, unique message code.

Related reference:
- “Conventions for National Language Feature information” on page 4
- Db2 QMF is available in several different languages, each of which is provided by a National Language Feature (NLF).
Considerations for QMF form objects in applications

Some tips might help you create and use a QMF form in an application.

When using a QMF form in an application, keep in mind the following points:

Creating a form data set or CICS data queue outside of QMF

If you create a form outside of QMF (not with EXPORT FORM), it is not necessary to have a complete form object to import it successfully into QMF. You need only 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 is imported.

When you create your own form data set or CICS data queue, it does not have to be exactly like the data set or 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. However, you can use an asterisk (*) signifying that the data value is delimited by the end of the record when you import a form.

QMF keeps the excess lines if an R record count in an imported form is less than the number of default lines it allocated for the associated area in the default form.

Checking the object level in the header record

The object level in the header record of a data set or queue that contains a form indicates 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.) Make sure that your application properly interprets the contents of the data set or data queue that contains the form. Check that the object level represents the format upon which your application is based.

Using application data records

The application data records can be useful in your application program. Use application data records to include your own comments within a data set or CICS data queue for a form object. You can place the records anywhere in the data set or CICS data queue following the header record. When QMF reads such a record, it ignores all data in the record that follows the * character. The record, therefore, has no effect on the import process.

Restrictions for using forms in CICS

REXX is not available under QMF for 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. REXX calculations, conditional row formatting, and column definitions are not available to QMF for CICS users.

Related concepts:

“Importing forms and prompted queries” on page 128
Be aware of the rules for importing a form or prompted query.

Related reference:

“Exporting a form” on page 103
The form object contains all the information specified in all the QMF form panels. When you export a form, QMF converts to the encoded format any form panels whose values deviate from the default values.
Application data records of exported forms, reports, or prompted queries allow application programs to associate their own data with an object in the external file. Application programs frequently use these records as comment records to further describe the object in the file.

Most records of exported forms, reports, or prompted queries have a variable format. However, header records have a fixed format, even though the data set or data queue that contains the records can be of variable format.

Exporting a standard report

When QMF displays a report, you see the result of interaction between the form and the data object 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 an application
- Modify the appearance of the report for printing or redisplay by the application

A sample report (before export)

This sample shows a tabular report with a level 1 break.

For a list of the field numbers, see “Interpreting the report header record in the exported data set or queue” on page 115.
Figure 21. A tabular QMF report before exporting

How the exported report looks

Here is the format of the exported report from the sample tabular report.

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

DEPT JOB AVERAGE 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

COMPANY NAME
REPORT 17
```

Figure 21. A tabular QMF report before exporting

How the exported report looks

Here is the format of the exported report from the sample tabular 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, or L, records
- Data continuation, or C, records

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 data set or CICS data queue. This data set or CICS data queue contains only the formatted data without any layout information.

**Interpreting the report header record in the exported data set or queue**

The following example shows a header record for a QMF report:

```
H QMF 19 R 01 E V W E R 02 03 13/01/15 16:20
```

This table explains this example.

*Table 37. Example of a header record for a report*

<table>
<thead>
<tr>
<th>Value from example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>H QMF 19 R</td>
<td>This record is a QMF report header record for Version 12.1.</td>
</tr>
<tr>
<td>01</td>
<td>The structure of the report is at object level 1.</td>
</tr>
<tr>
<td>E</td>
<td>The format is encoded (the format used for exported forms, reports, and prompted queries).</td>
</tr>
<tr>
<td>V</td>
<td>The exported report does not contain any errors or warnings.</td>
</tr>
<tr>
<td>W</td>
<td>The file contains the entire report.</td>
</tr>
<tr>
<td>E</td>
<td>English was the national language in use when the object was exported.</td>
</tr>
<tr>
<td>R</td>
<td>This indicator is ignored.</td>
</tr>
<tr>
<td>02</td>
<td>The length of the control area is 2 bytes.</td>
</tr>
<tr>
<td>03</td>
<td>The length of integer length fields is 3 bytes.</td>
</tr>
<tr>
<td>13/01/15</td>
<td>The date stamp specifies January 15, 2013.</td>
</tr>
<tr>
<td>16:20</td>
<td>The timestamp specifies a time of 4:20 PM.</td>
</tr>
</tbody>
</table>
Interpreting the records of the exported report

This table shows the table numbers for T records and field numbers for V records of the exported report shown in “How the exported report looks” on page 114.

Table 38. Table and field numbers for an exported report

<table>
<thead>
<tr>
<th>Table or field number</th>
<th>Record type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>V</td>
<td>Profile DECIMAL option</td>
</tr>
<tr>
<td>1002</td>
<td>V</td>
<td>Length of L record control area + fixed area</td>
</tr>
<tr>
<td>1010</td>
<td>T</td>
<td>Formatted report table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For each formatted data column in the report:</td>
</tr>
<tr>
<td>1012</td>
<td>T</td>
<td>For all usage codes except OMIT</td>
</tr>
<tr>
<td>1013</td>
<td>T</td>
<td>Edit code by which data is formatted</td>
</tr>
<tr>
<td>1014</td>
<td>T</td>
<td>Starting position for field that contains formatted data (including indent area)</td>
</tr>
<tr>
<td>1015</td>
<td>T</td>
<td>Starting position for field that contains formatted data (excluding indent area)</td>
</tr>
<tr>
<td>1016</td>
<td>T</td>
<td>Ending position for field that contains formatted data</td>
</tr>
<tr>
<td>1017</td>
<td>T</td>
<td>Number of relative physical report line within logical report line in which formatted column value appears</td>
</tr>
</tbody>
</table>

When working with table and field numbers in an exported report, note the following points:

• Position 1 of the report line immediately follows the L-record fixed area.
• 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.
• Continuation records are written for the report object when the maximum record length would otherwise be exceeded.

Related reference:

“Exporting an across-style report” on page 119
Exported across-style reports include fields not found in standard exported reports

“Data continuation records (C)” on page 102
A C record in an exported form, report, or prompted query is used to continue a value or set of values across more than one record. It immediately follows the record that is continued. The format of a C record corresponds to the format of the original record that is continued. QMF uses C records to continue L records only.

“Report line records (L)” on page 100
Each formatted line in a report is described by an L record. There is one L record for each line in the report.
Exporting a report in HTML format

When you export a report in HTML format, QMF places the necessary HTML tags before and after the body of your report. You can then place the report on a web server and display it in an HTML-compliant web browser.

This sample illustrates the HTML coding that QMF places around the report. 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 documentation.

```html
<HTML>
<HEAD>
<TITLE>
Report
</TITLE>
</HEAD>
<BODY>
<pre>
J & H SUPPLY COMPANY
AVERAGE SALARY (DEPTS 10, 15, 20)
REPORT 17

DEPT JOB          AVERAGE 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>
```

This table briefly explains this HTML coding:

**Table 39. HTML tags used in exported HTML reports**

<table>
<thead>
<tr>
<th>Tag set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;HTML&gt;&lt;/HTML&gt;</td>
<td>These tags define the file as an HTML document.</td>
</tr>
<tr>
<td>&lt;HEAD&gt;&lt;/HEAD&gt;</td>
<td>These tags mark the boundaries of the document header.</td>
</tr>
</tbody>
</table>
Table 39. HTML tags used in exported HTML reports (continued)

<table>
<thead>
<tr>
<th>Tag set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;TITLE&gt;&lt;/TITLE&gt;</td>
<td>QMF inserts the word &quot;Report&quot; between these tags. Content between these tags is included in the HTML document title. Placemen of the title is browser- and platform-dependent. These tags are placed within the header.</td>
</tr>
<tr>
<td>&lt;BODY&gt;&lt;/BODY&gt;</td>
<td>These tags follow the header and contain the body of the document. Report output is placed in the body of the document.</td>
</tr>
<tr>
<td>&lt;PRE&gt;&lt;/PRE&gt;</td>
<td>The 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.</td>
</tr>
</tbody>
</table>

Exporting a report without control information

When you export a report, QMF places control information around the report by default.

For example an exported report might have the following format:

```
H QMF 17 R 01 E V W E R 02 03 14/03/05 11:07
V 1001 006 PERIOD
V 1002 003 016
T 1010 005 006 1013 005 1014 006 1015 006 1016 006 1017 006 1012 008
R L 000001 000003 000010 000001
R C 000011 000013 000026 000001
R L 000027 000029 000035 000001
R C 000036 000038 000047 000001
R C 000048 000050 000062 000001
L 110 10000000
L 110 10000000
L 170 10000000 DEPTNUMB DEPTNAME MANAGER DIVISION LOCATION
L 170 10000000 10 HEAD OFFICE 160 CORPORATE NEW YORK
L 180 11000000 15 NEW ENGLAND 50 EASTERN BOSTON
L 180 11000000 20 MID ATLANTIC 10 EASTERN WASHINGTON
L 180 11000000 38 SOUTH ATLANTIC 30 EASTERN ATLANTA
L 180 11000000 42 GREAT LAKES 100 MIDWEST CHICAGO
L 180 11000000 51 PLAINS 140 MIDWEST DALLAS
L 180 11000000 66 PACIFIC 270 WESTERN SAN FRANCISCO
L 180 11000000 84 MOUNTAIN 290 WESTERN DENVER
L 120 10000000
L 120 10000000
E
```

Figure 22. Sample exported report with control information.

If you specify the DATAFORMAT=TEXT option on your EXPORT REPORT command, you can export reports without the control information, as in the following example.
Exporting an across-style report

Exported across-style reports include fields not found in standard exported reports.

This sample shows an exported across-style report.

The following encoded format is the result of exporting the sample across-style report.

H QMF 19 R 01 E V W E R 02 03 13/01/15 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

Figure 23. Sample report exported without control information

Exporting an across-style report

Exported across-style reports include fields not found in standard exported reports.

This sample shows an exported across-style report.

The following encoded format is the result of exporting the sample across-style report.

H QMF 19 R 01 E V W E R 02 03 13/01/15 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
Table 38 on page 116 explains field numbers that are common to both standard reports and across-style reports. The following table shows the additional fields that you see in the exported across-style reports.

### Table 40. Field numbers for exported across-style report

<table>
<thead>
<tr>
<th>Field number</th>
<th>Record type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>V</td>
<td>Edit code by which across value is formatted</td>
</tr>
<tr>
<td>2002</td>
<td>V</td>
<td>Number of data lines per across group</td>
</tr>
<tr>
<td>2003</td>
<td>V</td>
<td>Indicates whether the across summary column exists</td>
</tr>
<tr>
<td>2010</td>
<td>T</td>
<td>Across report table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For each across value:</td>
</tr>
<tr>
<td>2012</td>
<td>T</td>
<td>Starting position for formatted across value (the across value appears in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the column heading lines)</td>
</tr>
<tr>
<td>2013</td>
<td>T</td>
<td>Ending position for formatted across value</td>
</tr>
<tr>
<td>2014</td>
<td>T</td>
<td>Starting position for the set of report columns associated with this across</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value, including preceding indent area</td>
</tr>
</tbody>
</table>

For aggregated columns in an across report, fields 1014, 1015, and 1016 describe the relative starting and ending positions of the field within the set of aggregated columns of an across value. (See field 2014 in the table.)

**Exporting a prompted query**

An exported prompted query object contains the information displayed in the echo area of the Prompted Query primary panel.

**A sample query (before export)**

A data set or data queue that contains an exported prompted query can be imported into the QMF in two ways. The data set or data queue can either be imported into the QMF temporary storage area or directly into the database. When you import a prompted query, QMF checks whether the incoming query is consistent with the data in the database. For example, if the prompted query that
is 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.

This example shows sample echo text that appears on the Prompted Query primary panel before exporting.

```
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
```

How the exported query looks

This example shows the format of the exported prompted query.

```
H QMF 19 T 01 E V W E R 01 03 13/01/15 16:20
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
```
Interpreting the header record in the exported data set or queue

This following table shows the meaning of this header record in the exported prompted query shown in “How the exported query looks” on page 121.

<table>
<thead>
<tr>
<th>Value from example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>H QMF 19 T</td>
<td>This prompted query header record specifies QMF Version 12.1.</td>
</tr>
<tr>
<td>01</td>
<td>The structure of the prompted query is at object level 1. If the exported query object contains a period specification, object level 2 is specified instead.</td>
</tr>
<tr>
<td>E</td>
<td>The format is encoded (the format used for exported forms, reports, and prompted queries).</td>
</tr>
<tr>
<td>V</td>
<td>The exported prompted query does not contain any errors or warnings.</td>
</tr>
<tr>
<td>W</td>
<td>The file contains the entire prompted query.</td>
</tr>
<tr>
<td>E</td>
<td>English was the national language in use when the object was exported.</td>
</tr>
<tr>
<td>R</td>
<td>When importing, the object in the temporary storage area is replaced.</td>
</tr>
<tr>
<td>01</td>
<td>The length of the control area is 1 byte.</td>
</tr>
<tr>
<td>03</td>
<td>The length of integer length fields is 3 bytes.</td>
</tr>
<tr>
<td>13/01/15</td>
<td>The date stamp specifies January 15, 2013.</td>
</tr>
<tr>
<td>16:20</td>
<td>The timestamp specifies a time of 4:20 PM.</td>
</tr>
</tbody>
</table>

See “How the exported query looks” on page 121 for a complete example of the Prompted Query encoded format.

Interpreting the records of the exported prompted query

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 an H record followed by the T record of the encoded table. If no tables are specified, an empty query is imported. Join conditions are not required unless more than one table is selected.
Table 42. Table and field numbers for an exported prompted query object

<table>
<thead>
<tr>
<th>Record type</th>
<th>Table number</th>
<th>Field number</th>
<th>Field description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>1110</td>
<td>-</td>
<td>Table definitions table</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The T record in this section of the exported prompted query in “How the exported query looks” on page 121 identifies this section as the portion that contains the table names involved in the query: T 1110 003 002 1112 001 1113 050</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>‘003’ refers to 3 tables, while ‘002’ refers to 2 field numbers (1112 and 1113). If the exported query object contains a period specification, a value of ‘003’ is used to indicate 3 field numbers (1112, 1113, and 1114) instead.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Each T record is followed by R records and, in this example, the R records identify the tables involved in the prompted query join: R A Q.STAFF R B Q.ORG R C Q.STAFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This portion of the exported file corresponds to the following part of the prompted query shown in “A sample query (before export)” on page 120: Tables: Q.STAFF(A) Q.ORG(B) Q.STAFF(C)</td>
</tr>
<tr>
<td></td>
<td>1112</td>
<td></td>
<td>Table ID (valid table IDs are A-Z, and #,$,@)</td>
</tr>
<tr>
<td></td>
<td>1113</td>
<td></td>
<td>Table name (maximum of 280 characters)</td>
</tr>
<tr>
<td></td>
<td>1114</td>
<td></td>
<td>Period specification (maximum of 560 characters). This field number is included only if the exported query object contains a period specification.</td>
</tr>
</tbody>
</table>
Table 42. Table and field numbers for an exported prompted query object (continued)

<table>
<thead>
<tr>
<th>Record type</th>
<th>Table number</th>
<th>Field number</th>
<th>Field description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>1150</td>
<td>-</td>
<td>Join conditions table</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The T record in this section of the exported prompted query in “How the exported query looks” on page 121 identifies this section as the portion that contains the join conditions involved in the query. Each T record is followed by R records that identify which tables will be joined:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>T 1150 002 002 1152 020 1153 020</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R A.DEPT B.DEPTNUMB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R A.ID C.ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This portion of the sample exported query corresponds to the following part of the sample prompted query shown in “A sample query (before export)” on page 120:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Join Tables: A.DEPT And B.DEPTNUMB And A.ID And C.ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1152</td>
<td>Column 1 name: Short length (22) Expanded length (34)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1153</td>
<td>Column 2 name: Short length (22) Expanded length (34)</td>
</tr>
<tr>
<td>T</td>
<td>1210</td>
<td>-</td>
<td>Columns table</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The T record in this section of the exported prompted query in “How the exported query looks” on page 121 identifies this section as the portion that contains the column names involved in the query. Each T record is followed by R records that identify the column names. The section appears as follows in the exported query:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>T 1210 007 002 1212 001 1213 255</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R C A.ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R C A.DEPT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R C A.JOB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R C A.SALARY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R C B.DEPTNUMB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R C C.SALARY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R C C.SALARY+A.COMM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This section of the exported query corresponds to the following section of the sample query shown in “A sample query (before export)” on page 120:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Columns: A.ID A.DEPT A.JOB A.SALARY DEPTNUMB C.SALARY C.SALARY+A.COMM</td>
</tr>
</tbody>
</table>
Table 42. Table and field numbers for an exported prompted query object (continued)

<table>
<thead>
<tr>
<th>Record type</th>
<th>Table number</th>
<th>Field number</th>
<th>Field description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1212</td>
<td>Column type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• C=column</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• E=expression</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• S=summary function with expression</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• F=summary function with only a column</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1213</td>
<td>Column name, expression, or summary function: Short length (255) Expanded length (560)</td>
</tr>
<tr>
<td>T</td>
<td>1310</td>
<td>-</td>
<td>Row selection conditions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The T record in this section of the exported prompted query in &quot;How the exported query looks&quot; on page 121 identifies this section of the exported query as the portion that contains the query conditions. Each T record is followed by R records that characterize each condition. The section appears as follows in the exported prompted query:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>T 1310 009 003 1312 001 1313 008 1314 255</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R 1 C A.SALARY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R 2 IS GT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R 3 10000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R 4 I</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R 1 C A.DEPT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R 2 IS EQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R 3 84</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R 3 96</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R 4 A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This section of the exported query corresponds to the following section of the query shown in &quot;A sample query (before export)&quot; on page 120:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Row Conditions:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If A.SALARY Is Greater Than 10000 And A.DEPT Is Equal To 84 or 96</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1312</td>
<td>Entry type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1 - left of operator</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2 - operator</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 3 - right of operator</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 4 - connector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1313</td>
<td>For entry type '1', identifies column type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• C=column</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• E=expression</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• S=summary function</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• F=summary function (column name only specified)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For entry type '2', identifies the verb:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• IS for 'is' (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• ISN for 'is not'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For entry type '3' (not used)</td>
</tr>
</tbody>
</table>
Table 42. Table and field numbers for an exported prompted query object (continued)

<table>
<thead>
<tr>
<th>Record type</th>
<th>Table number</th>
<th>Field number</th>
<th>Field description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>For entry type ‘4’, identifies a connector:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• O for ‘or’</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• A for ‘and’ (default)</td>
</tr>
<tr>
<td></td>
<td>1314</td>
<td></td>
<td>For entry type ‘1’ this field is a column name, expression, or summary function: Short length (255) Expanded length (560)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1410</td>
<td>Sort conditions table</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The T record in this section of the exported prompted query in “How the exported query looks” on page 121 identifies this section as the portion that contains the sort conditions for the query. Each T record is followed by R records that characterize each sort condition. This section appears as follows in the exported prompted query:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>T 1410 001 002 1412 001 1413 255 R D C.SALARY+A.COMM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This section of the exported query corresponds to the following section of the sample query in “A sample query (before export)” on page 120:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sort:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Descending by C.SALARY+A.COMM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1412</td>
<td>Sort direction:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• A for ‘ascending’</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• D for ‘descending’</td>
</tr>
<tr>
<td></td>
<td>1413</td>
<td></td>
<td>Column: Short length (255) Expanded length (560)</td>
</tr>
</tbody>
</table>

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)
Table 42. Table and field numbers for an exported prompted query object (continued)

<table>
<thead>
<tr>
<th>Record type</th>
<th>Table number</th>
<th>Field number</th>
<th>Field description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>1501</td>
<td></td>
<td>Treatment of duplicate rows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• K for ‘keep’</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• D for ‘discard’</td>
</tr>
</tbody>
</table>

For example, the following line in the sample exported prompted query in “How the exported query looks” on page 121 shows that the record length of the value K is 1 ("001"). The line also shows that the user who built the query specified to keep duplicate rows ("K"):  
V 1501 001 K

This section of the exported query corresponds to the following section of the sample query shown in “A sample query (before export)” on page 120:

Duplicate Rows:  
Keep duplicate rows

The meaning of values for fields 1313 and 1314 depends on the sequence number indicated in field number 1312 in table number 1310.

Related reference:
“Header records” on page 94

Most records of exported forms, reports, or prompted queries have a variable format. However, header records have a fixed format, even though the data set or data queue that contains the records can be of variable format.

“Table row records (R)” on page 99

R records of exported forms, reports, or prompted queries 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.

Ensuring that the exported prompted query has a valid format

Importing a prompted query object that your application modified is subject to certain rules.

If you want to import a prompted query object that your application modified, be aware of the following conditions:

• When a prompted query file is imported, the incoming records must be in this specific order after the header (H) record:
  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 (table number 1310) must be in order within each condition according to the entry type sequence number (field number 1312). In other words, the records must be in the same order in which row data appears in the Prompted Query echo area.
The remaining records can be in any order.

- The Table definitions table (T record 1110) 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 that precedes 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 pre-allocated 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. The codes used when exporting a prompted query are described in the list of table and field numbers for an exported prompted query object.

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

### Importing forms and prompted queries

Be aware of the rules for importing a form or prompted query.

When you import a form or prompted query:

- The file can consist of variable-length or fixed-length records.
- 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, with the following exception: the last data field in a record does not need to be followed by a delimiter because the end-of-record acts like a delimiter. (The examples in this information use the blank character as the delimiter.)
- If QMF encounters a duplicate data value or table while importing, 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 cannot be changed after the first COLUMNS table is processed.
- Table numbers, field numbers, and numeric lengths can contain leading zeros or leading blanks. However, trailing blanks (except for the blank delimiter) are not allowed; fields must be right-aligned.
- When * is used instead of a length or count, it must be left-aligned 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 the value supplied is longer than the field, it is truncated.
If the record is shorter than its fixed-format length, those fields left unspecified are assumed to be blank.

**Related reference:**

“Exporting a form” on page 103

The form object contains all the information specified in all the QMF form panels. When you export a form, QMF converts to the encoded format any form panels whose values deviate from the default values.

“Size specifications for externalized objects” on page 130

External tables and objects have both record size and record format specifications that vary by the type of object.

---

**Procedures and SQL queries**

The format of the TSO data set or CICS data queue that represents these objects is the simplest of all the formats. Each record in the data set or data queue is essentially an image of a line as it is displayed on the screen (a fixed-length record of 79 bytes).

Although each line of these objects is 79 bytes, the logical record length (LRECL) for new and existing data sets can be 79 - 32,760 bytes. If you export to a new data set, the LRECL is the value that is specified by global variable DSQEC_DSLRECL1. If the LRECL is greater than 79, QMF pads each object record with blanks during export.

Here is an example of a simple SQL query:

```
SQL query
SELECT *
FROM Q.STAFF
```

*Figure 24. A simple SQL query*

This example shows the query in its externalized format:

```
SELECT *
FROM Q.STAFF
```

Because of the simplicity of the record format, creating or editing an SQL query or procedure outside of QMF is straightforward. An SQL query or procedure consists of fixed-length data in columns 1 - 79. Any data in columns 80 - 32,760 is ignored during import. When you import the resulting data set or data queue, your query or procedure is in the QMF temporary storage area, ready to be run.

---

**Exported form-based charts and QBE queries**

You can export form-based charts and Query-by-Example (QBE) query objects for processing outside of the QMF environment.

**Exported form-based charts**

A form-based chart cannot be saved as a QMF object in the database or retrieved from the database. You cannot import form-based charts into QMF.
When you export a chart in QMF, it converts the data from the report to Graphics Data Format (GDF). GDF, a GDDM format, is an existing standard for data interchange. You can print the exported chart data by using GDDM utilities, or include it in documents.

You can use an exported chart object just as you would any GDF-formatted data set. For example, through the Document Composition Facility (DCF), an application can combine a QMF report (that uses a printed or exported report) with a QMF chart (that uses an exported chart) and send the formatted information to a printer.

**Exported QBE queries**

QBE query objects are exported by using a format internal to QMF. This format cannot be altered in any way.

### Size specifications for externalized objects

External tables and objects have both record size and record format specifications that vary by the type of object.

The following table contains specifications for both TSO and CICS import and export files. For CICS, record sizes are not enforced. For example, you can import an SQL query from a temporary storage queue with a record size of 32 KB and QMF truncates it to 79 bytes.

Record format is not a factor for CICS temporary storage or transient data queues. A temporary storage queue holds records without regard to their format. A transient data queue is defined to a destination control table (DCT) and ignores the record format.

You must specify a name for your data set or CICS data queue on the EXPORT or IMPORT command. Queue names have no default prefix or suffix. CICS temporary storage queue names are 8 bytes; transient data queue names are 4 bytes.

The following abbreviations are used for record formats in the table:

- FB - fixed block format
- VB - variable block format
- VBS - variable block spanned format

<table>
<thead>
<tr>
<th>Object</th>
<th>Record size</th>
<th>Record format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data or table (QMF format)</td>
<td>Maximum size: 7,000 bytes</td>
<td>Fixed length (FB)</td>
</tr>
</tbody>
</table>
Table 43. File and data set attributes (continued)

<table>
<thead>
<tr>
<th>Object</th>
<th>Record size</th>
<th>Record format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data or table (IXF format)</td>
<td>Maximum size: 32,756</td>
<td>Variable length (VB)</td>
</tr>
<tr>
<td></td>
<td>The minimum LRECL for an exported form that includes defined columns is 161 bytes.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The minimum LRECL that QMF accepts for an IXF data set or CICS data queue during import is 49 bytes.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Record size is normally the length of a row of data in the table that is being exported (including space for null indicators and DBCS delimiters) plus the length of the IXF D-type record count field (5 bytes).</td>
<td></td>
</tr>
<tr>
<td>Data or table (XML format)</td>
<td>Maximum size: 2 GB</td>
<td>Variable length (VBS)</td>
</tr>
<tr>
<td>Data or table (CSV format)</td>
<td>The maximum LRECL for exporting to new data sets is calculated based on whether YES or NO is specified for the HEADER option of the EXPORT command.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• If HEADER=YES is specified, the following formula is used:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>max(Column_Names_Total_Length, Column_Data_Value_Total_Length) + (number of columns * 3 - 1) + 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• If HEADER=NO is specified, the following formula is used:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(length of column data values) + (number of columns - 1) + (number of columns * 2) + 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For exporting to existing data sets, the LRECL of the existing data set is the maximum LRECL that can be exported.</td>
<td></td>
</tr>
<tr>
<td>Prompted query</td>
<td>Maximum: 7,290 bytes</td>
<td>Variable length (VB) on EXPORT</td>
</tr>
<tr>
<td></td>
<td>Minimum: 266 bytes on EXPORT; 41 bytes on IMPORT</td>
<td>Either fixed length (FB) or variable length (VB) on IMPORT</td>
</tr>
<tr>
<td>SQL query</td>
<td>Must be 79 - 32,760 bytes on EXPORT to new and existing data sets; can be any size on IMPORT, but is truncated to 79 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Either fixed length (FB) or variable length (VB) on EXPORT to existing data sets; fixed length (FB) on EXPORT to new data sets</td>
<td></td>
</tr>
<tr>
<td>QBE query</td>
<td>Must be 1,024 bytes</td>
<td>Variable length (VB)</td>
</tr>
<tr>
<td></td>
<td>An empty QBE query is 828 bytes.</td>
<td></td>
</tr>
<tr>
<td>Form</td>
<td>Maximum: 7,290 bytes</td>
<td>Fixed length (FB) on EXPORT</td>
</tr>
<tr>
<td></td>
<td>Minimum: 161 bytes on EXPORT; 23 bytes on IMPORT</td>
<td>Either fixed length (FB) or variable length (VB) on IMPORT</td>
</tr>
</tbody>
</table>

Chapter 9. Exporting and importing objects 131
Table 43. File and data set attributes (continued)

<table>
<thead>
<tr>
<th>Object</th>
<th>Record size</th>
<th>Record format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proc</td>
<td>Must be 79 - 32,760 bytes on EXPORT to new and existing data sets; can be any size on IMPORT, but is truncated to 79 bytes</td>
<td>Either fixed length (FB) or variable length (VB) on EXPORT to existing data sets; fixed length (FB) on EXPORT to new data sets</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Either fixed length (FB) or variable length (VB) on IMPORT</td>
</tr>
<tr>
<td>Report</td>
<td>Maximum: 7,290 bytes</td>
<td>Variable length (VB)</td>
</tr>
<tr>
<td>HTML report</td>
<td>Maximum: 32,000 bytes</td>
<td>Variable length (VB)</td>
</tr>
</tbody>
</table>

Storage considerations

When you import and export objects to CICS data queues and TSO data sets, be aware of how QMF handles storage.

CICS data queues

For objects exported to a CICS data queue, understand how QMF handles the queues.

When you export an object to a CICS data queue, keep in mind the following conditions:

- In CICS, both the IMPORT and EXPORT commands require that you specify the QUEUETYPE option. There is no default.
- When importing an object from a transient data (TD) queue in CICS, you must specify the correct object type. The queue is emptied after 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 correct object type) by using the same queue, because that queue is now empty.
- In CICS, the transient data or temporary storage (TS) queue must contain a single, completed QMF object before you issue the IMPORT command.
- If you export to a transient data queue, the queue must be open, enabled, and empty before you issue the EXPORT command.

QMF handles CICS transient data queues differently than temporary storage queues.

- Transient data queues
  QMF imports the entire transient data queue before displaying the object on the screen. This means that the contents of the entire queue must fit into your storage or spill area. You can use the DSQSPILL parameter to specify use of spill storage. There might be a delay before the object is displayed if the object is large and you are using a file for spill data.
  A CICS intrapartition transient data queue can hold up to 32 KB rows of data. An extrapartition transient data queue can be as large as it needs to be to hold the object.

- Temporary storage queues
By default, QMF reads approximately 100 rows of temporary storage before displaying them to the user. A temporary storage queue can hold up to 32 KB rows of data.

QMF uses the SUSPEND parameter on the IMPORT and EXPORT commands to allow CICS to 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 setting tells CICS to wait until the queue is available before writing the QMF object to the queue. The wait ensures that the QMF transaction does not interfere with any other jobs that are 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.

**TSO data sets**

For objects exported to a TSO data set, be sure that you configure your storage management system appropriately.

If you are using standard DASD devices, be sure that your storage management software is configured to handle dynamic allocation of extended data sets. When configuring these data sets, specify the default storage classes. When your storage management system is configured in this manner and you export an object, QMF dynamically allocates a data set. QMF uses the name specified on the EXPORT command if the data set does not exist. If you are exporting data in XML format, you could receive dynamic allocation errors if you have not properly configured your data sets. For more information about how to configure dynamic allocation of extended data sets, see the information provided with your storage management software.

If you are not using standard DASD devices, you must pre-allocate your data sets before using the EXPORT command. You can use global variables to specify the type and size of new data sets that will contain exported objects:

- Use global variable DSQEC_PO to specify the type of partitioned data set to create when you export an object to a member of a new data set. The type can be the default type for your site, a PDS data set, or a PDSE data set.
- Use global variable DSQEC_DSALLOC_DIR to specify the number of directory blocks when you export a member of a new PDS data set. The default is 20.
- Use global variable DSQEC_DSALLOC_PRI to specify the primary space allocation in tracks. The default is 15 tracks.
- Use global variable DSQEC_DSALLOC_SEC to specify the secondary space allocation in tracks. The default is 105 tracks.

**Related reference:**

“Global variables that control how commands and procedures are executed” on page 218

DSQEC global variables control how commands and procedures are executed. All of these global variables can be modified by the SET GLOBAL command.
Chapter 10. Debugging your QMF applications

In addition to error handling, QMF provides debugging facilities for your callable interface applications.

You can use the REXX trace facility through the REXX TRACE statement.

Related concepts:

Chapter 6, “Writing QMF applications that use ISPF services,” on page 31

You can bypass the QMF panels by writing applications that have their own user interfaces. You can use either the callable interface or the command interface to write applications that use ISPF.

Debugging your callable interface applications

QMF provides two trace options, L and A, and various levels of trace detail for debugging your applications.

The L option for tracing

The L option writes messages and commands to an external TSO data set or CICS data queue.

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.

You can set the L option in one of two ways:

- Issue the DISPLAY PROFILE command, and when the PROFILE is displayed, change the TRACE option to either L1 or L2.
- Issue the command:

  ```
  SET PROFILE (TRACE=x)
  ```

  In this statement, x is either L1 or L2.

Related concepts:

“Allocating the QMF trace data output” on page 136

You must allocate the QMF trace data output before you start QMF if tracing is to be used.

The A option for tracing

You can use the A option to specify a level of tracing for QMF application support services.

The A option can be A0, A1, or A2. A0 is the default and provides no A-tracing at all. A1 and A2 provide increasingly detailed results. This pattern is also 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 PROFILE command, or by entering it on the screen after you issue the DISPLAY PROFILE command. For example, you can enter the following statement immediately before you call the application you are debugging:
SET PROFILE (TRACE=L2A1)

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 for the settings A0, A1, or A2. You can use the value of DSQAO_APPL_TRACE to select the tracing you want in your application, as in the REXX application shown here:

```rexx
/* 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
end
```

*Figure 25. Structure of a sample REXX program that you can use to set tracing for application support services*

**Turning the tracing off**

To turn tracing off, use the SET PROFILE command.

If you need to turn the tracing off for any reason, issue the following command:

```
SET PROFILE (TRACE=NONE
```

This command discontinues tracing for the rest of the QMF session, but does not affect the permanent QMF profile.

**Allocating the QMF trace data output**

You must allocate the QMF trace data output before you start QMF if tracing is to be used.

You might want to reallocate the data set or data queue if the original allocation does not meet your needs.

For examples of how to allocate QMF trace data output for TSO, see the programming language specification for the language you are using.

The commands in the examples allocate a sequential trace data set or data queue that you can examine after your QMF session is over. The output consists of fixed-length, 80-character records.

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. Because trace data that exceeds the size of the queue is discarded, use CICS temporary storage only for trace data from messages or small applications.

**Related reference:**
The QMF application programming interface is available for several programming languages.

You can specify keywords on the START command.

### The QMF MESSAGE command for tracing

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 record messages, run the application with the L tracing option set to L1 or L2. Every message processed through the MESSAGE command is recorded, along with other QMF messages in the QMF trace data output. If the L tracing option is set to L2, commands are also recorded.

By placing MESSAGE commands at strategic points in your program, you can log useful information in the QMF trace file. You can examine the information either on a display device or in printed output.

The following lines show an example of how to turn on tracing and issue meaningful messages that are displayed in the trace output:

```
call dsqcix "SET PROFILE (TRACE=L2"
    :
    call dsqcix "MESSAGE (TEXT='QUERYA COMPLETED SUCCESSFULLY')"
    :
    call dsqcix "MESSAGE (TEXT='EXECB ENTERED WITH VALUE OF 7')"
    :
```

In this example, records containing the messages “QUERYA COMPLETED SUCCESSFULLY” and “EXECB ENTERED WITH A VALUE OF 7” are written to the QMF trace data output.

Because QMF messages might change from one release to the next, do not use the QMF trace data output as input to an application.

**Related concepts:**

- “Allocating the QMF trace data output” on page 136
- “The L option for tracing” on page 135

The L option writes messages and commands to an external TSO data set or CICS data queue.

---

### Errors on the START or other QMF commands

Depending on its level, the DSQCOMM might contain message text. If the START command (or any QMF command) fails, you can use this message text to troubleshoot problems.
Chapter 11. Programming language specifications for using the callable interface

The QMF application programming interface is available for several programming languages.

IBM provides information about how to assemble (or compile) and link-edit the programs and how to run them using the callable interface. IBM does not provide the REXX execs, JCL, or CLISTs in the examples, but you can copy them and alter them to suit your needs.

Assembler language interface

You can use the Assembler language with the callable interface in QMF.

Interface communications area mapping for Assembler (DSQCOMMA)

DSQCOMMA provides DSQCOMM mapping for the Assembler language; it is provided with the product.

This table shows the values for DSQCOMMA.

Table 44. Contents of the DSQCOMMA interface communications area

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQ_RETURN_CODE</td>
<td>DS F</td>
<td>Indicates the status of a QMF command after it runs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Its values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DSQ_SUCCESS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Successful execution of the request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DSQ_WARNING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Normal completion with warnings</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DSQ_FAILURE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Command did not execute correctly</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DSQ_SEVERE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Severe error; QMF session terminated</td>
</tr>
<tr>
<td>DSQ_INSTANCE_ID</td>
<td>DS F</td>
<td>Identifier established by QMF during execution of the START command</td>
</tr>
<tr>
<td>DSQ_COMM_LEVEL</td>
<td>DS CL12</td>
<td>Identifies the level of the DSQCOMM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In your application, include instructions that initialize this variable to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the value of DSQ_CURRENT_COMM_LEVEL before issuing the QMF START command</td>
</tr>
<tr>
<td>DSQ_PRODUCT</td>
<td>DS CL2</td>
<td>Identifies the IBM query product in use</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Variables that begin with DSQ_QMF specify QMF for TSO and QMF for CICS</td>
</tr>
</tbody>
</table>
Table 44. Contents of the DSQCOMMA interface communications area  (continued)

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQ_PRODUCT_RELEASE</td>
<td>DS CL2</td>
<td>Release level of QMF in use</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Variable DSQ_QMF_V11R1 specifies QMF Version 12 Release 1.</td>
</tr>
<tr>
<td>DSQ_RESERVE1</td>
<td>DS XL28</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>DSQ_MESSAGE_ID</td>
<td>DS CL8</td>
<td>Completion message ID</td>
</tr>
<tr>
<td>DSQ_Q_MESSAGE_ID</td>
<td>DS CL8</td>
<td>Query message ID</td>
</tr>
<tr>
<td>DSQ_START_PARM_ERROR</td>
<td>DS CL8</td>
<td>Name of the parameter in error when the START command failed due to a parameter error</td>
</tr>
<tr>
<td>DSQCANCEL_IND</td>
<td>DS C</td>
<td>Contains one of two values, depending on whether the user canceled the QMF session while a QMF command was running: DSQ_CANCEL_YES, DSQ_CANCEL_NO</td>
</tr>
<tr>
<td>DSQ_RESERVE2</td>
<td>DS XL23</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>DSQ_RESERVE3</td>
<td>DS XL156</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>DSQ_MESSAGE_TEXT</td>
<td>DS CL128</td>
<td>Completion message text</td>
</tr>
<tr>
<td>DSQ_Q_MESSAGE_TEXT</td>
<td>DS CL128</td>
<td>Query message text</td>
</tr>
</tbody>
</table>

Function calls for Assembler language

QMF provides one function call, DSQCIA, for Assembler-language programs. The function call has two formats: regular syntax and extended syntax.

**DSQCIA, regular syntax**

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**
- The QMF command issued on the function call; 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 call is for the QMF commands that require access to application program variables: START, TRACE, 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**
The length of the command string (CMDSTR); a FULLWORD parameter.

**CMDSTR**
The QMF command to execute; an uppercase character string of the length specified by CMDLTH.

**PNUM**
The number of command keywords or trace areas; a FULLWORD parameter.

**KLTH**
The length of each specified keyword or trace title; a FULLWORD parameter or array of FULLWORD parameters.

**KWORD**
QMF keyword, keywords, or address of trace titles; a character, array of characters, or array of addresses to trace titles whose lengths are specified by KLTH.

**VLTH**
The length of each value that is associated with the keyword or trace title; a FULLWORD parameter or array of FULLWORD parameters.

**VALUE**
The value that is associated with each keyword or the address of a value that is associated with a trace title.

Its type is specified in the VTYPE parameter and can be a character, array of characters, FULLWORD parameter, or array of FULLWORD parameters. For trace data, VTYPE must be FINT.

**VTYPE**
Data type of the contents of the VALUE parameter.

This parameter has one of two values, which are provided in the interface communications area, 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 that are specified in the VALUE field must have the data type that is specified in VTYPE.

**VL** is the Assembler VARIABLE LIST statement.

**Assembler programming example**
IBM provides a sample Assembler program for CICS and TSO. This sample program is a member of the library QMF1210.SDSQSAPn (where n is a national language identifier).

**Sample Assembler program for CICS**
IBM provides a sample Assembler program for CICS named DSQABFAC
This sample Assembler program for CICS is in the QMF1210.SDSQSAP\(n\) library (where \(n\) is a national language identifier).

This sample program for the Assembler callable interface performs the following functions:

- Starts QMF
- Sets three global variables
- Runs a query called Q1
- Prints the resulting report by using form F1
- Ends the QMF session

QMF does not supply query Q1 or form F1, but the sample program uses these objects.

```
TITLE 'Sample HLASM Query Callable Interface'
***********************************************************************
*  Sample Program: DSQABFAC                                        *
*  Assembler Version of the QMF Callable Interface for CICS          *
***********************************************************************
DSQABFAC DFHEIENT CODEREG=(12),DATAREG=(13),EIBREG=(11)
DSQABFAC AMODE 31
DSQABFAC RMODE ANY
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
ST R4,QMF1 Address of DSQCOMMA
LA R1,STARTQIL Address of START command length
ST R1,QMF2
LA R1,STARTQI Address of START command
ST R1,QMF3
LA R1,1 One Start command parameter
ST R1,NUMPARMS
LA R1,NUMPARMS Address of number of parameters
ST R1,QMF4
LA R1,STARTKYL Address of keyword lengths
ST R1,QMF5
LA R1,STARTKY Address of keywords
ST R1,QMF6
LA R1,STARTVL Address of value lengths
ST R1,QMF7
LA R1,STARTV Address of values
ST R1,QMF8
LA R1,DSQ_VARIABLE_CHAR Address of value data type
ST R1,QMF9
OI QMF9,'X'80' Set end of parameter list
LA R1,QMFPLIST Address of parameter list
CALL DSQCIA
SPACE 1
***********************************************************************
* Set numeric values into query using SET command                    *
***********************************************************************
SPACE 1
LA R1,20 Set values for SET GLOBAL command
ST R1,VVAL1
LA R1,40
ST R1,VVAL2
LA R1,84
```

Developing Db2 QMF applications
Chapter 11. Programming language specifications for using the callable interface

***********************************************************************
* Run a query                                               *
***********************************************************************
LA R1,QUERYL Addr of RUN QUERY command length
ST R1,QMF2
LA R1,QUERY Address of RUN QUERY command
ST R1,QMF3
OI QMF3,X'80' Set end of parameter list
LA R1,QMFPLIST Address of parameter list
CALL DSQCIA
SPACE 1
***********************************************************************
* Print the result of the query                             *
***********************************************************************
LA R1,REPTL Addr of PRINT Report command length
ST R1,QMF2
LA R1,REPT Address of PRINT Report command
ST R1,QMF3
OI QMF3,X'80' Set end of parameter list
LA R1,QMFPLIST Address of parameter list
CALL DSQCIA
SPACE 1
***********************************************************************
* End the query interface session                          *
***********************************************************************
LA R1,ENDQIL Address of EXIT command length
ST R1,QMF2
LA R1,ENDQI Address of EXIT command
ST R1,QMF3
OI QMF3,X'80' Set end of parameter list
LA R1,QMFPLIST Address of parameter list
CALL DSQCIA
SPACE 1
***********************************************************************
* Free Keyboard                                               *
***********************************************************************
EXEC CICS SEND CONTROL FREEKB
SPACE 1
***********************************************************************
* Return                                                     *
***********************************************************************
SPACE 1
XR R15,R15 ZERO RETURN CODE
DFHEIRET RCREG=15
**Data Areas**

**SPACE 1**

* Query Interface commands

**SPACE 1**

**STARTQI DC C'START'**

**SETG DC C'SET GLOBAL'**

**QUERY DC C'RUN QUERY Q1'**

**REPT DC C'PRINT REPORT (FORM=F1,QUEUEN=DSQP,QUEUET=TS)'**

**ENDQI DC C'EXIT'**

**SPACE 1**

**DS OF**

**STARTQIL DC AL4(L'STARTQI')**

**SETGL DC AL4(L'SETG')**

**QUERYL DC AL4(L'QUERY')**

**REPTL DC AL4(L'REPT')**

**ENDQIL DC AL4(L'ENDQI')**

**SPACE 1**

* START command keyword

**SPACE 1**

**STARTKY DC C'DSQSMODE'**

**STARTV DC C'INTERACTIVE'**

**DS OF**

**STARTKYL DC AL4(L'STARTKY')**

**STARTVL DC AL4(L'STARTV')**

**SPACE 1**

* SET GLOBAL command variable names

**SPACE 1**

**VNAME1 DC C'MYVAR01'**

**VNAME2 DC C'SHORT'**

**VNAME3 DC C'MYVAR03'**

**DS OF**

**VNAME1L DC AL4(L'VNAME1')**

**VNAME2L DC AL4(L'VNAME2')**

**VNAME3L DC AL4(L'VNAME3')**

**SPACE 1**

* SET GLOBAL command values

**SPACE 1**

**VVAL1 DC AL4(L'VVAL1')**

**VVAL2 DC AL4(L'VVAL2')**

**VVAL3 DC AL4(L'VVAL3')**

**DS OF**

**NUMPARMS DS F**

**NUMPARMS**

**SPACE 1**

* Callable interface communications definition

**DSQCOMMA**

* Equates for registers 0-15

**R0 EQU 00**

**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**

* Local variables located in CICS working storage

**DFHEIUSR DSECT**

**ORG DFHEIUSR**

**NUMPARMS DS F**

**NUMPARMS**

* QMF SET GLOBAL command values

**VVAL1 DS F**

**VVAL1**

144 Developing Db2 QMF applications
Db2 QMF is available in several different languages, each of which is provided by a National Language Feature (NLF).

Sample Assembler program for TSO
IBM provides a sample Assembler program for TSO named DSQABFA.

This sample Assembler program for TSO is in the QMF1210.SDSQSAPl library (where n is a national language identifier).

This sample program for the Assembler callable interface performs the following functions:
- Starts QMF
- Sets three global variables
- Runs a query called Q1
- Prints the resulting report by using form F1
- Ends the QMF session

QMF does not supply query Q1 or form F1, but the sample program uses these objects.
LA R4,CICOMM  ESTABLISH ACCESS TO DSQCOMM
USING DSQCOMM,R4
SPACE 1
MVC DSQ_COMM_LEVEL,DSQ_CURRENT_COMM_LEVEL
LA R1,I 1 PARAMETER
ST R1,NUMPARMS
CALL DSQCIA,
(CICOMM,        QI COMMON AREA
STARTQI,       START COMMAND LENGTH
STARTQI,       START COMMAND
NUMPARMS,      NUMBER OF KEYWORDS
STARTKYL,      KEYWORD LENGTHS
STARTKY,       KEYWORDS
STARTVL,       VALUE LENGTHS
STARTV,        VALUES
DSQ_VARIABLE_CHAR),VL VALUES ARE CHARACTERS
SPACE 1
***********************************************************************
* 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 R1,33  3 PARAMETERS
ST R1,NUMPARMS
SPACE 1
CALL DSQCIA,
(CICOMM,
SETGL,       SET GLOBAL COMMAND LENGTH
SETG,        SET GLOBAL COMMAND
NUMPARMS,    NUM OF VARIABLES TO BE SET
VNAME1L,     VARIABLE NAME LENGTHS
VNAME1,      VARIABLE NAMES
VVAL1L,      VALUE LENGTHS
VVAL1,       VALUES
DSQ_VARIABLE_FINT),VL VALUES ARE INTEGERS
SPACE 1
***********************************************************************
* Run a query                                                         *
***********************************************************************
SPACE 1
CALL DSQCIA,
(CICOMM,
QUERYL,      QUERY COMMAND LENGTH
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,ENQQL,ENQI),VL
SPACE 1
***********************************************************************
* Return                                                              *
***********************************************************************
SPACE 1
SR R15,R15  SET RETURN CODE
L R13,4(R13)
L R14,12(R13)
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'

SPACE 1
STARTKYL DC AL4(L'STARTKY)
STARTVYL DC AL4(L'STARTV')

SPACE 1
* SET GLOBAL command variable names
SPACE 1
VNAME1 DC C'MYVAR01'
VNAME2 DC C'SHORT'
VNAME3 DC C'MYVAR03'

SPACE 1
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
NUPARMS DS F                    NUMBER OF KEYWORDS

SPACE 1
* Callable interface communications area
SPACE 1
CICOMM DS CL(DSQCCOMM_LEN)

SPACE 1
SAVEAREA DS 18F
EJECT
DSQCOMMA

R0 EQU 00                EQUATES FOR REGISTERS 0-15
R1 EQU 01
R2 EQU 02
R3 EQU 03
R4 EQU 04
R5 EQU 05

Chapter 11. Programming language specifications for using the callable interface 147
DSQCOMM for Assembler

The Assembler interface communications area file is named DSQCOMMA.

The DSQCOMMA file is provided in the QMF1210.SDSQSAPn library (where n is a national language identifier). A copy of the file is shown here:

MACRO
DSQCOMM
******************************************************************************
* Callable interface - variable constants                                  *
******************************************************************************
*
* Communications Level ID
*
DSQ_CURRENT_COMM_LEVEL DC CL12'DSQL>001002<'
*
* Query Product IDs
*
DSQ_QRW DC C'01'
DSQ_QMF DC C'02'
DSQ_QM4 DC C'03'
*
* Query Product Release IDs
*
DSQ_QRW_V1R2 DC C'01'
DSQ_QRW_V1R3 DC C'02'
DSQ_QMF_V2R4 DC C'01'
DSQ_QMF_V3R1 DC C'02'
DSQ_QMF_V3R1M1 DC C'03'
DSQ_QMF_V3R2 DC C'04'
DSQ_QMF_V3R3 DC C'05'
DSQ_QMF_V6R1 DC C'06'
DSQ_QMF_V7R1 DC C'07'
DSQ_QM4_V1R1 DC C'01'
DSQ_QMF_V7R2 DC C'08'
DSQ_QMF_V8R1 DC C'09'
DSQ_QMF_V9R1 DC C'10'
DSQ_QMF_V10R1 DC C'11'
*
* Extended parameter data types
*
DSQ_VARIABLE_CHAR DC C'CHAR'
DSQ_VARIABLE_FINT DC C'FINT'
*
* Return codes
*
DSQ_SUCCESS EQU 0
DSQ_WARNING EQU 4

Related reference:
"Conventions for National Language Feature information" on page 4
Db2 QMF is available in several different languages, each of which is provided by a National Language Feature (NLF).
DSQ_FAILURE EQU 8
DSQ_SEVERE EQU 16
*
* Instance ID values
*
DSQ_CONTINUE EQU 0
*
* Cancel indicator
*
DSQ_CANCEL_YES EQU C'1'
DSQ_CANCEL_NO EQU C'0'
*
*
DSQ_INTERACTIVE EQU C'1'
DSQ_BATCH EQU C'2'
*
*
DSQ_YES EQU C'1'
DSQ_NO EQU C'2'
*
**********************************************************************
* Callable interface communications area  *
**********************************************************************
DSQCOMM DSECT
DSQ_RETURN_CODE DS F FUNCTION RETURN CODE
DSQ_INSTANCE_ID DS F ID ESTABLISHED IN START CMD
DSQ_COMM_LEVEL DS CL12 COMMUNICATIONS LEVEL ID
DSQ_PRODUCT DS CL2 QUERY PRODUCT ID
DSQ_PRODUCT_RELEASE DS CL2 QUERY PRODUCT RELEASE ID
DSQ_RESERVE1 DS CL28 RESERVED
DSQ_MESSAGE_ID DS CL28 COMPLETION MESSAGE ID
DSQ_Q_MESSAGE_ID DS CL28 QUERY MESSAGE ID
DSQ_START_PARAM_ERROR DS CL28 START PARAMETER IN ERROR
DSQCANCEL_IND DS C CMD CANCEL INDICATOR
DSQ_RESERVE2 DS CL23 RESERVED
DSQ_RESERVE3 DS CL156 RESERVED
DSQ_MESSAGE_TEXT DS CL128 COMPLETION MESSAGE
DSQ_Q_MESSAGE_TEXT DS CL128 QUERY MESSAGE
SPACE 1
DSQCOMM_LEN EQU *-DSQCOMM LENGTH OF DSQCOMM AREA
MEND
Related reference:
"Conventions for National Language Feature information" on page 4
Db2 QMF is available in several different languages, each of which is provided by a National Language Feature (NLF).

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 REXX JCL and CLISTs in these examples are not provided with QMF, but you can copy them from here, altering them to suit your needs.

When you translate, assemble, and link-edit a program that uses the QMF callable interface, be aware of the following conditions:
• The interface communications area, 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 step of your program.

The JCL shown here is an example of how to use the procedure DFHEBTAL, supplied with CICS.
Running your Assembler programs in TSO

You must assemble and link-edit your program before you can run it in TSO.

Assembling and link-editing in TSO

This listing shows a sample job that assembles and link-edits your program. Some parameters might vary from one QMF installation to the next.

Running in TSO with ISPF

After your program is assembled successfully, you can run it under ISPF.
Run your program in TSO under ISPF by writing a program similar to the CLIST shown here:

```
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('QMF1210.SDSQCLTE', 'ISR.ISRCLIB')  
ALLOC FI(SYSEXEC) DA('QMF1210.SDSQEXCE')  
/**************************  */  
/* Datasets used by ISPF */  
**************************  */  
ALLOC FI(ISPLLIB) SHR REUSE + DA('QMF1210.SDSQLOAD', 'ADM.GDDMLOAD', 'DSN.DSNEXIT', 'DSN.DSNLOAD')  
ALLOC FI(ISPMLIB) SHR REUSE + DA('QMF1210.SDSQMLBE', 'ISR.ISRMLIB', 'ISP.ISPMLIB')  
ALLOC FI(ISPPLIB) SHR REUSE + DA('QMF1210.SDSQPLBE', 'ISR.ISRPLIB', 'ISP.ISPPLIB')  
ALLOC FI(ISPSLIB) SHR REUSE + DA('ISR.ISRSLIB', 'ISP.ISPSLIB')  
/**************************  */  
/* QMF/GDDM Datasets */  
**************************  */  
ALLOC FI(ADMGGMAP) DA('QMF1210.SDSQMAPE') SHR REUSE  
ALLOC FI(ADMCFORM) DA('QMF1210.SDSQCFORM') SHR REUSE  
ALLOC FI(DSQUCFRM) DA('QMF1210.SDSQUCFRM') SHR REUSE  
ALLOC FI(ADMSYMBL) DA('ADM.GDDMSYM') SHR REUSE  
ALLOC FI(ADMDF) 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(DEBUDGDCB)  
ALLOC FI(DSQDUMP) SYSOUT(X) USING(UDDUMPDCB)  
ALLOC FI(DSQPNLE) NEW UNIT(SYSDA) SPACE(1,1) TRACKS  
ALLOC FI(DSQEDIT) NEW UNIT(SYSDA) USING(EDITDCB)  
**************************  */  
/* Start your program as the initial ISPF dialog */  
**************************  */  
ISPSTART PGM(sampasm) NEWAPPL(DSQE)  
EXIT CODE(4)  
```

The EXIT CODE(4) statement suppresses the ISPF disposition panel.

**Running in TSO without ISPF**

After your program is assembled successfully, you can run it without ISPF.

Run your program in TSO without ISPF by writing a program similar to the CLIST shown here:

```
PROC 0  
CONTROL ASIS  
/**************************  */  
/* Note: QMF, DB2 and GDDM load libraries must be allocated */  
```
C language interface

You can use the C language with the callable interface in QMF.

Interface communications area mapping for C language (DSQCOMMC)

DSQCOMMC provides DSQCOMM mapping for C language programs and is provided with QMF.

The table shows the values for DSQCOMMC.
### Table 45. Interface communications area for DSQCOMMC

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQ_RETURN_CODE</td>
<td>signed long integer</td>
<td>Indicates the status of a QMF command after it is run</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Its values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DSQ_SUCCESS</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Successful execution of the request</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DSQ_WARNING</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Normal completion with warnings</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DSQ_FAILURE</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Command did not run correctly</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DSQ_SEVERE</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Severe error; QMF session terminated</td>
</tr>
<tr>
<td>DSQ_INSTANCE_ID</td>
<td>signed long integer</td>
<td>Identifier established by QMF during execution of the START command</td>
</tr>
<tr>
<td>DSQ_COMM_LEVEL</td>
<td>character, length 12</td>
<td>Identifies the level of the DSQCOMM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In your application, include instructions that initialize this variable to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the value of DSQ_CURRENT_COMM_LEVEL before issuing the QMF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>START command.</td>
</tr>
<tr>
<td>DSQ_PRODUCT</td>
<td>character, length 2</td>
<td>Identifies the IBM query product in use</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Variables that begin with DSQ_QMF specify QMF for TSO and QMF for CICS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>versions.</td>
</tr>
<tr>
<td>DSQ_PRODUCT_RELEASE</td>
<td>character, length 2</td>
<td>Release level of QMF in use</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Variable DSQ_QMF_V11R1 specifies QMF Version 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Release 1.</td>
</tr>
<tr>
<td>DSQ_RESERVE1</td>
<td>character, length 28</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>DSQ_MESSAGE_ID</td>
<td>character, length 8</td>
<td>Completion message ID</td>
</tr>
<tr>
<td>DSQ_Q_MESSAGE_ID</td>
<td>character, length 8</td>
<td>Query message ID</td>
</tr>
<tr>
<td>DSQ_START_PARM_ERROR</td>
<td>character, length 8</td>
<td>Parameter in error when START failed due to a parameter error</td>
</tr>
<tr>
<td>DSQ_CANCEL_IND</td>
<td>character, length 1</td>
<td>Contains one of two values, depending on whether the user canceled while a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QMF command was running:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DSQ_CANCEL_YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DSQ_CANCEL_NO</td>
</tr>
<tr>
<td>DSQ_RESERVE2</td>
<td>character, length 23</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>DSQ_RESERVE3</td>
<td>character, length 156</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>DSQ_MESSAGE_TEXT</td>
<td>character, length 128</td>
<td>Completion message text</td>
</tr>
<tr>
<td>DSQ_Q_MESSAGE_TEXT</td>
<td>character, length 128</td>
<td>Query message text</td>
</tr>
</tbody>
</table>

### 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; its syntax is as follows:

```
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**
  The QMF command to run, specified as an array of unsigned character type of the length specified by CMDLTH

  The QMF command must be in uppercase.

**DSQCICE**

This call has an extended syntax for the QMF commands that require access to application program variables: START, TRACE, 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; an array of unsigned character type. The QMF command must be in uppercase.

- **PNUM**
  Number of command keywords or trace areas; a long integer parameter.

- **KLTH**
  The length of each specified keyword (KWORD) or trace title; a long integer parameter or an array of long integer parameters.

- **KWORD**
  QMF keyword, keywords, or address of trace titles; each is a character, array of characters, or array of addresses to trace titles.

- **VLTH**
  The length of each value that is associated with the keyword or trace title; a long integer parameter or array of long integer parameters.

- **VALUE**
  The value that is associated with each keyword or the address of a value that is associated with a trace title.
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. For trace data, VTYPE must be FINT.

**VTYPE**

Data type of the contents of the VALUE parameter.

This parameter has one of two values, which are provided in the interface communications area, DSQCOMMC:

- DSQ_VARIABLE_CHAR for unsigned character type
- DSQ_VARIABLE_FINT for long integer

All of the values that are specified in the VALUE field must have the data type that is specified by VTYPE.

The C language interface has the following parameter considerations:

- Command strings and the START, GET, and SET command parameters are all input character strings. With these strings, C requires you to pass a storage area that is terminated with a null value, which must be included in the length of the parameter. Use the compile-time length function to obtain the parameter length that is passed to the QMF interface.
- If the string is not terminated by a null value 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 storage area of the application. QMF also 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.

**C language programming example**

The sample C program, DSQABFC, is provided with QMF. The sample program is a member of the library QMF1210.SDSQSAPn (where n is a national language identifier).

The sample program for the IBM C language callable interface performs the following functions:

- Starts QMF
- Sets three global variables
- Runs a query called Q1
- Prints the resulting report by using form F1
- Ends the QMF session

QMF does not supply query Q1 or form F1, but the sample program uses these objects.

```c
/***************************************************************************/
/* Sample Program: DSQABFC */
/***************************************************************************/

/***************************************************************************/
/* 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 */

    /*************************************************************************
    /* 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]);

/*******************************************************************/
/* 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]);

command_length = sizeof(end_query_interface);
dsqcic(&communication_area,&command_length,;
&end_query_interface[0]);
exit(0);
}

DSQCOMM for C
The interface communications area file for the C language is named DSQCOMMC.

The DSQCOMMC include file, shown here, is provided with QMF.

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_QMF_V7R1 "07"
# define DSQ_QM4_V1R1 "01"
# define DSQ_QMF_V7R2 "08"
# define DSQ_QMF_V8R1 "09"
# define DSQ_QMF_V9R1 "10"
# define DSQ_QMF_V10R1 "11"

/* INSTANCE CODES */

# define DSQ_CONTINUE 0

/* CANCELLED INDICATOR */

# define DSQCANCEL_YES "1"
# define DSQCANCEL_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)

Running your C programs in CICS

After you write your program in C, you need to translate, compile, and link-edit it before you can run it.

These examples show the necessary steps to translate, compile, and link-edit your program. The REXX JCL and CLISTs in these examples are not provided with QMF, but you can copy them from here, altering them to suit your needs.

When you translate, compile, and link-edit a program that uses the QMF callable interface under CICS, consider the following conditions:

- The interface communications area 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 step of your program.
• Programs written in C must be link-edited with AMODE=31.

The example shown here uses the procedure DFHYITDL, supplied with CICS.

```
//sampleC JOB
// EXEC PROC=DFHYITDL
//TRN.SYSIN DD *
.
.
.
Your program or copy of QMF sample DSQABFC
.
.
/*
  // Provide Access to QMF Communications Macro DSQCOMM
  //C.SYSLIB DD DSN=QMF1210.SDSQSAPE,DISP=SHR
  // Allocation for target library
  //LKED.SYSLMOD DD
  // Allocation for QMF load library
  //LKED.QMLOAD DD DSN=QMF1210.SDSQLOAD,DISP=SHR
  //LKED.SYSIN DD *
  INCLUDE QMFLOAD(DSQCICX)
  NAME sampleC(R)
  */
```

Running your C programs in TSO
To run your C program in TSO, compile and link-edit the program, and then run it in either with or without ISPF.

Compiling and link-editing in TSO
You must compile and link-edit your C program before you can run it in TSO.

This job compiles and link-edits your callable interface application by using the IBM C compiler for z/OS. Some parameters might vary from one QMF installation to the next.

```
//sampleC JOB
//STEP1 EXEC PROC=EDCCB,
// INFILE='name of dataset that contains source code',
// OUTFILE='name of dataset that contains executable'
// Provide Access to QMF Communications Macro DSQCOMM
//COMPILE.SYSLIB DD DSN=QMF1210.SAMPLIB,DISP=SHR
//BIND.QMLOAD DD DSN=QMF1210.SDSQLOAD,DISP=SHR
//BIND.SYSIN DD *
  INCLUDE QMFLOAD(DSQCICX)
  */
```

Running your programs in TSO without ISPF
After your C program compiles successfully, you can run it without ISPF.

Run your program in TSO without ISPF by writing a program similar to the CLIST shown:

```
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 "QMF1210.SDSQLOAD". */
/*****************************/
/* Specify attribute list for dataset allocations */
```
Running your programs in TSO under ISPF

After your C program compiles successfully, you can run it under ISPF.

Run your program in TSO under ISPF by writing a program similar to the CLIST for running DSQABFC shown here:

```c
PROC 0
CONTROL ASIS

/********************皇子*************************************************************************/
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)
/********************皇子*************************************************************************/
/* Datasets used by TSO */
ALLOC FI(SYSPROC) DA('QMF1210.DSQCLTE')
ALLOC FI(SYSEXEC) DA('QMF1210.DSQEXCE')
/********************皇子*************************************************************************/
/* QMF/GDDM Datasets */
ALLOC FI(ADMGGMAP) DA('QMF1210.DSQMAPE') SHR REUSE
ALLOC FI(ADMCFORM) DA('QMF1210.DSQCFORM') SHR REUSE
ALLOC FI(DSQUCFRM) DA('QMF1210.DSQUCFRM') SHR REUSE
ALLOC FI(ADMDEF) DA('QMF1210.DSQADEF') SHR REUSE
/********************皇子*************************************************************************/
/* Datasets used by QMF */
ALLOC FI(DSQPRINT) SYSTOUT(X) USING(PRINTDCB)
ALLOC FI(DSQDEBUG) SYSTOUT(X) USING(DEBUGDCB)
ALLOC FI(DSQUDUMP) SYSTOUT(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('QMF1210.DSQPNLE') SHR REUSE
/********************皇子*************************************************************************/
/* Start your program using TSO CALL command */
CALL sampleC
EXIT CODE(0)
```

Chapter 11. Programming language specifications for using the callable interface 161
ALLOC FI(ISPLIB) SHR REUSE +
   DA('ISR.ISRTLIB','ISP.ISPLIB')
/*****************************/
/* QMF/GDDM Datasets        */
/*****************************/
ALLOC FI(ADMGGMAP) DA('QM1210.SDSQMAPE') SHR REUSE
ALLOC FI(ADMCFORM) DA('QM1210.DSQCFORM') SHR REUSE
ALLOC FI(DSQCFORM) DA('QM1210.DSQCFORM') SHR REUSE
ALLOC FI(ADMSYMCL) DA('ADM.GDDMSYM') SHR REUSE
ALLOC FI(ADMDF) DA('ADM.GDDM.CHARTLIB') SHR REUSE
ALLOC FI(ADMDF) DA('ADM.GDDM.NICKNAME') SHR REUSE
/*****************************/
/* Datasets used by QMF     */
/*****************************/
ALLOC FI(DSQPRINT) SYSPUT(X) USING(PRINTDCB)
ALLOC FI(DSQDEBUG) SYSPUT(X) USING(DEBUGDCB)
ALLOC FI(DSQDUMP) SYSPUT(X) USING(UDUMPDCB)
ALLOC FI(DSQFORM) NEW UNIT(SYSDA) SPACE(1,1) TRACKS
ALLOC FI(DSQEDIT) NEW UNIT(SYSA) USING(EDITDCB)
ALLOC FI(DSQPNLE) DA('QM1210.DSQPNLE') SHR REUSE
/*****************************/
/* Start your program as the initial ISPF dialog */
/*****************************/
ISPSTART PGM(sampleC) NEWAPPL(DSQE)
EXIT CODE(4)

The EXIT CODE(4) statement suppresses the ISPF disposition panel.

**COBOL language interface**

You can use the COBOL language with the callable interface in QMF.

**Interface communications area mapping for COBOL (DSQCOMMB)**

DSQCOMMB provides DSQCOMM mapping for COBOL language programs and is provided with QMF.

The table shows the values for DSQCOMMB.

*Table 46. Interface communications area for COBOL (DSQCOMMB)*

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQ-RETURN-CODE</td>
<td>PIC 9(8)</td>
<td>Indicates the status of a QMF command after is run</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Its values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DSQ-SUCCESS</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Successful execution of the request</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DSQ-WARNING</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Normal completion with warnings</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DSQ-FAILURE</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Command did not run correctly</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DSQ-SEVERE</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Severe error; QMF session terminated</td>
</tr>
<tr>
<td>DSQ-INSTANCE-ID</td>
<td>PIC 9(8)</td>
<td>Identifier established by QMF during execution of the START command</td>
</tr>
</tbody>
</table>
Table 46. Interface communications area for COBOL (DSQCOMMB) (continued)

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQ-COMM-LEVEL</td>
<td>PIC X(12)</td>
<td>Identifies the level of the DSQCOMM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In your application, include instructions that initialize this variable to the value of DSQ_CURRENT_COMM_LEVEL before issuing the QMF START command.</td>
</tr>
<tr>
<td>DSQ-PRODUCT</td>
<td>PIC X(2)</td>
<td>Identifies the IBM query product in use</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Variables that begin with DSQ-QMF specify QMF for TSO and QMF for CICS versions.</td>
</tr>
<tr>
<td>DSQ-PRODUCT-RELEASE</td>
<td>PIC X(2)</td>
<td>Release level of QMF in use</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Variable DSQ_QMF_V11R1 specifies QMF Version 12 Release 1.</td>
</tr>
<tr>
<td>DSQ-RESERVE1</td>
<td>PIC X(28)</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>DSQ-MESSAGE-ID</td>
<td>PIC X(8)</td>
<td>Completion message ID</td>
</tr>
<tr>
<td>DSQ-Q-MESSAGE-ID</td>
<td>PIC X(8)</td>
<td>Query message ID</td>
</tr>
<tr>
<td>DSQ-START-PARM-ERROR</td>
<td>PIC X(8)</td>
<td>Parameter in error when START failed due to a parameter error</td>
</tr>
<tr>
<td>DSQ-CANCEL-IND</td>
<td>PIC X(1)</td>
<td>Contains one of two values, depending on whether the user canceled while a QMF command was running:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DSQ-CANCEL-YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DSQ-CANCEL-NO</td>
</tr>
<tr>
<td>DSQ-RESERVE2</td>
<td>PIC X(23)</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>DSQ-RESERVE3</td>
<td>PIC X(156)</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>DSQ-MESSAGE-TEXT</td>
<td>PIC X(128)</td>
<td>Completion message text</td>
</tr>
<tr>
<td>DSQ-Q-MESSAGE-TEXT</td>
<td>PIC X(128)</td>
<td>Query message text</td>
</tr>
</tbody>
</table>

Function calls for COBOL

QMF provides one function call for the COBOL language: DSQCIB. The 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); an integer parameter

**CMDSTR**

QMF command to run; an uppercase character string of the length specified by CMDLTH
DSQCIB (extended format)

This call has an extended syntax for the QMF commands that require access to application program variables: START, TRACE, 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**
The length of the command string (CMDSTR); an integer parameter.

**CMDSTR**
The QMF command to run; an uppercase character string of the length specified by CMDLTH.

**PNUM**
The number of command keywords or trace areas; an integer parameter.

**KLTH**
The length of each specified keyword or trace title; an integer parameter or an array of integer parameters.

**KWORD**
QMF keyword, keywords, or address of trace titles.

Each is a character, array of characters, or array of addresses to trace titles whose lengths are specified by KLTH. If all the keywords have the same length, you can use an array of characters.

**VLTH**
The length of each value that is associated with the keyword or trace title; an integer parameter or an array of integer parameters.

**VALUE**
The value that is associated with each keyword or the address of a value that is associated with a trace title.

Its type is specified in the VTYPE parameter, and can be a character, an array of characters, an integer parameter, or an array of integer parameters. For trace data, VTYPE must be FINT.

**VTYPE**
Data type of the contents of the VALUE parameter.

This parameter has one of two values, which are provided in the communications area, DSQCOMMB:

- DSQ-VARIABLE-CHAR for character values
- DSQ-VARIABLE-FINT for integer values

All values that are specified in the VALUE field must have the data type that is specified by VTYPE.

The ISPF LIBDEF service with COBOL

Change dynamic calls to the QMF interface (DSQCIB) to static calls to use the LIBDEF function in your QMF application.
For example, consider the following call identifier statement:

```
CALL DSQCIB USING ...
```

You can change this statement to its call literal form as follows:

```
CALL "DSQCIB" USING ...
```

### COBOL programming example

The sample COBOL program, DSQABFCO, is provided with QMF. The sample program is a member of the library QMF1210.5DSQSAPn (where n is a national language identifier).

The sample program for the COBOL callable interface performs the following functions:

- Starts QMF
- Sets three global variables
- Runs a query called Q1
- Prints the resulting report by using form F1
- Ends the QMF session

QMF does not supply query Q1 or form F1, but the sample program uses these objects.

For CICS, the STOP RUN statement must be changed to a GOBACK statement.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DSQABFCO.
DATE-COMPILED.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

COPY DSQCOMMB.

COPY DSQCOMMB definition - contains query interface variables

* 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 ENDIQ 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 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.  
PROCEDURE DIVISION.  
* 
* Start a query interface session  
  MOVE DSQ-CURRENT-COMM-LEVEL TO DSQ-COMM-LEVEL.  
  MOVE 5 TO QICLTH.  
  MOVE 8 TO KLT(1).  
  MOVE 11 TO VLTHS(1).  
  MOVE 1 TO QIPNUM.  
  CALL DSQCIB USING DSQCOMM, QICLTH, STARTQI,  
          QIPNUM, QIKLT, VNAMES,  
* 
* Set numeric values into query variables using SET GLOBAL command
  MOVE 10 TO QICLTH.  
  MOVE 7 TO KLT(1).  
  MOVE 5 TO KLT(2).  
  MOVE 7 TO KLT(3).  
  MOVE 4 TO VLTHS(1).  
  MOVE 4 TO VLTHS(2).  
  MOVE 4 TO VLTHS(3).  
  MOVE 20 TO VVALUES(1).  
  MOVE 40 TO VVALUES(2).  
  MOVE 84 TO VVALUES(3).  
  MOVE 3 TO QIPNUM.  
  CALL DSQCIB USING DSQCOMM, QICLTH, SETQ,  
          QIPNUM, QIKLT, 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, ENQI.

STOP RUN.

Related reference:

"Conventions for National Language Feature information" on page 4

Db2 QMF is available in several different languages, each of which is provided by a National Language Feature (NLF).

**DSQCOMM for COBOL**

The interface communications area file for the COBOL language is named DSQCOMMB.

The DSQCOMMB include file shown here, is provided with QMF:

```
************************************************************
* COBOL INCLUDE FOR QUERY CALLABLE INTERFACE
************************************************************
```
* STRUCTURE DECLARE FOR COMMUNICATIONS AREA

01 DSQCOMM.

  03 DSQ-RETURN-CODE PIC 9(8) USAGE IS COMP.
  * FUNCTION RETURN CODE
  03 DSQ-INSTANCE-ID PIC 9(8) USAGE IS COMP.
  IDENTIFIER FROM START CMD
  03 DSQ-COMM-LEVEL PIC X(12).
  * COMMUNICATIONS LEVEL
  03 DSQ-PRODUCT PIC X(2).
  * QUERY PRODUCT ID
  03 DSQ-PRODUCT-RELEASE PIC X(2).
  * QUERY PRODUCT RELEASE
  03 DSQ-RESERVE1 PIC X(28).
  * RESERVED AREA
  03 DSQ-MESSAGE-TEXT PIC X(8).
  * COMPLETION MESSAGE ID
  03 DSQ-Q-MESSAGE-TEXT PIC X(8).
  * QUERY MESSAGE ID
  03 DSQ-START-PARM-ERROR PIC X(8).
  * START PARAMETER IN ERROR
  03 DSQ-CANCEL-IND PIC X(1).
  * 1 = COMMAND CANCELLED
     0 = COMMAND NOT CANCELLED
  03 DSQ-RESERVE2 PIC X(23).
  * RESERVED AREA
  03 DSQ-RESERVE3 PIC X(156).
  * RESERVED AREA
  03 DSQ-MESSAGE-TEXT PIC X(128).
  * QMF MESSAGE TEXT
  03 DSQ-Q-MESSAGE-TEXT PIC X(128).
  * QMF QUERY MESSAGE TEXT
  * 512 BYTES TOTAL

* VALUES FOR DSQ-RETURN-CODE

  01 DSQ-SUCCESS PIC 9(8) USAGE IS COMP VALUE 0.
  01 DSQ-WARNING PIC 9(8) USAGE IS COMP VALUE 4.
  01 DSQ-FAILURE PIC 9(8) USAGE IS COMP VALUE 8.
  01 DSQ-SEVERE PIC 9(8) USAGE IS COMP VALUE 16.

* VALUES FOR DSQ-INSTANCE-ID

  01 DSQ-CONTINUE PIC 9(8) USAGE IS COMP VALUE 0.

* VALUES FOR DSQ-COMM-LEVEL

  01 DSQ-CURRENT-COMM-LEVEL PIC X(12) VALUE "DSQL>001002<".

* VALUES FOR DSQ-PRODUCT

  01 DSQ-QRW PIC X(2) VALUE "01".
  01 DSQ-QMF PIC X(2) VALUE "02".
  01 DSQ-QM4 PIC X(2) VALUE "03".

* VALUES FOR DSQ-PRODUCT-RELEASE

  01 DSQ-QRW-V1R2 PIC X(2) VALUE "01".
  01 DSQ-QRW-V1R3 PIC X(2) VALUE "02".
  01 DSQ-QMF-V2R4 PIC X(2) VALUE "01".
  01 DSQ-QMF-V3R1 PIC X(2) VALUE "02".
  01 DSQ-QMF-V3R1M1 PIC X(2) VALUE "03".
  01 DSQ-QMF-V3R2 PIC X(2) VALUE "04".
  01 DSQ-QMF-V3R3 PIC X(2) VALUE "05".
  01 DSQ-QMF-V6R1 PIC X(2) VALUE "06".
Considerations for running your COBOL callable interface program

Pay attention to the details about running a COBOL program that uses the QMF callable interface.

When you translate, compile, and link-edit a program that uses the QMF callable interface, consider the following conditions:

- **The execution environment**
  QMF is run as an Assembler program in the COBOL environment. Your COBOL program must call the QMF interface program, DSQCIB, by using a COBOL dynamic call.

- **Whether to use quotation marks or apostrophes**
  You must use either double quotation marks (""') 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 (DSQABFSCO) as distributed by QMF use quotations to delimit literals. If your site or program uses apostrophes instead of quotation marks, change DSQCOMMB or copy the structure to your program, changing quotation marks to apostrophes.

- **Availability of the communications area (DSQCOMMB)**
  The communications area DSQCOMMB must be available to the COBOL compile step or copied into your program as a control structure.

- **Availability of the interface module (DSQCIB)**
The QMF interface module must be available during the link-edit step of your program.

**Running your COBOL programs in CICS**

After you write your program in COBOL, you need to translate, compile, and link-edit it before you can run it.

The JCL in these examples is not provided with QMF, but you can copy it from here, altering it to suit your needs.

The example shows the procedure DFHEBTVL, supplied with CICS, and which supports COBOL.

```plaintext
//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 DSQCOMM
  //COBOL.SYSLIB DD DSN=QMF1210.SDSQAPE,DISP=SHR
  //* Allocation for target library
  //LKED.SYSLMOD DD
  //* Allocation for QMF load library
  //LKED.QMFLOAD DD DSN=QMF1210.SDSQLOAD,DISP=SHR
  //LKED.SYSIN DD *
  INCLUDE CICSLOAD(DFHECI)
  INCLUDE QMFLOAD(DSQCIB)
ORDER DFHECI
ENTRY samCOBOL
MODE AMODE(31) RMODE(31)
NAME samCOBOL(R)
*/
```

**Running your COBOL programs in TSO**

To run your COBOL program in TSO, compile and link-edit the program, and then run it in either with or without ISPF.

**Compiling and link-editing in TSO**

You must compile and link-edit your COBOL program before you can run it in TSO.

This job uses the COBOL compiler to compile your callable interface application. It then link-edits your application. Some parameters might vary from one QMF installation to the next.

```plaintext
//samCOBOL JOB
//STEP1 EXEC PROC=IGYWCL
//* Provide access to QMF communications macro DSQCOMM
//COBOL.SYSLIB DD DSN=QMF1210.SAMPLIB,DISP=SHR
//COBOL.SYSIN DD *
  .
  .
  .
Your program or copy of QMF sample DSQABFCO
  .
  .
  .
```
Running your programs in TSO without ISPF
After your COBOL program compiles successfully, you can run it with JCL without ISPF.

Run the COBOL compiler and linkage editor in TSO without ISPF by writing a program similar to the CLIST shown here:

PROCO
CONTROL ASIS

*******************************************************************************/
/* Note: QMF, DB2, GDDM and COBOL load libraries must be allocated before running this CLIST. */
/* Name of QMF load library is "QMF1210.SDSQLOAD". */
/* Specify attribute list for dataset allocations */
*******************************************************************************/
ATTR PRINTDCB LRECL(133) RECFM(FBA) BLKSIZE(1330)
ATTR DEBUGDCB LRECL(80) RECFM(FB) BLKSIZE(3120)
ATTR UDUMPDCB LRECL(125) RECFM(VBA) BLKSIZE(1632)
ATTR EDITDCB LRECL(79) RECFM(FBA) BLKSIZE(4029)
*******************************************************************************/
/* Datasets used by TSO */
*******************************************************************************/
ALLOC FI(SYSPROC) DA('QMF1210.SDSQCLTE')
ALLOC FI(SYSEXEC) DA('QMF1210.SDSQEXCE')
*******************************************************************************/
/* QMF/GDDM Datasets */
*******************************************************************************/
ALLOC FI(ADMGGMAP) DA('QMF1210.SDSQMAPE') SHR REUSE
ALLOC FI(ADMCFORM) DA('QMF1210.SDSQCFRM') SHR REUSE
ALLOC FI(DSQUCFRM) DA('QMF1210.SDSQUCFRM') SHR REUSE
ALLOC FI(ADMSYMBL) DA('ADM.GDDMSYM') SHR REUSE
ALLOC FI(ADMDF) 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(DSQDUMP) 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('QMF1210.DSQPNLE') SHR REUSE
*******************************************************************************/
/* Start your program using TSO CALL command */
*******************************************************************************/
CALL samCOBOL
EXIT CODE(0)

Running your programs in TSO under ISPF
After your COBOL program compiles successfully, you can run it under ISPF.
Run your program in TSO under ISPF by writing a program similar to the CLIST shown here:

```clist
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('QMF1210.SDSQCLTE', 'ISR.ISRCLIB')
ALLOC FI(SYSEXEC) DA('QMF1210.SDSQEXCE')
/*****************************/
/* Datasets used by ISPF */
/*****************************/
ALLOC FI(ISPLLIB) SHR REUSE + DA('QMF1210.SDSQLOAD', 'ADM.GDDMLOAD', 'DSN.DSNEXIT', 'DSN.DSNLOAD', + 'PRODUCT.COB2LIB')
ALLOC FI(ISPMLIB) SHR REUSE + DA('QMF1210.SDSQMLBE', 'ISR.ISRMLIB', 'ISP.ISPMLIB')
ALLOC FI(ISPPLIB) SHR REUSE + DA('QMF1210.SDSQPLBE', 'ISR.ISRPLIB', 'ISP.ISPPLIB')
ALLOC FI(ISPSLIB) SHR REUSE + DA('QMF1210.SDSQSLBE', 'ISR.ISPSLIB', 'ISP.ISPSLIB')
ALLOC FI(ISPTLIB) SHR REUSE + DA('ISR.ISRTLIB', 'ISP.ISPTLIB')
/*****************************/
/* QMF/GDDM Datasets */
/*****************************/
ALLOC FI(ADMGGMAP) DA('QMF1210.SDSQMAPE') SHR REUSE
ALLOC FI(ADMCFORM) DA('QMF1210.DSQCFORM') SHR REUSE
ALLOC FI(DSQUCFRM) DA('QMF1210.DSQUCFRM') SHR REUSE
ALLOC FI(ADMSYMBL) DA('ADM.GDDMSYM') SHR REUSE
ALLOC FI(ADMDF) DA('ADM.GDOM.CHARTLIB') SHR REUSE
ALLOC FI(ADMDEFS) DA('ADM.GDOM.NICKNAME') SHR REUSE
/*****************************/
/* Datasets used by QMF */
/*****************************/
ALLOC FI(DSQPRINT) SYOUT(X) USING(PRINTDCB)
ALLOC FI(DSQDEBUG) SYOUT(X) USING(DEBUGDCB)
ALLOC FI(DSQUDUMP) SYOUT(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('QMF1210.DSQPNLE') SHR REUSE
/*****************************/
/* Start your program as the initial ISPF dialog */
/*****************************/
ISPSTART PGM(samCOBOL) NEWAPPL(DSQE)
EXIT CODE(4)
```

The EXIT CODE(4) statement suppresses the display of the ISPF disposition panel.

---

**Fortran language interface**

You can use the Fortran language with the callable interface in QMF for TSO.

**Restriction:** 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 language programs and is provided with QMF.

The table shows the information for DSQCOMMF, which you must not alter:

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQ_RETURN_CODE</td>
<td>INTEGER</td>
<td>Indicates the status of a QMF command after it is run</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Its values are:</td>
</tr>
<tr>
<td>DSQ_SUCCESS</td>
<td></td>
<td>Successful execution of the request</td>
</tr>
<tr>
<td>DSQ_WARNING</td>
<td></td>
<td>Normal completion with warnings</td>
</tr>
<tr>
<td>DSQ_FAILURE</td>
<td></td>
<td>Command did not run correctly</td>
</tr>
<tr>
<td>DSQ_SEVERE</td>
<td></td>
<td>Severe error; QMF session terminated</td>
</tr>
<tr>
<td>DSQ_INSTANCE_ID</td>
<td>INTEGER</td>
<td>Identifier established by QMF during execution of the START command</td>
</tr>
<tr>
<td>DSQ_COMM_LEVEL</td>
<td>CHARACTER(12)</td>
<td>Identifies the level of the DSQCOMM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In your application, include instructions that initialize this variable to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the value of DSQ_CURRENT_COMM_LEVEL before issuing the QMF START command.</td>
</tr>
<tr>
<td>DSQ_PRODUCT</td>
<td>CHARACTER(2)</td>
<td>Identifies the IBM query product in use</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Variables that begin with DSQ_QMF specify QMF for TSO versions.</td>
</tr>
<tr>
<td>DSQ_PRODUCT_RELEASE</td>
<td>CHARACTER(2)</td>
<td>Release level of QMF in use</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Variable DSQ_QMF_V1R1 specifies QMF Version 12 Release 1.</td>
</tr>
<tr>
<td>DSQ_RESERVE1</td>
<td>CHARACTER(28)</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>DSQ_MESSAGE_ID</td>
<td>CHARACTER(8)</td>
<td>Completion message ID</td>
</tr>
<tr>
<td>DSQ_Q_MESSAGE_ID</td>
<td>CHARACTER(8)</td>
<td>Query message ID</td>
</tr>
<tr>
<td>DSQ_START_PARM_ERROR</td>
<td>CHARACTER(8)</td>
<td>Parameter in error when START failed due to a parameter error</td>
</tr>
<tr>
<td>DSQ_CANCEL_IND</td>
<td>CHARACTER(1)</td>
<td>Contains one of two values, depending on whether the user canceled while a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QMF command was running:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DSQ_CANCEL_YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DSQ_CANCEL_NO</td>
</tr>
<tr>
<td>DSQ_RESERVE2</td>
<td>CHARACTER(23)</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>DSQ_RESERVE3</td>
<td>CHARACTER(156)</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>DSQ_MESSAGE_TEXT</td>
<td>CHARACTER(128)</td>
<td>Completion message text</td>
</tr>
<tr>
<td>DSQ_Q_MESSAGE_TEXT</td>
<td>CHARACTER(128)</td>
<td>Query message text</td>
</tr>
</tbody>
</table>

Table 47. Interface communications area for Fortran (DSQCOMMF)
Function calls for Fortran

QMF provides two function calls for the Fortran language: DSQCIF and DSQCIFE. Both calls are described in the communications area (DSQCOMMF).

**DSQCIF**

This call is for QMF commands that do not require access to application program variables. Use this call for most QMF commands.

\[
\text{RC} = \text{DSQCIF(DSQCOMM,}
+ \text{CMDLTH},
+ \text{CMDSTR})
\]

The parameters have the following values:

**DSQCOMM**
The communications area.

**CMDLTH**
The length of the command string (**CMDSTR**); an integer parameter.

**CMDSTR**
QMF command to run; an uppercase character string of the length that is specified by **CMDLTH**.

**DSQCIFE**

This call has an extended syntax for the commands that require access to application program variables: START, TRACE, and the extended formats of GET GLOBAL and SET GLOBAL.

The syntax for this call is:

\[
\text{RC} = \text{DSQCIFE(DSQCOMM,}
+ \text{CMDLTH},
+ \text{CMDSTR},
+ \text{PNUM},
+ \text{KLTH},
+ \text{KWORD},
+ \text{VLTH},
+ \text{VALUE},
+ \text{VTYPE})
\]

The parameters have the following values:

**DSQCOMM**
The interface communications area.

**CMDLTH**
The length of the command string (**CMDSTR**); an integer parameter.

**CMDSTR**
The QMF command to run; an uppercase character string of the length that is specified by **CMDLTH**.

**PNUM**
The number of command keywords or trace areas; an integer parameter.

**KLTH**
The length of each specified keyword or trace title; an integer parameter or parameter array.
**KWORD**

QMF keyword, keywords, or address of trace titles; a character, array of characters, or array of addresses to trace titles whose lengths are specified by **KLTH**.

You can use an array of characters if all of the keywords have the same length. The keywords must be in contiguous storage and not separated by any special delimiters.

**KLTH**

The length of each value that is associated with the keyword or trace title; an integer parameter or parameter array.

**VALUE**

The value that is associated with each keyword or the address of a value that is associated with a trace title.

Its type is specified in the **VTYPE** parameter and can be a character, array of characters, integer parameter, or parameter array. For trace data, **VTYPE** must be FINT. If you have character values, the values must be in contiguous storage and not separated by any special delimiters.

**VTYPE**

Data type of the contents of the **VALUE** parameter.

This parameter has one of two values, which are provided in the communications area, DSQCOMMF:

- DSQ_VARIABLE_CHAR for character values
- DSQ_VARIABLE_FINT for integer values

All values that are specified in the **VALUE** field must have the data type that is specified by **VTYPE**.

**Fortran programming example**

The sample program, DSQABFF, is provided with QMF. The sample program is a member of the library QMF1210.SDSQSAP$n$ (where $n$ is a national language identifier).

The sample program for the Fortran callable interface performs the following functions:

- Starts QMF
- Sets three global variables
- Runs a query called Q1
- Prints the resulting report by using form F1
- Ends the QMF session

QMF does not supply query Q1 or form F1, but the sample program uses these objects.

```fortran
C***********************************************************************
C Sample program: DSQABFF
C FORTRAN version of QMF 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. Set global query manager numeric variables.

d. Run a query manager query using the callable interface.

e. Print a report using the callable interface.

f. Exit the query manager session.

Prerequisites:
1. Create the SAMPLE database.
2. Create a prompted query, Q1, which has a SELECT state
3. Create a form, F1, that displays data for query Q1.

Chapter 11. Programming language specifications for using the callable interface
EQUIVALENCE (SET_VALUES(1), SET_VALUE_1),
+ (SET_VALUES(5), SET_VALUE_2),
+ (SET_VALUES(9), SET_VALUE_3)

C Declarer command length and return code variables
C***********************************************************************
INTEGER LEN,
+ RC

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 /

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 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,
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

Related reference:
"Conventions for National Language Feature information" on page 4
Db2 QMF is available in several different languages, each of which is provided by
a National Language Feature (NLF).

**DSQCOMM for Fortran**

The interface communications area file for the Fortran language is named
DSQCOMM.F.

The DSQCOMM.F include file shown here, is provided with QMF:

C**********************************************************************
C FORTRAN include file for callable interface
C**********************************************************************
C Return codes
INTEGER DSQ_SUCCESS, DSQ_WARNING, DSQ_FAILURE, DSQ_SEVERE
PARAMETER(
+ DSQ_SUCCESS = 0,
+ DSQ_WARNING = 4,
+ DSQ_FAILURE = 8,
+ DSQ_SEVERE = 16)

C Communications level
CHARACTER DSQ_CURRENT_COMM_LEVEL*12
PARAMETER(
+ DSQ_CURRENT_COMM_LEVEL = 'DSQL>001002<')

C Query product IDs
CHARACTER DSQ_QRN*2, DSQ_QMF*2, DSQ_QM4*2
PARAMETER(
+ DSQ_QRN = '01',
+ DSQ_QMF = '02',
+ DSQ_QM4 = '03')
C  Query product release levels
CHARACTER DSQ_QRW_V1R2*2, DSQ_QRW_V1R3*2,
+ DSQ_QMW_V2R4*2, DSQ_QMF_V3R1*2,
+ DSQ_QMF_V3R1M1*2, DSQ_QMF_V3R2*2,
+ DSQ_QMF_V3R3*2, DSQ_QMF_V6R1*2,
+ DSQ_QMF_V7R1*2, DSQ_QMF_V8R1*2,
+ DSQ_QMF_V9R1*2, DSQ_QMF_V10R1*2
PARAMETER(
+ DSQ_QRW_V1R2  =  '01',
+ DSQ_QRW_V1R3  =  '02',
+ DSQ_QMW_V2R4  =  '01',
+ DSQ_QMF_V3R1  =  '02',
+ DSQ_QMF_V3R1M1 =  '03',
+ DSQ_QMF_V3R2  =  '04',
+ DSQ_QMF_V3R3  =  '05',
+ DSQ_QMF_V6R1  =  '06',
+ DSQ_QMF_V7R1  =  '07',
+ DSQ_QMF_V7R1  =  '01',
+ DSQ_QMF_V7R2  =  '08',
+ DSQ_QMF_V8R1  =  '09',
+ DSQ_QMF_V9R1  =  '10',
+ DSQ_QMF_V10R1 =  '11')

C  Host variable types
CHARACTER DSQ_VARIABLE_CHAR*4, DSQ_VARIABLE_FINT*4
PARAMETER(
+ DSQ_VARIABLE_CHAR  =  'CHAR',
+ DSQ_VARIABLE_FINT  =  'FINT')

C  Cancel indicator
CHARACTER DSQ_CANCEL_YES, DSQ_CANCEL_NO
PARAMETER(
+ DSQ_CANCEL_YES  =  '1',
+ DSQ_CANCEL_NO   =  '0')
CHARACTER DSQCOMM(512)
INTEGER DSQ_RETURN_CODE, DSQ_INSTANCE_ID
CHARACTER DSQ_COMM_LEVEL*12,
+ DSQ_PRODUCT=2,
+ DSQ_PRODUCT_RELEASE=2,
+ DSQ_RESERVE1=28,
+ DSQ_MESSAGE_ID=8,
+ DSQ_Q_MESSAGE_ID=8,
+ DSQ_START_PARM_ERROR=8,
+ DSQ_CANCEL_IND=1,
+ DSQ_RESERVE2=23,
+ DSQ_RESERVE3=156,
+ DSQ_MESSAGE_TEXT=128,
+ DSQ_Q_MESSAGE_TEXT=128
EQUIVALENCE (DSQCOMM( 1), DSQ_RETURN_CODE ),
+ (DSQCOMM( 5), DSQ_INSTANCE_ID ),
+ (DSQCOMM( 9), DSQ_COMM_LEVEL ),
+ (DSQCOMM(21), DSQ_PRODUCT ),
+ (DSQCOMM(23), DSQ_PRODUCT_RELEASE ),
+ (DSQCOMM(25), DSQ_RESERVE1 ),
+ (DSQCOMM(53), DSQ_MESSAGE_ID ),
+ (DSQCOMM(61), DSQ_Q_MESSAGE_ID ),
+ (DSQCOMM(69), DSQ_START_PARM_ERROR ),
+ (DSQCOMM(77), DSQ_CANCEL_IND ),
+ (DSQCOMM(78), DSQ_RESERVE2 ),
+ (DSQCOMM(101), DSQ_RESERVE3 ),
+ (DSQCOMM(257), DSQ_MESSAGE_TEXT ),
+ (DSQCOMM(385), DSQ_Q_MESSAGE_TEXT )
Callable interface normal and extended calls
EXTERNAL DSQCIF
EXTERNAL DSQCIFE

Running your Fortran programs

To run your Fortran program in TSO, compile and link-edit the program, and then run it in either with or without ISPF.

Compiling and link-editing your program

You must compile and link-edit your Fortran program before you can run it in TSO.

JCL for running the Fortran compiler and linkage editor uses the Fortran compiler for z/OS. Some parameters might vary from one QMF installation to the next.

The JCL in this example is not provided with QMF, but you can copy it from here, altering it to suit your needs.

```
//samFORT JOB
//STEP1 EXEC PROC=VSF2CL
//* Provide access to QMF communications macro DSQCOMM
//FORT.SYSLIB DD DSN=QMF1210.SAMPLIB,DISP=SHR
//FORT.SYSIN DD *
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
/*
//* Provide access to QMF interface module
//* Allocation for target library
//LKED.SYSLMOD DD
//* Allocation for QMF load library
//LKED.QMFLOAD DD DSN=QMF1210.SDSQLOAD,DISP=SHR
//LKED.SYSIN DD *
INCLUDE QMFLOAD(DSQCIF)
INCLUDE QMFLOAD(DSQCIFE)
Enter samFORT
MODE AMODE(31) RMODE(31)
NAME samFORT(R)
/*
```

Running your programs in TSO without ISPF

After your Fortran program compiles successfully, you can run it with JCL without ISPF.

The JCL in this example is not provided with QMF, but you can copy it from here, altering it to suit your needs.

The program shown here runs your callable interface application by using the Fortran compiler. Some parameters can vary from one QMF installation to the next:

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 "QMF1210.SDSQLOAD". */
/*****************************/
/* Specify attribute list for dataset allocations */
Running in TSO under ISPF

After your Fortran program compiles successfully, you can run it under ISPF.

The CLIST in this example is not provided with QMF, but you can copy it from here, altering it to suit your needs.

The CLIST shown here runs your callable interface application by using the Fortran compiler. Some parameters can vary from one QMF installation to the next:

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('QMF1210.SDSQCLTE')
ALLOC FI(SYSEXEC) DA('QMF1210.SDSQEXCE')
/*******************************************************************************/
/* QMF/GDDM Datasets */
/*******************************************************************************/
ALLOC FI(ADMGGMAP) DA('QMF1210.SDSQMAPE') SHR REUSE
ALLOC FI(ADMCFORM) DA('QMF1210.DSQCFORM') SHR REUSE
ALLOC FI(DSQUCFRM) DA('QMF1210.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('QMF1210.DSQPNLE') SHR REUSE
/*******************************************************************************/
/* Start your program using TSO CALL command */
/*******************************************************************************/
CALL samFORT
EXIT CODE(0)
The EXIT CODE(4) statement suppresses the display of the ISPF disposition panel.

PL/I language interface

You can use the PL/I language with the callable interface in QMF.

Interface communications area mapping for PL/I (DSQCOMML)

DSQCOMML provides DSQCOMM mapping for PL/I and is provided with QMF.

The table shows the values for DSQCOMML.

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQ_RETURN_CODE</td>
<td>FIXED BIN(31)</td>
<td>Indicates the status of a QMF command after it is run</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Its values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DSQ_SUCCESS</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Successful execution of the request</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DSQ_WARNING</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Normal completion with warnings</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DSQ_FAILURE</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Command did not run correctly</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DSQ_SEVERE</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Severe error; QMF session terminated</td>
</tr>
<tr>
<td>DSQ_INSTANCE_ID</td>
<td>FIXED BIN(31)</td>
<td>Identifier established by QMF during execution of the START command</td>
</tr>
</tbody>
</table>
### Table 48. Interface communications area for PL/I (DSQCOMML) (continued)

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQ_COMM_LEVEL</td>
<td>CHAR(12)</td>
<td>Identifies the level of the DSQCOMM communications area</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In your application, include instructions that initialize this variable to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the value of DSQ_CURRENT_COMM_LEVEL before issuing the QMF START command.</td>
</tr>
<tr>
<td>DSQ_PRODUCT</td>
<td>CHAR(2)</td>
<td>Identifies the IBM query product in use. Variables that begin with DSQ_QMF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>specify QMF for TSO and CICS versions.</td>
</tr>
<tr>
<td>DSQ_PRODUCT_RELEASE</td>
<td>CHAR(2)</td>
<td>Release level of QMF in use. Variable DSQ_QMF_V11R1 specifies QMF Version</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12 Release 1.</td>
</tr>
<tr>
<td>DSQ_RESERVE1</td>
<td>CHAR(28)</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>DSQ_MESSAGE_ID</td>
<td>CHAR(8)</td>
<td>Completion message ID</td>
</tr>
<tr>
<td>DSQ_Q_MESSAGE_ID</td>
<td>CHAR(8)</td>
<td>Query message ID</td>
</tr>
<tr>
<td>DSQ_START_PARM_ERROR</td>
<td>CHAR(8)</td>
<td>Parameter in error when START failed due to a parameter error</td>
</tr>
<tr>
<td>DSQ_CANCEL_IND</td>
<td>CHAR(1)</td>
<td>Contains one of two values, depending on whether the user canceled while a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QMF command was running:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DSQ_CANCEL_YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DSQ_CANCEL_NO</td>
</tr>
<tr>
<td>DSQ_RESERVE2</td>
<td>CHAR(23)</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>DSQ_RESERVE3</td>
<td>CHAR(156)</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>DSQ_MESSAGE_TEXT</td>
<td>CHAR(128)</td>
<td>Completion message text</td>
</tr>
<tr>
<td>DSQ_Q_MESSAGE_TEXT</td>
<td>CHAR(128)</td>
<td>Query message text</td>
</tr>
</tbody>
</table>

### Function calls for PL/I

QMF provides two function calls for PL/I: DSQCIPL and DSQCIPX. Both calls are described in the communications area (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.

```plaintext
CALL DSQCIPL(DSQCOMM, CMLTH, CMDSTR)
```

The parameters have the following values:

**DSQCOMM**

The interface communications area.

**CMLTH**

The length of the command string (CMDSTR).

**CMDSTR**

The QMF command to run; an uppercase character string of the length specified by CMLTH.
**DSQCIPX syntax**

This call is for the commands that require access to application program variables: START, TRACE, 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**

The length of the command string (CMDSTR); an integer FIXED BIN(31) parameter.

**CMDSTR**

The QMF command to run; an uppercase character string of the length specified by CMDLTH.

**PNUM**

The number of command keywords or trace areas; an integer FIXED BIN(31) parameter.

**KLTH**

The length of each specified keyword or trace title; an integer FIXED BIN(31) parameter or parameter array.

**KWORD**

The QMF keyword, keywords, or address of trace titles.

Each is a character, array of characters, or array of addresses to trace titles whose lengths are specified by KLTH. You can use an array of characters if all of the keywords have the same length. The keywords must be in contiguous storage and not separated by any special delimiters.

**VLTH**

The length of each value that is associated with the keyword or trace title; an integer FIXED BIN(31) parameter or parameter array.

**VALUE**

The value that is associated with each keyword or the address of a value that is associated with a trace title.

Its type is specified in the VTYPE parameter and can be a character, array of characters, integer FIXED BIN(31) parameter, or parameter array. If you have character values, the values must be in contiguous storage and not separated by any special delimiters.

**VTYPE**

Data type of the contents of the VALUE parameter.

This parameter has one of two values, which are provided in the DSQCOMML communications area:
• DSQ_VARIABLE_CHAR for character values
• DSQ_VARIABLE_FINT for integer FIXED BIN(31) values

All values that are specified in the VALUE field must have the data type that is specified in VTYPE.

PL/I programming example

The sample program, DSQABFP, is provided with QMF and uses PL/I. The sample program is a member of the library QMF1210.SDSQSAPn (where n is a national language identifier).

The sample program for the PL/I language callable interface performs the following functions:
• Starts QMF
• Sets three global variables
• Runs a query called Q1
• Prints the resulting report by using form F1
• Ends the QMF session

QMF does not supply query Q1 or form F1, but the sample program uses these objects.

DSQABFP: PROCEDURE OPTIONS(MAIN REENTRANT) REORDER;
/*******************************************************************************
/* Sample program: DSQABFP */
/* PL/I version of the QMF callable interface */
*******************************************************************************

*******************************************************************************
/* Include and declare query interface communications area */
*******************************************************************************
INCLUDE SYSLIB(DSQCOMML);

*******************************************************************************
/* Built in function */
*******************************************************************************
DCL LENGTH BUILTIN;

*******************************************************************************
/* Query interface command length and commands */
*******************************************************************************
DCL COMMAND_LENGTH FIXED BIN(31);
DCL START_QUERY_INTERFACE CHAR(5) INIT('START');
DCL SET_GLOBAL_VARIABLES CHAR(10) INIT('SET GLOBAL');
DCL RUN_QUERY CHAR(12) INIT('RUN QUERY Q1');
DCL PRINT_REPORT CHAR(22) INIT('PRINT REPORT (FORM=F1)');
DCL END_QUERY_INTERFACE CHAR(4) INIT('EXIT');

*******************************************************************************
/* Query command extension, number of parameters and lengths */
*******************************************************************************
DCL NUMBER_OF_PARAMETERS FIXED BIN(31); /* number of variables */
DCL KEYWORD_LENGTHS(10) FIXED BIN(31); /* lengths of keyword names*/
DCL DATA_LENGTHS(10) FIXED BIN(31); /* lengths of variable data*/

*******************************************************************************
/* Keyword parameter and value for START command */
*******************************************************************************
DCL START_KEYWORDS CHAR(8) INIT('DSQSMODE');
DCL START_KEYWORD_VALUES CHAR(11) INIT('INTERACTIVE');

*******************************************************************************
/* Keyword parameter and value for SET command */
/***************************************************/
DCL 1 SET_KEYWORDS,
    3 SET_KEYWORDS_1 CHAR(7) INIT('MYVAR01'),
    3 SET_KEYWORDS_2 CHAR(5) INIT('SHORT'),
    3 SET_KEYWORDS_3 CHAR(7) INIT('MYVAR03');

DCL 1 SET_VALUES,
    3 SET_VALUES_1 FIXED BIN(31),
    3 SET_VALUES_2 FIXED BIN(31),
    3 SET_VALUES_3 FIXED BIN(31);

/****************************************************************************************************/
/* Main program */
/****************************************************************************************************/
DSQCOMM = '';
DSQ_COMM_LEVEL = DSQ_CURRENT_COMM_LEVEL;

/****************************************************************************************************/
/* Start a query interface session */
/****************************************************************************************************/
NUMBER_OF_PARAMETERS = 1;
COMMAND_LENGTH = LENGTH(START_QUERY_INTERFACE);
KEYWORD_LENGTHS(1) = LENGTH(START_KEYWORDS);
DATA_LENGTHS(1) = LENGTH(START_KEYWORD_VALUES);
CALL DSQCIPX(DSQCOMM,
    COMMAND_LENGTH,
    START_QUERY_INTERFACE,
    NUMBER_OF_PARAMETERS,
    KEYWORD_LENGTHS,
    START_KEYWORDS,
    DATA_LENGTHS,
    START_KEYWORD_VALUES,
    DSQ_VARIABLE_CHAR);

/****************************************************************************************************/
/* Set numeric values into query using SET command */
/****************************************************************************************************/
NUMBER_OF_PARAMETERS = 3;
COMMAND_LENGTH = LENGTH(SET_GLOBAL_VARIABLES);
KEYWORD_LENGTHS(1) = LENGTH(SET_KEYWORDS_1);
KEYWORD_LENGTHS(2) = LENGTH(SET_KEYWORDS_2);
KEYWORD_LENGTHS(3) = LENGTH(SET_KEYWORDS_3);
DATA_LENGTHS(1) = 4;
DATA_LENGTHS(2) = 4;
DATA_LENGTHS(3) = 4;
SET_VALUES_1 = 20;
SET_VALUES_2 = 40;
SET_VALUES_3 = 84;
CALL DSQCIPX(DSQCOMM,
    COMMAND_LENGTH,
    SET_GLOBAL_VARIABLES,
    NUMBER_OF_PARAMETERS,
    KEYWORD_LENGTHS,
    SET_KEYWORDS,
    DATA_LENGTHS,
    SET_VALUES,
    DSQ_VARIABLE_FINT);

/****************************************************************************************************/
/* Run a query */
/****************************************************************************************************/
COMMAND_LENGTH = LENGTH(RUN_QUERY);
CALL DSQCIPL(DSQCOMM,
COMMAND_LENGTH,
RUN_QUERY);

/******************************
/* Print the results of the query */
/******************************
COMMAND_LENGTH = LENGTH(PRINT_REPORT);

CALL DSQCIPL(DSQCOMM,
COMMAND_LENGTH,
PRINT_REPORT);

/******************************
/* End the query interface session */
/******************************
COMMAND_LENGTH = LENGTH(END_QUERY_INTERFACE);

CALL DSQCIPL(DSQCOMM,
COMMAND_LENGTH,
END_QUERY_INTERFACE);

END DSQABFP;

Related reference:
"Conventions for National Language Feature information" on page 4
Db2 QMF is available in several different languages, each of which is provided by a National Language Feature (NLF).

**DSQCOMM for PL/I**

The interface communications area for PL/I, is named DSQCOMML.

/******************************/
/* PL/I include for query callable interface */
/******************************/

/* Structure declare for communications area */
DCL
 1 DSQCOMM,
  3 DSQ_RETURN_CODE FIXED BIN(31), /* Function return code */
  3 DSQ_INSTANCE_ID FIXED BIN(31), /* Start ID */
  3 DSQ_COMM_LEVEL CHAR(12), /* Communications level */
  3 DSQ_PRODUCT CHAR(2), /* Query product ID */
  3 DSQ_PRODUCT_RELEASE CHAR(2), /* Query product release */
  3 DSQ_RESERVE1 CHAR(28), /* Reserved */
  3 DSQ_MESSAGE_ID CHAR(8), /* Completion message ID */
  3 DSQ_Q_MESSAGE_ID CHAR(8), /* Query message ID */
  3 DSQ_START_PARM_ERROR CHAR(8), /* Start parms in error */
  3 DSQ_CANCEL_IND CHAR(1), /* Cmd cancel indicator */
  /* 1 = cancelled, 0 = not cancelled */
  3 DSQ_reserve2 CHAR(23), /* Reserved */
  3 DSQ_RESERVE3 CHAR(156), /* Reserved */
  3 DSQ_MESSAGE_TEXT CHAR(128), /* QMF command message */
  3 DSQ_Q_MESSAGE_TEXT CHAR(128); /* QMF query message */

/* Return codes */
DCL
  DSQ_SUCCESS FIXED BIN(31) INIT(0) STATIC,
  DSQ_WARNING FIXED BIN(31) INIT(4) STATIC,
  DSQ_FAILURE FIXED BIN(31) INIT(8) STATIC,
  DSQ_SEVERE FIXED BIN(31) INIT(16) STATIC;

/* Communications level */
DCL
  DSQ_CURRENT_COMM_LEVEL CHAR(12) INIT('DSQL>001002<') STATIC;

/* Query product ID */

Running your programs under CICS

After you write your program in PL/I, you need to compile and run it.

When you translate, compile, and link-edit a program that uses the QMF callable interface, consider the following conditions:
• The communications area (DSQCOMML) must be available in the compile step or copied into your program.
• The QMF interface modules DSQCIPL and DSQCIPX must be available during the link-edit step of your program.

This example uses the procedure DFHVTITVL supplied with CICS. This JCL is not provided with QMF, but you can copy it and alter it to suit your needs.

//samPLI JOB
// EXEC PROC=DFHVTITVL
//PLI.SYSIN DD *

Your program or copy of QMF sample DSQABFP
.
.

Running your programs under TSO
To run your PL/I program in TSO, compile and link-edit the program, and then run it in either with or without ISPF.

Compiling and link-editing in TSO
You must compile and link-edit your PL/I program before you can run it in TSO.

This JCL uses the PL/I compiler to compile your callable interface application and then link-edits the application. Some parameters can vary from one QMF installation to the next.

//samPLI JOB
//STEP1 EXEC PROC=IBMZCPL
// Provide Access to QMF Communications Macro DSQCOMML
//PLI.SYSLIB DD DSN=QMF1210.SDSQAPE,DISP=SHR
// Provide access to QMF interface module
//PLI.SYSIN DD *

Your program or copy of QMF sample DSQABFP
.
.

/*
 // Provide access to QMF communications macro DSQCOMML
 // PLI.SYSLIB DD DSN=QMF1210.SDSQAPE,DISP=SHR
 /* Provide access to QMF interface module
 // Allocation for target library
 //LKED.SYSLMOD DD
 /* Allocation for QMF load library
 //LKED.QMLOAD DD DSN=QMF1210.SDSQLOAD,DISP=SHR
 //LKED.SYSIN DD *
 INCLUDE QMLOAD(DSQCIPL)
 INCLUDE QMLOAD(DSQCIPX)
 MODE AMODE(31) RMODE(31)
 NAME sampPLI(R)
 */
ENTRY CEESTART
MODE AMODE(31) RMODE(ANY)
NAME sampPLI(R)
/

Running in TSO without ISPF
After your PL/I program is assembled successfully, you can run it without ISPF.

Run your program in TSO without ISPF by writing a program similar to the CLIST shown here:

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 "QMF1210.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('QMF1210.SDSQCLTE')
ALLOC FI(SYSEXEC) DA('QMF1210.SDSQEXCE')
/**************************************************************************/
/* QMF/GDDM Datasets */
**************************************************************************/
ALLOC FI(ADMGGMAP) DA('QMF1210.SDSQMAPE') SHR REUSE
ALLOC FI(ADMCFORM) DA('QMF1210.DSQCFORM') SHR REUSE
ALLOC FI(DSQUCFRM) DA('QMF1210.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('QMF1210.DSQPNLE') SHR REUSE
/**************************************************************************/
/* Start your program using TSO CALL command */
**************************************************************************/
CALL sampPLI
EXIT CODE(0)

Running in TSO under ISPF
After your PL/I program is assembled successfully, you can run it under ISPF.

Run your program in TSO under ISPF by writing a program similar to the CLIST shown here:

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)
The EXIT CODE(4) statement suppresses the ISPF disposition panel.

REXX language interface

REXX is an interpretive language; it does not have to be compiled.

However, programs that use compiled REXX or other compiled languages have better performance than the same programs written that use interpretive REXX. A REXX compiler is available for REXX programs, but not for procedures with logic.

Under TSO, start QMF with the REXX callable interface when you use procedures with logic and certain QMF form functions (calculations, defined columns, and conditions). The REXX callable interface can reduce resources required to use REXX services.

For example, fewer resources are required to issue PRINT REPORT or BOTTOM commands on the REPORT panel if QMF is started with the REXX callable
interface. The reduction of resource consumption can be substantial and is most noticeable when running QMF under TSO.

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.

**Restriction:** Because REXX is not available under QMF for CICS, the QMF callable interface for REXX does not work under CICS.

**Related concepts:**

- Chapter 5, “ADDRESS QRW and the QMF command environment,” on page 29

When QMF is started in TSO, ISPF, or native z/OS, QMF creates a REXX command environment called QRW. When you are executing a REXX program, you can set the default command environment to QRW by issuing the REXX ADDRESS command ADDRESS QRW. With ADDRESS QRW, QMF remains the default command environment until you issue another ADDRESS command.

**Interface communications variables for REXX**

The interface communications variables consist of several REXX variables. They are set after the completion of every call and must not be altered by the calling program.

The interface communications variables for REXX variables are shown in this table:

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dsq_return_code</td>
<td>Integer that indicates the results of executing a QMF command</td>
</tr>
<tr>
<td></td>
<td>Possible values are:</td>
</tr>
<tr>
<td></td>
<td>- dsq_success: Successful processing of the request</td>
</tr>
<tr>
<td></td>
<td>- dsq_warning: Normal completion with warnings</td>
</tr>
<tr>
<td></td>
<td>- dsq_failure: Command did not process correctly</td>
</tr>
<tr>
<td></td>
<td>- dsq_severe: Severe error that forces the QMF session to end</td>
</tr>
<tr>
<td></td>
<td>Additional calls to QMF cannot be made with this instance ID.</td>
</tr>
<tr>
<td></td>
<td>The value of dsq_return_code is also placed in the REXX variable rc.</td>
</tr>
<tr>
<td>dsq_instance_id</td>
<td>Identifier that is established by QMF during processing of the START command</td>
</tr>
<tr>
<td>dsq_product</td>
<td>Identifies the IBM query product in use</td>
</tr>
<tr>
<td></td>
<td>Variables that begin with dsq_qmf specify QMF for TSO versions.</td>
</tr>
<tr>
<td>dsq_product_release</td>
<td>Release level of QMF in use</td>
</tr>
<tr>
<td></td>
<td>Variable dsq_qmf_v12r1 specifies QMF Version 12 Release 1.</td>
</tr>
<tr>
<td>dsq_message_id</td>
<td>Completion message ID</td>
</tr>
</tbody>
</table>
Table 49. Interface communications variables for REXX (continued)

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dsq_q_message_id</td>
<td>Query message ID</td>
</tr>
<tr>
<td>dsq_start_parm_error</td>
<td>Parameter in error when START failed due to a parameter error</td>
</tr>
<tr>
<td>dsq_cancel_ind</td>
<td>Command cancel indicator that indicates whether the user canceled command processing while QMF was running a command; possible values are:</td>
</tr>
<tr>
<td></td>
<td><strong>dsq_cancel_yes</strong></td>
</tr>
<tr>
<td></td>
<td>The user canceled the command</td>
</tr>
<tr>
<td></td>
<td><strong>dsq_cancel_no</strong></td>
</tr>
<tr>
<td></td>
<td>The user did not cancel the command</td>
</tr>
<tr>
<td>dsq_message_text</td>
<td>Completion message text</td>
</tr>
<tr>
<td>dsq_q_message_text</td>
<td>Query message text</td>
</tr>
</tbody>
</table>

**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 QMF commands issued through the callable interface.

The syntax for the DSQCIX function call is as follows:

```rexx
call DSQCIX cmd parmlist
```

In this syntax, `cmd` is a QMF command written as an uppercase character string and `parmlist` is a list of parameter and value pairs.

**Syntax of the parameter list for the DSQCIX function call**

```rexx
parmname=value,
```

Pass the entire command, including the `parmlist`, 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, do not enclose the argument. For example:

```rexx
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`

Here are some examples of function calls:

```rexx
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 (which is not required)
- Equal sign (=) between parmname and value (a space produces the same result)

Each of the following would produce the same results:

```plaintext
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 sample REXX program, DSQABFX, is provided with QMF.

You can look at the sample source code listing here or you can access it online. The sample program is a member of the library QMF1210.SDSQEXCE.

The sample program for the REXX callable interface performs the following functions:

- Starts QMF
- Sets three global variables
- Runs a query called Q1
- Prints the resulting report by using form F1
- Ends the QMF session

QMF does not supply query Q1 or form F1, but the sample program uses these objects.

```plaintext
/*REXX*******************************************************************/
/* Sample program:  DSQABFX                                          */
/* REXX version of the QMF 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

//************************************************************************
```

Chapter 11. Programming language specifications for using the callable interface  193
Run your REXX programs

After you write your program in REXX, you need to run it.

You can run your REXX program in TSO by writing a program similar to the one shown here:

/* 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 UDUMPCB LRECL(125) RECFM(V B A) BLKSIZE(1632)"
"ATTR EDITDCB LRECL(79) RECFM(F B A) BLKSIZE(4029)"
"ALLOC FI(SYSproc) SHR REUSE",
"DA('QMF1210.DSQCLSTE,'",
 " 'DSN.DSNCLIST')"
"ALLOC FI(SYSexec) SHR REUSE",
"DA('QMF1210.SDSQEXEC')"
"ALLOC FI(ISPLLIB) SHR REUSE",
"DA('QMF1210.SDSQLOAD',"
 " 'ADM.GDDM.GDDMLOAD','"
 " 'ADM.GDDM.GDDMLOAD')"
"ALLOC FI(DSQPNLE) DA('QMF1210.DSQPNLE') SHR REUSE"
"ALLOC FI(DSQPRINT) SYST USING(PRINTDCB)"
"ALLOC FI(SYSPrTR) SYST OUT(X) LRECL(132) RECFM(FBA) BLKSIZE(132)"
"ALLOC FI(DSQDEBUG) SYST OUT(X) USING(DEBUGDCB)"
"ALLOC FI(DSQUDUMP) SYST OUT(X) USING(UDUMPCB)"
"ALLOC FI(DSQSPILL) NEW UNIT(SYSDA) SPACE(1,1) TRACKS"
"ALLOC DDNAME(DSQEDIT) UNIT(SYSDA) NEW USING(EDEDCB)"
"ALLOC FI(ADMGDF) DA('ADM.GDDM.NICKNAME') SHR REUSE"
"ALLOC FI(ADMGGMAP) DA('QMF1210.SDSQMAPE') SHR REUSE"
"ALLOC FI(ADMCFORM) DA('QMF1210.DSQCHART') SHR REUSE"
"ALLOC FI(DSQUCFRM) DA('GDDM.ADMGDF') SHR REUSE"
"ALLOC FI(ADMSYMBL) DA('ADM.GDDM.GDDMSYM') SHR REUSE"

/* The beginning of your REXX program ....... */

A REXX example of using an INTERACT loop

You can make the END command in an interactive session behave similarly to the way END behaves in interactive QMF.
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.

Add the logic in the following example to your program to allow the END command to behave similarly to 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 */
/******************************getChildID return_code pathological**********/
;/* 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
&view GLOBAL to show panel IDs
/*****************************getChildID return_code pathological**********/
call dsqcix "SET GLOBAL (DSQDC_SHOW_PANID=1"
if dsq_return_code = dsq_severe then exit dsq_return_code
&view GLOBAL to show panel IDs
/******************************getChildID return_code pathological**********/
call dsqcix "MESSAGE (TEXT='OK, you may enter a command.')"
if dsq_return_code = dsq_severe then exit dsq_return_code
&view GLOBAL to show panel IDs
/******************************getChildID return_code pathological**********/
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
```
## Appendix A. Product interface macros

This table lists macros that are provided with QMF as General Use Programming Interfaces for customers.

**Important**: Do not use any QMF macros as programming interfaces other than those macros identified here.

### Table 50. Macros that provide interfaces to QMF functions

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Macro names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product interface macro</td>
<td>DSQMF&lt;sub&gt;n&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td>In this program name, &lt;sub&gt;n&lt;/sub&gt; is a national language identifier). For English, this identifier is E.</td>
</tr>
<tr>
<td>Callable interface macros</td>
<td>• Assembler</td>
</tr>
<tr>
<td></td>
<td>- DSQCLA</td>
</tr>
<tr>
<td></td>
<td>- DSQCOMMA</td>
</tr>
<tr>
<td></td>
<td>• COBOL</td>
</tr>
<tr>
<td></td>
<td>- DSQCLB</td>
</tr>
<tr>
<td></td>
<td>- DSQCOMMB</td>
</tr>
<tr>
<td></td>
<td>• C/C++</td>
</tr>
<tr>
<td></td>
<td>- DSQCLC</td>
</tr>
<tr>
<td></td>
<td>- DSQCLICE</td>
</tr>
<tr>
<td></td>
<td>- DSQCOMMC</td>
</tr>
<tr>
<td></td>
<td>• Fortran</td>
</tr>
<tr>
<td></td>
<td>- DSQCLF</td>
</tr>
<tr>
<td></td>
<td>- DSQCLIFE</td>
</tr>
<tr>
<td></td>
<td>- DSQCOMMF</td>
</tr>
<tr>
<td></td>
<td>• PL/I</td>
</tr>
<tr>
<td></td>
<td>- DSQCLPL</td>
</tr>
<tr>
<td></td>
<td>- DSQCLIX</td>
</tr>
<tr>
<td></td>
<td>• REXX</td>
</tr>
<tr>
<td></td>
<td>- DSQCLX</td>
</tr>
<tr>
<td>Command interface macro</td>
<td>DSQCCI</td>
</tr>
<tr>
<td>QMF governor exit routine interface macros</td>
<td>• DXEGOVA</td>
</tr>
<tr>
<td>QMF user edit exit routine macro</td>
<td>DXEECS</td>
</tr>
</tbody>
</table>

**Related reference:**

“Conventions for National Language Feature information” on page 4

Db2 QMF is available in several different languages, each of which is provided by a National Language Feature (NLF).
Appendix B. QMF global variables

QMF provides many global variables that help you control aspects of your QMF session, QMF commands, and panel display. The global variables also help you control behavior of QMF functions in procedures and applications.

Naming convention for QMF global variables

The naming convention for most global variables that are provided with QMF is DSQcc_xxxxxxxxxxx. cc identifies the category of variable, and xxxxxxxxxx is a descriptive name up to 12 characters long. An underscore character (_) is included after cc.

c can be any one of the following identifiers:

- AP Variables for profile-related state information
- AO Variables for other (not profile-related) state information
- CM Variables for information about the message produced by the previous command
- CP Variables for information about the Table Editor
- DC Variables that control how QMF displays information displayed on the screen
- EC Variables that control how QMF executes commands and procedures
- QC Variables whose values are produced by a CONVERT QUERY option
- QM Variables that contain RUN QUERY error message information
- QW Variables unique to QMF for Workstation

Session variables

Session variables follow a different naming convention. Session variables are global variables that store the values that users enter in some fields on some panels if the DSQEC_SESSGLV_SAV global variable is set to 1 or 2. The naming convention for session variables is as follows:

\[ \text{DXYnnnppp_in_dd} \]

where:

- \( n \) is the national language identifier
- \( nppp \) is the last four letters of the panel ID
- \( in \) is an ID that is associated with the field
- \( dd \) is an ID that is associated with the field and is used only if the field is dependent on another field

Setting and displaying values for global variables

If the value you want to assign to a global variable is 55 or fewer bytes, use the SET GLOBAL command to assign the value. If the variable is over 55 bytes, use the SHOW GLOBALS command.
About this task

By default, a global variable value is retained until you reset it or end the QMF session. However, the DSQEC_USERGLV_SAV global variable can be set to save global variable values from one session to another.

To customize global variables at initialization, see the information in Installing and Managing Db2 QMF for TSO and CICS about initializing global variables and QMF session behavior when QMF starts.

Procedure

To assign a value that is over 55 bytes to a global variable:
1. Use the SHOW GLOBALS command to display the GLOBALS panel.
2. Press the Show Field key to display the entire entry field. The maximum length for a global variable on the Show Global Variable screen is 32,768 bytes.
3. Type the value for the variable on the lines provided.

Global variables for state information not related to the profile

DSQAO global variables contain status information or settings of parameters or flags. None of these global variables can be modified by the SET GLOBAL command.

Table 51. Global variables for state information not related to the profile

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQAO_APPL_TRACE</td>
<td>DSQATRAC</td>
<td>01</td>
<td></td>
</tr>
<tr>
<td>DSQAO_ATTENTION</td>
<td>DSQCATTN</td>
<td>01</td>
<td>User attention flag.</td>
</tr>
<tr>
<td>DSQAO_BATCH</td>
<td>DSQABATC</td>
<td>01</td>
<td>Batch or interactive mode; values can be: 1 for an interactive session, 2 for a batch-mode session</td>
</tr>
<tr>
<td>DSQAO_CONNECT_ID</td>
<td>DSQAAUTH</td>
<td>128</td>
<td>The user ID used to connect to the database and under which work is done. The value of this variable changes when you issue the following command or statement: • Issue a QMF CONNECT command to reconnect to the database under a different authorization ID • Issue a SET CURRENT SQLID statement on a DB2 for z/OS database.</td>
</tr>
<tr>
<td>DSQAO_CONNECT_LOC</td>
<td>None</td>
<td>18</td>
<td>The location name of the database to which you are currently connected; the name is 16 characters (padded to the right with blanks, if necessary).</td>
</tr>
<tr>
<td>DSQAO_CURSOR_OPEN</td>
<td>DSQACRSR</td>
<td>01*</td>
<td>Database cursor status; values can be: 1 if the cursor is open, 2 if the cursor is closed</td>
</tr>
</tbody>
</table>
Table 51. Global variables for state information not related to the profile (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQAO_DATE_FORMAT</td>
<td>None</td>
<td>05</td>
<td>Contains the value that is specified in SYSIBM.DATE_FORMAT. Values can be ISO, USA, EUR, JIS, or LOCAL.</td>
</tr>
<tr>
<td>DSQAO_DB_MANAGER</td>
<td>DSQADBMG</td>
<td>01</td>
<td>Database manager, indicated by one of the following values: 1. DB2 for VSE and VM 2. Db2 for z/OS 3. Db2 for Linux, UNIX, and Windows 4. Db2 for iSeries</td>
</tr>
<tr>
<td>DSQAO_DBCS</td>
<td>DSQADBCS</td>
<td>01</td>
<td>DBCS support status; values can be: 1. for DBCS support 2. for no DBCS support</td>
</tr>
<tr>
<td>DSQAO_DSQSBSSTG</td>
<td>None</td>
<td>10</td>
<td>Contains the value specified by the DSQSBSSTG parameter or the default if the parameter was not specified.</td>
</tr>
<tr>
<td>DSQAO_DSQSFISO</td>
<td>None</td>
<td>01</td>
<td>Contains the value that is specified by the DSQSFISO parameter or the default if the parameter was not specified. The following values are used: 0. QMF is not precompiled with DATE(ISO) and TIME(ISO). 1. QMF is precompiled with DATE(ISO) and TIME(ISO). This is the default.</td>
</tr>
<tr>
<td>DSQAO_DSQSMRFI</td>
<td>None</td>
<td>01</td>
<td>This field reflects the value that was specified for the DSQSMRFI program parameter when QMF was started. 0. NO was specified for the DSQSMRFI program parameter, meaning that Db2 single-row fetch and insert is used. 1. YES was specified for the DSQSMRFI program parameter, meaning that Db2 multirow fetch and insert is used. Multirow fetch uses a rowset cursor.</td>
</tr>
<tr>
<td>DSQAO_DSQSMTHD</td>
<td>None</td>
<td>01</td>
<td>Contains the value specified by the DSQSMTHD program parameter or the default if the parameter was not specified. The following values are used: 0. NO was specified; QMF runs with one thread. This is the default. 1. YES was specified; QMF will run with a second thread that will be used for commands (RUN QUERY, DISPLAY TABLE) and subsequent scrolling (BOTTOM, TOP, FORWARD, BACKWARD, RIGHT and LEFT) of reports with open cursors.</td>
</tr>
</tbody>
</table>
Table 51. Global variables for state information not related to the profile (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQAO_DSQSPILL</td>
<td>None</td>
<td>01</td>
<td>Contains the value specified by the DSQSPILL parameter or the default if the parameter was not specified. The following values are used: 0 for not using spill storage. This value corresponds with a DSQSPILL parameter value of NO. 1 for using spill storage. This value corresponds with a DSQSPILL parameter value of YES.</td>
</tr>
<tr>
<td>DSQAO_DSQSPTYP</td>
<td>None</td>
<td>5</td>
<td>Contains the value specified by the DSQSPTYP parameter or the default if the parameter was not specified. The following values are used: FILE for spilling data to a file. 64BIT for spilling data to extended virtual storage.</td>
</tr>
<tr>
<td>DSQAO_DSQSRSTG</td>
<td>None</td>
<td>8</td>
<td>Contains the value specified by the DSQSRSTG parameter or the default if the parameter was not specified.</td>
</tr>
<tr>
<td>DSQAO_FORM_PANEL</td>
<td>DSQASUBP</td>
<td>02</td>
<td>Current form panel; values can be: 1 for FORM.MAIN 2 for FORM.COLUMNS 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 that the form does not exist in QMF temporary storage.</td>
</tr>
<tr>
<td>DSQAO_INTERACT</td>
<td>DSQAIACT</td>
<td>01</td>
<td>Setting of the interact flag; values can be: 0 for no interactive execution 1 when interactive execution is allowed</td>
</tr>
</tbody>
</table>
Table 51. Global variables for state information not related to the profile  (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQAO_LOCAL_DB2</td>
<td>None</td>
<td>18</td>
<td>The location name of the local Db2 for z/OS database. This value 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).</td>
</tr>
<tr>
<td>DSQAO_LOCATION</td>
<td>DSQAITLO</td>
<td>18</td>
<td>Location name of the current object, if any. This value is applicable only if a three-part name was used. The name is 16 characters (padded to the right with blanks, if necessary).</td>
</tr>
<tr>
<td>DSQAO_NLF_LANG</td>
<td>DSQALANG</td>
<td>01</td>
<td>National language of user; for the English language environment, this value is ‘E’.</td>
</tr>
<tr>
<td>DSQAO_NUM_FETCHED</td>
<td>DSQAROWS</td>
<td>16</td>
<td>Fetched data rows; contains ‘0’ when the DATA object is empty.</td>
</tr>
<tr>
<td>DSQAO_OBJ_NAME</td>
<td>DSQAITMN</td>
<td>128</td>
<td>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.</td>
</tr>
<tr>
<td>DSQAO_OBJ_OWNER</td>
<td>DSQAITMO</td>
<td>128</td>
<td>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.</td>
</tr>
<tr>
<td>DSQAO_OTC_LICENSE</td>
<td>None</td>
<td>01</td>
<td>Indicates whether the Db2 QMF for z/OS OTC standalone product is installed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 Indicates that the product identifier for the Db2 QMF for z/OS OTC standalone product, 5697-QMF, was not found.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 Indicates that the product identifier for the Db2 QMF for z/OS standalone product, 5697-QMF, was found.</td>
</tr>
<tr>
<td>DSQAO_PANEL_TYPE</td>
<td>DSQAITEM</td>
<td>01</td>
<td>Type of current panel; values can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 for HOME</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 for QUERY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 for REPORT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4 for FORM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5 for PROC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6 for PROFILE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7 for CHART</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8 for LIST</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9 for Table Editor</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A for GLOBALS</td>
</tr>
<tr>
<td>Callable interface variable name</td>
<td>Command interface variable name</td>
<td>Length</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>DSQAO_QMF_RELEASE</td>
<td>DSQAREVN</td>
<td>02</td>
<td>Numeric release number of QMF, which is displayed in header records for exported forms, reports, and prompted queries. For QMF Version 12 Release 1, this value is '19'.</td>
</tr>
<tr>
<td>DSQAO_QMF_VER_RLS</td>
<td>DSQAQMFM</td>
<td>10</td>
<td>Version and release of QMF. For QMF Version 12 Release 1, this value is 'QMFV12R1.0'.</td>
</tr>
<tr>
<td>DSQAO_QMFADM</td>
<td>None</td>
<td>01</td>
<td>QMF administrator authority:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 Current authorization ID does not have QMF administrator authority.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 Current authorization ID has administrator authority.</td>
</tr>
<tr>
<td>DSQAO_QRY_SUBTYPE</td>
<td>DSQASUBI</td>
<td>01</td>
<td>Query subtype; values can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 for a subtype of SQL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 for a subtype of QBE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 for a subtype of PROMPTED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Blank means that the current panel is not QUERY.</td>
</tr>
<tr>
<td>DSQAO_QUERY_MODEL</td>
<td>DSQAMODL</td>
<td>01</td>
<td>Model of current query; value can be only '1' (for relational data model).</td>
</tr>
<tr>
<td>DSQAOSAME_CMD</td>
<td>DSQACMDM</td>
<td>01</td>
<td>Values can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 if the two commands are not the same</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 if the two commands are the same</td>
</tr>
<tr>
<td>DSQAO_STO_PROC_INT</td>
<td>None</td>
<td>01</td>
<td>Shows whether QMF for TSO was started as a Db2 for z/OS stored procedure. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 QMF was not started as a stored procedure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 QMF was started as a stored procedure.</td>
</tr>
<tr>
<td>DSQAO_SUBSYS_ID</td>
<td>None</td>
<td>04</td>
<td>If QMF is running in TSO, this value is the ID of the local Db2 subsystem to which QMF is attached.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If you specify a value for the DSQSSUBS program parameter in CICS, this global variable contains that value. The parameter is tolerated and the value is not processed. The value is placed in the global variable field and nothing is done with it. This logic permits the same program to be used in multiple environments.</td>
</tr>
<tr>
<td>DSQAO_SYSTEM_ID</td>
<td>DSQASYST</td>
<td>01</td>
<td>Current operating system; values can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 TSO under z/OS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 TSO or native z/OS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5 CICS</td>
</tr>
<tr>
<td>DSQAO_TERMINATE</td>
<td>DSQCSESC</td>
<td>01</td>
<td>QMF termination flag; values can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 if the session was not marked for termination</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 if the session was marked for termination</td>
</tr>
<tr>
<td>Callable interface variable name</td>
<td>Command interface variable name</td>
<td>Length</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>DSQAO_TIME_FORMAT</td>
<td>None</td>
<td>05</td>
<td>Contains the value that is specified in SYSIBM.TIME_FORMAT. Values can be ISO, USA, EUR, JIS, or LOCAL.</td>
</tr>
<tr>
<td>DSQAO_VARIATION</td>
<td>DSQAVARN</td>
<td>02</td>
<td>Form panel variation number; blank means FORM.DETAIL is not the current panel.</td>
</tr>
</tbody>
</table>

### Global variables for profile-related state information

DSQAP global variables store information related to QMF profile settings. None of these global variables can be modified by the SET GLOBAL command.

#### Table 52. Global variables for profile-related state information

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQAP_CASE</td>
<td>DSQAPCAS</td>
<td>01</td>
<td>CASE parameter; values can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 for UPPER</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 for MIXED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 for STRING</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If your site uses RACF support for mixed-case passwords under TSO, set this value to 2. Without this setting, all input (including passwords) is converted to uppercase, causing the CONNECT command to fail. When you set CASE to MIXED, ensure that you enter all input in uppercase, because QMF recognizes commands only in uppercase.</td>
</tr>
<tr>
<td>DSQAP_CONFIRM</td>
<td>DSQAPRMP</td>
<td>01</td>
<td>CONFIRM parameter; values can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 for NO</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 for YES</td>
</tr>
<tr>
<td>DSQAP_DECIMAL</td>
<td>DSQAPDEC</td>
<td>01</td>
<td>DECIMAL parameter; values can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 for PERIOD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 for COMMA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 for FRENCH</td>
</tr>
<tr>
<td>DSQAP_LENGTH</td>
<td>DSQAPLEN</td>
<td>18</td>
<td>LENGTH parameter; its value is that of the parameter (‘1’ through ‘999’ or ‘CONT’).</td>
</tr>
<tr>
<td>DSQAP_PFKEY_TABLE</td>
<td>DSQAPPPFK</td>
<td>31</td>
<td>Name of the function keys table.</td>
</tr>
<tr>
<td>DSQAP_PRINTER</td>
<td>DSQAPPRRT</td>
<td>08</td>
<td>PRINTER parameter; values can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• A nickname for a GDDM printer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Blanks for the printer associated with DSQPRINT.</td>
</tr>
<tr>
<td>DSQAP_QUERY_LANG</td>
<td>DSQAPLNG</td>
<td>01</td>
<td>LANGUAGE parameter; values can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 for SQL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 for QBE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 for PROMPTED</td>
</tr>
<tr>
<td>DSQAP_QUERY_MODEL</td>
<td>DSQAMODP</td>
<td>01</td>
<td>MODEL parameter (value is ‘1’ for relational).</td>
</tr>
</tbody>
</table>
Table 52. Global variables for profile-related state information  (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQAP_RESOURC_GRP</td>
<td>DSQAPGRP</td>
<td>16</td>
<td>RESOURCE GROUP parameter.</td>
</tr>
<tr>
<td>DSQAP_SPACE</td>
<td>DSQAPSPC</td>
<td>50</td>
<td>SPACE parameter; its value is that of the parameter.</td>
</tr>
<tr>
<td>DSQAP_SYNONYM_TBL</td>
<td>DSQAPSYN</td>
<td>31</td>
<td>Name of the synonyms table used for the current QMF session. When a user enters a command synonym, the synonym definition must be stored in the table named here or the command fails.</td>
</tr>
<tr>
<td>DSQAP_TRACE</td>
<td>DSQAPTRC</td>
<td>18</td>
<td>TRACE parameter; values can be: ALL (maximum tracing) NONE (minimum tracing) You can also specify a series of letters and numbers that specifies which components are to be traced at which levels of detail (for example, A2L2C1).</td>
</tr>
<tr>
<td>DSQAP_WIDTH</td>
<td>DSQAPWID</td>
<td>18</td>
<td>WIDTH parameter; its value is that of the parameter ('22' through '999').</td>
</tr>
</tbody>
</table>

Global variables associated with CICS

DSQAP global variables are associated with CICS environments. Only DSQAP_CICS_PQNAME and DSQAP_CICS_PQTYPE can be modified by the SET GLOBAL command.

When the queue type is transient data (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.

Table 53. Global variables associated with the CICS environment

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQAP_CICS_PQNAME</td>
<td>None</td>
<td>08</td>
<td>Names the CICS data queue to contain the QMF print output.</td>
</tr>
<tr>
<td>DSQAP_CICS_PQTYPE</td>
<td></td>
<td></td>
<td>Type of CICS storage used to contain the QMF print output: T5 Writes the QMF print to a CICS temporary storage queue on an auxiliary storage device. This value is the default. T0 Writes the QMF print to a CICS transient data queue.</td>
</tr>
<tr>
<td>DSQAO_CICS_SQNAME</td>
<td>None</td>
<td>08</td>
<td>Names the CICS data queue to be used as the spill file.</td>
</tr>
</tbody>
</table>
Table 53. Global variables associated with the CICS environment (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQAO_CICS_SQTYPE</td>
<td>None</td>
<td>02</td>
<td>Type of CICS storage used to contain the QMF spill file:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>TS</strong> Writes the QMF spill data to a CICS temporary storage queue on an auxiliary storage device. This value is the default.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>TD</strong> Writes the QMF spill data to a CICS transient data queue.</td>
</tr>
<tr>
<td>DSQAO_CICS_TQNAME</td>
<td>None</td>
<td>08</td>
<td>Names the CICS data queue to contain the QMF trace data.</td>
</tr>
<tr>
<td>DSQAO_CICS_TQTYPE</td>
<td>None</td>
<td>02</td>
<td>Type of CICS storage used to contain the QMF trace data:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>TS</strong> Writes the QMF trace to a CICS temporary storage queue on an auxiliary storage device.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>TD</strong> Writes the QMF trace to a CICS transient data queue. This value is the default.</td>
</tr>
</tbody>
</table>

Global variables related to a message produced by the most recent command

DSQCM global variables contain information about the most recent QMF command that was issued. None of these global variables can be modified by the SET GLOBAL command.

Table 54. Global variables that capture information about the most recently issued command

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQCM_MESSAGE</td>
<td>DSQCM_MESSAGE</td>
<td>80</td>
<td>Message text.</td>
</tr>
<tr>
<td>DSQCM_MESSAGE_ALL</td>
<td>DSQCIMSA</td>
<td>360</td>
<td>Complete message text.</td>
</tr>
<tr>
<td>DSQCM_MSG_HELP</td>
<td>DSQCIMID</td>
<td>08</td>
<td>ID of message help panel.</td>
</tr>
<tr>
<td>DSQCM_MSG_NUMBER</td>
<td>DSQCIMNO</td>
<td>08</td>
<td>Message number.</td>
</tr>
<tr>
<td>DSQCM_SUB_TXT_n</td>
<td>DSQCIMnn</td>
<td>20</td>
<td>Substitution value nn.</td>
</tr>
</tbody>
</table>

Global variables associated with the Table Editor

DSQCP global variables are associated with the operations of the Table Editor. All of these global variables can be modified by the SET GLOBAL command.

The following table shows global variables that are associated with the operations of the 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 that are 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.

### Table 55. Global variables associated with the Table Editor

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
</table>
| DSQCP_RMV_BLANKS                | None                            | 01     | Retains or removes trailing blanks of VARCHAR columns. This variable affects only the Table Editor in Change mode. Values can be:
|                                 |                                 |        | 0  Trailing blanks of VARCHAR columns are not removed. 1  Trailing blanks of VARCHAR columns are removed. This value is the default. |
| 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 value 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 value 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 value 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 4-byte mixed string, composed of one DBCS character, preceded by the shift-out character, and followed by the shift-in character. It is initially set to a DBCS plus sign (+) character. This global variable is used only in a DBCS environment. |
Table 55. Global variables associated with the Table Editor (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQCP_TEEND</td>
<td>None</td>
<td>01</td>
<td>Displays a confirmation panel when you issue an END subcommand or a CANCEL subcommand to terminate a Table Editor subsession. The panel can be displayed in several variations: • If END or CANCEL is issued • If modifications are made to the database • If the screen contains modified data when END or CANCEL is 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 value is the default.</td>
</tr>
<tr>
<td>DSQCP_TEMOD</td>
<td>None</td>
<td>01</td>
<td>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.</td>
</tr>
<tr>
<td>DSQCP_TENULL</td>
<td>None</td>
<td>01</td>
<td>The reserved character used to indicate the null value for a column in the Table Editor; initially set to a hyphen (-) character.</td>
</tr>
<tr>
<td>DSQCP_TENULL_DBCS</td>
<td>None</td>
<td>04</td>
<td>The reserved DBCS character used to indicate the null value for a graphic-string column in the Table Editor. The character is also used to indicate ignore in the context of search criteria. The value must be a 4-byte mixed string composed of one DBCS character, preceded by the shift-out character, and followed by the shift-in character. It is initially set to a DBCS hyphen (-) character. This global variable is used only in a DBCS environment.</td>
</tr>
</tbody>
</table>
Global variables that control various displays

DSQDC global variables control the display of certain kinds of information. All of these global variables can be modified by the SET GLOBAL command.

Table 56. Global variables that control the display of certain types of information

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQDC_COL_LABELS</td>
<td>None</td>
<td>01</td>
<td>Controls whether the column heading shown in FORM.MAIN and FORM.COLUMNS defaults to the database label assigned to the column or the name of the column in the table from which it was selected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>DSQDC_COST_EST</td>
<td>None</td>
<td>01</td>
<td>Controls the display of the database cost estimate; values can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>DSQDC_CURRENCY</td>
<td>None</td>
<td>18</td>
<td>The currency symbol used when the DC edit code is specified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If you require a currency symbol that is not represented on the keyboard, you can still specify the symbol. Set the DSQDC_CURRENCY variable in a procedure with logic to the hex value that is equivalent to the correct symbol. For example, the following procedure sets the currency symbol to HEX '9F', which specifies the euro currency symbol in English QMF:</td>
</tr>
</tbody>
</table>
|                                 |                                 |        | /* */
<p>|                                 |                                 |        | &quot;SET GLOBAL (DSQDC_CURRENCY =&quot; '9F' ')' |
|                                 |                                 |        | If trailing blanks are needed for the currency symbol, put the currency symbol in single quotation marks. This example shows the blanks for French QMF: |
|                                 |                                 |        | SET GLOBAL (DSQDC_CURRENCY = 'FR' ' |
|                                 |                                 |        | You can issue this command from the command line or in a linear procedure. |</p>
<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQDC_DISPLAY_RPT</td>
<td>DSQADPAN</td>
<td>01</td>
<td>Displays a report after <code>RUN QUERY</code>; values can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0  QMF does not display the resulting report from a <code>RUN QUERY</code> command. This value is the default if QMF is started interactively with DSQQMF (where n is a National Language Feature identifier). This value is also the default if QMF is started in batch mode. Changing this variable when QMF is started in batch mode does not cause any QMF screen to display.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1  QMF automatically displays the report. This value is the default if QMF is started with the callable interface. The value can be overridden with the <code>DSQADPAN</code> program parameter on the <code>START</code> command.</td>
</tr>
</tbody>
</table>

When setting this global variable to 1, you can review the displayed report and choose whether to commit or roll back changes. To do this, press F3 (END) when you have finished reviewing your changes. You will then be prompted to either commit or roll back changes. Select 1 to commit or 2 to roll back your changes and then press Enter.
Table 56: Global variables that control the display of certain types of information (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQDC_EC_CHAR</td>
<td>None</td>
<td>05</td>
<td>User-defined default edit code for character data (fixed character, varying character, and very long character).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CW</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CDx</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Uxxxx</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Vxxxx</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BW</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>XM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M</td>
</tr>
</tbody>
</table>
Table 56. Global variables that control the display of certain types of information  (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQDC_EC_DATE</td>
<td>None</td>
<td>05</td>
<td>Default edit code for DATE data. Values can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TDYx</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Four-digit year with year first.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TDMx</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Four-digit year with month first.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TDDx</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Four-digit year with day first.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TD(Y)ax</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Abbreviated two-digit year with year first.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TD(Y)Mx</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Abbreviated two-digit year with month first.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TD(Y)Dx</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Abbreviated two-digit year with day first.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TDL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Locally defined date format.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Default date format of the database system. This is the default value for this global variable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Represents the character that you specify to serve as the delimiter between parts of the date.</td>
</tr>
<tr>
<td>Callable interface variable name</td>
<td>Command interface variable name</td>
<td>Length</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>DSQDC_EC_DEC</td>
<td>None</td>
<td>05</td>
<td>User defined default edit code for decimal data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>E or EZ</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Scientific notation. A Z in the second position of the edit code suppresses zero values.</td>
</tr>
<tr>
<td>D, DC, DZ, DZC, I, IZ, J, JZ, K, KZ, L, LZ, P, PZ</td>
<td></td>
<td></td>
<td>Decimal notation with different combinations of leading zeros, minus signs for negative numbers, thousands separators, currency symbols, and percent signs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Each code can be followed by a number (from 0 to 99) that tells how many places to allow after the decimal point.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A C in the second or third position of the D edit code displays a user-defined currency symbol instead of the standard currency symbol.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A Z in the second position of the edit code suppresses zero values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The default value is L. When L* is specified, QMF will format decimal data based on the column definition of the database. This behavior is consistent with previous releases of QMF.</td>
</tr>
<tr>
<td>Uxxxx</td>
<td></td>
<td></td>
<td>User-defined formatting. Replace xxxx with 0 - 4 characters (letters, digits, or special characters).</td>
</tr>
<tr>
<td>Vxxxx</td>
<td></td>
<td></td>
<td>User-defined formatting. Replace xxxx with 0 - 4 characters (letters, digits, or special characters).</td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td>Displays metadata (data type and length) instead of the actual data.</td>
</tr>
</tbody>
</table>
Table 56. Global variables that control the display of certain types of information (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQDC_EC_NUM</td>
<td>None</td>
<td>05</td>
<td>User-defined default edit code for numeric data (integer, small integer, and big integer.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>E or EZ</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Scientific notation. A Z in the second position of the edit code suppresses zero values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>D, DC, DZ, DZC, I, IZ, J, JZ, K, KZ, L, LZ, P, PZ</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Decimal notation with different combinations of leading zeros, minus signs for negative numbers, thousands separators, currency symbols, and percent signs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A C in the second or third position of the D edit code displays a user-defined currency symbol instead of the standard currency symbol.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A Z in the second position of the edit code suppresses zero values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The default value is L.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Uxxxx</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>User-defined formatting. Replace xxxx with 0 - 4 characters (letters, digits, or special characters).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Vxxxx</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>User-defined formatting. Replace xxxx with 0 - 4 characters (letters, digits, or special characters).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>M</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Displays metadata (data type and length) instead of the actual data.</td>
</tr>
<tr>
<td>Callable interface variable name</td>
<td>Command interface variable name</td>
<td>Length</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| DSQDC_EC_TIME                   | None                            | 05     | Default edit code for TIME data. Values can be:  
TTSx  
24-hour clock format (including seconds).  
TTCx  
12-hour clock format (including seconds).  
TTAx  
Abbreviated clock format (no seconds).  
TTAN  
Abbreviated clock format (no seconds, no delimiter).  
TTUX  
USA format.  
TTL  
Locally defined time format.  
TT  
Default time format of the database system. This is the default value for this global variable.  
x represents the character that you specify to serve as the delimiter between parts of the time. |
| 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 uses the default order.  
2  
The list is sorted by object owner.  
3  
The list is sorted by object name.  
4  
The list is sorted by object type.  
5  
The list is sorted by date modified.  
6  
The list is sorted by date last used. The list of commands that cause this date to be updated is set by the DSQEC_LAST_RUN global variable.  
Values for the second character can be:  
A  
The list is sorted in ascending order.  
D  
The list is 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. |
Table 56. Global variables that control the display of certain types of information (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQDC_POS_SQLCODE</td>
<td>None</td>
<td>01</td>
<td>Sets the action QMF takes when a positive SQL code is returned from the database. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>DSQDC_SCROLL_AMT</td>
<td>None</td>
<td>04</td>
<td>Sets the scroll amount for QMF panels; values can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Csr</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Half</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Page</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>n</td>
</tr>
</tbody>
</table>
### Table 56: Global variables that control the display of certain types of information (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQDC_SHORT_EXPT</td>
<td>None</td>
<td>01</td>
<td>Applies to data or tables exported with a value of QMF on the DATAFORMAT parameter of the EXPORT command. Controls the length of all column name fields in the header records. Possible values are: 0 QMF sets the length of column fields in the header records to 30 bytes. This length is the default length for: * Db2 for z/OS Version 8.1.5, or later * DB2 for iSeries Version 5.2, or later * Db2 for Linux, UNIX, and Windows, Version 8.1, or later 1 QMF sets the length of column fields in the header records to 18 bytes. This length is the default length for: * Db2 for z/OS, Version 8.1.5, or earlier * DB2 for iSeries, Version 5.2, or earlier * Db2 for Linux, UNIX, and Windows, Version 8.1, or earlier * All DB2 Server for VSE and VM databases</td>
</tr>
<tr>
<td>DSQDC_SHOW_PANID</td>
<td>DSQCPDSP</td>
<td>01</td>
<td>Displays panel IDs of QMF panels; values can be: 0 Suppresses panel identifiers. This value is the default. 1 Displays panel identifiers.</td>
</tr>
</tbody>
</table>

**Related reference:**

"Global variables that control how commands and procedures are executed"

DSQEC global variables control how commands and procedures are executed. All of these global variables can be modified by the SET GLOBAL command.

---

### Global variables that control how commands and procedures are executed

DSQEC global variables control how commands and procedures are executed. All of these global variables can be modified by the SET GLOBAL command.

**Table 57: Global variables that control how commands and procedures are executed**

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQEC_ALIASES</td>
<td>None</td>
<td>31</td>
<td>View for retrieving lists of table and view aliases when you request a list of tables from a Db2 for z/OS location. Also applies if the current server is Db2 for z/OS or Db2 for Linux, UNIX, and Windows.</td>
</tr>
<tr>
<td>DSQEC_BUFFER_SIZE</td>
<td>None</td>
<td>03</td>
<td>Sets the length of the data buffer used to fetch data from the database. Valid values range from 4 - 256 (each integer is 1KB: for example, 4 equals 4K, 256 equals 256K, etc.). The default value is 4 (4KB).</td>
</tr>
<tr>
<td>Callable interface variable name</td>
<td>Command interface variable name</td>
<td>Length</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>DSQEC_CC</td>
<td>None</td>
<td>01</td>
<td>Suppresses the carriage control characters in the report output format; values can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0  No carriage control character in column 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1  Carriage control is in effect; the report has a carriage control character in column 1.</td>
</tr>
<tr>
<td>DSQEC_COLS_LDB2</td>
<td>None</td>
<td>31</td>
<td>View for retrieving column information for a table at the current location, if that location is Db2 for z/OS.</td>
</tr>
<tr>
<td>DSQEC_COLS_RDB2</td>
<td>None</td>
<td>31</td>
<td>View for retrieving column information for a table at a remote Db2 for z/OS location (if it is not the current location).</td>
</tr>
<tr>
<td>DSQEC_COLS_SQL</td>
<td>None</td>
<td>31</td>
<td>View for retrieving column information for a table in a DB2 for VSE and VM database.</td>
</tr>
<tr>
<td>DSQEC_CON_ACC_RES</td>
<td>None</td>
<td>01</td>
<td>Applies to executable SELECT queries that QMF submits to Db2 for z/OS. Use this variable to specify how you want the database to proceed when the data to be selected is locked by an insert, update, or delete operation. When you set this variable, QMF specifies the clause associated with the variable value on the concurrent-access-resolution attribute of the PREPARE statement for the SELECT query. Executable SELECT queries can result not only from QMF queries (such as SQL SELECT queries, prompted queries, or QBE P. queries), but also from other QMF operations such as DISPLAY TABLE. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0  QMF specifies no concurrent access resolution options on the PREPARE statement associated with the pending SQL SELECT statement. This value is the default.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1  SKIP LOCKED DATA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This value can be specified for executable SELECT statements directed to Db2 for z/OS Version 9 (New Function Mode), or later.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2  USE CURRENTLY COMMITTED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This value can be specified for executable SELECT statements directed to Db2 for z/OS Version 10 (New Function Mode), or later.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3  WAIT FOR OUTCOME</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This value can be specified for executable SELECT statements directed to Db2 for z/OS Version 10 (New Function Mode), or later.</td>
</tr>
<tr>
<td>DSQEC_CON_CSWL</td>
<td>None</td>
<td>01</td>
<td>This global variable enables the use of the DB2 for z/OS statement concentration with literals feature. It applies to dynamic SQL SELECT statements submitted to DB2 for z/OS through QMF commands such as RUN QUERY and DISPLAY, EXPORT and PRINT TABLE. When you set this variable, QMF specifies support through the DB2 for z/OS CONCENTRATE STATEMENTS WITH LITERALS prepare attribute:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0  Do not enable DB2 for z/OS statement concentration with literals. This is the default.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1  Enable DB2 for z/OS statement concentration with literals.</td>
</tr>
</tbody>
</table>
Table 57. Global variables that control how commands and procedures are executed (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQEC_CURR_FOLDER</td>
<td>None</td>
<td>128</td>
<td>Specifies the name of the current folder to be used for QMF commands that allow folder processing (SAVE, LIST, and ERASE). The default is blank. When a folder name is identified in this global variable, that folder is used when any QMF command that uses QMF folder objects is processed. For example, when DSQEC_CURR_FOLDER is set and the SAVE QUERY AS Q1 command is executed, the query will be saved and the query object will be included in the folder that is identified in the global variable. You can override this global variable by specifying a folder name with the FOLDER keyword with the QMF command. In this case, the folder name that is specified with the FOLDER keyword overrides the folder name that is specified in the DSQEC_CURR_FOLDER global variable. If this global variable is blank and the FOLDER keyword is not specified, folder processing is not used. <strong>Restriction:</strong> This global variable is not supported when QMF is connected to DB2 Server for VSE and VM.</td>
</tr>
<tr>
<td>DSQEC_DISABLEADM</td>
<td>None</td>
<td>01</td>
<td>Suppression of QMF administrator authority. When the value of this global variable is changed, the effect is immediate. Possible values can be: 0 QMF administrator authority is available (if the authorization ID has QMF administrator authority). 1 QMF administrator authority is suppressed (regardless of the authority of the authorization ID). The initial default value for this global variable can be overridden by the DSQUOPTS initialization exit routine.</td>
</tr>
<tr>
<td>DSQEC_DSALLOC_DIR</td>
<td>None</td>
<td>03</td>
<td>Specifies the number of directory blocks to be used when exporting a member of a new PDS data set in TSO. The value must be greater than zero for PDS data sets. If you are using the site default type of data set or PDSE data sets, QMF ignores the value of this global variable. To use the site default type of data set, set DSQEC_PO to 0. To use PDSE data sets, set DSQEC_PO to 2. If your site uses sequential data sets, set this global variable to zero.</td>
</tr>
<tr>
<td>DSQEC_DSALLOC_PRI</td>
<td>None</td>
<td>08</td>
<td>QMF allocates data sets in tracks. This global variable specifies the primary quantity of tracks for the TSO data set that is used to store the results of the QMF EXPORT command. Values can be from 1 to the maximum size allowed by the storage device and operating system. The default value is 15. A value of zero is not allowed. PS, PDS, and PDSE data sets can have a maximum value of 16777215 tracks.</td>
</tr>
<tr>
<td>DSQEC_DSALLOC_SEC</td>
<td>None</td>
<td>08</td>
<td>QMF allocates data sets in tracks. This global variable specifies the secondary quantity of tracks for the TSO data set that is used to store the results of the QMF EXPORT command. Values can be from zero to the maximum size allowed by the storage device and operating system. The default value is 105 tracks. PS and PDS data sets can have a maximum value of 65535 tracks; PDSE data sets can have a maximum value of 16777215 tracks.</td>
</tr>
</tbody>
</table>
Table 57. Global variables that control how commands and procedures are executed (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQEC_DSLRECL1</td>
<td>None</td>
<td>05</td>
<td>Specifies the logical record length (LRECL) that is to be used when an SQL query or QMF procedure is exported to a new data set. Valid values are 79 - 32760. The default value is 79.</td>
</tr>
</tbody>
</table>
| DSQEC_DSQSFISO                  | None                            | 01     | Specifies the format of CHAR(datetime-expression) data within a QMF report. The following values are used:  
|                                 |                                  |        | 0 The result of CHAR(datetime-expression) data is in the format specified in the DATE FORMAT and TIME FORMAT fields on Db2 installation panel DSNTIP4. The current Db2 DATE and TIME format values can be found by referencing global variables DSQAO_DATE_FORMAT and DSQAO_TIME_FORMAT.  
|                                 |                                  |        | 1 The result of CHAR(datetime-expression) data is in ISO format. DSQEC_DSQSFISO takes its default value from the value of program parameter DSQSFISO. The DSQSFISO program parameter setting may be seen in state global variable DSQAO_DSQSFISO. Note that if DSQEC_DSQSFISO is modified, the value of DSQAO_DSQSFISO will not change. DSQEC_DSQSFISO should be referenced for the current behavior settings. |
| DSQEC_DS_NOPAR                  | None                            | 01     | Indicates whether Parallelism is currently in use.  
|                                 |                                  |        | 0 Parallelism is currently in use (default).  
|                                 |                                  |        | 1 Parallelism is not currently in use. |
| DSQEC_DS_PAR                    | None                            | 02     | The valid values:  
|                                 |                                  |        | -1 No restrictions are placed on QDS (default)  
|                                 |                                  |        | 0 QDS will advise DVS that Map Reduce may be used, but Map Reduce Client may not be used.  
|                                 |                                  |        | 1 Neither Map Reduce nor Map Reduce Client are allowed.  
|                                 |                                  |        | 2-10 Both Map Reduce and Map Reduce Client can be used, but the degree of Map Reduce Client parallelism is limited to the number specified (For example, 2 means that 2 parallel paths can be used, 3 means 3 can be used, and so on.) |
|                                 |                                  |        | NOTE: If DSQEC_DS_NOPAR is set to 1 then the value of DSQEC_DS_PAR is ignored and no parallelism is in use. |
| DSQEC_DS_SUPPOR                  | None                            | 01     | Provides support for QMF Data Service (QDS)  
|                                 |                                  |        | 0 Do not allow access to QMF Data Service (default).  
|                                 |                                  |        | 1 Allow access to QMF Data Service. This global variable controls whether RUN QUERY (SQL, PQ or QBE), DISPLAY TABLE, DRAW, EXPORT, and PRINT TABLE commands should be analyzed by the QMF Data Service component. If an object that is referenced in the command is defined to the QMF Data Service component, then the entire command is executed by QMF Data Service. If none of the objects referenced in the command access an object defined to QMF Data Service, then the command is executed by the current DB2 connection. If the QDS service could not be loaded or is not available, then this value is ignored and all requests are routed to Db2. |
Table 57. Global variables that control how commands and procedures are executed (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQEC_EDITOR</td>
<td>None</td>
<td>18</td>
<td>Specifies the value to use for the EDITOR keyword on the EDIT command when the EDITOR keyword is not specified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The valid values for this global variable are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PDF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The ISPF/PDF editor is used to edit the procedure or query. To use the PDF editor to edit a query or procedure, start QMF as an ISPF dialog.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The SQL QUERY or PROC enhanced editor is used to edit the procedure or query.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>editorname</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The name of any other editor that is available to you. You can also specify the name of a CLIST that starts an editor. For more information about available editors, see your QMF administrator.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The default value is blank.</td>
</tr>
<tr>
<td>DSQEC_EDITOR_PVIEW</td>
<td>None</td>
<td>1</td>
<td>Controls the QMF Enhanced Editor preview command.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Do not allow the preview command to run. Message DYQE069 is issued to warn the user that the command is inactive.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Allow access to QMF Data Service. Allow the preview command to run. This is the default.</td>
</tr>
<tr>
<td>DSQEC_EXPL_MODE</td>
<td>None</td>
<td>07</td>
<td>Specifies the setting that is to be used for the Db2 special register CURRENT EXPLAIN MODE when the RUN QUER Y command is issued. The special register controls the behavior of the EXPLAIN facility for eligible dynamic SQL statements. Before a query is run, QMF sets the CURRENT EXPLAIN MODE special register to the value that is specified by this global variable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The valid values for this global variable are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The EXPLAIN facility is disabled and no EXPLAIN information is captured when explainable dynamic statements are run. This is the default value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The EXPLAIN facility is enabled and EXPLAIN information is inserted into the EXPLAIN tables for eligible dynamic SQL statements after the statement is prepared and run. All dynamic SQL statements are compiled and run.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EXPLAIN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The EXPLAIN facility is enabled and EXPLAIN information is inserted into the EXPLAIN tables for eligible dynamic SQL statements after the statement is prepared. Dynamic statements, except for SET statements, are not run. For servers other than Db2 for Linux, UNIX, and Windows or Db2 10 for z/OS (New Function Mode) or later, the only valid value is NO.</td>
</tr>
<tr>
<td>Callable interface variable name</td>
<td>Command interface variable name</td>
<td>Length</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>DSQEC_EXTND_STG</td>
<td>None</td>
<td>31</td>
<td>Specifies the number of megabytes of extended storage that QMF acquires on each request to the extended storage manager when the DSQSPTYP program parameter is set to 64BIT. This program parameter is available in QMF for TSO only. When an operation requires extended storage, QMF requests the specified amount until the operation is complete or extended storage is exhausted. When setting this global variable, consider the average size of DATA objects with which your QMF users work. If the average size is large and you set the value low, QMF issues many calls to the extended storage manager to complete the DATA object. These repeated calls might affect performance. Values can be from 1 to 1000. The default value is 25, indicating that QMF requests 25 MB of storage on each request.</td>
</tr>
<tr>
<td>DSQEC_FORM_LANG</td>
<td>None</td>
<td>01</td>
<td>Establishes the default NLF language in a saved, exported, or imported form; values can be: 0 The form uses the presiding NLF language. 1 The form uses English. This value is the default.</td>
</tr>
<tr>
<td>DSQEC_ISOLATION</td>
<td>None</td>
<td>01</td>
<td>Default query isolation level. Values can be: 0 Isolation level UR (uncommitted read) Uncommitted read can be useful in a distributed environment. However, if you are using uncommitted read, any reports that users view might contain data that was deleted from the database after the report was displayed. 1 Isolation level CS (cursor stability) This value is the default. When using cursor stability, QMF does not display the report until all database commands that affects the data in the report are complete.</td>
</tr>
</tbody>
</table>
Table 57. Global variables that control how commands and procedures are executed (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
</table>
| DSQEC_KEEP_THREAD               | None                            | 01     | Specifies whether a thread is released or kept active at the end of a query.  
This global variable does not affect threads that are created for procedures that run in batch mode or threads that are created when QMF is connected to a remote database through the CONNECT command. When procedures are run in batch mode, threads persist until the procedure completes. When QMF is connected to a remote database, threads persist until the connection ends.  
The valid values for this global variable are: |
|                                 |                                 |        | 0 The thread is released at the end of the query. This is the default value.  
If this setting is used, the SET Db2 global variable statement fails unless it is run in one of the following situations:  
• The statement is included in a procedure that is run in batch mode. The Db2 global variable is reset to its default value after the procedure completes.  
• The QMF CONNECT command is issued to connect to a remote database and the SET Db2 global variable statement is run on the remote database.  
• The SET Db2 global variable statement is included in a multistatement query and the QMF DSQEC_RUN_MQ global variable is set to 1. The Db2 global variable is reset to its default value after the query completes.  
1 The thread is kept active until the end of the QMF session or the DSQEC_KEEP_THREAD global variable is set to 0. This setting allows users to run the SET Db2 global variable statement to set Db2 global variables.  
If you set any Db2 global variables while DSQEC_KEEP_THREAD is set to 1 and then change DSQEC_KEEP_THREAD to 0, those Db2 global variables revert to their default values. |
| DSQEC_LAST_RUN                  | None                            | 01     | Specifies the set of commands that cause the LAST_USED field on QMF object lists to be updated. This field is based on the LAST_USED column of the Q.OBJECT_DIRECTORY control table. The value in the LAST_USED column is updated regardless of whether the issued command is successful. However, in some cases, the LAST_USED column is not updated immediately, and if QMF is terminated abnormally, the column might not be updated.  
Possible values are: |
|                                 |                                 |        | 0 QMF updates the LAST_USED timestamp whenever any of the following commands is issued:  
• CONVERT  
• DISPLAY  
• EXPORT  
• IMPORT  
• LAYOUT  
• PRINT  
• RUN  
• SAVE  
This value is the default.  
1 QMF restricts updates of the LAST_USED timestamp to RUN, SAVE, and IMPORT commands only.  
2 QMF restricts updates of the LAST_USED timestamp to the RUN command only. |
<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
</table>
| DSQEC_LIST_OWNER                 | None                            | 128    | Provides the default value for the OWNER parameter of the LIST command. Specify an authorization ID up to 128 characters long. This variable is blank by default, resulting in a list of objects owned by the current authorization ID. You can use selection symbols in the variable value. Use an underscore (_) in place of a single character and a percent sign (%) in place of zero or more characters. For example, the following command followed by a LIST command instructs QMF to list only objects that are owned by user IDs that begin with the characters RO:  
  
  SET GLOBAL (DSQEC_LIST_OWNER=RO%  
  
  The following command sets the default owner to any user IDs that begin with I, have any character in the second position, and any characters in the remaining positions:  
  
  SET GLOBAL (DSQEC_LIST_OWNER=I_%  
  
  The value you set with this global variable does not apply to lists displayed when you press the List key on QMF panels other than the home panel. |
| DSQEC_LOB_COLMAX                 | None                            | 10     | Specifies the maximum data size of a LOB column that is to be retrieved, in bytes, up to the maximum LOB size of 2147483637, or 2 GB. By default, LOB metadata is retrieved instead of LOB data. However, if an edit code other than M is specified or if the DSQEC_LOB_RETRV global variable is set to 3, LOB data is retrieved instead of metadata. In this case, if a user queries a table that contains LOB data that is larger than the maximum, an error is issued and no report data is displayed. If a user issues an EXPORT TABLE, PRINT TABLE, SAVE DATA, or EXPORT DATA command for a table or data object that contains LOB data that is larger than the maximum, an error is issued and the command is terminated. The default is 0, which specifies no maximum. |
| DSQEC_LOB_RETRV                  | None                            | 01     | Specifies how LOB data or metadata is retrieved. The valid values are:  
  
  1 Displays LOB metadata in results. To display actual LOB data, you can change the M edit code to another edit code. When this value is specified, QMF uses LOB locators to access LOB data. This is the default setting.  
  
  2 Displays LOB metadata only in results. The M edit code is the only valid edit code for LOB data. When this value is specified, QMF does not use LOB locators.  
  
  3 Retrieves and displays actual LOB data in results. When this value is specified, QMF does not use LOB locators to access LOB data. |
Table 57. Global variables that control how commands and procedures are executed (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQEC_LOB_SAVE</td>
<td>None</td>
<td>01</td>
<td>Specifies whether users can save LOB data to a table in the database using the QMF SAVE DATA or IMPORT TABLE command. The valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 - Disable LOB Save</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Specifies that users cannot issue the QMF SAVE DATA or IMPORT TABLE commands to save data to a table in the database if any column contains LOB data. An error message is displayed and no data is saved if a LOB column exists.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - Enable LOB Save</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Specifies that users can save LOB data to a table in the database using the QMF SAVE DATA or IMPORT TABLE commands. This is the default value.</td>
</tr>
<tr>
<td>DSQEC_NLFCMD_LANG</td>
<td>None</td>
<td>01</td>
<td>Sets expected NLF language for commands. Values can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0  Commands must be in the presiding NLF language. This value is the default.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1  Commands must be in English.</td>
</tr>
<tr>
<td>DSQEC_PO</td>
<td>None</td>
<td>01</td>
<td>Specifies the type of partitioned (PO) data set to create when exporting a QMF object to a new TSO data set. Values can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0  Allocates a data set of the type listed as the default for your site. This type is specified in the IGDSMSxx member of the SYS1.PARMLIB. This value is the default value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1  Allocates a PDS data set for the exported data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2  Allocates a PDSE data set for the exported data.</td>
</tr>
<tr>
<td>DSQEC_PRO_ENABLE</td>
<td>None</td>
<td>01</td>
<td>Controls whether a confirmation panel is displayed before QMF overwrites or discards the contents of the QUERY, FORM, PROC, or PROFILE temporary storage areas. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0  No confirmation panel is displayed before the contents of the supported temporary storage areas are overwritten. This value is the default.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1  A confirmation panel is displayed if the global variable that corresponds to the temporary storage area in question is also set to 1. The following global variables individually control overwrites in each of the supported temporary storage areas:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• DSQEC_PRO_FORM controls overwrites of the FORM temporary storage area, which stores current QMF report formatting specifications.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• DSQEC_PRO_PROC controls overwrites of the PROC temporary storage area, which stores current QMF procedures.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• DSQEC_PRO_PROF controls overwrites of the PROFILE temporary storage area, which stores current QMF profile settings.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• DSQEC_PRO_QUERY controls overwrites of the QUERY temporary storage area, which stores the current QMF query.</td>
</tr>
<tr>
<td>Callable interface variable name</td>
<td>Command interface variable name</td>
<td>Length</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------------------------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>DSQEC_PRO_FORM</td>
<td>None</td>
<td>01</td>
<td>This variable controls whether a confirmation panel is displayed before QMF overwrites or discards the contents of the FORM temporary storage area. The DSQEC_PRO_ENABLE global variable must be set to 1. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0  No confirmation panel is displayed before the contents of the temporary storage area are discarded.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1  A confirmation panel is displayed, giving the user the opportunity to proceed or cancel the command that caused the pending discard. The contents of the temporary storage area can then be saved with the SAVE command.</td>
</tr>
<tr>
<td>DSQEC_PRO_PROC</td>
<td>None</td>
<td>01</td>
<td>This variable controls whether a confirmation panel is displayed before QMF overwrites or discards the contents of the PROC temporary storage area. The DSQEC_PRO_ENABLE global variable must be set to 1. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0  No confirmation panel is displayed before the contents of the temporary storage area are discarded.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1  A confirmation panel is displayed before the contents of the temporary storage area are discarded. The user can proceed or cancel the command that caused the pending discard. The contents of the temporary storage area can then be saved with the SAVE command.</td>
</tr>
<tr>
<td>DSQEC_PRO_PROF</td>
<td>None</td>
<td>01</td>
<td>This variable controls whether a confirmation panel is displayed before QMF overwrites or discards the contents of the PROFILE temporary storage area. The DSQEC_PRO_ENABLE global variable must be set to 1. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0  No confirmation panel is displayed before the contents of the temporary storage area are discarded.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1  A confirmation panel is displayed before the contents of the temporary storage area are discarded. The user can proceed or cancel the command that caused the pending discard. The contents of the temporary storage area can then be saved with the SAVE command.</td>
</tr>
<tr>
<td>DSQEC_PRO_QUERY</td>
<td>None</td>
<td>01</td>
<td>This variable controls whether a confirmation panel is displayed before QMF overwrites or discards the contents of the QUERY temporary storage area. The DSQEC_PRO_ENABLE global variable must be set to 1. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0  No confirmation panel is displayed before the contents of the temporary storage area are discarded.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1  A confirmation panel is displayed before the contents of the temporary storage area are discarded. The user can proceed or cancel the command that caused the pending discard. The contents of the temporary storage area can then be saved with the SAVE command.</td>
</tr>
<tr>
<td>DSQEC_RERUN_IPROC</td>
<td>None</td>
<td>01</td>
<td>Reruns the invocation procedure after the END command; values can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0  Suppresses rerun of the invocation procedure after the END command.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1  Reruns the invocation procedure after the END command. This value is the default.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If you start QMF with an invocation procedure, set this variable to '0'; QMF terminates instead of rerunning the procedure.</td>
</tr>
<tr>
<td>Callable interface variable name</td>
<td>Command interface variable name</td>
<td>Length</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>DSQEC_RESET_RPT</td>
<td>None</td>
<td>31</td>
<td>Determines whether QMF prompts you when an incomplete DATA object in temporary storage might be affecting performance; possible values are: &lt;br&gt;0: Reset Report prompt panel is not displayed and QMF completes the running report. This value is the default value. &lt;br&gt;1: Reset Report prompt panel is displayed; this panel prompts you to complete or reset the currently running report before starting the new command. &lt;br&gt;2: Reset Report prompt panel is not displayed and QMF resets the currently running report.</td>
</tr>
<tr>
<td>DSQEC_RUN_MQ</td>
<td>None</td>
<td>01</td>
<td>Specifies whether the RUN QUERY command supports multiple statements in an SQL query. Possible values are: &lt;br&gt;0: Multiple SQL statements are not supported. If you set this variable to 0 and run an SQL query that contains multiple statements, QMF ignores all statements after encountering the first semicolon. This value is the default. &lt;br&gt;1: Multiple SQL statements are supported. A semicolon must be placed at the end of each statement except the last. &lt;br&gt;Restrictions: Although a SELECT statement can be included with other statements in a query, only one SELECT statement can be included per query. CALL and CREATE PROCEDURE statements must be used alone in an SQL query.</td>
</tr>
<tr>
<td>DSQEC_SAV_ACCELNM</td>
<td>None</td>
<td>128</td>
<td>Specifies the name of the default accelerator to be used when creating accelerator-only tables from SAVE DATA, IMPORT TABLE and RUN QUERY to TABLE commands. This variable is referenced only if the ACCELERATOR keyword is not specified. &lt;br&gt;Although you can set this global variable to a blank, do not set it to blank if the DSQEC_SAV_ALLOWED global variable is set to ‘4’.</td>
</tr>
</tbody>
</table>
Table 57. Global variables that control how commands and procedures are executed (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQEC_SAV_ALLOWED</td>
<td>None</td>
<td>01</td>
<td>Controls whether users save data to a new table in the database or in an accelerator using the QMF SAVE DATA, RUN QUERY to TABLE, or IMPORT TABLE commands. Except for option 0, this field does not influence the location of existing tables that the replaced data is in or the data is appended to. Existing tables are replaced or appended to in the database or accelerator regardless of the setting of this variable. Valid values for this global variable are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 - Disable Save Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Users cannot issue the QMF SAVE DATA, RUN QUERY to TABLE, or IMPORT TABLE commands to save data to a table in the database or accelerator. An error message will be displayed and no data will be saved.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - Enable Save Data to database tables only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Users can save data to a table in the database by using the QMF SAVE DATA, RUN QUERY to TABLE, or IMPORT TABLE commands. Users cannot save data to accelerator-only tables. This setting is the default.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 - Enable Save Data to accelerator only tables only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Users can save data to an accelerator-only table by using the QMF SAVE DATA, RUN QUERY to TABLE, or IMPORT TABLE commands. Users cannot save data to database tables. The DSQEC_SAV_ACCELNM global variable contains the default name of the accelerator but can be overridden by the ACCELERATOR keyword.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 - Enable Save Data to either database or accelerator only tables (database default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Users can save data either to a table in the database or to an accelerator-only table by using the QMF SAVE DATA, RUN QUERY to TABLE, or IMPORT TABLE commands. If no command keyword overrides are present, such as SPACE or ACCEL, tables are saved in the database.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4 - Enable Save Data to either database or accelerator only tables (accelerator default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Users can save data either to a table in the database or to an accelerator-only table by using the QMF SAVE DATA, RUN QUERY to TABLE, or IMPORT TABLE commands. If no command keyword overrides are present, such as SPACE or ACCELERATOR, tables are saved in the accelerator. When this option is chosen, the DSQEC_SAV_ACCELNM global variable must contain the name of the accelerator.</td>
</tr>
</tbody>
</table>
Table 57. Global variables that control how commands and procedures are executed (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQEC_SESSGL_VSA</td>
<td>None</td>
<td>01</td>
<td>Controls whether user input in some data entry fields on some panels is saved within and across QMF sessions. User input is saved as session variables that are stored in the Q.GLOBAL_VARS table as global variables that are named with a DXY prefix. The DSQEC_SESSGL_VSA global variable is checked throughout the session, as well as when QMF starts and exits. The valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Callable interface variable name</td>
<td>Command interface variable name</td>
<td>Length</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------------------------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| DSQEC_SHARE                      | None                             | 31     | Specifies the default value for the SHARE parameter; possible values are:  
|                                 |                                  |        | 0  Do not share data with other users.  
|                                 |                                  |        | 1  Share data with other users.  |
| DSQEC_SP_RS_NUM                  | None                             | 04     | Indicates which result set returned by a stored procedure is used to create the report. Possible values are:  
|                                 |                                  |        | 0  Ignores result sets.  
|                                 |                                  |        | 1  Returns the first result set.  
|                                 |                                  |        | 2  Returns the second result set.  
|                                 |                                  |        | n  Returns the nth result set. The maximum value for n is 63.  
|                                 |                                  |        | ALL  Returns all results sets.  |
| DSQEC_SPAC_OVRIDE                | None                             | 01     | Specifies whether users can override the default table space that is specified in the QMF profile.  
|                                 |                                  |        | Valid values for this global variable are:  
|                                 |                                  |        | 0   - Disable Space Keyword Option  
|                                 |                                  |        | Users cannot issue the SAVE DATA, RUN QUERY to TABLE, or IMPORT TABLE commands with the SPACE keyword option.  
|                                 |                                  |        | 1   - Enable Space Keyword Option  
|                                 |                                  |        | Users can issue the SAVE DATA, RUN QUERY to TABLE, or IMPORT TABLE commands with the space keyword option. This setting is the default.  |
| DSQEC_SQLQRYSZ_2M                 | None                             | 01     | Controls whether SQL queries greater than 32,767 bytes (32 KB) in length are supported by the RUN QUERY command.  
|                                 |                                  |        | 0  SQL queries directed to Db2 for z/OS, DB2 for iSeries, and Db2 for Linux, UNIX, and Windows databases are limited to 32,767 bytes (32 KB). This value is the default.  
|                                 |                                  |        | 1  SQL queries can be greater than 32 KB. The maximum supported query size varies depending on the type of database to which the query is directed:  
|                                 |                                  |        | • Queries directed to Db2 for z/OS can be up to 2 MB in length.  
|                                 |                                  |        | • Queries directed to DB2 for iSeries or Db2 for Linux, UNIX, and Windows can be up to 65 KB in length.  
|                                 |                                  |        | These maximums assume that the version of the database to which the RUN QUERY command is directed supports queries of this size. SQL queries directed to Db2 for VSE and VM are limited to 8 KB.  
|                                 |                                  |        | Additional customization might be required to run queries larger than 32 KB from QMF for CICS.  |
| DSQEC_TABS_LDB2                   | None                             | 31     | View for retrieving lists of tables and views at the current server, if it is Db2 for z/OS or Db2 for Linux, UNIX, and Windows  |
| DSQEC_TABS_RDB2                   | None                             | 31     | View for retrieving lists of tables and views at remote Db2 subsystems.  |
| DSQEC_TABS_SQL                    | None                             | 31     | View for retrieving lists of tables and views for a Db2 for VSE and VM database.  |
### Table 57. Global variables that control how commands and procedures are executed (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQEC_TRACE_LIMIT</td>
<td>None</td>
<td>31</td>
<td>Limits the amount of trace output to the specified number of bytes. The valid range is 0 - 2147483647. This variable can be used to reduce the size of QMF trace output. This global variable is typically set as directed by IBM Software Support.</td>
</tr>
<tr>
<td>DSQEC_TRACE_MODULE</td>
<td>None</td>
<td>54</td>
<td>Contains the names of QMF modules to be traced. Up to 6 modules can be specified, separated by commas. After module names are specified in the global variable, initiate the trace by issuing the SET PROFILE command with the TRACE keyword to set to ALL. Example: SET PROFILE (TRACE=ALL) Note: If modules are specified via the SET GLOBAL command from the command line, the module names must be enclosed in single quotes.</td>
</tr>
<tr>
<td>DSQEC_TWO_GB_ROW</td>
<td>None</td>
<td>01</td>
<td>Controls the length of rows returned in QMF reports. Use one of the following values: 0 Limits the length of a data row in a QMF report to 32 KB, unless the report contains an XML or LOB column. 1 Allows the length of a data row to be greater than 32 KB, up to a maximum length of 2 GB. <strong>Important:</strong>  • Regardless of the DSQEC_TWO_GB_ROW global variable setting, up to 2 GB of XML, CLOB, or BLOB data, and up to 1 GB of DBCLOB data can be displayed by default. However, the maximum length of a LOB row can be restricted by the DSQEC_LOB_COLMAX global variable.  • Regardless of the DSQEC_TWO_GB_ROW global variable setting, a single table cannot have a maximum record size that is greater than the page size. Db2 stores records within pages that are 4 KB, 8 KB, 16 KB, or 32 KB in size. So, the maximum length of a data row that can be displayed remains at 32 KB when you display or select data from a single table. If you display or select data from a view that joins two or more tables, the row length can be up to 2 GB. Because of these page size considerations, the length of a data row in a QMF report that can be saved with the SAVE DATA command is also limited to 32 KB. The ability to save LOB data is controlled by the DSQEC_LOB_SAVE global variable.</td>
</tr>
</tbody>
</table>
Table 57. Global variables that control how commands and procedures are executed (continued)

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQEC_USERGLV_SAV</td>
<td>None</td>
<td>01</td>
<td>Determines whether global variables that were created or changed by the user, including those that start with “DSQ,” are saved when the QMF session ends. Values that are to be saved are stored in the Q.GLOBAL_VARS table and associated with the user ID of the session. If the values are saved, they are restored at the start of the user’s next QMF session. The valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

Related reference:
- [PREPARE statement for Db2](#)

See the information about the concurrent-access-resolution attribute of the PREPARE statement.

Global variables that store results of CONVERT QUERY

DSQQC global variables reflect the results of a CONVERT QUERY command. None of these global variables can be modified by the SET GLOBAL command.

Table 58. Global variables that reflect the results of a CONVERT QUERY command

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQQC_LENGTH_nnn</td>
<td>DSQCLENn</td>
<td>05</td>
<td>Length of converted result nnn.</td>
</tr>
<tr>
<td>DSQQC_QRY_COUNT</td>
<td>DSQCQCNT</td>
<td>03</td>
<td>Number of queries in converted result; value must always be ‘1’ unless the original query is a QBE I. or U. query.</td>
</tr>
<tr>
<td>DSQQC_QRY_LANG</td>
<td>DSQCQLNG</td>
<td>01</td>
<td>Language of converted query; values can be: 1 for SQL, 2 for QBE, 3 for Prompted</td>
</tr>
<tr>
<td>DSQQC_QRY_TYPE</td>
<td>DSQCQTRYP</td>
<td>Not specified</td>
<td>First word in converted results.</td>
</tr>
<tr>
<td>DSQQC_RESULT_nnn</td>
<td>DSQCQnmm</td>
<td>Not specified</td>
<td>nnn</td>
</tr>
</tbody>
</table>
Global variables that show RUN QUERY error message information

DSQQM global variables store the results of a RUN QUERY command. None of these global variables can be modified by the SET GLOBAL command.

<table>
<thead>
<tr>
<th>Callable interface variable name</th>
<th>Command interface variable name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSQQM_MESSAGE</td>
<td>DSQCICQMG</td>
<td>80</td>
<td>Text of query message.</td>
</tr>
<tr>
<td>DSQQM_MESSAGE_ALL</td>
<td>DSQCICQMA</td>
<td>360</td>
<td>Complete query message text.</td>
</tr>
<tr>
<td>DSQQM_MSG_HELP</td>
<td>DSQCICQID</td>
<td>08</td>
<td>ID of message help panel.</td>
</tr>
<tr>
<td>DSQQM_MSG_NUMBER</td>
<td>DSQCIQNO</td>
<td>08</td>
<td>Message number.</td>
</tr>
<tr>
<td>DSQQM_SQL_RC</td>
<td>DSQCIQLIQUE</td>
<td>16</td>
<td>The SQLCODE from the last command or query.</td>
</tr>
<tr>
<td>DSQQM_SQL_STATE</td>
<td>None</td>
<td>05</td>
<td>The SQLSTATE associated with the SQLCODE in DSQQM_SQL_RC, if SQLSTATE is returned by the database manager.</td>
</tr>
<tr>
<td>DSQQM_SUB_TXT_nnn</td>
<td>DSQCICQnn</td>
<td>20</td>
<td>Substitution value nnn.</td>
</tr>
<tr>
<td>DSQQM_SUBST_VARS</td>
<td>DSQCICQ00</td>
<td>04</td>
<td>Number of substitution variables.</td>
</tr>
</tbody>
</table>

Global variables that store panel input values

DXY global variables store the values that users enter in data entry fields if the DSQEC_SESSGLV_SAV global variable is set to 1 or 2. Input in only some data entry fields on some panels is saved. User input for fields that are not listed in the following table are not saved, regardless of the DSQEC_SESSGLV_SAV global variable setting.

All of these global variables can be modified by the SET GLOBAL command. However, use caution when changing or deleting these variables because doing so changes the values that are generated on command prompt panels.

<table>
<thead>
<tr>
<th>Global variable name (where n is a national language identifier and ln is an ID associated with a line of a multiline field)</th>
<th>Range of ln values</th>
<th>Command</th>
<th>Field name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DXYnPCO1_01</td>
<td>01 - 03</td>
<td>CONNECT</td>
<td>User</td>
</tr>
<tr>
<td>DXYnPCO1_05</td>
<td>–</td>
<td>CONNECT</td>
<td>Location</td>
</tr>
<tr>
<td>DXYnPC03_01</td>
<td>–</td>
<td>CONNECT (CICS)</td>
<td>Location</td>
</tr>
<tr>
<td>DXYnPENV_Ln</td>
<td>02 - 07</td>
<td>CONVERT</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPDSF_Ln</td>
<td>02 - 07</td>
<td>DISPLAY</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPDSF_Ln_01</td>
<td>02 - 07</td>
<td>DISPLAY QUERY</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPDSF_Ln_02</td>
<td>02 - 07</td>
<td>DISPLAY PROC</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPDSF_Ln_03</td>
<td>02 - 07</td>
<td>DISPLAY FORM</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPDSF_Ln_05</td>
<td>02 - 07</td>
<td>DISPLAY REPORT</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPDSF_Ln_07</td>
<td>02 - 07</td>
<td>DISPLAY CHART</td>
<td>Name</td>
</tr>
</tbody>
</table>
Table 60. Mapping between DXY global variables and panel field names (continued)

<table>
<thead>
<tr>
<th>Global variable name (where ( n ) is a national language identifier and ( ln ) is an ID associated with a line of a multiline field)</th>
<th>Range of ( ln ) values</th>
<th>Command</th>
<th>Field name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DXYnPDSP_ln_08</td>
<td>02 - 07</td>
<td>DISPLAY TABLE</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPDRS_ln</td>
<td>01 - 06</td>
<td>DRAW</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPDRS_07</td>
<td>–</td>
<td>DRAW</td>
<td>Type</td>
</tr>
<tr>
<td>DXYnPDRS_08</td>
<td>–</td>
<td>DRAW</td>
<td>Identifier</td>
</tr>
<tr>
<td>DXYnPEDT_01</td>
<td>–</td>
<td>EDIT</td>
<td>Type</td>
</tr>
<tr>
<td>DXYnPED1_ln</td>
<td>01 - 06</td>
<td>EDIT (QUERY or PROC)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPED2_ln</td>
<td>01 - 06</td>
<td>EDIT TABLE</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPED2_07</td>
<td>–</td>
<td>EDIT TABLE</td>
<td>Mode</td>
</tr>
<tr>
<td>DXYnPED3_ln</td>
<td>01 - 06</td>
<td>EDIT (QUERY or PROC), then make changes and exit without saving.</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPED3_09</td>
<td>–</td>
<td>EDIT (QUERY or PROC), then make changes and exit without saving.</td>
<td>Comment</td>
</tr>
<tr>
<td>DXYnPED3_ln</td>
<td>10 - 12</td>
<td>EDIT (QUERY or PROC), then make changes and exit without saving.</td>
<td>Folder</td>
</tr>
<tr>
<td>DXYnPERA_ln</td>
<td>02 - 07</td>
<td>ERASE</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPERA_ln_01</td>
<td>02 - 07</td>
<td>ERASE QUERY</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPERA_ln_02</td>
<td>02 - 07</td>
<td>ERASE PROC</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPERA_ln_03</td>
<td>02 - 07</td>
<td>ERASE FORM</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPERA_ln_08</td>
<td>02 - 07</td>
<td>ERASE TABLE</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPXM_ln</td>
<td>02 - 07</td>
<td>EXPORT</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPXM_ln_01</td>
<td>02 - 07</td>
<td>EXPORT QUERY</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPXM_ln_02</td>
<td>02 - 07</td>
<td>EXPORT PROC</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPXM_ln_03</td>
<td>02 - 07</td>
<td>EXPORT FORM</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPXM_ln_05</td>
<td>02 - 07</td>
<td>EXPORT REPORT</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPXM_ln_06</td>
<td>02 - 07</td>
<td>EXPORT DATA</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPXM_ln_07</td>
<td>02 - 07</td>
<td>EXPORT CHART</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPXM_ln_08</td>
<td>02 - 07</td>
<td>EXPORT TABLE</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPXM1_ln</td>
<td>01 - 05</td>
<td>EXPORT, then Enter (in TSO)</td>
<td>To</td>
</tr>
<tr>
<td>DXYnPXM1_ln_01</td>
<td>01 - 05</td>
<td>EXPORT QUERY, then Enter (in TSO)</td>
<td>To</td>
</tr>
<tr>
<td>DXYnPXM1_ln_02</td>
<td>01 - 05</td>
<td>EXPORT PROC, then Enter (in TSO)</td>
<td>To</td>
</tr>
</tbody>
</table>
Table 60. Mapping between DXY global variables and panel field names (continued)

<table>
<thead>
<tr>
<th>Global variable name (where $n$ is a national language identifier and $ln$ is an ID associated with a line of a multiline field)</th>
<th>Range of $ln$ values</th>
<th>Command</th>
<th>Field name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DXY$n$PXM1_06</td>
<td>–</td>
<td>EXPORT, then Enter (in TSO)</td>
<td>Member</td>
</tr>
<tr>
<td>DXY$n$PXM1_06_01</td>
<td>–</td>
<td>EXPORT QUERY, then Enter (in TSO)</td>
<td>Member</td>
</tr>
<tr>
<td>DXY$n$PXM1_06_02</td>
<td>–</td>
<td>EXPORT PROC, then Enter (in TSO)</td>
<td>Member</td>
</tr>
<tr>
<td>DXY$n$PXM2_01_07</td>
<td>–</td>
<td>EXPORT CHART, then Enter (in TSO)</td>
<td>Member</td>
</tr>
<tr>
<td>DXY$n$PXM3_01_05</td>
<td>01 - 05</td>
<td>EXPORT REPORT, then Enter (in TSO)</td>
<td>To</td>
</tr>
<tr>
<td>DXY$n$PXM3_06_05</td>
<td>–</td>
<td>EXPORT REPORT, then Enter (in TSO)</td>
<td>Member</td>
</tr>
<tr>
<td>DXY$n$PXM3_08_05</td>
<td>–</td>
<td>EXPORT REPORT, then Enter (in TSO)</td>
<td>Dataformat</td>
</tr>
<tr>
<td>DXY$n$PXM4_01_06</td>
<td>01 - 05</td>
<td>EXPORT DATA, then Enter (in TSO)</td>
<td>To</td>
</tr>
<tr>
<td>DXY$n$PXM4_06_06</td>
<td>–</td>
<td>EXPORT DATA, then Enter (in TSO)</td>
<td>Member</td>
</tr>
<tr>
<td>DXY$n$PXM4_08_06</td>
<td>–</td>
<td>EXPORT DATA, then Enter (in TSO)</td>
<td>Dataformat</td>
</tr>
<tr>
<td>DXY$n$PXM4_09_06</td>
<td>–</td>
<td>EXPORT DATA, then Enter (in TSO)</td>
<td>Outputmode</td>
</tr>
<tr>
<td>DXY$n$PXM4_10_06</td>
<td>–</td>
<td>EXPORT DATA, then Enter (in TSO)</td>
<td>Header</td>
</tr>
<tr>
<td>DXY$n$PXM4_01_08</td>
<td>01-05</td>
<td>EXPORT TABLE, then Enter (in TSO)</td>
<td>To</td>
</tr>
<tr>
<td>DXY$n$PXM4_06_08</td>
<td>–</td>
<td>EXPORT TABLE, then Enter (in TSO)</td>
<td>Member</td>
</tr>
<tr>
<td>DXY$n$PXM4_08_08</td>
<td>–</td>
<td>EXPORT TABLE, then Enter (in TSO)</td>
<td>Dataformat</td>
</tr>
<tr>
<td>DXY$n$PXM4_09_08</td>
<td>–</td>
<td>EXPORT TABLE, then Enter (in TSO)</td>
<td>Outputmode</td>
</tr>
<tr>
<td>DXY$n$PXM4_10_08</td>
<td>–</td>
<td>EXPORT TABLE, then Enter (in TSO)</td>
<td>Header</td>
</tr>
<tr>
<td>DXY$n$PXM5_01_03</td>
<td>01 - 05</td>
<td>EXPORT FORM, then Enter (in TSO)</td>
<td>To</td>
</tr>
<tr>
<td>DXY$n$PXM5_06_03</td>
<td>–</td>
<td>EXPORT FORM, then Enter (in TSO)</td>
<td>Member</td>
</tr>
<tr>
<td>DXY$n$PXM5_08_03</td>
<td>–</td>
<td>EXPORT FORM, then Enter (in TSO)</td>
<td>Language</td>
</tr>
<tr>
<td>DXY$n$PXC1_01</td>
<td>–</td>
<td>EXPORT, then Enter (in CICS)</td>
<td>Queue Name</td>
</tr>
</tbody>
</table>
Table 60. Mapping between DXY global variables and panel field names (continued)

<table>
<thead>
<tr>
<th>Global variable name (where ( n ) is a national language identifier and ( ln ) is an ID associated with a line of a multiline field)</th>
<th>Range of ( ln ) values</th>
<th>Command</th>
<th>Field name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DXY(_n)PXC1(_01)(_01)</td>
<td>–</td>
<td>EXPORT QUERY, then Enter (in CICS)</td>
<td>Queue Name</td>
</tr>
<tr>
<td>DXY(_n)PXC1(_01)(_02)</td>
<td>–</td>
<td>EXPORT PROC, then Enter (in CICS)</td>
<td>Queue Name</td>
</tr>
<tr>
<td>DXY(_n)PXC1(_02)</td>
<td>–</td>
<td>EXPORT, then Enter (in CICS)</td>
<td>Queue Type</td>
</tr>
<tr>
<td>DXY(_n)PXC1(_02)(_01)</td>
<td>–</td>
<td>EXPORT QUERY, then Enter (in CICS)</td>
<td>Queue Type</td>
</tr>
<tr>
<td>DXY(_n)PXC1(_02)(_02)</td>
<td>–</td>
<td>EXPORT PROC, then Enter (in CICS)</td>
<td>Queue Type</td>
</tr>
<tr>
<td>DXY(_n)PXC1(_04)</td>
<td>–</td>
<td>EXPORT, then Enter (in CICS)</td>
<td>Suspend</td>
</tr>
<tr>
<td>DXY(_n)PXC1(_04)(_01)</td>
<td>–</td>
<td>EXPORT QUERY, then Enter (in CICS)</td>
<td>Suspend</td>
</tr>
<tr>
<td>DXY(_n)PXC1(_04)(_02)</td>
<td>–</td>
<td>EXPORT PROC, then Enter (in CICS)</td>
<td>Suspend</td>
</tr>
<tr>
<td>DXY(_n)PXC3(_01)(_05)</td>
<td>–</td>
<td>EXPORT REPORT, then Enter (in CICS)</td>
<td>Queue Name</td>
</tr>
<tr>
<td>DXY(_n)PXC3(_02)(_05)</td>
<td>–</td>
<td>EXPORT REPORT, then Enter (in CICS)</td>
<td>Queue Type</td>
</tr>
<tr>
<td>DXY(_n)PXC3(_04)(_05)</td>
<td>–</td>
<td>EXPORT REPORT, then Enter (in CICS)</td>
<td>Suspend</td>
</tr>
<tr>
<td>DXY(_n)PXC3(_05)(_05)</td>
<td>–</td>
<td>EXPORT REPORT, then Enter (in CICS)</td>
<td>Dataformat</td>
</tr>
<tr>
<td>DXY(_n)PXC4(_01)(_06)</td>
<td>–</td>
<td>EXPORT DATA, then Enter (in CICS)</td>
<td>Queue Name</td>
</tr>
<tr>
<td>DXY(_n)PXC4(_02)(_06)</td>
<td>–</td>
<td>EXPORT DATA, then Enter (in CICS)</td>
<td>Queue Type</td>
</tr>
<tr>
<td>DXY(_n)PXC4(_04)(_06)</td>
<td>–</td>
<td>EXPORT DATA, then Enter (in CICS)</td>
<td>Suspend</td>
</tr>
<tr>
<td>DXY(_n)PXC4(_05)(_06)</td>
<td>–</td>
<td>EXPORT DATA, then Enter (in CICS)</td>
<td>Dataformat</td>
</tr>
<tr>
<td>DXY(_n)PXC4(_06)(_06)</td>
<td>–</td>
<td>EXPORT DATA, then Enter (in CICS)</td>
<td>Outputmode</td>
</tr>
<tr>
<td>DXY(_n)PXC4(_07)(_06)</td>
<td>–</td>
<td>EXPORT DATA, then Enter (in CICS)</td>
<td>Header</td>
</tr>
<tr>
<td>DXY(_n)PXC5(_01)(_03)</td>
<td>–</td>
<td>EXPORT FORM, then Enter (in CICS)</td>
<td>Queue Name</td>
</tr>
<tr>
<td>DXY(_n)PXC5(_02)(_03)</td>
<td>–</td>
<td>EXPORT FORM, then Enter (in CICS)</td>
<td>Queue Type</td>
</tr>
<tr>
<td>DXY(_n)PXC5(_04)(_03)</td>
<td>–</td>
<td>EXPORT FORM, then Enter (in CICS)</td>
<td>Suspend</td>
</tr>
</tbody>
</table>
Table 60. Mapping between DXY global variables and panel field names  (continued)

<table>
<thead>
<tr>
<th>Global variable name (where ( n ) is a national language identifier and ( ln ) is an ID associated with a line of a multiline field)</th>
<th>Range of ( ln ) values</th>
<th>Command</th>
<th>Field name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DXYnlPXC5_05_03</td>
<td>–</td>
<td>EXPORT FORM, then Enter (in CICS)</td>
<td>Language</td>
</tr>
<tr>
<td>DXYnlPIMM_In</td>
<td>02 - 07</td>
<td>IMPORT (in TSO)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnlPIMM_In_01</td>
<td>02 - 07</td>
<td>IMPORT QUERY (in TSO)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnlPIMM_In_02</td>
<td>02 - 07</td>
<td>IMPORT PROC (in TSO)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnlPIMM_In_03</td>
<td>02 - 07</td>
<td>IMPORT FORM (in TSO)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnlPIMM_In_06</td>
<td>02 - 07</td>
<td>IMPORT DATA (in TSO)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnlPIMM_In_08</td>
<td>02 - 07</td>
<td>IMPORT TABLE (in TSO)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnlPIMM_In</td>
<td>08 - 12</td>
<td>IMPORT (in TSO)</td>
<td>From</td>
</tr>
<tr>
<td>DXYnlPIMM_In_01</td>
<td>08 - 12</td>
<td>IMPORT QUERY (in TSO)</td>
<td>From</td>
</tr>
<tr>
<td>DXYnlPIMM_In_02</td>
<td>08 - 12</td>
<td>IMPORT PROC (in TSO)</td>
<td>From</td>
</tr>
<tr>
<td>DXYnlPIMM_In_03</td>
<td>08 - 12</td>
<td>IMPORT FORM (in TSO)</td>
<td>From</td>
</tr>
<tr>
<td>DXYnlPIMM_In_06</td>
<td>08 - 12</td>
<td>IMPORT DATA (in TSO)</td>
<td>From</td>
</tr>
<tr>
<td>DXYnlPIMM_In_08</td>
<td>08 - 12</td>
<td>IMPORT TABLE (in TSO)</td>
<td>From</td>
</tr>
<tr>
<td>DXYnlPIMM_13</td>
<td>–</td>
<td>IMPORT (in TSO)</td>
<td>Member</td>
</tr>
<tr>
<td>DXYnlPIMM_13_01</td>
<td>–</td>
<td>IMPORT QUERY (in TSO)</td>
<td>Member</td>
</tr>
<tr>
<td>DXYnlPIMM_13_02</td>
<td>–</td>
<td>IMPORT PROC (in TSO)</td>
<td>Member</td>
</tr>
<tr>
<td>DXYnlPIMM_13_03</td>
<td>–</td>
<td>IMPORT FORM (in TSO)</td>
<td>Member</td>
</tr>
<tr>
<td>DXYnlPIMM_13_06</td>
<td>–</td>
<td>IMPORT DATA (in TSO)</td>
<td>Member</td>
</tr>
<tr>
<td>DXYnlPIMM_13_08</td>
<td>–</td>
<td>IMPORT TABLE (in TSO)</td>
<td>Member</td>
</tr>
<tr>
<td>DXYnlPIQF_03</td>
<td>–</td>
<td>IMPORT, then Enter (in TSO)</td>
<td>Comment</td>
</tr>
<tr>
<td>DXYnlPIQF_03_01</td>
<td>–</td>
<td>IMPORT QUERY, then Enter (in TSO)</td>
<td>Comment</td>
</tr>
<tr>
<td>DXYnlPIQF_03_02</td>
<td>–</td>
<td>IMPORT PROC, then Enter (in TSO)</td>
<td>Comment</td>
</tr>
</tbody>
</table>
Table 60. Mapping between DXY global variables and panel field names  (continued)

<table>
<thead>
<tr>
<th>Global variable name (where ( n ) is a national language identifier and ( ln ) is an ID associated with a line of a multiline field)</th>
<th>Range of ( ln ) values</th>
<th>Command</th>
<th>Field name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DXY( n )PIQL_03_03</td>
<td>–</td>
<td>IMPORT FORM, then Enter (in TSO)</td>
<td>Comment</td>
</tr>
<tr>
<td>DXY( n )PIQL_04_03</td>
<td>–</td>
<td>IMPORT FORM, then Enter (in TSO)</td>
<td>Language</td>
</tr>
<tr>
<td>DXY( n )PITB_02_08</td>
<td>–</td>
<td>IMPORT TABLE, then Enter (in TSO)</td>
<td>Comment</td>
</tr>
<tr>
<td>DXY( n )PITB_04_08</td>
<td>–</td>
<td>IMPORT TABLE, then Enter (in TSO)</td>
<td>Space</td>
</tr>
<tr>
<td>DXY( n )PITB_( ln )08</td>
<td>05 - 07</td>
<td>IMPORT TABLE, then Enter (in TSO)</td>
<td>Accelerator</td>
</tr>
<tr>
<td>DXY( n )PIMC_( ln )</td>
<td>02 - 07</td>
<td>IMPORT (in CICS)</td>
<td>Name</td>
</tr>
<tr>
<td>DXY( n )PIMC_( ln )01</td>
<td>02 - 07</td>
<td>IMPORT QUERY (in CICS)</td>
<td>Name</td>
</tr>
<tr>
<td>DXY( n )PIMC_( ln )02</td>
<td>02 - 07</td>
<td>IMPORT PROC (in CICS)</td>
<td>Name</td>
</tr>
<tr>
<td>DXY( n )PIMC_( ln )03</td>
<td>02 - 07</td>
<td>IMPORT FORM (in CICS)</td>
<td>Name</td>
</tr>
<tr>
<td>DXY( n )PIMC_( ln )06</td>
<td>02 - 07</td>
<td>IMPORT DATA (in CICS)</td>
<td>Name</td>
</tr>
<tr>
<td>DXY( n )PIMC_( ln )08</td>
<td>02 - 07</td>
<td>IMPORT TABLE (in CICS)</td>
<td>Name</td>
</tr>
<tr>
<td>DXY( n )PIMC_08</td>
<td>–</td>
<td>IMPORT (in CICS)</td>
<td>Queue Name</td>
</tr>
<tr>
<td>DXY( n )PIMC_08_01</td>
<td>–</td>
<td>IMPORT QUERY (in CICS)</td>
<td>Queue Name</td>
</tr>
<tr>
<td>DXY( n )PIMC_08_02</td>
<td>–</td>
<td>IMPORT PROC (in CICS)</td>
<td>Queue Name</td>
</tr>
<tr>
<td>DXY( n )PIMC_08_03</td>
<td>–</td>
<td>IMPORT FORM (in CICS)</td>
<td>Queue Name</td>
</tr>
<tr>
<td>DXY( n )PIMC_08_06</td>
<td>–</td>
<td>IMPORT DATA (in CICS)</td>
<td>Queue Name</td>
</tr>
<tr>
<td>DXY( n )PIMC_08_08</td>
<td>–</td>
<td>IMPORT TABLE (in CICS)</td>
<td>Queue Name</td>
</tr>
<tr>
<td>DXY( n )PIMC_09</td>
<td>–</td>
<td>IMPORT (in CICS)</td>
<td>Queue Type</td>
</tr>
<tr>
<td>DXY( n )PIMC_09_01</td>
<td>–</td>
<td>IMPORT QUERY (in CICS)</td>
<td>Queue Type</td>
</tr>
<tr>
<td>DXY( n )PIMC_09_02</td>
<td>–</td>
<td>IMPORT PROC (in CICS)</td>
<td>Queue Type</td>
</tr>
<tr>
<td>DXY( n )PIMC_09_03</td>
<td>–</td>
<td>IMPORT FORM (in CICS)</td>
<td>Queue Type</td>
</tr>
<tr>
<td>DXY( n )PIMC_09_06</td>
<td>–</td>
<td>IMPORT DATA (in CICS)</td>
<td>Queue Type</td>
</tr>
</tbody>
</table>
Table 60. Mapping between DXY global variables and panel field names (continued)

<table>
<thead>
<tr>
<th>Global variable name (where n is a national language identifier and ln is an ID associated with a line of a multiline field)</th>
<th>Range of ln values</th>
<th>Command</th>
<th>Field name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DXYnPIMC_09_08</td>
<td>--</td>
<td>IMPORT TABLE (in CICS)</td>
<td>Queue Type</td>
</tr>
<tr>
<td>DXYnPIMC_10</td>
<td>--</td>
<td>IMPORT (in CICS)</td>
<td>Suspend</td>
</tr>
<tr>
<td>DXYnPIMC_10_01</td>
<td>--</td>
<td>IMPORT QUERY (in CICS)</td>
<td>Suspend</td>
</tr>
<tr>
<td>DXYnPIMC_10_02</td>
<td>--</td>
<td>IMPORT PROC (in CICS)</td>
<td>Suspend</td>
</tr>
<tr>
<td>DXYnPIMC_10_03</td>
<td>--</td>
<td>IMPORT FORM (in CICS)</td>
<td>Suspend</td>
</tr>
<tr>
<td>DXYnPIMC_10_06</td>
<td>--</td>
<td>IMPORT DATA (in CICS)</td>
<td>Suspend</td>
</tr>
<tr>
<td>DXYnPIMC_10_08</td>
<td>--</td>
<td>IMPORT TABLE (in CICS)</td>
<td>Suspend</td>
</tr>
<tr>
<td>DXYnPLST_01</td>
<td>--</td>
<td>LIST (QUERIES, PROCS, FORMS, ANALYTICS, QMF, TABLES, or ALL)</td>
<td>Type</td>
</tr>
<tr>
<td>DXYnPLST_08</td>
<td>--</td>
<td>LIST (QUERIES, PROCS, FORMS, ANALYTICS, QMF, TABLES, or ALL)</td>
<td>Location</td>
</tr>
<tr>
<td>DXYnPPRT_08</td>
<td>02 - 07</td>
<td>PRINT (in TSO)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPPRT_01</td>
<td>02 - 07</td>
<td>PRINT QUERY (in TSO)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPPRT_02</td>
<td>02 - 07</td>
<td>PRINT PROC (in TSO)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPPRT_03</td>
<td>02 - 07</td>
<td>PRINT FORM (in TSO)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPPRT_04</td>
<td>02 - 07</td>
<td>PRINT PROFILE (in TSO)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPPRT_05</td>
<td>02 - 07</td>
<td>PRINT REPORT (in TSO)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPPRT_07</td>
<td>02 - 07</td>
<td>PRINT CHART (in TSO)</td>
<td>Name</td>
</tr>
</tbody>
</table>
Table 60. Mapping between DXY global variables and panel field names (continued)

<table>
<thead>
<tr>
<th>Global variable name (where ( n ) is a national language identifier and ( ln ) is an ID associated with a line of a multiline field)</th>
<th>Range of ( ln ) values</th>
<th>Command</th>
<th>Field name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DXYnPPRT_ln_08</td>
<td>02 - 07</td>
<td>PRINT TABLE (in TSO)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPPR2_01_07</td>
<td>–</td>
<td>PRINT CHART, then Enter (in TSO)</td>
<td>Printer</td>
</tr>
<tr>
<td>DXYnPPR3_01_01</td>
<td>–</td>
<td>PRINT QUERY, then Enter (in TSO)</td>
<td>Printer</td>
</tr>
<tr>
<td>DXYnPPR3_01_02</td>
<td>–</td>
<td>PRINT PROC, then Enter (in TSO)</td>
<td>Printer</td>
</tr>
<tr>
<td>DXYnPPR3_01_03</td>
<td>–</td>
<td>PRINT FORM, then Enter (in TSO)</td>
<td>Printer</td>
</tr>
<tr>
<td>DXYnPPR3_01_04</td>
<td>–</td>
<td>PRINT PROFILE, then Enter (in TSO)</td>
<td>Printer</td>
</tr>
<tr>
<td>DXYnPPR3_01_08</td>
<td>–</td>
<td>PRINT TABLE, then Enter (in TSO)</td>
<td>Printer</td>
</tr>
<tr>
<td>DXYnPPR4_01_05</td>
<td>–</td>
<td>PRINT REPORT, then Enter (in TSO)</td>
<td>Printer</td>
</tr>
<tr>
<td>DXYnPPR5_ln</td>
<td>02 - 07</td>
<td>PRINT (in CICS)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPPR5_ln_01</td>
<td>02 - 07</td>
<td>PRINT QUERY (in CICS)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPPR5_ln_02</td>
<td>02 - 07</td>
<td>PRINT PROC (in CICS)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPPR5_ln_03</td>
<td>02 - 07</td>
<td>PRINT FORM (in CICS)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPPR5_ln_04</td>
<td>02 - 07</td>
<td>PRINT PROFILE (in CICS)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPPR5_ln_05</td>
<td>02 - 07</td>
<td>PRINT REPORT (in CICS)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPPR5_ln_07</td>
<td>02 - 07</td>
<td>PRINT CHART (in CICS)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPPR5_ln_08</td>
<td>02 - 07</td>
<td>PRINT TABLE (in CICS)</td>
<td>Name</td>
</tr>
<tr>
<td>DXYnPPR5_08</td>
<td>–</td>
<td>PRINT (in CICS)</td>
<td>Queue Name</td>
</tr>
<tr>
<td>DXYnPPR5_08_01</td>
<td>–</td>
<td>PRINT QUERY (in CICS)</td>
<td>Queue Name</td>
</tr>
<tr>
<td>DXYnPPR5_08_02</td>
<td>–</td>
<td>PRINT PROC (in CICS)</td>
<td>Queue Name</td>
</tr>
<tr>
<td>DXYnPPR5_08_03</td>
<td>–</td>
<td>PRINT FORM (in CICS)</td>
<td>Queue Name</td>
</tr>
<tr>
<td>DXYnPPR5_08_04</td>
<td>–</td>
<td>PRINT PROFILE (in CICS)</td>
<td>Queue Name</td>
</tr>
<tr>
<td>DXYnPPR5_08_05</td>
<td>–</td>
<td>PRINT REPORT (in CICS)</td>
<td>Queue Name</td>
</tr>
</tbody>
</table>
Table 60. Mapping between DXY global variables and panel field names (continued)

<table>
<thead>
<tr>
<th>Global variable name (where ( n ) is a national language identifier and ( ln ) is an ID associated with a line of a multiline field)</th>
<th>Range of ( ln ) values</th>
<th>Command</th>
<th>Field name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DXYnnPPR5_08_07</td>
<td>–</td>
<td>PRINT CHART (in CICS)</td>
<td>Queue Name</td>
</tr>
<tr>
<td>DXYnnPPR5_08_08</td>
<td>–</td>
<td>PRINT TABLE (in CICS)</td>
<td>Queue Name</td>
</tr>
<tr>
<td>DXYnnPPR5_09</td>
<td>–</td>
<td>PRINT (in CICS)</td>
<td>Queue Type</td>
</tr>
<tr>
<td>DXYnnPPR5_09_01</td>
<td>–</td>
<td>PRINT QUERY (in CICS)</td>
<td>Queue Type</td>
</tr>
<tr>
<td>DXYnnPPR5_09_02</td>
<td>–</td>
<td>PRINT PROC (in CICS)</td>
<td>Queue Type</td>
</tr>
<tr>
<td>DXYnnPPR5_09_03</td>
<td>–</td>
<td>PRINT FORM (in CICS)</td>
<td>Queue Type</td>
</tr>
<tr>
<td>DXYnnPPR5_09_04</td>
<td>–</td>
<td>PRINT PROFILE (in CICS)</td>
<td>Queue Type</td>
</tr>
<tr>
<td>DXYnnPPR5_09_05</td>
<td>–</td>
<td>PRINT REPORT (in CICS)</td>
<td>Queue Type</td>
</tr>
<tr>
<td>DXYnnPPR5_09_07</td>
<td>–</td>
<td>PRINT CHART (in CICS)</td>
<td>Queue Type</td>
</tr>
<tr>
<td>DXYnnPPR5_09_08</td>
<td>–</td>
<td>PRINT TABLE (in CICS)</td>
<td>Queue Type</td>
</tr>
<tr>
<td>DXYnnPPR5_10</td>
<td>–</td>
<td>PRINT (in CICS)</td>
<td>Suspend</td>
</tr>
<tr>
<td>DXYnnPPR5_10_01</td>
<td>–</td>
<td>PRINT QUERY (in CICS)</td>
<td>Suspend</td>
</tr>
<tr>
<td>DXYnnPPR5_10_02</td>
<td>–</td>
<td>PRINT PROC (in CICS)</td>
<td>Suspend</td>
</tr>
<tr>
<td>DXYnnPPR5_10_03</td>
<td>–</td>
<td>PRINT FORM (in CICS)</td>
<td>Suspend</td>
</tr>
<tr>
<td>DXYnnPPR5_10_04</td>
<td>–</td>
<td>PRINT PROFILE (in CICS)</td>
<td>Suspend</td>
</tr>
<tr>
<td>DXYnnPPR5_10_05</td>
<td>–</td>
<td>PRINT REPORT (in CICS)</td>
<td>Suspend</td>
</tr>
<tr>
<td>DXYnnPPR5_10_07</td>
<td>–</td>
<td>PRINT CHART (in CICS)</td>
<td>Suspend</td>
</tr>
<tr>
<td>DXYnnPPR5_10_08</td>
<td>–</td>
<td>PRINT TABLE (in CICS)</td>
<td>Suspend</td>
</tr>
<tr>
<td>DXYnnPRNM_( ln )</td>
<td>02 - 07</td>
<td>RENAME</td>
<td>Old Name</td>
</tr>
<tr>
<td>DXYnnPRNM_( ln )</td>
<td>08 - 10</td>
<td>RENAME</td>
<td>New Name</td>
</tr>
<tr>
<td>DXYnnPRST_01</td>
<td>–</td>
<td>RESET</td>
<td>Type</td>
</tr>
</tbody>
</table>
| DXYnnPRSG_01 | – | RESET GLOBAL | Enter ALL ...
| DXYnnPRSG_\( ln \) | 02 - 11 | RESET GLOBAL | Global variable name |
| DXYnnPRUN_\( ln \) | 02 - 07 | RUN | Name |
| DXYnnPRUN_\( ln \)_01 | 02 - 07 | RUN QUERY | Name |
Table 60. Mapping between DXY global variables and panel field names (continued)

<table>
<thead>
<tr>
<th>Global variable name (where ( n ) is a national language identifier and ( ln ) is an ID associated with a line of a multiline field)</th>
<th>Range of ( ln ) values</th>
<th>Command</th>
<th>Field name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DXY( n )PRUN(_{ln} _02)</td>
<td>02 - 07</td>
<td>RUN PROC</td>
<td>Name</td>
</tr>
<tr>
<td>DXY( n )PRU3(_{ln} )</td>
<td>01 - 06</td>
<td>RUN QUERY, then Enter</td>
<td>Form</td>
</tr>
<tr>
<td>DXY( n )PRU3(_{08} )</td>
<td>–</td>
<td>RUN QUERY, then Enter</td>
<td>Rowlimit</td>
</tr>
<tr>
<td>DXY( n )PRU3(_{ln} )</td>
<td>09 - 14</td>
<td>RUN QUERY, then Enter</td>
<td>Analytic</td>
</tr>
<tr>
<td>DXY( n )PRU3(_{ln} )</td>
<td>15 - 20</td>
<td>RUN QUERY, then Enter</td>
<td>Table</td>
</tr>
<tr>
<td>DXY( n )PRU3(_{22} )</td>
<td>–</td>
<td>RUN QUERY, then Enter</td>
<td>Comment</td>
</tr>
<tr>
<td>DXY( n )PRU3(_{23} )</td>
<td>–</td>
<td>RUN QUERY, then Enter</td>
<td>Space</td>
</tr>
<tr>
<td>DXY( n )PRU3(_{ln} )</td>
<td>24 - 26</td>
<td>RUN QUERY, then Enter</td>
<td>Accelerator</td>
</tr>
<tr>
<td>DXY( n )PRU4(_{01} )</td>
<td>–</td>
<td>RUN PROC, then Enter</td>
<td>Arg</td>
</tr>
<tr>
<td>DXY( n )PSAV(_{01} )</td>
<td>–</td>
<td>SAVE DATA</td>
<td>Name</td>
</tr>
<tr>
<td>DXY( n )PSA2(_{ln} )</td>
<td>01 - 06</td>
<td>SAVE DATA</td>
<td>Name</td>
</tr>
<tr>
<td>DXY( n )PSA2(_{08} )</td>
<td>–</td>
<td>SAVE DATA</td>
<td>Comment</td>
</tr>
<tr>
<td>DXY( n )PSA2(_{10} )</td>
<td>–</td>
<td>SAVE DATA</td>
<td>Space</td>
</tr>
<tr>
<td>DXY( n )PSA2(_{ln} )</td>
<td>11 - 13</td>
<td>SAVE DATA</td>
<td>Accelerator</td>
</tr>
<tr>
<td>DXY( n )PSA3(_{ln} _01)</td>
<td>01 - 06</td>
<td>SAVE QUERY</td>
<td>Name</td>
</tr>
<tr>
<td>DXY( n )PSA3(_{ln} _02)</td>
<td>01 - 06</td>
<td>SAVE PROC</td>
<td>Name</td>
</tr>
<tr>
<td>DXY( n )PSA3(_{09} _01)</td>
<td>–</td>
<td>SAVE QUERY</td>
<td>Comment</td>
</tr>
<tr>
<td>DXY( n )PSA3(_{09} _02)</td>
<td>–</td>
<td>SAVE PROC</td>
<td>Comment</td>
</tr>
<tr>
<td>DXY( n )PSA3(_{ln} _01)</td>
<td>10 - 12</td>
<td>SAVE QUERY</td>
<td>Folder</td>
</tr>
<tr>
<td>DXY( n )PSA3(_{ln} _02)</td>
<td>10 - 12</td>
<td>SAVE PROC</td>
<td>Folder</td>
</tr>
<tr>
<td>DXY( n )PSA4(_{ln} )</td>
<td>01 - 06</td>
<td>SAVE FORM</td>
<td>Name</td>
</tr>
<tr>
<td>DXY( n )PSA4(_{09} )</td>
<td>–</td>
<td>SAVE FORM</td>
<td>Comment</td>
</tr>
<tr>
<td>DXY( n )PSA4(_{ln} )</td>
<td>11 - 13</td>
<td>SAVE FORM</td>
<td>Folder</td>
</tr>
<tr>
<td>DXY( n )PSET(_{01} )</td>
<td>–</td>
<td>SET</td>
<td>Type</td>
</tr>
<tr>
<td>DXY( n )PSGL(_{ln} )</td>
<td>01 - 19 (even numbers)</td>
<td>SET GLOBAL</td>
<td>Var</td>
</tr>
<tr>
<td>DXY( n )PSGL(_{ln} )</td>
<td>02 - 20 (odd numbers)</td>
<td>SET GLOBAL</td>
<td>Value</td>
</tr>
<tr>
<td>DXY( n )PSHO(_{01} )</td>
<td>–</td>
<td>SHOW</td>
<td>Enter the name ...</td>
</tr>
</tbody>
</table>
Notices

This information was developed for products and services offered in the US. This material may be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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, MD-NC119
Armonk, NY 10504-1785
US

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

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbash-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

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

© Copyright IBM Corp. 1982, 2016
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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 Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

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 document 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.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate 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. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as shown below.

© (your company name) (year).
Portions of this code are derived from IBM Corp. Sample Programs.
© Copyright IBM Corp. _enter the year or years_.

Programmings interface information

This publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of QMF.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at http://www.ibm.com/legal/copytrade.shtml.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

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

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

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

abnormal end of task (abend)
The termination of a task, job, or subsystem because of an error condition that recovery facilities cannot resolve during execution.

address space
The range of addresses available to a computer program or process. Address space can refer to physical storage, virtual storage, or both.

Advanced Program-to-Program Communication
See APPC.

aggregate 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 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
An alternative name used to identify a table, view, database, or nickname. An alias can be used in SQL statements to refer to a table, view, or database in the same Db2 system or subsystem or in a remote Db2 system or subsystem.

APAR (Authorized Program Analysis Report)
A request for correction of a defect in a supported release of an program supplied by IBM.

APF (authorized program facility)
In a z/OS environment, a facility that permits the identification of programs that are authorized to use restricted functions.

API (application programming interface)
An interface that allows an application program that is written in a high-level language to use specific data or functions of the operating system or another program.

application
One or more computer programs or software components that use QMF services to provide functionality in direct support of a specific business process or processes.

APPC (Advanced Program-to-Program Communication)
An implementation of the SNA LU 6.2 protocol that allows interconnected systems to communicate and share the processing of programs.

application plan
The control structure that is produced during the bind process. The default name for the QMF Version 12.1 application plan is QMF12.

application programming interface
See API.

application requester
The source of a request to a remote DRDA-enabled relational database management system (RDBMS). Only Db2 for z/OS databases can function as application requesters because this is the only type of database in which QMF can be started.

application server
The target of a request from an application requester. The database management system (DBMS) at the application server site services the request. Connectivity with remote servers is not supported when QMF for TSO is running as a Db2 for z/OS stored procedure.

argument
A value passed to or returned from a function or procedure at run time.

authorization identifier (authorization ID)
A character string that designates a set of privileges and can be used to verify authority. An authorization ID can represent an object, an individual user, an...
organizational group, a function, or a database role. QMF authenticates either the database authorization ID or, optionally, the QMF TSO logon ID, against the CREATOR column of the Q.PROFILES table during QMF initialization.

**Authorized Program Analysis Report**
See APAR.

**Authorized program facility**
See APF.

**auxiliary table**
A table that stores columns outside the table in which they are defined. See also base table.

**base product**
The English-language version of QMF, established when QMF is installed. Any other language environment is established after installation by installing the National Language Feature (NLF) associated with that language.

**base table**
A table that is created by the SQL CREATE TABLE statement and that holds persistent data.

**binary string**
A sequence of bytes that is not associated with a coded character set and therefore is never converted. For example, the BLOB data type is a binary string. See also CCSID.

**bind**
To convert the output from the DBMS precompiler to a usable control structure, such as an access plan, an application plan, or a package.

**bit data**
Data with a data type of CHAR or VARCHAR that is not associated with a coded character set and therefore is never converted.

**buffer pool**
An area of memory into which data pages are read and in which they are modified and held during processing. See also address space.

**built-in function**
A strongly typed, high-performance function that is integral to the Db2 database. A built-in function can be referenced in SQL statements anywhere that an expression is valid.

**CAF (call attachment facility)**
A Db2 for z/OS attachment facility for application programs that run in TSO or z/OS batch. The CAF is an alternative to the DSN command processor and provides greater control over the execution environment.

**call attachment facility**
See CAF.

**callable interface**
A programming interface that provides access to QMF objects and services.

**cascade delete**
A process by which the Db2 database manager enforces referential constraints by deleting all descendent rows of a deleted parent row.

**catalog**
A collection of tables and views that contains descriptions of objects such as tables, views, and indexes. See also QMF object catalog.

**CCSID (coded character set identifier)**
A 16-bit number that includes a specific set of encoding scheme identifiers, character set identifiers, code page identifiers, and other information that uniquely identifies the coded graphic-character representation. Because QMF uses display services provided by GDDM, the GDDM application code page must agree with the CCSIDs in use for the database. See also binary string.

**character string**
A sequence of bytes that represents bit data, single-byte characters, or a mixture of single-byte and multibyte characters.

**check constraint**
A user-defined constraint that specifies the values that specific columns of a base table can contain. See also constraint.

**CICS (Customer Information Control System)**
An IBM licensed program that provides online transaction-processing services and management for business applications.

**clause**
In SQL, a distinct part of a statement in the language structure, such as a SELECT clause or a WHERE clause.
CM (Compatibility Mode)

An installation mode of QMF Version 8.1 and QMF Version 9.1 that limited owner and object names in the QMF object catalog to eight and 18 characters, respectively. See also NFM.

code page

A particular assignment of code points to graphic characters. Within a given code page, a code point can have only one specific meaning. A code page also identifies how undefined code points are handled.

coded character set identifier

See CCSID.

colleague

The state during which two QMF releases exist in the same Db2 subsystem. QMF Version 12.1 can coexist with QMF Version 9.1 New Function Mode or QMF Version 8.1 New Function Mode only.

column

The vertical component of a database table. A column has a name and a particular data type (for example, character, decimal, or integer).

column function

See aggregate function.

column wrapping

The value formatting in a report where the values occupy several lines within a column. Column wrapping is often used when a column contains values whose length exceeds the column width, such as cases requiring the display of XML data.

command interface

An interface for issuing QMF commands. 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.

command synonym

The verb or verb/object part of a site-defined command. After command synonyms are defined and activated in the QMF profile, users can enter the synonyms on the QMF command line as they do with regular QMF commands.

command synonym table

A table that stores one site-defined command in each row. You assign a set of command synonyms to a user by storing the name of this table in the user’s profile.

comparison operator

In SQL, a symbol used in comparison expressions to specify a relationship between two values. Comparison operators are = (equal to), <> (not equal to), < (less than), > (greater than), <= (less than or equal to), and >= (greater than or equal to).

Compatibility Mode

See CM.

commit

To end a unit of work by releasing locks so that the database changes made by that unit of work can be perceived by other processes. This operation makes the data changes permanent.

concatenation

Joining two characters or strings to form one string.

connection

In data communication, an association established between entities for conveying information. See also SQL connection. Connectivity with remote servers is not supported when QMF for TSO is running as a Db2 for z/OS stored procedure.

constant

A language element that specifies an unchanging value. Constants are classified as string constants or numeric constants.

constraint

A rule that limits the values that can be inserted, deleted, or updated in a table.

control section

See CSECT.

control tables

A set of tables that QMF uses to store information about QMF objects and manage QMF operations. See also QMF object catalog.

correlated reference

A reference to a column of a table or view that is outside a subquery.

correlation name

An identifier specified and used within a single SQL statement as the exposed name for objects such as a table, view,
database administrator
A person who is responsible for the design, development, operation, security, maintenance, and use of a database.

database management system
See DBMS.

database manager
A program that manages data by providing centralized control, data independence, and complex physical structures for efficient access, integrity, recovery, concurrency control, privacy, and security.

database server
A software program that uses a database manager to provide database services to other software programs or computers.

DBCS (double-byte character set)
A set of characters in which each character is represented by two bytes. These character sets are commonly used by national languages such as Japanese and Chinese, which have more symbols than can be represented by a single byte. See also SBCS.

DBMS (database management system)
A software system that controls the creation, organization, and modification of a database and the access to the data that is stored within it.

DCT (destination control table)
A table describing each of the transient data destinations used in CICS. This table contains an entry for each extrapartition, intrapartition, and indirect destination.

default form
The QMF form created when a saved form is not specified on the RUN QUERY command.

default value
A predetermined value, attribute, or option that is assumed when no other value is specified. A default value can be defined for column data in Db2 tables by specifying the DEFAULT keyword in an SQL statement that changes data (such as INSERT, UPDATE, and MERGE).

dependent row
A row that contains a foreign key that matches the value of a parent key in the parent row. The foreign key value

current location
The application server to which the QMF session is currently connected. After the connection is made, this server processes all SQL statements. When initializing QMF, the current location can be indicated using the DSQSDBNM startup parameter. Connectivity with remote servers is not supported when QMF for TSO is running as a Db2 for z/OS stored procedure.

current object
A QMF object that is held in temporary storage so that, with each use, it can be readily accessed without requiring database retrieval. There are seven temporary storage areas: QUERY, FORM, PROC, PROFILE, REPORT, DATA, and CHART. Users can navigate to all areas but the DATA area using the SHOW and DISPLAY commands. See also temporary storage.

cursor
A named control structure used by an application program to point to and select a row of data from a set.

Customer Information Control System
See CICS.

data type
A classification identifying one of various kinds of data. In SQL, the data type is an attribute of columns, literals, host variables, special registers, parameters, and the results of functions and expressions.

database
A collection of interrelated or independent data items that are stored together to serve one or more applications.
represents a reference from the dependent row to the parent row.

dependent table
A table that is a dependent of an object. For example, a table with a foreign key is a dependent of the table containing the corresponding primary key.

destination control table
See DCT.

detail block text
The text in the body of a report that is associated with a particular row of data.

detail heading text
The text in the heading of a report.

detail variation
A data formatting definition specified on a FORM.DETAIL panel that can be used to conditionally format a report or part of a report.

distinct type
A user-defined data type that shares a common representation with built-in data types.

distributed data
Data that is stored on more than one system and is available to remote users and application programs.

distributed database
A database that appears to users as a logical whole, locally accessible database, but consists of databases in multiple locations that are connected by a data communications network.

Distributed Relational Database Architecture™
See DRDA.

distributed unit of work
A form of distributed relational database processing that enables a user or application program to read or update data at multiple locations within a unit of work. Within one unit of work, an application, such as QMF, running in one system can direct SQL requests to multiple remote database management systems using the SQL supported by those systems. The request is made through a QMF command that includes a three-part table or view name. QMF commands with three-part names cannot be directed to Db2 for VM or VSE databases or used when QMF for TSO has been started as a Db2 for z/OS stored procedure. Three-part names in QMF commands also cannot refer to a table that contains large object (LOB) data types.

double-byte character set
See DBCS.

double-precision floating-point number
A 64-bit approximate representation of a real number.

DRDA (Distributed Relational Database Architecture)
The architecture that defines formats and protocols for providing transparent access to remote data. DRDA defines two types of functions: the application requester function and the application server function.

environment
A named collection of logical and physical resources used to support the performance of a function.

exit routine
A program that receives control from another program to perform specific functions.

Extensible Markup Language
See XML.

extended syntax
Syntax that is used for the QMF SET GLOBAL and GET GLOBAL commands and certain function calls in a callable interface application. Extended syntax defines parameters used by QMF callable interface applications written in Assembler, C, COBOL, Fortran, or PL/I.

fallback
The process of returning to a prior release of a software program after attempting or completing migration to a current release.

fetch
The process of retrieving rows from the database or a file to create a QMF DATA object. QMF supports multirow fetch through the use of the DSQSMRFI parameter.

foreign key
In a relational database, a key in one table that references the primary key in another table.
GDDM (Graphical Data Display Manager)
Graphics software that defines and displays text and graphics for output on a display device or printer.

global variable
A named entity whose value persists for the duration of a QMF session by default. QMF uses global variables to manage both session and database activity. Some global variables can be set with the SET GLOBAL command, while others record information about the state of the current QMF session and therefore cannot be set.

graphic string
A sequence of double-byte character set (DBCS) characters.

Graphical Data Display Manager
See GDDM.

host
The controlling or highest-level system in a data communications configuration.

HTML (hypertext markup language)
A markup language that conforms to the Standard Generalized Markup Language (SGML) standard and was designed primarily to support the online display of textual and graphical information, including hypertext links.

hypertext markup language
See HTML.

ICU (Interactive Chart Utility)
A menu-driven component of IBM's Graphical Data Display Manager (GDDM) product that allows non-programmers to display, print, or plot charts, graphs, and diagrams.

identity column
A column that provides a way for the Db2 database manager to automatically generate a numeric value for each row that is inserted into a table. Identity columns are defined with the AS IDENTITY clause. A table can have no more than one identity column.

index
A set of pointers that is logically ordered by the values of a key. Indexes provide quick access to data and can enforce uniqueness of the key values for the rows in the table.

inner join
The result of a join operation that includes only the matched rows of both tables that are being joined. See also outer join.

installation verification procedure
See IVP.

Integrated Exchange Format
See IXF.

Interactive Chart Utility
See ICU.

Interactive System Productivity Facility
See ISPF.

ISPF (Interactive System Productivity Facility)
An IBM licensed program that serves as a full-screen editor and dialog manager. Used for writing application programs, it provides a means of generating standard screen panels and interactive dialogs between the application programmer and terminal user.

IVP (installation verification procedure)
A procedure or program whose purpose is to verify that a product has been correctly installed.

IXF (Integrated Exchange Format)
A protocol for transferring tabular data among various software products.

JCL (job control language)
A command language that identifies a job to an operating system and describes the job's requirements.

job control language
See JCL.

join
An SQL relational operation that allows retrieval of data from two or more tables based on matching column values.

key
A column or an ordered collection of columns that is identified in the description of a table, index, or referential constraint. The same column can be part of more than one key.

keyword
One of the predefined words of a programming language, artificial language, application, or command.

keyword parameter
A parameter that consists of a keyword followed by one or more values. See also positional parameter.
large object

See LOB.

link-edit

To create a loadable computer program by means of a linkage editor.

linkage editor

A computer program for creating load modules from one or more object modules or load modules by resolving cross-references among the modules and, if necessary, adjusting addresses.

literal

A character string whose value is defined by the characters themselves. For example, the numeric constant 7 has the value 7, and the character constant 'CHARACTERS' has the value CHARACTERS.

linear procedure

A sequenced set of QMF commands or command synonyms that can be used to perform several operations at once. 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.

load module

A program in a form suitable for loading into main storage for execution.

LOB (large object)

A sequence of bytes with a size ranging from 0 bytes to 2 gigabytes (less 1 byte). There are three LOB data types: binary large object (BLOB), character large object (CLOB, which can include single-byte characters only or a mixture of single-byte and double-byte characters), and double-byte character large object (DBCLOB). QMF supports a LOB column size of up to 32 KB.

local

Pertaining to databases, objects, or applications that are installed or stored in the system in which QMF is currently running.

location

A specific relational database server in a distributed relational database system. Each location has a unique location name.

location name

The unique name of a database server. An application uses the location name to access a Db2 database server.

lock

A means of serializing a sequence of events or serializing access to data.

log

A collection of records that sequentially describes the events that occur in a system.

LUW

An abbreviation for Linux, UNIX, and Windows.

National Language Feature

See NLF.

New Function Mode

See NFM.

NFM (New Function Mode)

An installation mode of QMF Version 8.1 and QMF Version 9.1 that allowed owner and object names in the QMF object catalog to be the maximum length allowed by the database. QMF Version 12.1 allows owner and object names to be as long as the database allows as well. See also CM.

NLF (National Language Feature)

Any of several optional features available with QMF. NLFs allow users to interact with QMF in specific native languages.

object

A named storage space that consists of a set of characteristics that describe the space and, in some cases, data. An object is anything that occupies space in storage, can be located in a library or directory, can be secured, and on which defined operations can be performed. See also QMF object.

outer join

The result of a join operation that includes the matched rows of both tables that are being joined and preserves some or all of the unmatched rows of the tables that are being joined. See also inner join.

package

A control-structure database object produced during program preparation that can contain both executable forms of static SQL statements or XQuery expressions and placement holders for executable forms of dynamic SQL statements.

panel

A formatted display of information on a screen that can also include entry fields.
**parameter**
A value or reference passed to a function, command, or program that serves as input or controls actions. The value is supplied by a user or by another program or process.

**partition**
A portion of a page set. Each partition corresponds to a single, independently extendable data set. Partitions can be extended to a maximum size of 1, 2, or 4 gigabytes, depending on the number of partitions in the partitioned page set. All partitions of a given page set have the same maximum size.

**plan**
See *application plan*.

**positional parameter**
A parameter that must appear in a specified location, relative to other parameters. See also *keyword parameter*.

**precision**
An attribute of a number that describes the total number of significant digits.

**predicate**
An element of a search condition that expresses or implies a comparison operation.

**primary authorization ID**
The authorization identifier used to identify an application process to Db2 for z/OS.

**primary key**
In a relational database, a key that uniquely identifies one row of a database table.

**privilege**
In SQL, a capability given to a user by the processing of a GRANT statement.

**procedure**
A sequenced set of statements or commands used to perform one or more tasks. See also *linear procedure* and *procedure with logic*.

**procedure with logic**
A set of statements that performs one or more tasks. A procedure with logic begins with a REXX comment and allows conditional logic (which uses REXX), calculations, build strings, and TSO or CICS commands. See also *linear procedure*.

**profile**
An object that contains information about the characteristics of the user’s session.

**program temporary fix**
See *PTF*.

**prompted query**
A menu-driven query controlled by user-provided parameters.

**PTF (program temporary fix)**
For System i®, System p, and System z®, products, a fix that is tested by IBM and is made available to all customers.

**QBE (Query-by-Example)**
A component of QMF that allows users to create queries graphically.

**QMF administrator authority**
Authority that allows a user to insert or delete rows in the Q.PROFILES control table. Users with this authority can perform the following commands on QMF queries, forms, and procedures that are owned by other users without forcing the owners to share these objects with all users: *SAVE, ERASE, IMPORT, EXPORT, and DISPLAY*. QMF checks each user ID for administrator authority during initialization; you can disable this checking by setting the DSQEC_DISABLEADM variable in the DSQUOPTS exit routine or in another program of your choice.

**QMF administrator**
A user who has QMF administrator authority.

**Query-by-Example**
See *QBE*.

**QMF object**
An object used by QMF users to query, format, and present data or otherwise manage interaction between QMF and the database. QMF objects include queries and query result data, forms, procedures, reports, charts, and the QMF profile. Each QMF object has a named temporary storage area that is used to display the object. All objects except reports and charts can be saved in the database; reports and charts are created dynamically upon user request by applying the formatting specifications of a particular QMF form to result data that
has been returned from the database. See also temporary storage.

**QMF object catalog**
A set of control tables that stores information about QMF queries, procedures, forms, folders, and analytics objects. These control tables include Q.OBJECT_DIRECTORY, Q.OBJECT_DATA, and Q.OBJECT_REMARKS.

**qualifier**
When referring to a QMF object, the part of the name that identifies the owner or the location of an object. 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**
A request for information from a database based on specific conditions: for example, a request for a list of all customers in a customer table whose balances are greater than $1000. In QMF, a query also refers to SQL statements submitted from the Prompted Query, QBE, or SQL query panel, even if these statements are not requests for information (SELECT statements).

**RCT (resource control table)**
A Db2 control table that defines the relationship between CICS transactions and Db2 resources.

**RDBMS (relational database management system)**
A collection of hardware and software that organizes and provides access to a relational database.

**RDO (resource definition online)**
In CICS, a facility that allows the user to define certain CICS resources interactively while CICS is running. Specifically, RDO allows the user to define terminals, programs, and transactions interactively.

**record**
The storage representation of a row or other data.

**record length**
The length of storage that represents a row or other data.

**reentrant**
Executable code that can reside in storage as one shared copy for all database threads. Reentrant code is not self-modifying and provides separate storage areas for each thread.

**referential constraint**
The requirement that the nonnull values of a designated foreign key are valid only if they also appear as values of the primary key of the parent table. The referential constraint is always defined from the perspective of the dependent file.

**relational database**
A database that can be perceived as a set of tables and manipulated in accordance with the relational model of data. Each database includes a set of system catalog tables that describe the logical and physical structure of the data, a configuration file containing the parameter values allocated for the database, and a recovery log with ongoing transactions and archivable transactions.

**relational database management system**
See RDBMS.

**remote**
Pertaining to databases, objects, or applications that are installed or stored on a system other than the system where QMF is currently executing. You can access objects (including QMF queries, forms, procedures, folders, and analytics objects) at a remote server by using the QMF CONNECT command. You can also use a QMF command with a three-part table or view name if you want to access just tables or views at a remote location. Remote access is not permitted when QMF for TSO is running as a Db2 for z/OS stored procedure.

**remote unit of work**
A form of distributed relational database processing in which an application program, such as QMF, can access data on a remote database within a unit of work. The connection is established by the QMF CONNECT command. The CONNECT command cannot be used when QMF for TSO is running as a Db2 for z/OS stored procedure.

**requester**
See application requester.
resource
The object of a lock or claim, which could be a table space, an index space, a data partition, an index partition, or a logical partition.

resource control table
See RCT.

resource definition online
See RDO.

Restructured Extended Executor
See REXX.

REXX (Restructured Extended Executor)
A general-purpose, high-level programming language, particularly suitable for EXEC procedures or programs for personal computing.

roll back
To restore data that is changed by an SQL statement to the state at its last commit point. If a failure occurs in a query that contains multiple statements and no COMMIT statements, all statements, except those that affect the QMF session (such as SET), are rolled back. If a failure occurs in a query that contains one or more COMMIT statements, all updates after the last successful COMMIT statement are rolled back. In either case, the query ends after the failure.

routine
A program or sequence of instructions called by a program. Typically, a routine has a general purpose and is frequently used.

row
The horizontal component of a table, consisting of a sequence of values, one for each column of the table.

runtime 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 runtime variable is only available in the current procedure or query. See also global variable.

SBCS (single-byte character set)
A coded character set in which each character is represented by a 1-byte code. A 1-byte code point allows representation of up to 256 characters. See also double-byte character set.

scalar function
An SQL function that optionally accepts arguments and that returns a single scalar value each time that it is invoked. A scalar function can be referenced in an SQL statement wherever an expression is valid.

scratchpad area
A work area used in conversational processing to retain information from an application program across executions of the program.

search condition
A criterion for selecting rows from a table. A search condition consists of one or more predicates.

secondary authorization ID
In Db2 for z/OS, an authorization identifier that is associated with a primary authorization ID by an authorization exit routine. See also primary authorization ID.

segmented table space
A table space that is divided into equal-sized groups of pages called segments. Segments are assigned to tables so that rows of different tables are never stored in the same segment. See also table space.

server
See application server.

session
All interactions between the user and QMF from the time the user invokes QMF until the EXIT command is issued.

shift-in character
A control character (X'0F') that is used in EBCDIC systems to denote that the subsequent bytes represent SBCS characters. See also shift-out character.

shift-out character
A control character (X'0E') that is used in EBCDIC systems to denote that the subsequent bytes, up to the next shift-in control character, represent DBCS characters. See also shift-in character.

single-byte character set
See SBCS.

single-precision floating-point number
A 32-bit approximate representation of a real number.
SQL (Structured Query Language)
A standardized language for defining and manipulating data in a relational database.

SQL authorization ID
See SQLID.

SQL connection
An association between an application process and a local or remote application server or database server. See also remote unit of work, distributed unit of work.

SQL function
A function that is implemented entirely by using a subset of SQL statements and SQL PL statements.

SQL ID (SQL authorization ID)
In Db2 for z/OS, the ID that is used for checking the authorization of dynamic SQL statements in some situations.

SQL return code
The SQLSTATE or SQLCODE that indicates whether the previously run SQL statement completed successfully, with one or more warnings, or with an error.

SQLCA (Structured Query Language Communication Area)
A set of variables that provides an application program with information about the execution of its SQL statements or requests from the database manager. When an error is associated with an SQL code, the QMF message help (available by pressing the Help key) displays the contents of the SQLCA.

stored procedure
A routine that can be invoked using the SQL CALL statement to perform operations that can include both host language statements and SQL statements.

stored procedure interface
An interface to QMF for TSO that allows you to start QMF as a Db2 for z/OS stored procedure, pass the name of a QMF query or procedure that performs the work you require, and receive up to 21 result sets back, including a result set for trace output. QMF for TSO can be started in this manner from any product that can run a Db2 for z/OS stored procedure.

Structured Query Language
See SQL.

Structured Query Language Communication Area
See SQLCA.

subquery
A complete SQL query that appears in a WHERE or HAVING clause of another query.

substitution variable
(1) A variable in a procedure or query whose value is specified either by a global variable or by a runtime variable. (2) A variable in a QMF form whose value is specified by a global variable.

substring
A part of a character string.

subsystem
In Db2 for z/OS, a distinct instance of a relational database management system (RDBMS).

table
In a relational database, a database object that consists of a specific number of columns and is used to store an unordered set of rows. See also base table.

table space
A logical unit of storage in a database. In Db2 for z/OS, a table space is a page set and can contain one or more tables. In Db2 for Linux, UNIX, and Windows, a table space is a collection of containers, and the data, index, long field, and LOB portions of a table can be stored in the same table space or in separate table spaces.

temporary storage
An area used to store a QMF object temporarily while the user is working on it so that, with each use, it can be readily accessed without further database retrieval. There are seven temporary storage areas: QUERY, DATA, FORM, PROC, REPORT, CHART, or PROFILE. With the exception of query result data (the DATA object), the QMF objects in these areas can be displayed using the SHOW command followed by the name of the storage area. Though the contents of the DATA area cannot be directly displayed, users can issue the SHOW REPORT or SHOW CHART commands to see the query result data formatted with...
the specifications of the form currently in the FORM area. See also QMF object, current object.

temporary storage queue
In CICS, a queue of data items which can be read and reread, in any sequence. The queue is created by a task, and persists until the same task or another task deletes it. See also transient data queue.

thread
The Db2 structure that describes an application’s connection, traces its progress, processes resource functions, and delimits its accessibility to Db2 resources and services. Most Db2 functions execute under a thread structure.

three-part name
The full name of a table, view, or alias that consists of a location name, an authorization identifier, and an object name, separated by periods. QMF commands that include three-part names can be initiated only from Db2 for z/OS databases and can be directed to all databases except Db2 for VM or VSE. When QMF for TSO has been started as a Db2 for z/OS stored procedure, QMF commands with three-part names are not supported.

Time Sharing Option
See TSO.

trace
A record of the processing of a computer program or transaction. The information collected from a trace can be used to assess problems and performance.

transaction
A unit of processing consisting of one or more application programs, affecting one or more objects, that is initiated by a single request.

transient data queue
A CICS storage area where objects are stored for subsequent internal or external processing. See also temporary storage queue.

trigger
A database object that is associated with a single base table or view and that defines a rule. The rule consists of a set of SQL statements that runs when an insert, update, or delete database operation occurs on the associated base table or view.

TSO (Time Sharing Option)
A base element of the z/OS operating system that allows users to work interactively with the system.

two-phase commit
A two-step process by which recoverable resources in an external subsystem are committed. During the first step, the database manager subsystems are polled to ensure that they are ready to commit. If all subsystems respond positively, the database manager instructs them to commit.

UDF (user-defined function)
A function that is defined to the Db2 database system by using the CREATE FUNCTION statement and that can be referenced thereafter in SQL statements. A UDF can be an external function or an SQL function.

Unicode
A character encoding standard that supports the interchange, processing, and display of text that is written in the common languages around the world, plus some classical and historical texts. The Unicode standard has a 16-bit character set defined by ISO 10646.

unit of recovery (UR)
A sequence of operations within a unit of work between points of consistency.

unit of work (UOW)
A recoverable sequence of operations within an application process. At any time, an application process is a single UOW, but the life of an application process can involve many UOWs as a result of commit or rollback operations. In a multisite update operation, a single UOW can include several units of recovery. In QMF SQL queries that include multiple statements and no COMMIT statements, all statements comprise a single unit of work, so all statements except those that affect the session (such as SET) are rolled back in the event of a failure. In QMF SQL queries that include multiple statements and one or more COMMIT statements, a unit of work consists of a COMMIT.
statement and all previous statements
back to the beginning of the query or the
last COMMIT statement. If a failure
occurs, all updates after the last successful
COMMIT statement are rolled back.

user-defined function
See UDF.

view A logical table that is based on data
stored in an underlying set of tables. The
data returned by a view is determined by
a SELECT statement that is run on the
underlying tables.

XML (Extensible Markup Language)
A standard metalanguage for defining
markup languages that is based on
Standard Generalized Markup Language
(SGML).

z/OS An IBM mainframe operating system that
uses 64-bit real storage.
Index

Special characters
+ sign in Table Editor columns, changing 207

Numerics
64BIT option, DSQSPTYP parameter 51

A
A option for debugging 135
ADD command
  Table Editor confirmation 207
ADDRESS command 13, 29
administrator authority, global variables for 200, 218
alias
  view that retrieves aliases for LIST 218
application plan
default ID, V12.1 51
application programming interfaces
callable interface
  See callable interface command interface
  See commands, interface stored procedure (TSO only)
  See stored procedure interface
applications
  bilingual 37
  CICS environment 3
  command synonym 2
  commands 1
    INTERACT 44
    overview 41
    processing 17
  controlling 1
data records 100
debugging 135
developing 1
  implementation methods 3
  ISPF requirements 31
  procedures 7
  procedures with logic 3
  starting 2
types 1
ARG statement 11
arguments 11
ascending order for lists 210
Assembler
  CICS 139
    sample program 142
    z/OS 149
  communications area 148
  function calls 140
  High Level Assembler (HLASM) 139
  language interface 139
  macros 197
  sample program 141
  TSO sample programs 145, 150
attention flag for applications 200
callable interface (continued)
  ISPF 3
  languages 3, 17, 139
  macros 197
  PL/I 181
  program 3
  return codes 20
  REXX
    communications variables 191
    description 190
    invoking with 7
    uses 3
  sample programs 3
    Assembler 141
    C 155
    COBOL 165
  START command 3
    starting QMF 21
    syntax 50
CANCEL command
  Table Editor confirmation 207
carriage control characters, suppressing 218
CASE parameter of QMF profile 205
CCSID used for XML exports 90
cDATA tags in exported XML 90
cell tags in exported XML 90
CHANGE command
  Table Editor confirmation 207
CHAR data type
column width on export 69
  exporting
  column data format, IXF 75
  edit code keywords, exported form 103
SQLTYPE codes on export (QMF format) 69
CHAR keyword seen in exported forms 103
chart objects 129
CICS environment
  Assembler 3
  z/OS requirements 149
callable interface 3
  COBOL programs 169
  CONNECT command 8
data queue 3
  IXF format 75
  transient data queues 132
  using to transfer QMF objects 68
Db2 interaction 22
global variables related to 206
  program start parameter overrides 22
  region 22
CLEAR command
  Table Editor confirmation 207
COBOL
callable interface 162
  CICS 169
  communications area 162
COBOL (continued)
delimiters 168
DSQCOMM 166
execution requirements 168
function calls 163
ISPF 170
macros 197
sample program 165
TSO 169
coded character set identifier, XML exports 90
codes, SQL
See SQL codes
column
C records 75
data format 75
default indicator in Table Editor 207
DESCRIBE command
See DESCRIBE command
heading
labels vs. names 210
name lengths on EXPORT 69, 210
widths on export 69
command synonyms
creating 65
definitions 205
element 2
NLF table 35
commands
applications 41
bilingual applications 39
environment 29
global variables that support 200
globals that store message output 207
INTERACT 44
interface 2
description 25
invoking from a program 26
requirements 3
return codes 27
sample program 25
SELECT service 26
language variable 37
length 17
natural language, setting 218
return code 12
RUN 9
SET GLOBAL 48
system specific 8
comment
application data records 100, 112
exported formats 112
comments, sending to IBM v7
communications area
COBOL 162, 166
defining 18
FORTRAN 177
PL/I 186
Compatibility Mode and multirow fetch 51
concurrent access resolution 218
CONFIRM parameter of QMF profile 205
confirmation panels
Reset Report 218
temporary storage overwrites 218
CONNECT command
DB2 for VM or VSE 8
description 41
element 41
global variable for CONNECT ID 200
initial procedures 7
mixed-case passwords 8, 205
procedures 8
connectivity with remote servers
stored procedure interface
restrictions 41
control areas in exported objects
96
records of form files 96
records of report files 96
T records 98
control information
removing from reports 118
conventions for highlighting vi
CONVERT QUERY command
global variables for 233
restricting update of last used date 218
cost estimate for query, disabling 210
CSV data
exporting 93
CSV export format
file size maximum 130
currency symbol, changing 210
current location 41
cursor
stability, enabling 218
status of 200
D
D, DC, DZ, DZC edit codes
currency symbol, changing 210
data
D records 75
exporting 68
object
formats 69
IXF exported format 75
records, exporting 69, 90
type widths 69
DATA object
global variables related to 200
incomplete, enabling Reset Report panel 218
data record format
XML data type 90
data set, defining for exports 218
data types, export considerations 69, 103
database
SQL codes
See SQL codes
subsystem ID, global variable 200
uncommitted read vs. cursor stability 218
database manager, global that stores type 200
database remote connections 8
DATAFORMAT parameter values, EXPORT command
QMF 69, 210
DATE data type
column width on export 69
exporting
column data format, IXF 75
edit code keywords, exported form 103
ISO format on export 69
SQLTYPE codes on export (QMF format) 69
DATE keyword seen in exported forms 103
date last used, object lists 210, 218
date modified, ordering lists by 210
DB2 for VM or VSE
CONNECT command 8
remote connections 8
DBCS support
changing default indicator, Table Editor 207
changing null indicator, Table Editor 207
global variables related to 200
debugging applications
ISPF using 34
PDF dialog test 34
DECFLOAT data type
column width on export 69
exporting
column data format, IXF 75
edit code keywords, exported form 103
SQLTYPE codes on export (QMF format) 69
DECIMAL data type
column width on export 69
exporting
column data format, IXF 75
edit code keywords, exported form 103
SQLTYPE codes on export (QMF format) 69
decimal floating-point data type
See DECFLOAT data type
DECIMAL parameter of QMF profile 205
DELETE command
Table Editor confirmation 207
delimiters
between statements in SQL queries 218
descending order for lists 210
DESCRIBE command
views that support 218
directory blocks, specifying upon export 218
DISPLAY command
restricting update of last used date 218
dollar sign in reports, changing 210
dSQ parameters on START command
dSQADPAN 51
dSQALANG 51
dSQSBJSTG 51
dSQSCMD 51
dSQSDBCS 51
dSQSDBN 51
dSQSDBQ 51
DSQ parameters on START command (continued)

DSQSQDBQT 51
DSQSQDBUG 51
DSQSQIROW 51
DSQSMODE 51
DSQSMRFI 51
DSQSPILL 51
DSQSPLAN 51
DSQSPRIDE 51
DSQSTYP 51
DSQSRSTG 51
DSQSRUN 51
DSQQSPQON 51
DSQQSUBS 51
DSQ1SCEM schema file 90
DSQ1STSH style sheet file 90
DSQABFA 145
DSQABFAC 142
DSQAO global variables 200
DSQAP global variables 205, 206
DSQCLA 140
DSQCI subroutine 192
DSQCM global variables 207
DSQCOMM
Assembler 148
C 152, 158
COBOL 162
defining 18
DSQCOMMA 148
DSQCOMMC 158
error handling 22
message text 139
set fields 18
DSQCP global variables 207
DSQDEBUG trace log
See also tracing
logging positive SQL codes 210
DSQEC global variables 218
DSQEC_NLFCMD_LANG variable 37, 218
DSQQC global variables 233
DSQCOM global variables 234
DSQDSDBLG parameter 51
DSQEOPTS initialization routine 218
DSQY global variables 234

EXPORT command (continued)

DATA 68
data object 68
form, national language used 218
forms
  edit code keywords 103
  IXF option 75
restricting update of last used
  date 218
table object 68
TSO
  specifying storage 218
using CICS 132
XML specification used 90
exporting
  charts 129
  CSV data 93
data and tables 68
  file size maximums 130
form objects 67
forms 94
hex codes for data types 69
keywords used for edit codes 103
LOB data 90
object types 67
procedures 129
promoted queries 94
QBE queries 129
reports 94
SQL queries 129
storage considerations 132
versus saving 68
width calculations for data types 69
extended storage, using for spill data
  required program parameters 51
setting amount 218
extended syntax
  SET GLOBAL command 49
extended-format decimal floating-point data
  See DECIMAL data type
Extensible Markup Language (XML) data type
  See XML data type

form (continued)

FORM.MAIN
column names vs. labels 210
header records on export 103
importing 112
LAYOUT command 103
multicultural support for SAVE, EXPORT, IMPORT 218
panels
  globals related to 200
table numbers 103
translating 103
formats 113
column data 75
data, exporting 69
encoded
definition 67
form object 103
header record 69
IXF 67, 75
promoted query object 120
report object 113
no control information 118
table 68
FORTRAN
callable interface 171
communications area 172
DSQABFF 174
DSQCOMM 177
function calls 173
ISPF 179
macros 197
sample program 174
TSO 179
function keys
  where definitions are stored 205

G

GDDM (Graphical Data Display Manager) 44
GET GLOBAL command 20, 43
global variables
administrator authority 200
administrator authority 218
application trace level 200
batch vs. interactive operation 200
carriage control characters in
  printouts 218
CASE parameter of profile 205
CICS
  printing 206
  spill data 206
tracing 206
classes of
  DSQAO 200
  DSQAP 205, 206
  DSQCM 207
  DSQCP 207
  DSQEC 218
  DSQQC 233
  DSQQM 234
  DXY 234
global variables (continued)  
  classes of (continued)  
    stored procedure interface 200  
    column labels vs. names 210  
    command support 200  
    command synonym definitions 205  
    concurrent access resolution 218  
    CONNECT parameter of profile 205  
    CONNECTID 200  
    creating 48  
    creating variables 48  
    currency symbol 210  
    current form panel 200  
    current object 200  
    current panel name 200  
    database cursor status 200  
    database manager 200  
    DBCS support 200  
    EXPORT command storing/displaying 200  
    setting at initialization 200  
    setting/displaying 200  
    SHARE parameter of SAVE command 218  
    SPACE parameter of profile 205  
    SQL variables (continued)  
      SQL queries over 32 KB 218  
      stored procedure interface 200  
      subsystem ID 200  
      temporary storage overwrites 218  
      TRACE parameter of profile 205  
      user attention flag 200  
      version/release 200  
      WIDTH parameter of profile 205  
      GRAPHIC data type  
        column width on export 69  
        exporting  
        column data format, IXF 75  
        edit code keywords, exported  
        form 103  
      SQLTYPE codes on export (QMF format) 69  
      GRAPHIC keyword seen in exported  
      forms 103  
      Graphics Data Format (GDF) 129  

H  
  header record  
    fields 94  
    form object 103  
    format 69  
    IXF 75  
    object level 94  
    XML exports 90  
    headings, column  
    See heading  
    hex codes for exported data types 103  
    highlighting conventions vi  
    home panel 7  
    HTML format for reports  
    exporting in 117  
    file size maximum 130  

I  
  ICU (Interactive Chart Utility) 44  
  IMPORT command  
    accelerator tables 218  
    DATA option 93  
    definition 67  
    errors and warnings during execution  
    of 103  
    national language used, IMPORT  
    FORM 218  
    restricting update of last used  
    date 218  
    using CICS 132  
    XML specification used 90  
    importing 67, 94  
    form object 103  
    object level information 94  
    prompted query object 127  
    tables created outside QMF 68  
    incomplete data object  
    enabling Reset Report panel 218  
    prompt panel 93  
    initial procedures  
      bilingual applications 38  
      CONNECT command 7  
      name, specifying 7  

K  
  keywords 51  
  for edit codes, in exported file 103  
  START command 51  

L  
  L option for debugging 135  
  labels vs. names for column  
  headings 210  
  LANGUAGE parameter, QMF  
  profile 205  
  languages supported  
    programming for callable interface 3  
    translations  
    See multicultural support
last used date for objects
  limiting to RUN, SAVE, IMPORT 218
  sorting lists by 210
LAYOUT command 103
  restricting update of last used date 218
LENGTH parameter of QMF profile 205
  license agreement global variable for QMF VUE 200
  linear procedure
STOPPROC option 46
  suppressing 46
LIST command
  order of items, changing 210
  OWNER parameter default 218
  underlying views
    global variables that store view names 218
LOB data
  exporting 90
  location name
    See also remote data access
    global variable that stores 200
    locks on data
    concurrent access resolution options 218
    preventing escalation 218
    log, trace 210
LONG VARCHAR data type exporting
  column data format, IXF 75
LONG VARGRAPHIC data type exporting
  column data format, IXF 75
  long-format decimal floating-point data
    See DECFLOAT data type

M
  macros, product interface 197
  maximum lengths
    SQL queries 218
MESSAGE command
  description 46
  displaying text 46
  examples 46
  ISPF panels 46
  options 46
  QMF help panels 46
  suppressing linear procedure execution 46
  tracing 137
  messages
    global variables related to
      message support for positive SQL codes 210
      messages from prior command 207
    RUN QUERY messages 234
    migration information 94
    minisession
      invalid commands 62
      report 62
      valid commands 62
      mixed-case passwords 205
      mode of operation
        global variable that shows 200
        MODEL parameter, QMF profile 205
        monetary values, changing currency symbol 210
        MR parameter
          three-part name failures 51
          multicultural support
            forms (SAVE/EXPORT/IMPORT) 218
            global variables related to 200, 218
            multilingual environments 39
            multirow fetch 51
            three-part name failures 51
            multistatement queries 218
        names
          for columns, changing to database labels 210
          ordering lists by 210
          views that support LIST command, globals for 218
        National Language Feature (NLF)
          See multicultural support
        New Function Mode and multirow fetch 51
        NEXT command
          Table Editor confirmation 207
        NLF (National Language Feature)
          See also multicultural support
        defined 35
        language 35
        language ID 31
        multilingual environments 39
        panel requirements 35
        session environments 35
        notices
          legal 245
          notification of positive SQL codes 210
        numeric data
          currency symbol, changing 210
        NUMERIC keyword seen in exported forms 103
O
  object
    global variables related to
      current 200
      last used date 210, 218
      sharing 218
      type
        ordering lists by 210
        online help
          QMF message help
            displaying positive SQL codes 210
          operating system, global variable for 200
          order
            LIST command items, changing 210
    overwrites of temporary storage, preventing 218
    owner names
      default for LIST command 218
      global variables related to 200
      ordering lists by 210
P
  panels 44
    confirmation
      temporary storage overwrites, preventing 218
      current 44
      IDs
        displaying 210
      interactive 44
      names
        global variables related to 200
        relative cost estimate, disabling 210
        PARSE ARG statement 11
        parse services and exporting XML 90
        passwords, mixed-case 8
        PDF 33, 34
        PDS and PDSE data sets
          defining export storage 218
          defining type to QMF 218
        performance
          concurrent access resolution options 218
        PL/I
          callable interface 181
          CICS 187
          communications area 181
          DQABFP 184
          DSQCOMM 186
          function calls 182
          ISPF 189
          macros 197
          sample program 184
          TSO 188
          z/OS 187
        positive SQL codes, message support 210
        PREPARE statement, concurrent access resolution 218
        prerequisite D62 for z/OS knowledge v
        PREVIOUS command
          Table Editor confirmation 207
        primary space allocation upon export 218
        PRINT command
          CICS
            queue name/type 206
            global variables
              restricting last used date 218
              suppressing carriage control characters 218
        PRINTER parameter
          QMF profile 205
        procedure
          exporting
            file size maximum 130
            initialization, setting variables during 200
            invocation, rerunning 218
procedure (continued)
  preventing overwrites of PROC panel 218
  stored procedures
    See stored procedure
product interface macros 197
profile
  global variables related to 205
  preventing overwrites of unsaved values 218
program calls 17
programming interface information 246
prompted query
  data records 120
  export format 120
  exporting 94, 120
    file size maximum 130
  field numbers 120
  header records 120
  import/export file specifications 130
  table numbers 120
PS data sets, defining for export 218
records (continued)
  data continuation (C) 102
  data value (V) 97
  fixed format 94
  formats 69
  header 94
  table description (T) 75
  variable format 96
REFRESH command
  Table Editor confirmation 207
  relative cost estimate panel, disabling 210
  release number of QMF, global variable for 200
remote data access
  user ID for CONNECT 200
  remote unit of work command behavior 41
reports
  across 113
  displaying text 46
  export example 113
  export format 113
  export records 113
  export uses 113
  exported across 119
  exporting 68
    file size maximum 130
    exporting without control information 118
    field numbers 113
    HTML 117
    line (L) records 100
    minisession 62
    object 2
    across 113
    export format 113
    field numbers 113
    table numbers 113
  panel 2
  printing
    carriage control characters 218
    records 94
    row data 99
    row length, setting 218
    sample header 113
    stored procedure runs
      See stored procedure
    suppressing after query is run 210
    table data 98
    table numbers 113
  Reset Report panel, enabling 218
  resource contention, reducing 218
  RESOURCE GROUP parameter, QMF profile 205
result set
  records in exported XML 90
  specifying which to use for report 218
return codes
  callable interface 20
  command interface 27
  message 12
  nonzero 12
rows
  maximum length 218
RUN command
  embedded substitution variables 14
  prompt panel 9
  restricting update of last used date 218
  substitution variables 9
RUN QUERY command
  accelerator tables 218
  global variables for messages 234
  multistatement queries 218
  SQL queries over 32 KB 218
RUW (remote unit of work) 8
save command
  form, national language used 218
  restricting update of last used date 218
  SHARE parameter, global that sets 218
SAVE DATA command 68
accelerator tables 218
SAVE option
  EDIT TABLE command
    globals related to 207
  schema definition in exported XML file 90
  scroll amount, setting 210
  secondary space allocation upon export 218
SELECT PGM service, ISPF 32
SELECT statements
  concurrent access resolution options 218
  service trace 57
  PL/1 example 58
  session global variables 234
  session, variables that record state 200
SET GLOBAL command 200
  callable interface 20, 48
  extended syntax 49
  prompting for variables 10
  syntax 48
SHARE parameter of SAVE command 218
SHOW command
  SHOW CHANGE, Table Editor confirmation 207
  SHOW GLOBALS 200
  SHOW SEARCH, Table Editor confirmation 207
signal on error instruction 12
SKIP LOCKED DATA option for SELECT statements 218
SMALLINT data type
  column width on export 69
  exporting
    column data format, IXF 75
    edit code keywords, exported form 103
  SQLTYPE codes on export (QMF format) 69
sort order for LIST command 210
SPACE parameter, QMF profile 205
specification for XML export/import 90

268 Developing Db2 QMF applications
spill file
  global variables that support state of QMF session, variables for
  use of extended storage in TSO support 206
  SQL codes displaying from last command positive, enabling message support 234
  SQL keywords
  multistatement queries
    See multistatement queries
    SELECT concurrent access resolution options 218
  SQL queries 129
    exporting
      file size maximum lengths over 32 KB 130
      special register 200
      STATE information, displaying 234
    START command
      debugging errors interface communications area 18
      keywords
        DSQADPAN 51
        DSQALANG 51
        DSQBSTG 51
        DSQSCMD 51
        DSQSDBCS 51
        DSQSDBNM 51
        DSQSDBINQ 51
        DSQSDBQT 51
        DSQSDBUG 51
        DSQSIROW 51
        DSQSMODE 51
        DSQSMRFI 51
        DSQSPFIL 51
        DSQSPFRID 51
        DSQSPFTYP 51
        DSQSRSTG 51
        DSQSRUN 51
        DSQSSPQN 51
        DSQSSUBS 51
      list 51
      QMF startup 21, 50
      syntax 50
  state of QMF session, variables for storage export considerations XML data 132
  exported file size maximum extended, spill data (TJO 129
  specifying when exporting spill data extended virtual 218
  stored procedure
    specifying result set for report starting QMF for TSOAs global variable support 200
  stored procedure interface
    DSQSDBLG parameter 51
    remote data access restrictions 8
    style sheet for exported XML files substitution variables 9
    assigning values 9
    global variables, setting 9
    REXX program calls 14
    syntax 9

substitution variables 200
  support information
  symbol for currency, changing synonyms, command 65
  syntax diagrams, how to read

T
  table records in exported files
  tables creating outside QMF description records (T) 75, 98
  exporting form, numbers 103
  LIST command
    global variables related to object
      import/export file specifications 130
      import/export rules 93
      importing 68
      processing 68
      prompted query, numbers 120
      report, numbers 113
      row records (R) 99
    temporary storage 22
    CICS
      global related to printing 206
      global related to spill file 206
      confirmation for overwrites 218
      global variables for tracing 206
      modifying 22
      queue 132
      restrictions 3
      termination flag variable 200
    terms of VUE license agreement (global variable) 200
  three-part names in QMF commands failures when MR=YES 51
  TIME data type
    column width on export
    exporting
      column data format, IXF 75
      edit code keywords, exported form 103
    ISO format on export 69
    SQLITE types on export (QMF format) 69
  TIME keyword seen in exported forms 103
  TIMEST keyword seen in exported forms 103
  TIMESTKEY keyword seen in exported forms 103
  TIMESTAMP data type
    column width on export
    exporting
      column data format, IXF 75
      edit code keywords, exported form 103
    ISO format on export 69
    SQLITE types on export (QMF format) 69
  TIMESTAMP WITH TIME ZONE data type
    column width on export
    exporting
      column data format, IXF 75
      column record formats 75
  TIMESTAMP WITH TIME ZONE data type (continued)
    exporting
      edit code keywords, exported form 103
      IXF version numbers for export 75
      ISO format on export 69
      SQLITE types on export (QMF format) 69

TRACE command 57
  PL/I example 58
  traceability 57
  tracing
    A option 135
    allocating file for 136
    application trace level 200
    creating trace definitions example 137
    global variables for 206
    ISPF commands 34
    option 135
    positive SQL codes 210
    profile parameter for 205
    setting 135
    turning off 136
  transient data
    contrasted with temporary storage 132
    global related to printing 206
    global related to spill file 206
    global variables for tracing 206
    translatable applications 39
    translations available in QMF
      See multicultural support

TSO
  Assembler callable interface
    programs 150
  Assembler programs 150
  C callable interface programs 160
  C programs 160
  extended storage for spill data 51
  mixed-case passwords on RACF 8
  REXX callable interface
    programs 194
  REXX programs 194
  TSTMPZT keyword seen in exported forms 103

U
  uncommitted read, enabling 218
  UNKNOWN keyword seen in exported forms 103
  USE CURRENTLY COMMITTED option
    for SELECT statements 218
  user attention flag 200
  user ID, database connections 200
  user-defined edit codes
    keyword seen in exported forms 103

V
  validation of exported XML file 90
  VARBINARY data type
    column width on export 69
VARBINARY data type (continued)
exporting
  column data format, IXF 75
  edit code keywords, exported form 103
SQLTYPE codes on export (QMF format) 69

VARCHAR data type
  column width on export 69
  exporting
    column data format, IXF 75
    SQLTYPE codes on export (QMF format) 69
VARGRAPHIC data type
  column width on export 69
  exporting
    column data format, IXF 75
    edit code keywords, exported form 103
SQLTYPE codes on export (QMF format) 69

variables
  error handling 22
  global 9, 199
  substitution 9
  pool 17
  prompting for 10
  rc 12
  rules 49
  setting 9
  substitution 9
variation panels 103

VBS format, XML data exports 90

VM platform
  three-part name restrictions 51

VSE platform
  three-part name restrictions 51

VUE license agreement global variable 200

W
WAIT FOR OUTCOME option for
  SELECT statements 218
  WIDTH parameter, QMF profile 205

X
XML data type
  exporting 90
    file size maximum 130
    format of data records 90
    storage considerations 93