iSeries

DB2 Universal Database for iSeries Query Management Programming

Version 5

SC41-5703-05

iSeries

DB2 Universal Database for iSeries Query Management Programming

Version 5

SC41-5703-05

Note

Before using this information and the product it supports, be sure to read the information in Appendix E, “Notices” on page 241.

Sixth Edition (September 2002)

Contents

About DB2 UDB for iSeries Query Management Programming (SC41–5703)

DB2® UDB for iSeries Query Management provides a common method of accessing data and reporting the results from a relational database across the different DB2 platforms.

Query Management gives you the ability to design and format printed reports from processed queries. It is a powerful and flexible reporting tool. (DB2 UDB for iSeries Query Manager provides an easy to use front end to Query Management.) Queries can be included in programs written in RPG, COBOL, FORTRAN, C/400®, ILE C/400, or PLI. These can be run from within CL programs. This gives the programmer flexibility in setting up the environment.

Query Management is included within the OS/400® licensed program. OS/400 commands allow:

- The import of a query object from a source file, or from an existing Query for iSeries definition. This means that a query object is created from a source file definition.
- The import of a query form object from a source file, or from an existing Query for iSeries definition. This means that a form object is created from the relevant source definition.
- The export of a query object to a source file.
- The export of a query form object to a source file.
- You to run a query from a command line.
- You to delete relevant objects from the system.
- You to analyze existing Query for iSeries definitions prior to possible conversion.

Who should use the Query Management Programming book

Query Management itself is not an end-user tool. The people who can use it most effectively are data processing professionals. They have the ability to create queries that can be used within both CL and high-level language programs.

Selected end users could be trained to use DB2 UDB for iSeries Query Manager to create their own queries. Queries created and saved by DB2 UDB for iSeries Query Manager can be used by Query Management commands in batch mode. However, generally, if end users need to create their own reports, then Query for iSeries or Query Manager would be a better choice.

If compatibility across several machines is necessary, then Query Management should be considered.

This manual is intended to be used by:

- An application programmer familiar with the Query Management Common Programming Interface (CPI) and the query interface.
- An application programmer familiar with the Query for iSeries product and with using Query for iSeries definitions.
- A system operator who has had formal iSeries system training, is familiar with operating the iSeries system, and is familiar with query functions.
- A user who is performing problem analysis.
- IBM® Programming Service personnel who are responsible for resolving non-customer problems with system programs and this product.

Query Management performance must be reviewed if many lines of data are generated within a single report. Query Management is not recommended as an interactive transaction application, if the resulting report is very large.

However, performance may be better if a query generates a report that has summary calculations on a large file. For example, using the SUM, AVG, COUNT, MAX, or MIN calculations and only printing these and not the detail lines.

What Query Management does not do

Query Management does not pass back data to the program after accessing the database. If this type of processing is required, then embedded SQL should be considered.

Query Management does not provide an end-user interface for the definition of queries, the changing of queries, or for report formatting. Query Manager does this in a controlled manner.

Before you use this manual, you should be knowledgeable in the following:

- DB2 UDB Query Manager and SQL Development Kit licensed program
- iSeries system operation and functions
- Query for iSeries product

Prerequisite and related information

Use the iSeries Information Center as your starting point for looking up iSeries technical information.

You can access the Information Center two ways:

- From the following Web site:

  `http://www.ibm.com/eserver/iseries/infocenter`
- From CD-ROMs that ship with your Operating System/400 order:

  iSeries Information Center, SK3T-4091-02. This package also includes the PDF versions of iSeries manuals, *iSeries Information Center: Supplemental Manuals*, SK3T-4092-01, which replaces the Softcopy Library CD-ROM.

The iSeries Information Center contains advisors and important topics such as Java™, TCP/IP, Web serving, secured networks, logical partitions, clustering, CL commands, and system application programming interfaces (APIs). It also includes links to related IBM Redbooks™ and Internet links to other IBM Web sites such as the Technical Studio and the IBM home page.

With every new hardware order, you receive the *iSeries Setup and Operations CD-ROM*, SK3T-4098-01. This CD-ROM contains IBM @server iSeries Access for Windows and the EZ-Setup wizard. iSeries Access offers a powerful set of client and server capabilities for connecting PCs to iSeries servers. The EZ-Setup wizard automates many of the iSeries setup tasks.

iSeries Navigator

IBM iSeries Navigator is a powerful graphical interface for managing your iSeries servers. iSeries Navigator functionality includes system navigation, configuration, planning capabilities, and online help to guide you through your tasks. iSeries Navigator makes operation and administration of the server easier and more productive and is the only user interface to the new, advanced features of the OS/400 operating system. It also includes Management Central for managing multiple servers from a central server.

You can find more information on iSeries Navigator in the iSeries Information Center and at the following Web site:

`http://www.ibm.com/eserver/iseries/navigator/`

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 iSeries documentation, fill out the readers' comment form at the back of this book.

- If you prefer to send comments by mail, use the readers' comment form with the address that is printed on the back. If you are mailing a readers' comment form from a country other than the United States, you can give the form to the local IBM branch office or IBM representative for postage-paid mailing.
- If you prefer to send comments by FAX, use either of the following numbers:
  - United States, Canada, and Puerto Rico: 1-800-937-3430
  - Other countries: 1-507-253-5192
- If you prefer to send comments electronically, use one of these e-mail addresses:
  - Comments on books:

    RCHCLERK@us.ibm.com
  - Comments on the iSeries Information Center:

    RCHINFOC@us.ibm.com

Be sure to include the following:

- The name of the book or iSeries Information Center topic.
- The publication number of a book.
- The page number or topic of a book to which your comment applies.

Chapter 1. Introduction to Query Management

This chapter introduces the Operating System/400 (OS/400) query management system (DB2 UDB for iSeries Query Management) and describes some of its characteristics, specifications, and requirements, and its relationship to the Query for iSeries programs and functions.

Query Management overview

This manual covers the OS/400 implementation of Query Management Common Programming Interface (CPI). This manual describes the functions and user interface for the DB2 UDB for iSeries Query Management CPI.

Note: A working knowledge of Query Management Common Programming Interface (CPI) is recommended as a prerequisite to working with this product. A knowledge of Query for iSeries and Structured Query Language (SQL) would also be beneficial.

The CPI allows a user to access information in a relational database and control how this data appears when formatted into a report. The CPI provides services that fall into two major categories: querying and report writing.

Application programs can use query management services through a program-to-program callable interface using DB2 UDB for iSeries Query Management objects. The Query Management objects are created only through the CPI using files containing externalized query, procedure, and form definitions. The externalized files can be built using an editor, built by an application program, transferred from another system (from which they were exported), or created through conversion of definition (QRYDFN) objects that were created using Query Management.

Applications that use Query Management have the following advantages:

- Reduced requirements for error handling and interpretation. The application may not have to check for SQL error codes.
- Use of queries, procedures, and forms that are defined and stored outside of the application code. You can update your application by changing Query Management objects without having to change or recompile your application program.
- No need to understand and handle relational database manager protocols. The application can pass a few simple commands to allow access to data in an exported format.

The Query Management commands can be issued by processing a statement in a procedure or by running a program that uses the callable interface.

OS/400 and the Query Management environment

Query Management objects are created and maintained as OS/400 system objects. Table 1 shows the relationships between OS.400 system terms and the Query Management environment terms.

Table 1. iSeries and Query Management Terminology

OS/400 Term	DB2 UDB for iSeries Query Management Use
Library — A library groups related objects, allowing the user to find the objects by name.	**Collection** — A collection consists of a library, a journal, a journal receiver, a data dictionary, and an SQL catalog. A collection groups related objects, allowing the user to find the objects by name.
Physical file — A collected group of records.	**Table** — A collection of columns and rows.
Record — A collection of fields.	**Row** — The horizontal part of a table containing a serial collection of columns.

Table 1. iSeries and Query Management Terminology (continued)

OS/400 Term	DB2 UDB for iSeries Query Management Use
Field — One or more characters of related information of one data type.	**Column** — The vertical part of a table of one data type.
Logical file — A subset of fields and records of one or more physical files.	**View** — A subset of columns and rows of one or more tables.
User profile — A name that identifies a user and designates a set of privileges on the iSeries system.	**Authorization ID** — A character string of not more than 10 bytes that identifies a user.

Collection use by Query Management

Query Management treats all objects as belonging to a collection. The following list describes query management collection conventions:

- Query Management objects (queries, forms, procedures) are not part of a collection. As shown in Table 1 on page 1, a collection on the iSeries system is a library. A query management object may be part of a library, but if the library is a collection, there is no entry for the query management object within the collection catalogs or journals.
- Tables manipulated by the query management commands (ERASE and SAVE) are treated as SQL tables. The DB2 UDB for iSeries SQL rules applicable to the SQL table manipulation statements are in effect when manipulating tables through query management commands. The following list shows the correspondence between query management commands and SQL statements:

  ERASE
  Drop Table

  SAVE (to new table)
  Create Table and Insert

  SAVE REPLACE
  Delete and Insert

  SAVE APPEND
  Insert

Naming conventions in Query Management

The following rules apply when you create a table or view or name an object that you want to save in the database.

Naming conventions for Query Objects for Query Management

Query objects may be specified on commands using either SQL names or system names. The naming convention to use is specified in the query command procedure on the START command or the Start Query Manager Query (STRQMQRY) CL command. Query Management uses SQL naming as the default. The naming convention specified for a query instance is used throughout the entire instance and applies only to that instance. It cannot be changed after the START command has been issued for the query instance.

Note: These naming conventions only apply to the query object specified on a command. System naming conventions always apply on the file name specified on the IMPORT and EXPORT commands.

System naming in Query Management

When a query instance uses system names, the following rules apply for a query object name specified in a query command:

- Query objects may be specified using delimited names. A delimited name is a character string with the following characteristics:

  - One to eight characters long
  - Beginning and ending with quotation marks ("MYFORM")
  - Contains any character except:
    - A blank
    - An asterisk (*)
    - A question mark (?)
    - An apostrophe (')
    - Quotation marks (")
    - The numbers hex 00 through 3F, or hex FF

  A delimited name may be qualified, but the qualifier and the name must be surrounded by quotation marks separately from each other ("MYLIB"/"MYFORM").

- Query objects may be specified using simple names. Simple names are character strings up to 10 characters long and must begin with an alphabetic character (A through Z, $, #, or @). Periods and blanks are not allowed in simple names.
- A query command file name can be qualified by a library name up to 10 characters long. A slash (/) must separate the qualifying library name and the file name. For example, MYLIB/FILE1 (file FILE1 in library MYLIB) is a qualified name. The rules applying to OS/400 names in quotation marks and simple names apply to the library name used as the qualifier.
- Objects of the same type that are stored in the same library must have different names (you cannot have two files named TEST, for example). Queries and forms are different OS/400 object types; therefore, a query and form may have the same name. Names for procedures, tables, and views must be different because they are all OS/400 files. A procedure, table, or view can have the same name as a query object or form object, but not another procedure, table, or view.
- Names for queries, forms and procedures can use reserved words (like FORM, QUERY, COUNT, NULL and so on), though naming something with an SQL keyword is not recommended.
- If a Query Management query, form, or procedure specified in a query command is not qualified, OS/400 search conventions are followed. If an unqualified query management object name is specified, Query Management searches the library list (*LIBL) for the query management object. If the query management object is being created, Query Management places the object in the current library (*CURLIB).
- If an unqualified table or view name is specified on a query command, see the SQL Referencefor the search conventions followed.

For more information on system naming and OS/400 search conventions see the Control Language topic in the Information Center.

SQL naming in Query Management

When the query instance is using SQL names, the following rules apply for a query object name specified in a query command. These rules are similar to the OS/400 object naming conventions.

- Query objects may be specified using delimited names. A delimited name is a character string with the following characteristics:
  - One to eight characters long
  - Beginning and ending with quotation marks ("MYFORM")
  - Contains any character except
    - A blank
    - An asterisk (*)
    - A question mark (?)
    - An apostrophe (')
    - Quotation marks (")

  - The numbers hex 00 through 3F, or hex FF

  A delimited name may be qualified, but the qualifier and the name must be surrounded by quotation marks separately from each other ("MYLIB"."MYFORM").

- Query objects may be specified using simple names. Simple names are character strings up to 10 characters long and must begin with an alphabetic character (A through Z, $, #, or @). Periods and blanks are not allowed in simple names.
- A name can be qualified by another name (usually a user or an authorization identification) of up to 10 single byte characters with a period (.) separating the qualifier and the name. For example, Q.QUERY1 (the query in the Q collection) is a qualified name. Query Management uses the DB2 UDB for iSeries SQL conventions of treating the authorization ID as a user profile. If SQL naming conventions apply, Query Management attempts to find the object in the library with the same name as the authorization ID. The rules applying to delimited and simple names apply to the library name used as the qualifier.
- Objects of the same type that are stored in the same library must have different names (you cannot have two files named TEST, for example). Queries and forms are different OS/400 object types; therefore, a query and form may have the same name. Names for procedures, tables, and views must be different because they are all OS/400 files. A procedure, table, or view can have the same name as a query object or form object, but not another procedure, table, or view.
- If an unqualified query management object name is specified, Query Management searches the library with the same name as the current user profile for the query object. If the query object is being created, Query Management places the object in the library with same name as the current user profile. These are the same conventions followed by the DB2 UDB for iSeries Query Manager and SQL Development Kit licensed program.

Naming conventions for OS/400 Objects in Query Management: The *filename* specified on the IMPORT and EXPORT query commands will follow the OS/400 naming conventions for a source physical file.

- Query objects may be specified using delimited names. A delimited name is a character string with the following characteristics:
  - One to eight characters long
  - Beginning and ending with quotation marks ("MYLIB")
  - Contains any character except
    - A blank
    - An asterisk (*)
    - A question mark (?)
    - An apostrophe (')
    - Quotation marks (")
    - The numbers hex 00 through 3F, or hex FF

  A delimited name may be qualified, but the qualifier and the name must be surrounded by quotation marks separately from each other ("MYLIB"/"MYFORM").
- Query objects may be specified using simple names. Simple names are character strings up to 10 characters long and must begin with an alphabetic character (A through Z, $, #, or @). Periods and blanks are not allowed in simple names.
- OS/400 rules also apply when the *filename* is specified with a qualified name. A *filename* in a query command can be qualified by a library name of up to 10 characters with a slash (/) separating the qualifier and the name. For example, MYLIB/FILE1 (a file in library MYLIB) is a qualified name.
- If a *filename* specified in a query command is not qualified, Query Management searches the library list (*LIBL) for a source file named *filename.* If the file is being created, Query Management will place the file in the current library (*CURLIB).
- If the physical file is a multiple member source file, the following rules apply for specifying the member:

- – If no member name is specified, the member name used will default to *FIRST on the IMPORT and EXPORT commands.
- – Query Management will process a specified member in a physical file if a member name is given as part of the file name. The member name must follow the file name and be delimited by parenthesis with no intervening blanks. For example, the member MEMBER1 in the file FILE1 can be specified by entering a file name as follows:

  ```
  FILE1(MEMBER1)
  ```

Naming conventions using variable names in Query Management: See "Extended variable support in the Query Management CI" on page 81 for the rules that apply when you use variables in SQL queries across the callable interface. OS/400 specific rules are:

- Variable names must be preceded by an ampersand (&). The ampersand delimits the beginning of a variable name and is not included as one of the 18 characters allowed for the name. You cannot have more than one ampersand in a variable name, since each ampersand delimits the beginning of a distinct variable name.
- User-defined variables may not start with DSQ. An error is generated if an attempt is made to set a variable that starts with DSQ.
- Variable names within Query Management are case sensitive. Therefore, the variable i_owe_you, is not the same as the variable I_OWE_,YOU.

The following are valid variable names:

```
In an SQL Query              In the GET/SET Command
--------------------         -------------------------
&I_owe_you                   I_owe_you
&MYVAR123                    MYVAR123
&THIS_IS_A_BIG_NAME          THIS_IS_A_BIG_NAME
```

Naming conventions for other query names in Query Management: The naming convention being used by the query instance also applies to the SQL statements in any SQL query run during the instance. If system names are being used in the query instance, system names apply to the SQL query. See the SQL Reference for a description of the SQL and system naming conventions as followed by DB2 UDB for iSeries SQL.

Security and authorization in Query Management

Query Management uses the OS/400 security and authorization model. See *iSeries Security Reference* for information about security concepts for the iSeries system.

There are special security authorization considerations for query objects, OS/400 objects, and SQL:

- **Query objects:** When query objects are created through a query command, there are various ways to specify the type of public authority for the query object that you want to give to other users. The types of authority are specified using the DSQOAUTH keyword in the query command procedure or on the START command; see "START in Query Management" on page 45.
- **OS/400 objects:** Query Management uses the same public authority for creating a non-query object, such as the source physical file on an EXPORT, as it does when creating query objects.
- **SQL:** See the SQL Reference for information for object authority as it applies to the SQL statement within an SQL query.
- **Adopted Authority:** The user, while running a program, acquires the program owner's authority and thus acquires authority to the files associated with the program.

Objects in Query Management

Support for query management requires the definition of two object types: the query management query object and the query management form object. The OS/400 object type for the query management query object is QMQRY. The OS/400 object type for the query management form object is QMFORM. The query management procedure is stored as a single member source physical file.

CL commands that support objects in Query Management

The following query management CL commands support the QMQRY objects:

CRTQMQRY
Create Query Management Query

DLTQMQRY
Delete Query Management Query

RTVQMQRY
Retrieve Query Management Query

STRQMQRY
Start Query Management Query

WRKQMQRY
Work with Query Management Query

The WRKQMQRY CL command supports the following CL commands:

CHGOBJD
Change Object Description

DLTQMQRY
Delete Query Management Query

STRQMQRY
Start Query Management Query

The following query management CL commands support the QMFORM objects:

CRTQMFORM
Create Query Management Form

DLTQMFORM
Delete Query Management Form

RTVQMFORM
Retrieve Query Management Form

WRKQMFORM
Work with Query Management Form

The WRKQMFORM CL command supports the following CL commands:

CHGOBJD
Change Object Description

DLTQMFORM
Delete Query Management Form

Generic commands in Query Management

You can run the following generic commands against the QMQRY and QMFORM objects:

CHKOBJ
Check Object

CHGOBJD
Change Object Description

CHGOBJOWN
Change Object Authority

CRTDUPOBJ
Create Duplicate Object

DSPOBJAUT
Display Object Authority

DSPOBJD
Display Object Description

EDTOBJAUT
Edit Object Authority

GRTOBJAUT
Grant Object Authority

MOVOBJ
Move Object

RNMOBJ
Rename Object

RSTOBJ
Restore Object

RVKOBJAUT
Revoke Object Authority

SAVCHGOBJ
Save Changed Object

SAVOBJ
Save Object

WRKOBJ
Work with Object

Message descriptions in Query Management

The following error messages are possible when working with query management commands and functions:

FAILURE
The query management command issued failed, and processing stops. A message is returned to the display station that details the reason for the command failure and gives some possible solutions for correcting the error.

SEVERE
A severe error occurred when query management attempted to process the command issued, and all processing stops. A message is returned to the display station that details the reason for the error and gives some possible solutions for correcting the error.

SUCCESS
The query management command issued processed successfully, and data is available for use.

WARNING
Query Management encountered an error in the command or procedure issued, but processing continues. The error is ignored, or system defaults are used to correct the error. Check the error messages for an explanation of the warning and what, if anything, you can do to correct the error.

Naming Conventions

Chapter 2. Query capability in Query Management

DB2 UDB for iSeries Query Management supports queries against relational data using SQL. When a query is run remotely to an iSeries system, the local sort sequence table is used. When a query is run remotely to a system other than an iSeries system, a sort sequence table is **not** used.

Note: The remote iSeries system must be at Version 2 Release 3 or above in order to use the sort sequence support. Otherwise, no sort sequence table is used.

The basic statements, SELECT expressions, data definition, and authorization statements defined in the SQL Reference are specifically supported. By using these SQL features in a query, your application can perform table definitions, data access authorizations, database queries and data insertions, updates, and deletions. The results of a query can be displayed or printed as a report.

Query Management supports prompted queries and SQL queries. For more information about prompted queries, see *Query Manager Use*.

Queries can be created, named, stored, and retrieved. Stored queries can be shared among multiple users and applications. The queries used by your application can be defined and stored at application development time, or they can be created by your application and used by Query Management at application run time.

Stored queries allow flexibility in two ways. First, you can change a query and store it independently from your application program. Second, queries can contain variables, and the values assigned to the variables can be set prior to, or in conjunction with your application. Both of these capabilities allow data query parameters to be changed without rewriting or recompiling your application.

Creating queries in Query Management

In OS/400, a query is a QMQRY object. A query can be created using:

- The CRTQMQRY CL command
- An IMPORT QUERY command through the query management callable programming interface
- The DB2 UDB for iSeries Query Manager create query function

The query management query object is stored as an OS/400 *QMQRY object. The externalized query source member must contain a text string containing an SQL statement, which can optionally contain variables.

Variables can appear in any part of the query. They can represent anything that can be written into a query, such as:

- Column names
- Search conditions
- Subselects
- Specific values
- Multiple clauses
- Partial clauses

It is also possible to create a query that contains 1 or more variables.

The following is an example of a query:

```
-- This query lists the name, years of employment, and salary
-- for employees in a department. The department (DEPTNUM)
-- is a variable and should be set before the query is run.
```

```
H QM4 05 Q 01 E V W E R 01 03 92/10/24 11:07
V 1001 050 Department Query
V 5001 011 QSYS/QASCII
V 5002 003 ENV
  SELECT NAME, YEARS, SALARY -- names the columns used
  FROM Q.STAFF               -- names the table used
  WHERE DEPT=&DEPTNUM        -- variable selection condition
```

Note: The comments, H records, and V records are optional; only the SQL statement is required.

Query Management strips the comments and performs variable substitution before the query is passed to the database manager for processing.

Example: Creating queries in Query Management

You can create a query in query management by importing an externalized query source file member to create a query management query object.

1. To create a source file containing a member use the following command:

   ```
       CRTSRCPF FILE(MYLIB1/QQRYSRC) MBR(QUERY1)
   ```
2. Edit the source file member QUERY1 and add the following kind of information:

   ```
   H QM4 05 Q 01 E V W E R 01 03 92/10/24 11:07
   V 1001 050 Department Query
   V 5001 011 QSYS/QASCII
   V 5002 003 ENV
     SELECT NAME, YEARS, SALARY -- names the columns used
     FROM Q.STAFF               -- names the table used
     WHERE DEPT=&DEPTNUM        -- variable selection condition
   ```
3. Save the source file member QUERY1.
4. To create the query object in MYLIB1, use the Create Query Management Query (CRTQMQRY) command to import the source file member you just created. At an OS/400 command line, type:

   ```
   CRTQMQRY QMQRY(MYLIB1/QUERY1)
   SRCFILE(MYLIB1/QQRYSRC) SRCMBR(QUERY1)
   ```

   and press Enter.

Query Management strips the comments and performs variable substitution before the query is passed to the database manager for processing.

Query restrictions in Query Management

The following restrictions apply to queries handled by Query Management:

- A query is limited by the size of the source file.
- A single line of the query cannot exceed 79 bytes.
- Substitution variable values can be up to 55 characters long.
- Substitution variable names cannot exceed 30 characters.
- The externalized query source file should only contain an SQL statement. An externalized prompted query can be successfully imported to query management but would not run.
- Comments must be preceded by a double hyphen (--). Everything between the double hyphen and the end of the line is considered to be part of the comment.
- The total query cannot exceed 32767 bytes after comments and blanks are removed and variable substitution is made.
- An externalized query can contain an H record with a comment V record immediately following. The comment may be used as the text description when the object is imported.
- An externalized query can contain a sort sequence value, a language ID, or both. If these are present, they must immediately follow the H record or comment record.

Variable substitution in Queyr Management

The following rules apply to variable substitution in query management queries:

- Variable substitution is not done if the variable appears within a comment.
- Variable substitution is not done if the variable appears within a constant or a delimited name.
- A variable within an SQL query is defined as a string of characters that begins with an ampersand (&) and ends with any character that is not a valid variable name character.
- Query Management does not substitute extra blanks between variables. Therefore, you can use variable substitution as a concatenation device. As an example, the following query management SET commands are processed through the callable interface or within a procedure:

  ```
  SET GLOBAL (library='MYLIB'
  SET GLOBAL (table='MYTABLE'
  SET GLOBAL (dol=10
  SET GLOBAL (cnts=50
  ```

  Then running the following SQL query processes the ending SQL statement:

  ```
  SELECT * FROM &library.&table
           WHERE PRICE  = &dol.&cnts
  ```

  ```
  SELECT * FROM MYLIB.MYTABLE
           WHERE PRICE  = 10.50
  ```

Variable prompting in Query Management

Query Management sends a message to your display prompting you for the value to be used. This prompting happens if your job is running interactively and the variable specified in a query is not set in the global variable pool.

A valid value may be entered for the variable, and then the query is processed. If you press Enter without typing a value, or you press F3 (Exit) or F12 (Cancel), the query fails with an error. You can also replace the variable name with a single blank by entering the special value *BLANK.

If your job runs in batch mode and the variable specified in a query is not set in the global variable pool, the query fails. The variable is not substituted.

Any attempt to run a query with an incorrect variable name causes the query to fail with an error. For example, if the variable name is too long, or if the first character after the & is not a letter, the query will fail.

Comments in Query Management

Comments in query management queries are handled in the following ways:

- Comments are stripped from the SQL query prior to variable substitution. Comment delimiters within substituted variables are not stripped and may result in SQL errors or unpredictable results.
- Comment delimiters within strings enclosed in quotation marks are not treated as comments. These strings may either be delimited names, which are delimited by quotation marks (“”), or constants, which are delimited by apostrophes (’’).
- You cannot use intervening blanks between the two hyphens (--) that make up the comment delimiters.

Line continuations in Query Management

The following rules govern the use of line continuations in query management queries:

- Some SQL clauses may span multiple lines of the SQL query. SQL does not support a line continuation character. Therefore, for readability, start a new line at a point in the SQL statement where a blank could be inserted. If a clause spans multiple lines, the clause may be split as long as the last character in the previous line is part of the clause and the first character in the next line is part of the clause with no extra blanks. The last character in a clause to be continued on the next line must be in column 79.

- Constants and delimited names may span multiple lines.

  Note: Mixed single-byte character set (SBCS) and double-byte character set (DBCS) character strings may not successfully be split between multiple lines. In this case, use SQL concatenation.

Using sort sequence in Query Management

When the Query Management Common Programming Interface (CPI) interface is used to run a *QMQRY object that was defined with a sort sequence table, the sort sequence table associated with the object is used to run the query. The sort sequence table is also used for formatting all displayed and printed reports produced from the resulting data.

If you connect to a remote iSeries system that supports sort sequence, the sort sequence table associated with the object is used for SQL processing.

If a RUN QUERY is done to convert and run a Query for iSeries *QRYDFN object, Query Management uses the *HEX sort sequence table if the object was defined to use option 1, 2, or 3 in the Query for iSeries Select Collating Sequence display. For information about the Query for iSeries product, see *Query for iSeries Use*. If the *QRYDFN object was defined to use options 4 or 5, Query Management uses the same sort sequence that is used to run the query and format the report.

Chapter 3. Instance processing in Query Management

A query management instance is a progression of steps that results in creating a displayed or printed report from the data found in a database file or Query for iSeries definition. Query Management puts the data specified into a DATA set (the active information resulting from running a query) called a query management query (QMQRY) object, which is organized by the query management form (QMFORM) object. By changing the form, you can use the same QMQRY to create multiple reports that are organized according to your needs for a particular situation. This chapter describes how to create, change, and convert a query management instance that creates a report arranged to your needs.

Creating an instance in Query Management

Obtain access to the query management query function by beginning with a control language (CL) command or a user application. Once you access the query function, you can use query management commands to direct query management in creating an instance. An instance is a DATA set containing the data collected from the database file and the global variable pool that contains the DSQ variables used to define the query.

Using the query management instance created by the commands issued, build a printed or displayed report by creating or changing the form in a way that gives you the needed information. Figure 1 illustrates how a query management instance is created.

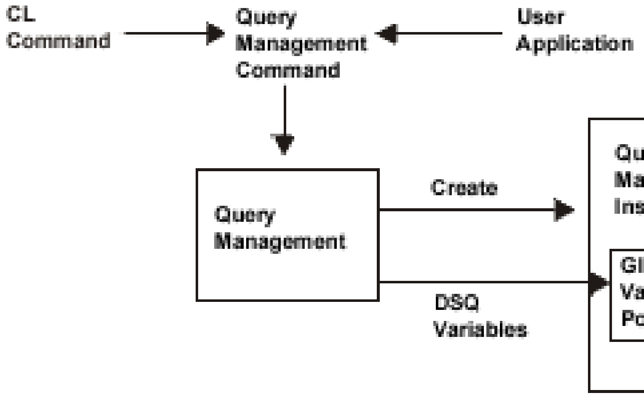


Figure 1. Creating a Query Management Instance

Exit the query management instance using the same procedure and the EXIT command. This destroys the instance.

Running a query in Query Management

Use one of the following methods to run a query management query:

- Specify the RUN QUERY command in a procedure (via STRQMPRC).
- Issue the Start Query Management Query (STRQMQRY) CL command.
- Run the query from a user application.

By using a Structured Query Language (SQL) SELECT statement, query management accesses an OS/400 database file and puts the information requested in the QMQRY into the DATA set contained in the instance.

The DATA set created by running the query remains in existence until another query is processed or the associated instance is ended by the EXIT command. A different DATA set is created for each query management instance.

Note: The DATA set is only created if the query is a SELECT statement.

Figure 2 illustrates how a query is run using query management.

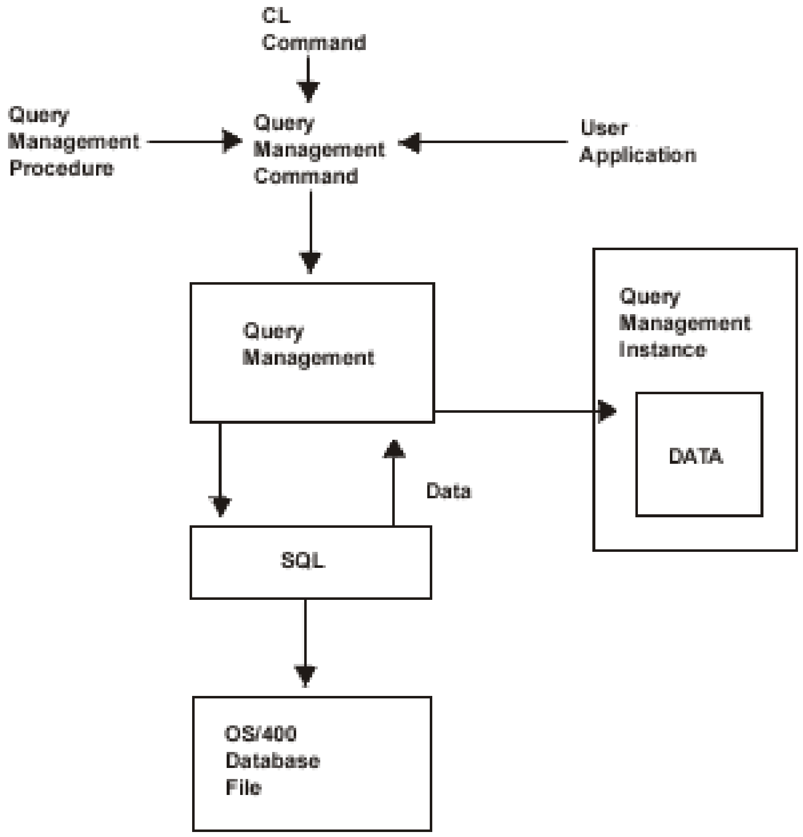


Figure 2. Running a Query Management Query

Global variable substitution in Query Management

If you run a query with global variable substitution specified, query management processes the request in the same manner as described previously, except the global variable pool defined when the instance was created is searched to resolve global variables. If the variable specified in the query is not set in the global variable pool, query management sends a message to the display prompting you for the value to be used in the field specified. Enter a valid value for the prompted field and the query will be run with the variable value entered at the prompt.

Creating reports in Query Management

Once you have created the DATA set, you can request that a report be printed or shown at the display station. Use the Create Query Management Form (CRTQMFORM) command or the Work with Query Management Form (WRKQMFORM) command to create or change a QMFORM object that puts the data in the DATA set into a form specific to your needs.

Once the QMFORM and the DATA set are created, use the Display Report display to show the report on your display station, or use the PRINT command to produce a printed version of the report data.

Note: Creating a report requires that you have already run a query to create a DATA set.

Figure 3 illustrates how to display or print a report using query management guidelines.

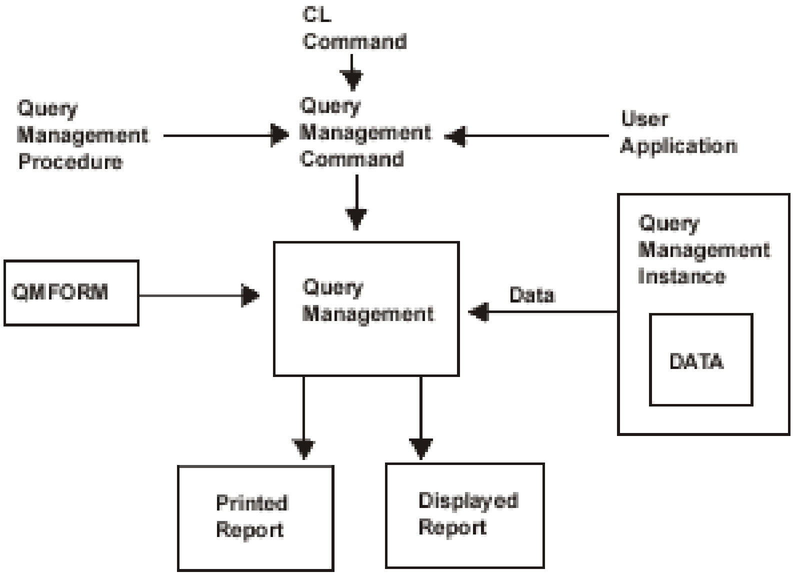


Figure 3. Creating a Query Management Report

Importing a query or form object in Query Management

Use one of the following methods to create a query or form object which you can use to create a query management report:

- Specify the IMPORT command in a procedure.
- Issue the Create Query Management Query (CRTQMQRY) or Create Query Management Form (CRTQMFORM) CL command.
- Import the query or form object from a user application using the IMPORT command.

A query management query or form is created from a source file member which contains the query or form source.

For information on how to create queries, see “Creating queries in Query Management” on page 9.

For information on how to create forms, see “Creating forms in Query Management” on page 57.

For information on how to create procedures, see “Creating procedures in Query Management” on page 53.

Exporting a query or form object in Query Management

Use one of the following methods to export a query or form object so you can change it.

- Specify the EXPORT command in a procedure.
- Issue the Retrieve Query Management Query (RTVQMQRY) or Retrieve Query Management Form (RTVQMFORM) CL command.
- Export the query or form from a user application using the EXPORT command.

The export process creates the query or form source in the specified source file member. The query or form source can then be changed and imported to produce a report containing the changes when the query is run again.

Note: Query Management queries with the attribute PROMPT are exported in the same way as queries with the attribute SQL.

Importing and exporting a procedure in Query Management

The process for importing and exporting query management procedures is the same as for importing and exporting queries and forms with one exception. A query management procedure is a source physical file member. An import or export of a procedure copies the information from one member to another. Therefore, it is not necessary to import or export a procedure, because it is already in the externalized format. Figure 4 illustrates the process for importing and exporting queries, forms, and procedures.

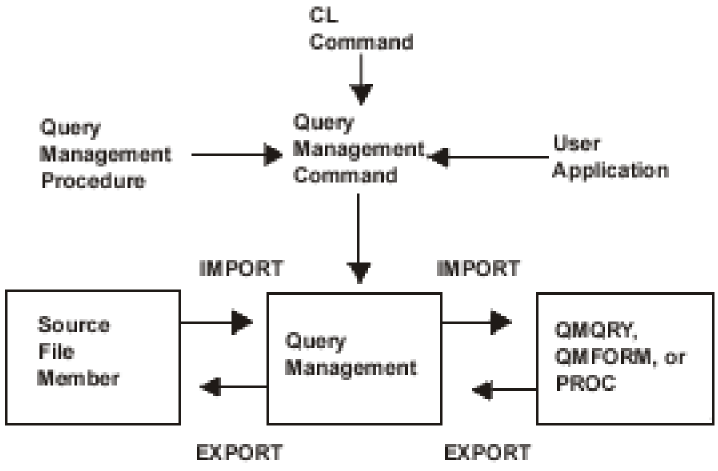


Figure 4. Importing and Exporting Query Management Members

Running a procedure in Query Management

Use one of the following methods to start working with procedures in the query management environment:

- Specify the procedure from inside another procedure using the RUN command.
- Issue the Start Query Management Procedure (STRQMPRC) CL command.
- Start the procedure from a user application using the RUN command.

Use query management commands to request the data from a source file member or another procedure for query management to use in creating a query management instance. The commands in the procedure are processed using the same instance as the instance associated with the RUN PROC command.

Query Management also allows you to call multiple procedures when creating an instance using this process. Figure 5 illustrates how to run a query management procedure.

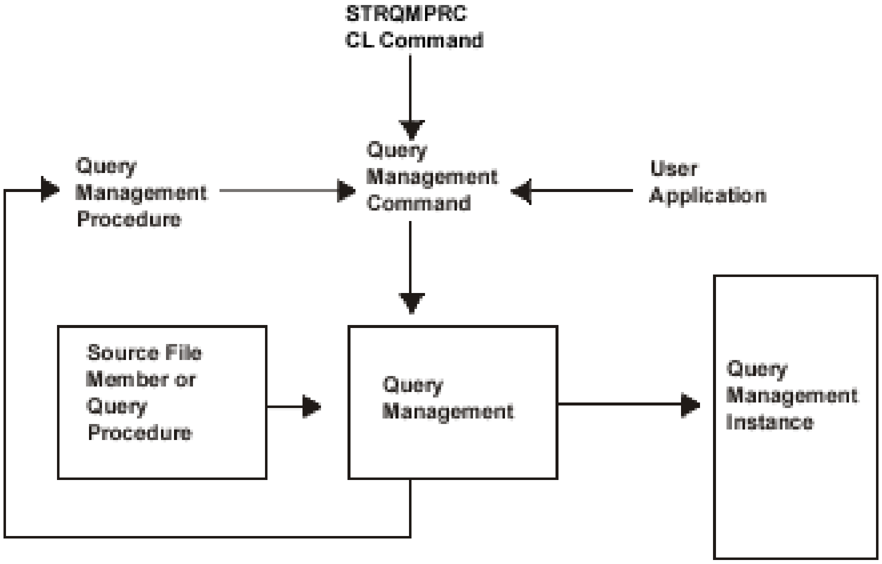

STRQMPRC CL Command
Query Management Procedure
Query Management Command
User Application
Source File Member or Query Procedure
Query Management
Query Management Instance

Figure 5. Running Query Management Procedures

Using the SAVE DATA AS command in Query Management

Query Management allows you to save the data created in the DATA set of your instance to a query management table. Use the following methods to start query management processing when working with the SAVE DATA AS command:

- Specify the save operation from a procedure using the SAVE DATA AS command.
- Issue the Start Query Management Query (STRQMQRY) CL command.
- Start the command from a user application using the SAVE DATA AS command.

Use query management commands to request that the data from the DATA set in an instance created previously be used to save the data in an OS/400 database file to a query management table. You must have processed a RUN QUERY command under the same instance to create the query management DATA set.

Figure 6 on page 18 illustrates how to save the data in a database file to a table using a query management instance.

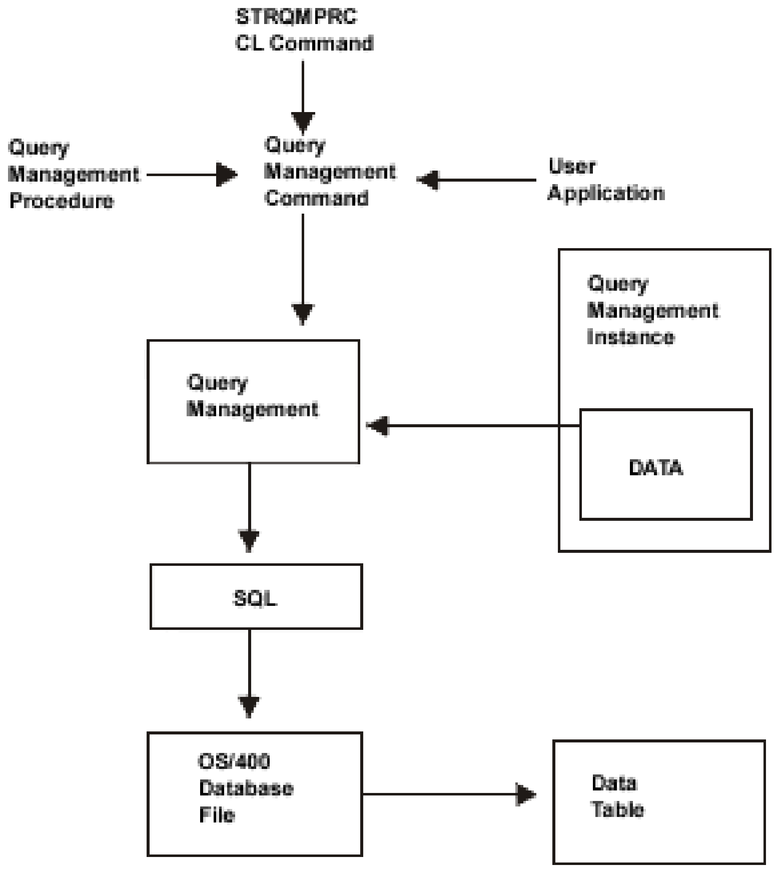


Figure 6. Saving Data to a Query Management Table

Using SET GLOBAL and GET GLOBAL commands in Query Management

Query Management allows you to get and change variables in the global variable pool. Use the GET GLOBAL command to get the value of a query management variable in the previously created instance and provide it to a user program or procedure.

Use the SET GLOBAL command to set or change the value of a query management variable in the previously created instance from a user program or procedure. Use the following methods to start query management processing when working with the GET GLOBAL and SET GLOBAL commands:

- Specify the command from a procedure.
- Issue the Start Query Management Query (STRQMQRY) CL command.
- Start the command from a user application.

Use query management commands to request that the values from the global variable pool in an instance created previously be set from a user program or procedure. The GET GLOBAL process is the same as the SET process, except query management gets the variable values *for* a user program in the GET GLOBAL process.

Figure 7 illustrates how query management uses the GET GLOBAL and SET GLOBAL commands to change or retrieve query management variables.

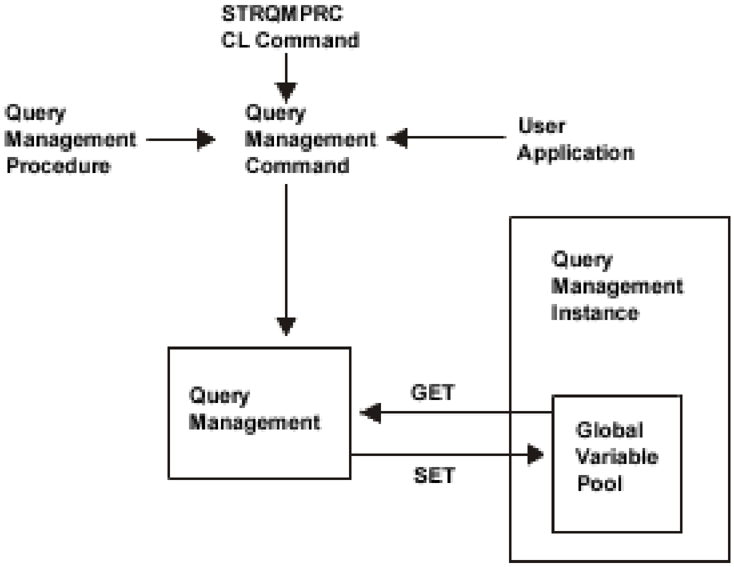


Figure 7. Using GET GLOBAL and SET GLOBAL Commands

Introducing activation groups in Query Management

An activation group is like a miniature job within a job. Each activation group is a substructure of a run-time job. Each consists of system resources (storage for program or procedure variables, commitment definitions, and open files) allocated to one or more programs.

Application programs continue to work as they always have as long as the programs are associated with the default activation group.

If you plan to use ILE C/400 you will need to understand more about how activation groups affect your programs because the iSeries system allows you to associate an application program with an activation group.

Only application programs created with ILE C/400 may be associated with a non-default activation group. The DB2 UDB for iSeries Query Manager function is associated with the default activation group. All query management CL commands also run in the default activation group.

A query management instance is associated with the activation group of the program which started the instance. Query Management instances created by programs other than ILE C/400 are associated with the default activation group. Query Management instances created using query management CL commands are always associated with the default activation group.

An application program can only use a query management instance if the activation group associated with the program is also associated with that query management instance. Activation groups cannot share a query management instance. Work done through a query management instance in one activation group, has no affect on work done by a query management instance in another activation group.

Chapter 4. Commands in Query Management

This chapter is a reference for the commands available to Query Management Use the following query management commands when writing applications to organize general reports from database files:

- COMMIT
- CONNECT
- DISCONNECT
- ERASE
- EXIT
- EXPORT
- GET
- IMPORT
- PRINT
- RELEASE
- RUN
- SAVE DATA AS
- SET CONNECTION
- SET GLOBAL
- START

Each command is described in full and has a syntax diagram provided as a means of quickly referring to the syntax of a command. For an explanation of the diagrams, see "How to read the syntax diagrams in Query Management" on page 22.

Specifying commands and keywords in Query Management

For consistency when moving applications across systems, you should specify all commands, whether used in procedures or passed through the callable interface, in uppercase letters. Similarly, when manipulating a query object, you should specify all character keywords in uppercase letters. Text lines (for example, page headings) may be specified in either uppercase or lowercase letters.

Command parsing in Query Management

For all commands, Query Management allows the keywords and variables associated with the commands to be presented as a part of the command string and as part of the extended parameter list on the callable interface. Keywords specified on the command string will take precedence over keywords specified in the extended parameter list if duplicates occur.

Parsing of keywords and values for command strings and for extended parameter lists differ in the following manner:

Command String Keywords and Variables

The following rules describe how Query Management uses command string keywords and variables:

- Query Management assumes all command values are character strings.
- Values can be delimited by either quotation marks (") or apostrophes (').
- Delimiters are not considered part of the value.
- Quotation marks must be doubled if found inside a value that is delimited by quotation marks ('Joe''s' or "Joe""s", for example). (This rule applies to both apostrophes and quotation marks.)
- Quotation marks found inside a value delimited by apostrophes need not be doubled and vice versa.

Specifying Commands and Keywords

- Blanks found at the beginning or end of values delimited by quotation marks will not be removed.
- Values that are not delimited by apostrophes or quotation marks will be delimited by a blank or by the end of a command string.
- The values for keywords that are used as integers (such as WIDTH and LENGTH on the PRINT command) can be presented to Query Management as integer values or as character string values. If presented as character strings, they will be converted to integers before being used.

Extended Parameter List Keywords and Variables

The following rules describe how Query Management uses parameter list keywords and variables:

- Leading and trailing blanks will be stripped from keyword names before being used.
- Trailing blanks will be stripped from keyword values, but leading blanks will not.
- Leading and trailing blanks will not be stripped from variable values before the variable name and value are added to the query management variable pool.
- Apostrophes and quotation marks will not be stripped by Query Management
- Query Management will not remove quotation marks or apostrophes.
- The values for keywords that are used as integers (such as WIDTH and LENGTH on the PRINT command) can be presented to Query Management as integer values or as character string values. If presented as character strings, they will be converted to integers before being used.

How to read the syntax diagrams in Query Management

In this chapter, syntax is described using the structure defined below. Within the command syntax, one or more blanks are allowed wherever a blank is used to delimit words in the command string.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The ►►── symbol indicates the beginning of a statement.

  The ──► symbol indicates that the statement syntax is continued on the next line.

  The ►── symbol indicates that a statement is continued from the previous line.

  The ──►◄ symbol indicates the end of a statement.

  Diagrams of syntactical units other than complete statements start with the ►── symbol and end with the ──► symbol.

- Required items appear on the horizontal line (the main path).

```
►►──STATEMENT──required_item───────────────────────────────────────────────►◄
```

- Optional items appear below the main path.

```
►►──STATEMENT──┬───────────────┬───────────────────────────────────────────►◄
               └─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.

```
►►──STATEMENT──┬─required_choice1─┬────────────────────────────────────────►◄
               └─required_choice2─┘
```

  If choosing one of the items is optional, the entire stack appears below the main path.

```
►►──STATEMENT──┬──────────────────┬────────────────────────────────────────►◄
               ├─optional_choice1─┤
               └─optional_choice2─┘
```

If one of the optional items is the default, it will appear above the main path and the remaining choices will be shown below.

```
            ┌default_choice──┐
►►──STATEMENT──┼────────────────┼──────────────────────────────────────────►◄
            ├optional_choice─┤
            └optional_choice─┘
```

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

```
              ┌────────────────┐
              ▼                │
►►──STATEMENT────repeatable_item─┴──────────────────────────────────────────►◄
```

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

- Keywords appear in uppercase (for example, `PARM1`). They must be spelled exactly as shown. Variables appear in all lowercase letters (for example, *parmx*). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or such symbols are shown, you must enter them as part of the syntax.

COMMIT in Query Management

The COMMIT command ends a unit of recovery and commits database changes that the unit of recovery made.

If the COMMIT command is successful, all work is committed.

If the COMMIT command is unsuccessful, the connection state of the activation group and the states of its connections are not changed.

The COMMIT command causes no change in the current connection unless there was a RELEASE command performed on the current connection. If a RELEASE command was previously performed, on the current connection, it is removed and the activation group placed in an unconnected state. In a case such as this, the next Query Management command should be a CONNECT or SET CONNECTION command. The Query Management function does not know that the current connection was removed. Any SQL statements issued before a CONNECT or SET CONNECTION command fails.

```
►►──COMMIT──┬──────┬────────────────────────────────────────────────────────►◄
            └─HOLD─┘
```

HOLD

Indicates a hold on resources. If specified, currently open cursors are not closed, prepared SQL statements are preserved, and all resources acquired during the unit of work are held. However, locks on specific rows and objects implicitly acquired during the transaction are released. If HOLD is omitted, open cursors, except those declared with a WITH HOLD clause, are closed, prepared SQL statements are discarded, and held resources are released.

Examples of COMMIT in Query Management

If you do not need the connection to RDBONE in the next unit of work, the following commands remove the connection, assuming the commit operation is successful.

```
RELEASE RDBONE
COMMIT
```

You want to commit the changes made to RDBTWO but hold the resources acquired during the current unit of work.

```
COMMIT HOLD
```

CONNECT in Query Management

You can use the CONNECT command to change the database that is associated with the query instance. It can be used to connect an application process to a database or return the connection to the local database manager. The connect command connects to a database and places it in the current state. The current state means the connection is used for issued SQL statements. Only one connection can be in the current state for each activation group.

If the CONNECT command is successful:

- The previously current connection, if any, is placed in the dormant state.
- The identified application server is placed in the current state.
- The global variables DSQSDBNM and DSQCMTLV are updated. DSQSDBNM contains the name of the current database. DSQCMTLV contains the commitment control level being used. The ConnectionInfo in the InstanceInfo is updated with the necessary information from the SQL to reflect the current state.

If the CONNECT command is unsuccessful, the connection state of the activation group and the states of its connection are unchanged. The instance information is not updated.

The function of the CONNECT depends on the connection management method you are using. The methods are distributed-unit-of-work (*DUW) and remote-unit-of-work (*RUW). The default, unless you change it, is *DUW. For information on changing the connection management method, see “START in Query Management” on page 45. Table 2 lists the functional differences depending on the connection management method.

*Table 2. CONNECT Command Differences Between *DUW and *RUW*

*DUW	*RUW
Multiple connections allowed	One connection allowed
CONNECT to additional database puts previous connection in dormant state. The previous connections are not disconnected.	CONNECT to additional database disconnects previous connections. The previous connection or connections are disconnected before performing the connection.
Consecutive CONNECTs to same database fails	Consecutive CONNECTs to same database results in no current connection change

A system running under DUW connecting to a system running under RUW may result in a read-only connection.

Note: A homogenous connection is read-only as far as commitment control is concerned. A read-only connection is not allowed to make updates under commitment control. However, a homogenous read-only connection can still make non-committable updates.

```
►►──CONNECT──┬─TO──────┬──RESET──┬──────────────────────────────────────────────────────┬──►◄
             └─rdbname─┘         │            ┌─*CURRENT─┐             ┌─*NONE────┐     │
                                 └─(──USER=───┴─username─┴──PASSWORD=──┴─password─┴─────┘
```

Parameter list of CONNECT in Query Management

TO
The keyword used prior to specifying the remote database to connect to.

rdbname

The name of the database instance that is to serve as the application server. It is used to specify the relational database that is to be accessed through DRDA®. It can:

- Be up to 18 characters in length
- Consist only of:
  - Uppercase characters (A-Z)
  - Numerics (0-9)
  - Underscore (_)

  The first character must be an uppercase character.

If you specify a remote database name, you can use the USER and PASSWORD keywords to specify the user identification and password to be used with the remote database. If you specify either of these keywords, you must specify both of them. If you do not specify these keywords, the default user identification is *CURRENT and the default password is *NONE.

RESET

Indicates that the current conversation to a remote database, if connected, should be released and the connection reset to the local database.

Note: The CONNECT command should be used to reset to the local database before using the SAVE DATA AS command. It should also be used to manage multiple query instances which have different connection information.

The application program must be in a connectable state before the CONNECT command is issued. When RUN QUERY and ERASE TABLE commands are directed to the connection, make sure that the application program that issued the call to the query callable programming interface remains in the callable stack. If it does not, you are implicitly disconnected when the application program ends.

Examples of CONNECT under RUW connection management

The following CONNECT command creates the RDB1 connection and places it in the current state:

```
CONNECT TO RDB1
```

The following command disconnects the previous connection and creates a connection to the local database:

```
CONNECT RESET
```

The following command disconnects the previous connection before performing the connect:

```
CONNECT TO RDB2 (USER=BIZET PASSWORD=CARMEN)
```

Examples of CONNECT under DUW connection management

The following CONNECT command creates the RDB1 connection and places it in the current state:

```
CONNECT TO RDB1
```

The following CONNECT command put the current connection in a dormant state and creates a connection to the local database. The local database becomes the current connection.

```
CONNECT RESET
```

The following command puts the current connection in a dormant state and creates a connection to RDB2. The RDB1 and local connections are now in a dormant state.

```
CONNECT TO RDB2 (USER=BIZET PASSWORD=CARMEN)
```

DISCONNECT in Query Management

The DISCONNECT command destroys one or more connections.

The connection identified in a DISCONNECT command cannot be a connection that was used to run SQL statements during the current unit of work. Likewise, it cannot be a connection for a protected conversation. To remove a connection on protected conversations, use the RELEASE command.

If the DISCONNECT command is successful, each identified connection is removed.

If the DISCONNECT command is unsuccessful, the connection state of the activation group and the states of its connections is not changed.

The creation and maintenance of connections use resources. You should disconnect connections that are not going to be reused. You should not disconnect connections that are needed in subsequent units of work.

You should run the DISCONNECT command immediately after a commit operation. If you remove the current connection, the activation group is left in an unconnected state. In a case such as this, the next Query Management command should be a CONNECT or SET CONNECTION command. The Query Management function does not know that the current connection was removed. Any SQL statements issued before a CONNECT or SET CONNECTION command fails.

```
►►──DISCONNECT──┬─rdbname─┬──────────────────────────────────────────────►◄
                ├─CURRENT─┤
                └─ALL─────┘
```

rdbname
: The globally unique name of a database that serves as the application server. It specifies the relational database to access. The relational database is the means of identifying a database that can be accessed through DRDB support. *rdbname* can:

- Be up to 18 characters long
- Consist of only uppercase characters (A-Z), numerics (0-9), and the underscore (_). The first character must be an uppercase character.

CURRENT
: Identifies the current connection of the activation group.

ALL
: Identifies all connections in the activation group. None of the connections can use protected conversations.

Examples of DISCONNECT in Query Management

The connection to RDBONE is not needed in the next unit of work. The following command is run after a commit operation.

```
DISCONNECT RDBONE
```

The current connection is not needed in the next unit of work. The following command is run after a commit operation.

```
DISCONNECT CURRENT
```

None of the existing connections are needed in the next unit of work. The following command is run after a commit operation.

```
DISCONNECT ALL
```

ERASE in Query Management

The ERASE command removes a FORM, PROC, QUERY, or TABLE from the database.

```
►►──ERASE──┬─FORM──┬──name──┬──────────────────────────┬──────────────────────────────►◄
           ├─PROC──┤        └─(──CONFIRM──=──┬─YES─┬───┘
           ├─QUERY─┤                         └─NO──┘
           └─TABLE─┘
```

Parameter list of ERASE in Query Management

name

Names a FORM, PROC, QUERY, or TABLE to be removed. You must have the appropriate ownership and database or query authorization to remove an object.

This name can be a qualified name of the form library/object or database object. Specify the naming convention you intend to use in your first query command procedure.

A user can only erase those objects to which he has been granted *ALL authority and must also have *CHANGE or *ALL authority to the library in which the object resides.

CONFIRM=YES | NO

This option provides for a check before performing your ERASE request. The confirmation request occurs only when an existing object in the database is about to be erased. You will be asked if you want the pending ERASE command performed.

CONFIRM=YES forces a display of the confirmation. CONFIRM=NO suppresses a display of the confirmation and erases the object.

If your job is running interactively, an inquiry message is sent to your display station and the job is suspended until you respond to the message. The message asks whether you want to erase the object. If your job is running as a batch job, or DSQSMODE was set to batch mode during START processing, CONFIRM=YES results in an error.

The default value is CONFIRM=YES. You can change the default by setting the DSQCONFIRM variable as a START command parameter or in the start procedure.

When you issue the ERASE command, the system returns the message:

```
Object exists. Do you want to replace it?
```

This message is displayed only if you specify the CONFIRM=YES option or if you omit the CONFIRM option.

Examples of ERASE in Query Management

```
ERASE TABLE EMP

ERASE TABLE SMITH.EMP (CONFIRM=YES
ERASE TABLE SMITH/EMP (CONFIRM=YES

ERASE PROC SMITH/MONTHEND (CONFIRM=NO
```

You can issue an `ERASE TABLE` command only on database physical files.

Command keywords are supported in the extended parameter list on the callable interface as well as in the command string. For more information on using the extended parameter list, see “START in Query Management” on page 45.

EXIT in Query Management

The EXIT command stops your application's session with Query Management and ends the associated instance of Query Management in your process. No parameters are allowed with this command. No messages are generated.

The EXIT command is valid only when issued through the Callable Interface.

An implied EXIT command is processed for all query instances when the job ends.

The EXIT command is not valid in a query procedure.

There are no authority considerations related to the EXIT command.

```
►►──EXIT─────────────────────────────────────────────────────────────────────►◄
```

Examples of EXIT in Query Management

See Chapter 7, "Callable interface in Query Management" for examples of programs that use the EXIT command.

EXPORT in Query Management

The EXPORT command is used to create a file containing the contents of certain Query Management objects. Chapter 8, "Exported and Imported Objects in Query Management" discusses exported objects in detail. The following objects can be exported: FORM, PROC, or QUERY. You cannot export the contents of a sort sequence table associated with a *QMQRY object.

For more information about sort sequence, see Appendix D, "Examples of sort sequence in Query Management" on page 231.

*QMQRY objects defined before Version 2 Release 3 of the iSeries system are always run using the hexadecimal sort sequence. When these objects are exported, SRTSEQ is set to *HEX and there is no LANGID parameter.

```
►►──EXPORT──┬─FORM──┬──name──TO──filename──┤ Options ├───────────────────────►◄
            ├─PROC──┤
            └─QUERY─┘
```

Options

```
├──┬──────────────────────────────────────────────────────────┬──────────────┤
   └─(──┬──────────────────┬──┬────────────────────────────────┬─┘
        └─CONFIRM=──┬─YES─┬┘  └─COMMENT=──'──comment for member──'─┘
                    └─NO──┘
```

Parameter list of EXPORT in Query Management

name
: Names a FORM, PROC, or QUERY to be exported.

This name can be a qualified name of the form library/object or database.object. Specify the naming convention you intend to use in your initial query command procedure.

If the form or query specified is not found, and DSQSCNVT=YES is specified on the START command, Query Management searches for a Query for iSeries definition with that name. If a query definition is found, the information is used to create a temporary query or form that is usable by Query Management

filename

Names the system file that receives the exported object.

To be consistent across systems, you should not specify more than a single file name without qualifiers or extensions. This enables system defaults to take effect.

If you are exporting an object to a system other than an iSeries system, it is recommended that the name of the file be from 1 to 8 characters long.

Query Management qualifies the single name to fit the data naming requirements of the operating environment.

You must be aware of these naming restrictions and be able to deal with them when transporting Query Management objects between operating environments.

CONFIRM=YES | NO

This option provides for a check before performing your EXPORT request. The confirmation request occurs only when an existing file is about to be replaced. You will be asked if you want the pending change to occur.

CONFIRM=YES forces a display of the confirmation. CONFIRM=NO suppresses a display of the confirmation.

The default value is CONFIRM=YES. You can change the default by setting the DSQCONFIRM variable as a START command parameter or in the start procedure.

COMMENT=comment for member

Use the comment option to specify the member text when exporting a form, procedure, or query object. Comments are useful for preserving information about the object. Because comments usually include embedded blanks, they must be enclosed in apostrophes. Apostrophes within a comment must be specified by two adjacent apostrophes.

Examples:

```
COMMENT='SALES QUERY'
COMMENT='THIS QUERY DOESN'T INCLUDE SALES'
```

The maximum comment length in Query Management is 50 characters.

CCSID considerations of EXPORT in Query Management

When a Query Manager query, form, or procedure is exported and the source file does not exist, the file is created with the CCSID of the job. The data for the form CCSID is converted to the CCSID of the source file. If the source file does exist and the CCSID of the source file is different from the CCSID of the Query Manager query, form, or procedure, data is converted from the CCSID of the query, form, or procedure to the CCSID of the source file.

When a Query Manager procedure is exported and the CCSID of the file it is exported from is different from the CCSID of the file it is exported to, the data is first converted to the CCSID of the job, then to the CCSID of the file it is going to. To avoid this extra conversion, copy the procedure instead of exporting it.

Examples of EXPORT in Query Management

```
EXPORT QUERY SAMP1 TO SAMP1EX
```

```
EXPORT FORM EXLIB/EX1 TO EXLIB/FILE(EX1F) (CONFIRM=YES
```

EXPORT

Command keywords are supported in the extended parameter list on the callable interface as well as in the command string. For more information on using the extended parameter list, see “START in Query Management” on page 45.

GET in Query Management

The GET command is used to get the value of a Query Management variable and provide it to a user program. When using the GET command from a program, the following command syntax must be used.

```
►►──GET GLOBAL───┬varnum──varlen──varname─┬──────────────────────────────────────────────►◄
                 └vallen──values──valtype─┘
```

GLOBAL
In Query Management, the variable varname located in the global variable pool is returned to the requester. If the variable is not found in the global pool, an error message is returned.

Extended Parameter List:

varnum

Number of varnames that are requested for this call.

varlen

Length of each varname that is specified.

varname

Name of the variable located in the Query Management variable pool.

vallen

Length of program storage that is to contain the varname value.

values

Program storage area that is to contain the varname value.

valtype

Data type of the storage area that is to contain the varname value.

Examples of GET in Query Management

For examples of using the GET command in a program, see Chapter 7, “Callable interface in Query Management”.

Command keywords are supported in the extended parameter list on the callable interface as well as in the command string. For more information on using the extended parameter list, see “START in Query Management” on page 45.

IMPORT in Query Management

You can use the IMPORT command to copy a file containing an exported object into one of the following Query Management objects: FORM, PROC, or QUERY. The IMPORT command does not affect the external file.

name
Name of the FORM, PROC, or QUERY to be imported. This can be a qualified name.

filename
Name of the system file that Query Management is to read (i.e., the source file for the imported object).

In order to be consistent across systems, do not specify any more than a single name without qualifiers or extensions; use the system defaults.

```
►►──IMPORT──┬─FORM──┬──name──FROM──filename──────────────────────────────────────────────────►
            ├─PROC──┤
            └─QUERY─┘

►──┬─────────────────────────────────────────────────────────────────────────────────────┬──►◄
   └─(──┬─────────────────────┬──┬───────────────────────────┬──┬──────────────────────────────────┬──┬─────────────────────────┬─┘
        └─CONFIRM=──┬─YES─┬───┘  └─COMMENT=──'──comment──'───┘  └─SRTSEQ=──┬─*SRC──────────────────┤  └─LANGID=──┬─*SRC──────┤
                    └─NO──┘                                                ├─*JOBRUN───────────────┤             ├─*JOBRUN───┤
                                                                           ├─*JOB──────────────────┤             ├─*JOB──────┤
                                                                           ├─*LANGIDSHR────────────┤             └─Lang. ID──┘
                                                                           ├─*LANGIDUNQ────────────┤
                                                                           ├─*HEX──────────────────┤
                                                                           └─┬─*LIBL/────┬──table──┘
                                                                             ├─*CURLIB/──┤
                                                                             ├─library/──┤
                                                                             └───────────┘
```

CONFIRM=YES | NO

This option provides for a check before performing your IMPORT request. The confirmation request occurs only when an existing object in the database is about to be replaced. You will be asked if you want the pending database changes to occur.

If your job is running interactively, an inquiry message is sent to your display, and the job is suspended until you respond to the message. The message asks whether you want to import the object. If your job is running in batch mode, or DSQSMODE was set to Batch during START processing, CONFIRM=YES results in an error.

CONFIRM=YES forces a display of the confirmation. CONFIRM=NO suppresses a display of the confirmation.

The default value is CONFIRM=YES. You can change the default by setting the DSQSCNRM variable as a START command parameter or in the start procedure.

COMMENT=comment

Use the comment option to specify the member text when exporting a form, procedure, or query object. Comments are useful for preserving information about the object. Because comments usually include embedded blanks, they must be enclosed within apostrophes. Apostrophes within a comment must be specified by two adjacent apostrophes.

Examples:

```
COMMENT='My form'
COMMENT='This form doesn''t include breaks'
```

The maximum comment length in Query Management is 50 characters.

SRTSEQ

Specifies the sort sequence used for this query. The sort sequence determines which sort sequence table is used when the query is run. The SRTSEQ parameter is allowed only at the time you create a query. The SRTSEQ parameter is not allowed when you create a form. The possible values are:

***SRC**

Contains the specification of the sort sequence used in creating the query. If SRTSEQ is not specified in the source file member, the default for SRTSEQ is *JOBRUN.

***JOBRUN**

Uses the SRTSEQ associated with the job at the time the query is run.

***JOB**

Uses the sort sequence associated with the job at the time the query is created.

***HEX**

Uses the binary value of the character to determine the sort sequence.

***LANGIDSHR**

Uses a system provided table with *shared* weights for each character as the sort sequence. The table used is determined by the value specified for the LANGID parameter.

***LANGIDUNQ**

Uses a system provided table with *unique* weights for some characters as the sort sequence. The table to be used is determined by the value specified for the LANGID parameter. If SRTSEQ=*LANGIDUNQ and the LANGID parameter is not specified, the default for LANGID is *JOBRUN.

tablename

Uses an external sort sequence table object which contains the sort sequence to use when the query is run.

LANGID=*SRC | *JOBRUN | *JOB | Language ID

The language identifier associated with the query. At this time, the only use of this parameter is to determine which IBM-supplied sort sequence table to use when the query is run. It can only be used when SRTSEQ=*LANGIDUNQ or SRTSEQ=*LANGIDSHR. Regardless of the SRTSEQ value, the LANGID value is validated and the created query contains the specified LANGID value.

***SRC**

Contains the specification of the language identifier to use when creating queries. If LANGID is not specified in the source, the default value of *JOBRUN is used.

***JOBRUN**

The language identifier to use when the query is run is determined at the time the query is run.

***JOB**

The language identifier to use when the query is run is determined at the time the query is created.

Language ID

A three-character language identifier.

The IMPORT command is typically used in the following situations:

- To copy or propagate FORM, PROC, and QUERY objects from one Query Management installation to another (the objects are exported by the sending installation and imported by the receiving installation).
- To use a full-function editor outside of Query Management The following is a typical scenario:
  1. First, export the Query Management object. This causes the creation of an external file.
  2. Next, invoke a text editor. Once inside the editor, you can perform normal editing functions like copying and moving.
  3. Once you finish editing, return to Query Management and import the file. The edited object is stored in the database.
- For application programmers who want to migrate SQL queries from program libraries that typically reside outside of Query Management to libraries inside Query Management, for purposes of modification and interactive processing (testing).

The IMPORT command copies the contents of the specified file into the database. For SQL queries and procedures, each record in the file becomes a separate line in the object. All files that were exported via the Query Management EXPORT command can be imported.

When importing files containing SQL queries and procedures, DB2 UDB for iSeries Query Management accepts records having a logical record length greater than 79, even though the resulting data may be truncated. If Query Management finds a logical record length greater than 79, it displays a warning message.

If the imported query has a fixed record format (and logical record length greater than 79), then Query Management accepts only data in positions 1 through 79 and ignores the rest.

If the imported form has a fixed record format (and logical record length greater than 150), then Query Management accepts only data in positions 1 through 150 and ignores the rest.

When importing query objects with a logical record length less than 79, DB2 UDB for iSeries Query Management pads the record with blanks up to and including position 79. If the line contains an open delimited string, this padding will then be included within the delimited string and may cause unexpected results.

When importing form objects with a logical record length less than 150, Query Management pads the record with blanks up to and including position 150. If the line contains an open delimited string, this padding will then be included within the delimited string and may cause unexpected results.

When importing SQL QUERY and PROC objects, Query Management does not perform any validation or semantic checking on the contents of the files. Therefore, it is possible to establish QUERY and PROC objects containing non-displayable characters (this could happen if a program's object file were imported as a QUERY). Also, it is possible to IMPORT SQL statements into the PROC object and vice versa. It is your responsibility to avoid such mistakes, since Query Management contains no provision for "re-categorizing" contents.

Query Management validates FORM objects. If some part of the file fails a validation test, then the object is brought into the database, but you are sent warning messages. It is possible for the file to pass the validation test, yet provide unpredictable results when used for formatting.

CCSID considerations of IMPORT in Query Management

When a Query Manager query or form is imported, it is tagged with the CCSID of the file that the query or form is imported from. When a Query Manager procedure is imported and the file does not exist, a file is created with the CCSID of the job. When a query management procedure is imported and the file does exist, the procedure is converted to the CCSID of the job and then to the CCSID of the file being imported. To avoid the additional conversion, copy the procedure instead of importing it.

Examples of IMPORT in Query Management

```
IMPORT FORM REPORT1 FROM REPT1EX

IMPORT QUERY SALARYWK FROM JENSON
```

Command keywords are supported in the extended parameter list on the callable interface as well as in the command string. For more information on using the extended parameter list, see "START in Query Management" on page 45.

PRINT in Query Management

The PRINT command is used to print a hard copy listing of an Query Management object. You cannot print the contents of a sort sequence table associated with a *QMQRY object. However, the name of the sort sequence table or sort sequence value associated with the query is printed.

The PRINT command uses standard system facilities for printing. Query Management does not externalize the printer attributes to your application. Nor does Query Management alter these attributes' values. Instead, it simply honors the printer definitions currently in effect.

An object's printed appearance is very much like its screen appearance. However, on a REPORT, there are some differences between display format and print format.

- The "panel title" line of the displayed REPORT object is replaced with a "page heading" at the top of each page in the printed report (assuming that a page heading has been defined).
- A "page footing" is provided at the bottom of each page of the printed report, but only once at the bottom of the displayed object.

PRINT

```
►►──PRINT──┬─FORM──┬──name──┬─────────────────────────────┬──────────────────────────────────────────►◄
           ├─PROC──┤        └─(──PRINTER──=──printer name─┘
           └─QUERY─┘


►►──PRINT──REPORT──┬────────────────────────────────────────────────────┬──────────────────────────►◄
                   │          ┌─────────────────────────────────────┐   │
                   │          ▼                                     │   │
                   └─(───────┬─WIDTH──=──max char per line──────────┬┴──┘
                             ├─LENGTH──=──max number lines──────────┤
                             ├─FORM──=──form name───────────────────┤
                             ├─PRINTER──=──printer name─────────────┤
                             ├─DATETIME──=──┬─YES─┬─────────────────┤
                             │              └─NO──┘                 │
                             └─PAGENO──=──┬─YES─┬───────────────────┘
                                          └─NO──┘
```

name
The name of the object to be printed. The name specified may be a FORM, PROC, or QUERY in the database.

If the form or query specified is not found, and DSQSCNVT=YES is specified on the START command, Query Management searches for a Query for iSeries definition with that name. If a query definition is found, the information is used to create a temporary query or form that is usable by Query Management

WIDTH=*maximum characters per print line*
An integer between 22 and 378 inclusive.

Reports that are wider than the print WIDTH will be split between pages. Query objects other than reports are not split between pages. If the object is wider than the print width, the lines in the printout will be truncated on the right.

It is important that you ensure the compatibility of WIDTH with the printer you are using. For example, if your current printer settings identify a 10 pitch device (10 characters per inch) mounted with 8.5 inch wide paper, then a WIDTH value of 132 results in modified output. The exact result may depend on printer hardware and software. Because Query Management does not know the width of the physical printer, no special message displays when this situation occurs.

If you do not specify this option, then Query Management uses the corresponding system or user default. If this value is not available, then the default is set to 80.

LENGTH=*maximum number of lines per page*
An integer between 1 and 255 inclusive.

When a report is to be printed and the value for LENGTH is inadequate (that is, if the value for LENGTH is less than the total number of lines needed for column headings, page headings and footings, plus the line needed to print the page number and/or date and time), then an error message is generated and the report is not printed.

For LENGTH values within the range allowed, Query Management performs a page eject whenever the number of lines of report data printed on a page is equal to LENGTH.

If you do not specify this option, then Query Management uses the corresponding system or user default. If this value is not available, then the default is set to 66.

FORM=*form name*
The name of the FORM that you want to use to format your data. If no form is specified, the form used in the previous RUN QUERY is used.

PRINTER=*printer name*

The name of the printer that produces the output.

Query Management can be directed to a different printer file through the use of the OS/400 OVRPTRF (Override Printer File) CL command. This command cannot be used to permanently change Query Management printer files. However, to run a PRINT command again and use the default page length and width values, use the CHGPRTF CL command to permanently change the printer file.

DATETIME=YES | NO

This option controls the generation and display of the system date and time on the bottom of each page. When DATETIME=YES, the date and time are placed on the last line of each page. When DATETIME=NO, the system date and time do not print. The default for this option is YES.

PAGENO=YES | NO

This option controls the printing of page numbers on the last line of each page. The default for this option is YES.

On OS/2, the default for this option is obtained from your profile.

Note: You may use any or all of the options associated with the PRINT REPORT command, but each option should only be used once. If an option is used more than once, the last one will be used.

CCSID considerations of PRINT in Query Management

When a Query Manager query is printed, it is converted from the CCSID that the query is tagged with to the CCSID of the job. When a Query Manager form is printed, no CCSID conversion takes place. When a Query Manager report is printed, the data is in the CCSID of the job. If a form is used on a PRINT REPORT command, no CCSID conversion is done and parts of the form may not be recognized. When a Query Manager procedure is printed, it is converted to the CCSID of the job.

Examples of PRINT in Query Management

```
PRINT QUERY queryname

PRINT FORM formname

PRINT PROC procname

PRINT REPORT

PRINT REPORT (WIDTH=80 LENGTH=60 DATETIME=YES PAGENO=YES

PRINT QUERY Q1 (PRINTER=PRT1

PRINT PROC LIBA/PROCA(MBRA)
```

Printer file use in Query Management

Default printer files called QPQXOBJPF and QPQXPRTF are included as part of query management and are in the QSYS library. These printer files are used when a PRINT QUERY, PRINT PROC, or PRINT REPORT command is issued. The printer file QPQXOBJPF has page length and width defaults of 66 lines and 132 characters, respectively. The printer file QPQXPRTF has page length and width defaults of 66 lines and 80 characters, respectively. The printer device name specified by the printer file is *JOB, which lets all printer output be directed to the printer set up for the job. Unless overridden, the printer files QPQXOBJPF and QPQXPRTF are used by query management for formatting the printed objects and report.

You can direct query management to use a different printer by specifying a value on the PRINT command or by changing the default value DSQAPRNM on the START command from *SAME to a printer name or *JOB.

PRINT

You can direct query management to use a different printer file by using the Override Printer File (OVRPRTF) CL command.

You can use the Change Printer File (CHGPRTF) CL command to permanently change the query management printer files QPQXOBJPF and QPQXPRTF. To use defaults again, issue another CHGPRTF CL command to change the attributes back.

On every install, the printer files are created again in the QSYS library. All changes to the printer files must be applied again. To save changes to a printer file, you can create your own printer file in your library with the desired attributes and use the OVRPRTF CL command to direct query management to this printer file. You can also copy the printer files to your own library and make the changes to the copy in that library. To use a printer file with the same name as the one in the QSYS library, your library must be in the library list before the QSYS library.

PRINT object formatting in Query Management

While processing the PRINT QUERY and PRINT PROC commands, query management formats the printed output into 132 column lines. The column lines are broken down into 123 bytes of text and 7 bytes for the line number, which is generated during the PRINT command.

The width of 132 is wide enough to handle the printing of most files with ease and is compatible with most iSeries printers.

Directing the printer output to a printer with a line width less than 132 characters results in possible loss of data unless the printer file has *YES specified for the Fold Record parameter. The default for the Fold Record parameter in the QPQXOBJPF printer file is *NO.

PRINT report formatting in Query Management

While processing the PRINT REPORT command, query management formats the printed report using the width specified on the PRINT command or the default from the printer file. If the report is wider than the print WIDTH, it is split between pages. In this case, multiple printer files are opened, and each segment of each report line is directed to the appropriate opened printer file.

If the report is directed to a printer with a width less than the WIDTH specified on the PRINT command or in the printer file, each print record is truncated. If the Fold Record option in the printer file is changed from the default to *YES, each print record is wrapped. For example, a report that formats to 200 columns when printed with a command of PRINT REPORT (WIDTH=200 PRINTER=xyz, results in line wrapping if the printer width is less than 200. The Record Wrap option on the printer file has been overridden to *YES. If the Record Wrap option is not overridden, the rightmost columns in the report are truncated.

Query Management uses the sort sequence table in effect when the query was run to do report formatting for:

- Calculations of
  - MIN
  - MAX
- BREAKn level processing against the following data types:
  - CHAR
  - VARCHAR
  - DBCS-Open
  - DBCS-Either data

Note: Query Management does not support the use of sort sequence for MIN and Max Calculations for BREAKn level processing for GRAPHIC and VARGRAPHIC data types. The sort sequence is not applied to DBCS data. Instead, the binary value of each byte of the DBCS data is used for comparison.

For more information on using sort sequence, see Appendix D, “Examples of sort sequence in Query Management” on page 231.

RELEASE in Query Management

The RELEASE command places one or more connections in the released state. The released state means a disconnect is to occur for the connection at the next successful commit operation. A rollback has no affect on connections. You can think of the released state as a pending disconnect.

If the RELEASE command is successful, each identified connection is placed in the released state and will be disconnected at the next successful commit operation.

If the RELEASE command is unsuccessful, the connection state of the activation group and the states of its connections are not changed.

The creation and maintenance of connections use resources. You should put connections that are not going to be reused in the released state. You should not release connections that are needed in subsequent units of work.

If the current connection is in the released state when a commit operation runs successfully, the connection is removed and the activation group is left in an unconnected state. In a case such as this, the next Query Management command should be a CONNECT or SET CONNECTION command. The Query Management function does not know that the current connection was removed. Any SQL statements issued before a CONNECT or SET CONNECTION command will fail.

```
►►──RELEASE──┬─rdbname─┬──────────────────────────────────────────────►◄
             ├─CURRENT─┤
             └─ALL─────┘
```

rdbname
: The globally unique name of a database that serves as the application server. It specifies the relational database to access. The relational database is the means of identifying a database that can be accessed through DRDA support. *rdbname* can:

- Be up to 18 characters long
- Consist of only uppercase characters (A-Z), numerics (0-9), and the underscore (_). The first character must be an uppercase character.

CURRENT
Identifies the current connection of the activation group.

ALL
Identifies all connections in the activation group.

Examples of RELEASE

The connection to RDBONE is not needed in the next unit of work. The following command causes it to be removed at the next successful commit operation.

```
RELEASE RDBONE
```

The current connection is not needed in the next unit of work. The following command causes it to be removed at the next successful commit operation.

RELEASE

```
RELEASE CURRENT
```

None of the existing connections are needed in the next unit of work. The following command causes them to be removed at the next successful commit operation.

```
RELEASE ALL
```

RUN in Query Management

The RUN command processes a PROC or QUERY. When you issue the RUN command, you must identify the PROC or QUERY that you want processed. Therefore, the PROC or QUERY must exist in the database prior to issuing the RUN command.

When used to process a QUERY (SELECT only), the RUN command produces new data, replacing the existing data produced from any previous RUN QUERY.

When running in interactive mode, the results of a query (SELECT only) will be displayed. When running in batch mode, the results will not be displayed.

When using a query that changes a field defined with a referential constraint, constraint checking is done and violation errors may occur. Check-pending errors may occur when Query Management changes or references a field in a dependent file or changes a field in a parent file.

Normally, a QUERY uses a FORM when run. This can occur in two ways:

1. An existing FORM is explicitly named on the RUN command via the FORM option.
2. A default FORM is created. This default form is constructed using rules that take into consideration the column attributes of the DATA.

```
►►──RUN──┬─PROC──name─────────────────────────────────┬──────────────────────────────►◄
         └─QUERY──name─┬────────────────────────────┬─┘
                       │         ┌────────────────┐ │
                       │         ▼                │ │
                       └─(────┬─FORM──=──form name─┬┴─┘
                              └─DISPLAY──=──┬─YES─┬┘
                                            └─NO──┘
```

name
: The name of the QUERY being run.

FORM=*form name*

This option is meaningful only for queries that contain a SELECT statement.

The FORM option specifies the FORM to be used in formatting the REPORT that RUN automatically displays when running in interactive mode.

If the form specified via the FORM option cannot be found (for instance, when the FORM does not exist), then the RUN command is rejected with an error message. If the form does exist but simply *will not work* with the DATA (perhaps different data types for the columns were specified), then Query Management responds with an error message.

If you omit the FORM option, the default FORM is used.

If the form or query specified is not found, and DSQSCNVT=YES is specified on the START command, Query Management searches for a Query for iSeries definition with that name. If a query definition is found, the information is used to create a temporary query or form that is usable by Query Management

DISPLAY=YES | NO

Use the DISPLAY keyword to indicate whether to display the report. This keyword defaults to YES if you are processing interactively. If you specified batch mode on the START command, this keyword is ignored.

You can use the extended parameter list format for this command. For more information on this format, see “GET in Query Management” on page 30.

RUN considerations in Query Management

Date format use for RUN in Query Management

When a Query Manager query contains a date format that is different from the job date format and the query contains global variables, the date format used to interpret the SQL statement is an SQL date format. This is done to avoid choosing between the job date format and the query date format when interpreting any date literal in the SQL statement.

CCSID use of RUN in Query Management

Query Management queries are interpreted in the CCSID that they are tagged with. When data is retrieved from a file which is in a different CCSID than the job, the data is converted to the CCSID of the job if it is displayed or printed. When data is saved to a database file, it is converted to the CCSID of the file if the file already exists or into the CCSID of the file that the data was retrieved from if a new file is created. If the data is saved and not displayed, there is no CCSID conversion if a new file is created. To avoid the conversion of data from the original file CCSID to the job CCSID to the file that the data is saved into, specify the DISPLAY=NO parameter on the RUN QUERY command before running the SAVE DATA AS command.

If a form is used that was created using a file with a different CCSID than the job, the text fields are not converted. This could result in unrecognizable text on the displayed report. To improve the appearance of the report, export the form to a file with the same CCSID as the job and import it again.

CALL SQL RUN limitations in Query Management

If the CALL SQL statement is run through Query Manager, and the called program runs locally, the called program cannot invoke the:

- Query Manager CPI interface
- Query Manager CL commands
- SQL Query Manager CL commands.

Query Manager cannot be run recursively. However, if the called program runs remotely, the called program can invoke Query Manager on the remote system. The CALL SQL statement fails with the SQL0469 error is an attempt is made to call a procedure that is defined with an output parameter. This is a run-time error, not a query definition error.

Examples of RUN in Query Management

```
RUN QUERY QN1
```

```
RUN PROC WEEKREPT
```

```
RUN QUERY SMITH.Q6 (FORM=SMITH.SAL_REPT
```

Command keywords are supported in the extended parameter list on the callable interface as well as in the command string. For more information on using the extended parameter list, see “START in Query Management” on page 45.

SAVE in Query Management

Use the SAVE command to save data in a table in the database. The saved TABLE is named according to the name you specify with the command.

If DATA is saved and a TABLE/VIEW is actually being replaced, then the data must be compatible with the existing definition. Compatible data has matching data types, lengths, and null attributes. Specifically, the number of columns in DATA must match the target, and the columns must have compatible data types and null characteristics. If the two objects are incompatible, then Query Management rejects the SAVE command and the database remains unchanged.

If the name on the SAVE command already exists as a view in the database, the table on which the view is defined will be changed according to the rules of updating a table through a view.

The column names for a saved TABLE that does not already exist are generated by Query Management using the same algorithm that is used in generating the default column headings in the FORM object. You cannot change the column names.

For information about using the SAVE command with double-byte character set (DBCS) graphic data (DBCS-graphic data type), see Appendix A, "DBCS data in Query Management".

```
►►──SAVE──DATA──AS──tablename──┬────────────────────────────────────────────────────┬──────────►◄
                               │            ┌─────────────────────────────────────┐ │
                               └─(──────▼───┬─CONFIRM──=──┬─YES─┬─────────────────┴─┘
                                            │             └─NO──┘                 │
                                            ├─COMMENT──=──'comment for table'─────┤
                                            └─ACTION──=──┬─REPLACE─┬──────────────┘
                                                         └─APPEND──┘
```

DATA
Refers to the active result from the previously running QUERY.

tablename
The name of the table or view in which the data is stored in the database. It is normally unqualified.

The table name must meet iSeries system naming conventions. SQL supported long names and names with special characters are not supported by the SAVE DATA AS command.

If the table name specified does not exist in the database, a new table is created. This name can be a qualified name of the form library/object or database.object. Specify the naming convention you intend to use in your initial query command procedure.

To save the data to a table, you must have proper SQL authority to change or create a table. Refer to the SQL Reference for table authorization rules. If ACTION=REPLACE, you must have authority to the Clear Physical File Member (CLRPFM) CL command also.

SQL conventions allow a view to be updated only if it is associated with just one table. Updating a view on multiple tables is not allowed, nor is updating a view associated with a table that is an SQL catalog.

CONFIRM=YES | NO
This option provides for a check before performing your SAVE request. The confirmation request occurs only when an existing object in the database is about to be replaced. You will be asked if you want the pending database changes to occur.

CONFIRM=YES forces a display of the confirmation. CONFIRM=NO suppresses a display of the confirmation. In either case, a confirmation message is generated to inform you that the SAVE operation is complete.

The default value is CONFIRM=YES. You can change the default by setting the DSQCONFIRM variable as a START command parameter or in the start procedure.

COMMENT='*comment for table*'

Use this option to supply a comment when saving data as a table. Comments are useful for preserving descriptive information about the object.

Because commentary usually consists of multiple words and embedded blanks, you must enclose it in apostrophes. Apostrophes embedded within the commentary must be entered as two adjacent apostrophes.

Examples:

```
COMMENT='The master EMPLOYEE table-see John (X3971)'
```

```
COMMENT='Don't ERASE this data without telling Phil!'
```

Query Management restricts object commentary to a maximum of 50 characters, excluding the apostrophes.

ACTION=REPLACE | APPEND

The ACTION=REPLACE option causes an existing table or view to be replaced. The ACTION=APPEND option causes the data to be added to the end of an existing table or view. The default is ACTION=REPLACE. The ACTION keyword is ignored if the table or view does not exist.

Note: You can use any or all of the options associated with the SAVE command, but each option should only be used once.

Examples of SAVE in Query Management

```
SAVE DATA AS EMP12
```

```
SAVE DATA AS EMP12 (COMMENT='CLASSIC TWO TABLE JOIN'
```

Command keywords are supported in the extended parameter list on the callable interface as well as in the command string. For more information on using the extended parameter list, see “START in Query Management” on page 45.

Null value considerations of SAVE in Query Management

Because null values can be specified for fields and columns, you should keep the following in mind when saving data using the SAVE DATA AS command.

- A column which resulted from an SQL function is null capable.
- If a report has a column from an SQL function using group by, and if no records were selected, the value for the column is null.
- A column that results from an SQL numeric expression is null capable.
- If a result field is created based on a null-capable field, it is null capable.
- If a null value is encountered during the calculation of a result field, the value for that result field is null. For example, if a field in concatenation contains a null value, the result contains a null value. If a field used in a substring operation contains a null value, the result contains a null value.

Referential constraint considerations of SAVE in Query Management

When using the OS/400 referential constraint capabilities and features, you should keep the following in mind using the SAVE DATA AS command.

- If a selected table has a referential constraint, the constraint is not propagated to the output table when it is created for the SAVE DATA AS command.
- When you output the result of a query to an existing table, the SAVE DATA AS command may fail if you are replacing the table and it is a primary with a constraint relationship.

SAVE

- When you output the result of a query to an existing table, the SAVE DATA AS command may fail if the output table has a referential constraint that is violated. This error may not be detected until after the existing table has been changed. For example, if data in the existing table is replaced, the table is cleared before the new data is inserted.
- Check-pending errors may occur when Query Management attempts to save data to a dependent or parent file when an established/enabled constraint is in check-pending. Check-pending errors should be caught before the existing table is changed. The SAVE DATA AS command processing indicates whether the file is changed at the time the error occurred.
- Take care when using the SAVE DATA AS command to replace data in a table. For example, referential constraints can be set up in such a way that if a record is deleted from a table, one or more records in another table may also be deleted. When doing a SAVE DATA AS command, if the data is being replaced a Clear Physical File Member (CLRPFM) command is executed against the file if it is a physical file or if an SQL DELETE is executed against a logical file. An SQL INSERT is done in any case to actually insert the data into the output table.

Long column name considerations of SAVE in Query Management

When the SAVE DATA AS command creates an output table, selected long column names and column names with special characters are retained in the output table. If the column name for the selected column is a duplicate of another column name or system column name, the column name is modified to make it unique. No warnings are generated indicating that the column name of the column in the outfile is different than the original column name. For example, if a query of files A and B:

Table Name	System Column Name	Column name
A	NAME	EMPLOYEENAME
B	EMPNAME	EMPLOYEENAME
B	CUSNAME	NAME

is output to table C, it will have the following names:

Table Name	System Column Name	Column Name
C	NAME	EMPLOYEENAME
C	EMPNAME	EMPLOYEENAME1
C	CUSNAME	NAME2

SET CONNECTION in Query Management

The SET CONNECTION changes the state of a connection from dormant to current. The dormant state means the connection is suspended. When the connection is in the dormant state, no SQL statements use the connection except for commits and rollbacks. The current state means the connection is used for issued SQL statements. Only one connection can be in the current state for each activation group. The SET CONNECTION command is useful only when running under DUW connection management.

If a SET CONNECTION command is successful:

- The previously current connection, if any, is placed in the dormant state.
- The identified application server is placed in the current state.
- The global variables DSQSDBNM and DSQCMTLV are updated. DSQSDBNM contains the name of the current database. DSQCMTLV contains the commitment control level being used. The connection information in a Query Management instance is updated with the necessary information from the SQL CA to reflect the current state.

If the SET CONNECTION command is unsuccessful, the connection state of the activation group and the states of its connection are unchanged. The instance information is not updated.

```
►►──SET CONNECTION──rdbname──────────────────────────────────────────────────────►◄
```

rdbname
The globally unique name of a database that is to serve as the application server. It specifies the relational database to access. The relational database is the means of identifying a database that can be accessed through DRDA support. *rdbname* can:

- Be up to 18 characters long
- Consist of only uppercase characters (A-Z), numerics (0-9), and the underscore (_). The first character must be an uppercase character.

Examples of SET CONNECTION in Query Management

The following commands send SQL statements to RDBONE after the first command, to RDBTWO after the second, and then to RDBONE again after the third command. These examples are contingent on running under DUW connection management.

```
CONNECT TO RDBONE
:
SQL statements
:
CONNECT TO RDBTWO
:
SQL statements
:
SET CONNECTION RDBONE
:
SQL statements
```

The first CONNECT command creates the RDBONE connection and places it in the current state. The second CONNECT command creates the RDBTWO connection, places it in the current state, and places RDBONE in the dormant state. The SET CONNECTION command returns RDBONE to the current state and places RDBTWO in the dormant state.

SET GLOBAL in Query Management

The SET GLOBAL command is used to set the value of an Query Management variable from the user program or procedure. When using the SET GLOBAL command from a procedure, the short version of the command syntax must be used. When using the SET GLOBAL command from a program, the extended version of the command syntax must be used.

```
►►──SET GLOBAL──┬─(varname=userval────────────────────────────┬──────────────────►◄
                └─varnum──varlen──varnames──vallen──values──valtype─┘
```

GLOBAL
In Query Management, the variable varname located in the global variable pool is set by the requester. If the variable does not exist, a new variable is created. If the variable does exist, its contents are replaced.

varname
Name of the variable located in the Query Management variable pool. For rules that apply to variable names used across the callable interface, see “Variable names in the Query Management CI” on page 81.

userval
The value that is to be associated with the variable name specified by varname. If it is a constant enclosed in apostrophes, the apostrophes are removed.

Extended Parameter List:

varnum
Number of varnames that are requested for this call.

varlen
Length of each varname that is specified.

varnames
Name of the variable located in the Query Management variable pool.

vallen
Length of program storage that is to contain the varname value.

values
Program storage area that is to contain the varname value.

valtype
Data type of the storage area that is to contain the varname value.

Examples of SET GLOBAL in Query Management

The following are examples of the SET GLOBAL command as used in a PROC. For examples of using the SET GLOBAL command in a program, see Chapter 7, “Callable interface in Query Management”.

```
SET GLOBAL (CHARVAR = 'abc'

SET GLOBAL (NUMBVAR = 199

SET GLOBAL (NAMEVAR = MYTABLE

SET GLOBAL (CHARVAR='abc'
```

Quotation marks in *varname* values when using SET GLOBAL in Query Management

Use two adjacent single quotation marks to represent a quotation mark in a character string *varname* value if the variable is set with the short command syntax. Use a single quotation mark in a character string *varname* value to represent a quotation mark if the variable is set through the extended parameter list format. See Appendix C, “Use of quotation marks and apostrophes when setting global variables in Query Management” for additional information.

Command keywords are supported in the extended parameter list on the callable interface as well as in the command string. For more information on using the extended parameter list, see “START in Query Management” on page 45.

Programming considerations of SET GLOBAL in Query Management

The SET GLOBAL command is useful for selecting run-time records. You do not need to save numerous QMQRY objects with different SELECT fields or WHERE conditions.

For example, you may want to run a query on a file that contains records for employees whose names start with letters in the first part of the alphabet. You may also want to run the same query with records for employees whose names start with letters in the last part of the alphabet. Your query object, named EMPREPORT, could contain the following SQL SELECT statement:

```
   SELECT  NAME,DEPT,EMPNO  FROM  MASTER
           WHERE  NAME  =  &STARTAL  AND  NAME  <  &ENDAL
```

You could then set up a procedure with the statements:

```
"SET  GLOBAL  (STARTAL='''A'''"
"SET  GLOBAL  (ENDAL='''J'''"
"RUN  QUERY  EMPREPORT"
"SET  GLOBAL   (STARTAL='''K'''"
"SET  GLOBAL   (ENDAL='''Z'''"
"RUN  QUERY  EMPREPORT"
```

You could also run the EMPREPORT query with the Start Query Management Query (STRQMQRY) CL command from an interactive mode. You are prompted for variables STARTAL and ENDAL before the SELECT statement is performed.

START in Query Management

The START command provides an interface to start an instance of Query Management This command is only valid when issued through the callable interface. The START command allows for values to be specified that indicate how the Query Management session is to be started.

```
►►──START──┬────────────────────────────────────────────────┬──────────────────►◄
           │   ┌─────────────────────────────────────────┐  │
           │   ▼                                         │  │
           └─(─┬─DSQSMODE=──┬─BATCH───────┬─────────────┬┴──┘
               │            └─INTERACTIVE─┘             │
               ├─DSQSCMD──=──procname───────────────────┤
               ├─DSQSRUN──=──procname───────────────────┤
               ├─DSQSNAME=──┬─*SAA─┬────────────────────┤
               │            └─*SYS─┘                    │
               ├─DSQSCNVT=──┬─YES──┬────────────────────┤
               │            ├─NO───┤                    │
               │            └─ONLY─┘                    │
               ├─DSQOAUTH=──┬─authorization-list─┬──────┤
               │            ├─*LIBCRTAUT─────────┤      │
               │            ├─*EXCLUDE───────────┤      │
               │            ├─*ALL───────────────┤      │
               │            ├─*CHANGE────────────┤      │
               │            └─*USE───────────────┘      │
               ├─DSQSDBNM=──┬─*CURRENT─┬────────────────┤
               │            ├─*NONE────┤                │
               │            └─rdbname──┘                │
               ├─DSQUSER=──┬─*CURRENT─┬─────────────────┤
               │           └─username─┘                 │
               ├─DSQPASSWORD=──┬─*NONE────┬─────────────┤
               │               └─password─┘             │
               ├─DSQCMTLV=──┬─NONE─┬────────────────────┤
               │            ├─UR───┤                    │
               │            ├─CS───┤                    │
               │            ├─RS───┤                    │
               │            └─RR───┘                    │
               ├─DSQRDBCNNMTH=──┬─*DUW─┬────────────────┤
               │                └─*RUW─┘                │
               └─DSQCONFIRM=──┬─YES─┬───────────────────┘
                              └─NO──┘

►►──┬──────────────────────────┬──┬──────────────────────────┬─────────────────►◄
    └─keynum──keylen──keywords─┘  └─vallen──values──valtype──┘
```

Extended parameter list of START in Query Management

keynum
Number of keywords that are passed with this call.

keylen Length of each specified keyword.

keywords
Name of the START command keyword that is being set.

START

The following keywords are used on the START command in query management:

- DSQSMODE indicates the mode of Query Management operation when subsequent commands are issued. Valid options are:

  INTERACTIVE

  Allows for the display of screens during Query Management processing. Any reports generated as a result of a RUN QUERY command will be displayed on your screen. Any confirmation messages requiring a response will be displayed on your screen to obtain your reply.

  BATCH

  No screens are displayed during Query Management processing. Any messages requiring a response will result in an error. All other messages will be sent to the job log.

  The keyword value set for the DSQSMODE variable on the START command will override any keyword value set for the DSQSMODE variable via the query command procedure.
- DSQSCMD is the name of a file that is used to start the Query Management session. The SET GLOBAL command is the only type of statement allowed in this procedure. If the DSQSNAME keyword is not specified on the START command, *SAA conventions will be used to find the query command procedure; otherwise, the naming conventions set by the DSQSNAME keyword will be used. If the DSQSCMD keyword is not specified on the START command, Query Management will search for and run a default procedure, DSQSCMDP. The START command does not fail if the default procedure is not found. Other than the parameters to the START command, this is the only place a DSQ data variable can be set.
- DSQSRUN names the Query Management procedure to process after initialization is started. All valid procedure commands are allowed.
- DSQSNAME is the naming convention to be used when processing query commands and the SQL query. See "Naming conventions for Query Objects for Query Management" on page 2 for more information. The keyword value set for DSQSNAME on the START command will override any keyword value set for DSQSNAME in the query command procedure.

  | ***SAA** The SQL naming convention is used. Any qualified query object name specified in commands or
  | query procedures will be of the format `'database.object'`

  ***SYS** Any qualified query object name specified in commands or query procedures will be of the format `'library/object'`
- DSQSCNVT indicates whether Query Management will search for a Query for iSeries definition object if DB2 UDB for iSeries Query Management object is not found. The information contained in the query definition is used to create a temporary Query Management object to be used in a command.

  For example, the command `RUN QUERY MYLIB/QRY1` tells Query Management to search for a query management query object named QRY1. If that object is not found, Query Management will search for a query definition object and use the information contained in it to run a query.

  YES Query Management will search for a Query for iSeries definition object if DB2 UDB for iSeries Query Management object is not found.

  NO Query Management will not search for a Query for iSeries definition object if DB2 UDB for iSeries Query Management object is not found.

  ONLY Query Management will only search for a Query for iSeries definition object.
- DSQOAUTH is the authority given to any object created by Query Management You can specify a default public authority for all objects created during a query instance by setting a value in the DSQOAUTH keyword in the query command procedure or on the START command. The values you can specify are:

  ***LIBCRTAUT**

  The authority for the object is the same as the value specified on the CRTAUT parameter of the library in which the object is being created. If the CRTAUT parameter is changed, the new value will not affect the authority of existing objects.

***CHANGE**
Change authority allows other users to perform all operations on the object except those limited to the owner or controlled by object existence authority and object management authority. A user can change or use the query object in any way, except for deleting it or transferring it to a new owner.

***ALL** All authority allows other users to perform all operations on the object except those limited to the owner or controlled by authorization list management rights. A user can do anything with the query object (including erasing it), except for transferring it to a new owner.

***EXCLUDE**
Exclude authority prevents other users from doing anything with the query object. Unless given specific types of authority, no user except its owner can use the query object.

***USE** Use authority allows other users to run, export, or print the query object, but prevents them from importing it or saving to it.

authorization list name
If you specify the name of an authorization list, its authority is used to control the users ability to use a query object. For more information, see the *Security Guide*

If you do not specify an authority through the query command procedure, other users will have *EXCLUDE access to the query object.

- DSQSDBNM is the remote database to which all SQL operations initiated by Query Management during the query instance are directed. If you do not specify this keyword on either the START or the query command procedure, the connection associated with the query instance is the CURRENT SERVER at the time of the START command. Values that you can specify are:

  ***CURRENT**
  The query instance inherits the connection associated with the CURRENT SERVER. The DSQSDBNM keyword is set to the remote database name (rdbname) of the CURRENT SERVER.

  Note: The Relational Database Directory contains the names of all the remote and local databases that the system is capable of accessing.
  If the local database name is not in the Relational Database Directory, the option will be set to *NONE. The default value is *CURRENT.

  ***NONE**
  The connection will be made to the local database manager. An entry for the local database does not need to exist in the Relational Database Directory.

  rdbname
  This stands for remote database name. This is how a database that can be accessed using Distributed Relational Database Architecture™ (DRDA) is identified. A rdbname can be up to 18 characters in length. It must consist of the uppercase characters (A-Z), numerics (0-9), or underscores (_). The first character must be a letter and an entry for the rdbname must be in the Relational Database Directory.

  The keyword value that is set for DSQSDBNM on the START command overrides any keyword value set for DSQSDBNM by a query command procedure.

- DSQUSER and DSQPASSWORD specify the user identification and password to be used with the remote database if you specify a remote database name with the DSQSDBNM keyword. If you specify either of these keywords, you must specify both of them. If you do not specify these keywords, the default user identification is *CURRENT and the default password is *NONE. Neither the keyword value for DSQUSER nor the keyword value for DSQPASSWORD can be set by a query command procedure. The DSQPASSWORD variable and value are not stored in the global variable pool.
- DSQCMTLV is the keyword used to specify the level of commitment control to be used during the session. The default value is NONE. If you set this keyword to any other value, Query Management will

run all SQL statements under commitment control. If you run the session with a commitment level other than NONE, you may run COMMIT and ROLLBACK SQL statements. The ERASE TABLE CPI command will have the commitment control level associated with the DSQCMTLV value of the query instance.

Note: SAVE DATA AS will always have a commitment control level of NONE.

You can specify the following for DSQCMTLV:

NONE Commitment control is not used. This is the same as *NONE.

UR Only the updated rows are locked until the end of the transaction. This is the same as *CHG.

CS Any row that a cursor is positioned on is locked until the cursor position changes. The updated rows are locked until the end of the transaction. This is the same as *CS.

RS All selected rows are locked until the end of the transaction. The updated rows are locked until the end of the transaction. This is the same as *ALL.

RR All selected rows are locked until the unit of work (UOW) completes. In addition to any exclusive locks, an application process running at isolation level RR (serializable) acquires at least share locks on all the rows it reads. The locking completely isolates the application process from concurrent application processes. This assures that repeated queries within a unit of work give the same results.

The keyword value set for DSQCMTLV on the START command overrides any DSQCMTLV value set by the query command procedure.

- DSQRDBCNNMTH indicates the connection management method to use during the session. The default value is *DUW. You can specify the following for DSQRDBCNNMTH:

  ***DUW** Connections to several relational databases are allowed. This is the default for DSQRDBCNNMTH. Consecutive START or CONNECT commands to additional relational databases do not disconnect previous connections. You can use SET CONNECTION to switch between connections. Read-only connections may result. Consecutive CONNECT commands to the same database fail. You can, however, issue consecutive START commands to the same database.

  ***RUW** Only one connection to a relational database is allowed. Consecutive START or CONNECT commands to additional relational databases disconnect the previous connections before establishing the new one. Consecutive START or CONNECT commands to the same database, do not change the current connection.

  Note: All previous connections are disconnected. For more information see “CONNECT in Query Management” on page 24.

- DSQCONFIRM is the keyword to specify the default value for the DB2 for iSeries Query Management commands that have a CONFIRM parameter. For example, if the CONFIRM keyword is not specified on an ERASE QUERY command, and DSQCONFIRM was set to NO, then no confirmation processing will take place. You can specify the following for DSQCONFIRM:

  YES If no CONFIRM keyword is specified on ERASE, IMPORT, or EXPORT commands then there will be confirmation processing.

  NO In this case, there will be no confirmation processing.

vallen

Length of program storage that is to contain the keyword value.

values

Program storage area that is to contain the keyword value.

valtype

Data type of the storage area that is to contain the keyword value.

Examples of START in Query Management

See Chapter 7, "Callable interface in Query Management" for examples of programs that use the START command.

START query command procedure in Query Management

Use the DSQSCMD keyword on the START command to specify the name of the query command procedure that is run as part of query management initialization. The procedure can also be used to set application-specific user variables.

The query command procedure used on the DSQSCMD parameter is the only place where users can set DSQ variables, other than the START command itself.

Note: The DSQ keywords specified on the START command will always override the same DSQ values set in DSQSCMD. For example:

```
START(DSQSCMD=MYPROC DSQSNAME=*SAA...

          MYPROC
          SET GLOBAL(DSQSNAME=*SYS...
```

would result in DSQSNAME=*SAA.

The following DSQ variables can be set using the query command procedure:

- DSQSMODE
- DSQSRUN
- DSQOAUTH
- DSQSNAME
- DSQCONFIRM
- DSQAPRNM
- DSQSDBNM
- DSQCMTLV
- DSQRDBCNNMTH
- DSQSCNVT

Any other DSQ variables set in the query command procedure are ignored. Defaults are applied to all DSQ variables that are not set in the user-supplied procedure. The possible parameters for the DSQ variables that users can set using the query command procedure are:

DSQSMODE = INTERACTIVE | BATCH
This parameter indicates the mode of query management operation when subsequent commands are issued. Valid options are:

INTERACTIVE
Allows displays to be shown during query management processing. Any reports generated as a result of a RUN QUERY command are shown on your display. Any confirmation messages requiring a response are shown on your display, and you can then reply to the messages.

BATCH
Does not show displays during query management processing. Any messages requiring responses result in errors. All other messages are sent to the job log.

DSQSRUN = *query procedure name*
The query procedure name names the query management procedure to run after initialization is started.

START

If DSQSRUN parameter is not specified on the START command and the DSQSRUN variable is not set in the query command procedure, no initialization procedure is run.

DSQOAUTH = *CHANGE, *EXCLUDE, *USE, *ALL, or authorization list name
If DSQOAUTH is not set by the query command procedure, it defaults to *EXCLUDE.

DSQSNAME = *SYS | *SAA
This parameter specifies the naming convention to be used when processing query management commands.

***SYS** Use the format library/object to specify any qualified names in commands or query management procedures.

***SAA** The SQL naming convention is used. Use the format database.object to specify any qualified names in commands or query management procedures.

If the DSQSNAME parameter is not specified on the START command and the DSQSNAME variable is not set in the query command procedure, the naming convention defaults to *SAA.

DSQCONFIRM = YES | NO
This keyword specifies the default to be taken when CONFIRM is not specified on a command that allows for confirmation processing (IMPORT, EXPORT, and SAVE DATA). If this DSQ variable is not specified in the query command procedure, the default is DSQCONFIRM = YES.

DSQSDBNM = *CURRENT | *NONE | remote database name
This keyword specifies the remote database to which all SQL operations initiated by Query Management during the query instance are directed. If you do not specify this keyword on either the START or the query command procedure, the connection associated with the query instance is the CURRENT SERVER at the time of the START command. Values that you can specify are:

DSQCMTLV = NONE | UR | CS | RS | RR
This keyword is used to specify the level of commitment control to be used during the session. The default value is NONE. If you set this keyword to any other value, Query Management will run all SQL statements under commitment control. If you run the session with a commitment level other than NONE, you may run COMMIT and ROLLBACK SQL statements. The ERASE TABLE CPI command will have the commitment control level associated with the DSQCMTLV value of the query instance.

DSQSCNVT = YES | NO | ONLY
This keyword indicates whether query and form information may be derived from a Query for iSeries definition (QRYDFN) if query management object information is not available. Specifying NO for this keyword causes the command to end with an error if the query or form specified on an EXPORT, PRINT, or RUN command is not found.

Specify YES for this keyword to request query management to attempt to use Query for iSeries information if the query or form specified on an EXPORT, PRINT, or RUN command is not found. If this keyword is not specified on the START command, it defaults to DSQSCNVT = NO.

If you specify ONLY for this keyword, the command ends with an error if a QRYDFN object cannot be found, whether or not there is a query management object of the appropriate type.

Example of the command procedure in Query Management

The following is an example of the contents of the default procedure included with the product:

```
'SET GLOBAL (DSQSMODE=BATCH'
'SET GLOBAL (DSQOAUTH=*EXCLUDE'
'SET GLOBAL (DSQSNAME=*SAA'
'SET GLOBAL (DSQCONFIRM=YES'
```

CL commands in Query Management

The following control language (CL) commands are commonly used when working with Query Management and writing applications to create Query Management reports. For further information on using these CL commands, see the *CL Programming* book.

ANZQRY (Analyze Query) in Query Management

The Analyze Query (ANZQRY) command allows you to analyze a Query for iSeries definition (QRYDFN) object for Query Management conversion problems. Query Management returns diagnostic messages about potential differences between Query for iSeries and Query Management use of query and form information derived from the analyzed QRYDFN object. A completion message shows the highest severity of the problems that are found.

CRTQMFORM (Create Query Manager Form) in Query Management

The Create Query Manager Form (CRTQMFORM) command allows you to create a query management form from a specified source. The form defines how to format DATA (from running a query) when printing or displaying a report. Form information is encoded in source file member records.

CRTQMQRY (Create Query Manager Query) in Query Management

The Create Query Manager Query (CRTQMQRY) command allows you to create a query from a specified source. A query is any single SQL statement that can contain variable substitution values. It can be spread over multiple records in a source file member.

DLTQMFORM (Delete Query Manager Form) in Query Management

The Delete Query Manager Form (DLTQMFORM) command allows you to delete an existing query management form from a library. A generic form name can be used to delete multiple forms from a library or list of libraries.

DLTQMQRY (Delete Query Manager Query) in Query Management

The Delete Query Manager Query (DLTQMQRY) command allows you to delete an existing query management query from a library. A generic query name can be used to delete multiple queries from a library or list of libraries.

RTVQMFORM (Retrieve Query Manager Form) in Query Management

The Retrieve Query Manager Form (RTVQMFORM) command allows you to retrieve encoded form source records from a query management form (QMFORM) object. The source records are placed into a source file member that can be edited.

You can also retrieve form source records from a QRYDFN object when the specified QMFORM does not exist.

RTVQMQRY (Retrieve Query Manager Query) in Query Management

The Retrieve Query Manager Query (RTVQMQRY) command allows you to retrieve an SQL source statement from query management query (QMQRY) object. The source records are placed into a source file member that can be edited.

You can also retrieve query source records from a QRYDFN object when the specified QMQRY object does not exist.

STRQMPRC (Start Query Manager Procedure)

The Start Query Manager Procedure (STRQMPRC) command allows you to run query management procedure that was saved as a member in a source file.

STRQMQRY (Start Query Manager Query)

The Start Query Manager Query (STRQMQRY) command allows you to run an existing query management query. The query runs the SQL statement saved in the query management query. The DATA collected from running an SQL SELECT statement can be displayed, printed, or stored in another database file.

You can also derive the SQL statement from a QRYDFN object when the specified QMQRY object does not exist.

WRKQMFORM (Work with Query Manager Form)

The Work with Query Manager Form (WRKQMFORM) command shows a list of query management forms from a user-specified subset of query management form names. Several query management form-related functions are available from this list.

WRKQMQRY (Work with Query Manager Query)

The Work with Query Manager Query (WRKQMQRY) command shows a list of query management queries from a user-specified subset of query management query names. Several query management query-related functions are available from this list.

Chapter 5. Procedures in Query Management

You may find yourself creating reports over and over again that use the same Query Management commands. If you do, consider processing these steps together by creating a *procedure*. A procedure allows you to process a set of Query Management commands with a single RUN command.

Procedures also allow flexibility in your application. Your application can be written to run a named procedure. At any time, the procedure can be updated or tailored to fit a new situation, without requiring you to change your application program.

Creating procedures in Query Management

A procedure allows you to run a set of Query Management CPI commands with a single RUN command. You can create a procedure for a common program section and then call that procedure with a single command rather than a series of commands.

Keep in mind the following rules when creating a procedure:

- Procedures are source file members.
- Procedures can contain Query Management commands and blank lines. (Blank lines have no effect on the processing of the commands.) They may also optionally contain an H record and a comment V record. The comment record may be used as a text descriptor when importing a procedure.
- Each Query Management command must be in uppercase English letters.
- All commands must be surrounded by apostrophes or quotation marks. If the command contains a quotation mark, the internal quotation marks are represented by two successive apostrophes ('') or quotation marks (“”), as shown in Example 2. See Appendix C, “Use of quotation marks and apostrophes when setting global variables in Query Management” for additional information.
- Procedures can contain a RUN command that runs another procedure or query.
- A single command is limited to 79 characters on a line.
- The width of a procedure line is limited to the source file record width.
- The width of a query command on a procedure line after procedure parsing is done is limited to 256 characters. Procedure parsing involves stripping leading and trailing blanks and reducing apostrophes and quotation marks.

Example of creating procedures in Query Management, 1

```
/*H QM4 01 P 01 E V W E R 01 03 90/3/19 14:27 */
/*V 1001 014 Monthly report */
/* This produces the monthly reports. */
'RUN QUERY A'                /* PAYROLL */
'PRINT REPORT (PRINTER=PRT01'
'RUN QUERY B'                /* ACCTS RECEIVABLE */
'PRINT REPORT (PRINTER=PRT01'
```

The format of the H and V records are described in Chapter 8, “Exported and Imported Objects in Query Management”. The text on the V record, Monthly report, is used on the IMPORT PROC command to set the text description on the source file member.

Example of creating procedures in Query Management, 2

```
'SAVE DATA AS LASTWKDATA (COMMENT='Last weeks'data'
'IMPORT FORM REPT4 FROM MYLIB/FORMS(REPT4) (CONFIRM=YES'
'SET GLOBAL (TBLENAME=MYFILE'
'SET GLOBAL (CMPVAL2=''Joe A. Customer'''
'RUN QUERY REPT4QRY (FORM=REPT4'
'SAVE DATA AS LASTWKDATA'
'PRINT REPORT'
```

Steps for creating a procedure in Query Management

To create a procedure called MYPROC, you would:

1. Edit the source member MYPROC and add information like the following:
   ```
   /*H QM4 01 P 01 E V W E R 01 03 90/3/19 14:27 */
   /*V 1001 014 Monthly report */
   /* This produces the monthly reports. */
   'RUN QUERY A'                /* PAYROLL */
   'PRINT REPORT (PRINTER=PRT01'
   'RUN QUERY B'                /* ACCTS RECEIVABLE */
   'PRINT REPORT (PRINTER=PRT01'
   ```
2. Save the member MYPROC.

The procedure is now ready to run.

The following rules apply when you use procedures:

- You can nest up to 12 procedures inside another but each procedure takes on the characteristics of the one it calls. Therefore, a PRINT command in a procedure which has just run a RUN QUERY command prints the data from the RUN QUERY command.
- Recursion is not allowed. That is, a procedure that contains a RUN PROC of itself is not allowed.
- The query command in a procedure must be delimited by quotation marks (″) or apostrophes, (’).
- Since query management procedures do not have high-level language constructs, the GET command is not functional. A GET command within a procedure sends an informational message to the job log. The message contains the variable name and the value.
- Query Management treats all variable values on a SET command as character strings. Therefore, it is not possible to set an integer variable within a procedure.
- A procedure can optionally contain an H record with a comment V record immediately following.
- A procedure can optionally contain comments. Comments are used to describe the action being taken in the procedure. Comments cannot span multiple lines or be nested within other comments. Comments are delimited with a **/*** at the start of the comment, and an ***/** at the end of the comment.

| On the iSeries server, the processing of each command depends on the mode for the particular instance
| in which the procedure is being processed.

The query management procedure is a member in a source physical file. Query management allows a specific member to be identified on the RUN PROC, IMPORT PROC, EXPORT PROC, PRINT PROC, and ERASE PROC commands. This is done by allowing members to be given as part of the query object name. The member name must follow the query object name and be delimited by a parenthesis with no intervening blanks.

If you do not specify a member as part of the object name, it is always assumed to be the first member of the file. If you issue an ERASE PROC and more than one member exists only the first member will be deleted. If you do not specify a member, and it is created as part of the IMPORT PROC processing, it is given the name of the procedure file.

User interaction in procedures in Query Management

Interaction with a query user depends on the interactive state of Query Management. This state is controlled by the startup parameter DSQSMODE on the START command (see “START in Query

Management” on page 45). If you allow the interactive state, there are further considerations when using a procedure. For example, specifying CONFIRM=YES on the ERASE command or DISPLAY=YES on the IMPORT command could cause the procedure to fail.

When a procedure that contains several queries is run, you will see a formatted report display as each query processes. You can then page the report. An exit from the report returns control to the procedure and causes the next statement to process.

Procedure interaction in Query Management

Refer to command descriptions in Chapter 4, “Commands in Query Management” to understand what will happen during the processing of each command depending on the mode the procedure is being processed in.

- Procedures are allowed to be called from inside another procedure; this practice is called nesting. Procedures located inside other procedures will use the parameters specified by the first procedure. Therefore, a PRINT command processed in a procedure which has just run a RUN QUERY command will print the data from the RUN QUERY command.
- A nesting level of 12 is allowed by Query Management.
- Recursion is not allowed. For example, PROCEDURE A cannot contain the command RUN PROC A.
- The GET command is not functional within a procedure. A GET command within a procedure will result in an informational message sent to the job log that contains the variable name and the value.
- Query Management treats all variable values on a SET command as character strings.

Procedure Objects in Query Management

The query procedure is a source physical file. Query Management allows a specific member to be specified on the RUN PROC, IMPORT PROC, EXPORT PROC, PRINT PROC, and ERASE PROC commands by allowing members to be given as part of the query object name. The member name must follow the query object name and be delimited by a parenthesis with no intervening blanks. The following examples show how each of the commands can be changed to point Query Management to a specific member:

```
RUN PROC MYLIB/MYPROCS(MYMEMBER)
PRINT PROC MYLIB/MYPROCS(MYMEMBER)
IMPORT PROC MYPROCS(MYMEMBER) FROM QQMQRYSRC
EXPORT PROC MYLIB/MYPROCS(MYMEMBER) TO QQMQRYSRC(MYMEMBER)
```

If a member is not specified as part of the object name, it is always assumed to be the first member of the file. If an ERASE PROC is issued and more than one member exists, only the first member will be deleted. If a member is not specified and is to be created as part of the IMPORT PROC processing, it will be created with the same name as the PROC file name.

Query Management will process the entire source file when running a RUN PROC or PRINT PROC command. If a PROC file is created on import or export, it will be created with a data length of 79 characters. Truncation occurs on any import or export from a file with a longer record width.

A query procedure can be run by:

1. Issuing the STRQMPRC CL command
2. Doing a RUN PROC query command through the query management callable programming interface
3. Doing a RUN PROC on the DB2 UDB for iSeries Query Manager query statement pop-up
4. Specifying a procedure name for the DSQSCMD keyword on the query management callable programming interface START command

Handling procedure errors in Query Management

Whenever an error with severity of FAILURE occurs, the processing of the procedure will be stopped and the completion code of the procedure will reflect the error. All messages encountered during the processing of the procedure will be queued to the job log and a summary message will be returned in the communications area.

Categories of procedure errors in Query Management

The following error categories are possible in query management procedures:

- Command not allowed in query management procedure.
- Command in query management procedure not valid.
- String in query management procedure not valid.
- Recursion not allowed in query management procedure.
- Maximum procedure nesting level exceeded. A nesting level of 15 is allowed.

Chapter 6. Report Forms in Query Management

This chapter explains how to create a form and describes query management reporting capabilities. You produce reports by formatting the results of a query using the formatting information that is specified in the FORM.

How applications can use the FORM in Query Management

An application can create or alter a FORM by directly changing or creating the exported FORM.

You may use an application to export an existing FORM from Query Management, change it, import the FORM, and then format a report. But a FORM does not have to be exported every time. An application can access and change an existing exported FORM, and then import it into Query Management for reporting.

You can also import just part of a FORM: only the header (H) record followed by the T and R records for column information, for example. The rest of the FORM fields can be filled in by defaults.

For information on how to create a form, see the next section, Creating forms in Query Management.

Creating forms in Query Management

You can create reports by formatting the results of a query using the information that is specified in a form. An application can create or alter a form by directly changing or creating the exported form.

You can use an application to export an existing form from query management, change it, import the form, and then format a report. You do not have to export the form every time. An application can access and change an existing exported form, and then import it into query management for reporting.

You can also import a form from a source that allows certain form fields to be filled by default. It is possible to use only the header (H) record followed by one T and one R record for each column that is formatted. The remaining form fields are filled in by query management default values. This allows you to create an entire form without having to type values for all the form attributes. Query Management forms can also be created using the DB2 UDB for iSeries Query Manager product.

Creating a default form in Query Management

1. Create a template for generating an external form object by creating a source member DEFAULT in the source file TESTFORM in library MYLIB1 with a record length of 162 characters. To do this,

   At an OS/400 command line, type:

   ```
   CRTSRCPF MYLIB1/TESTFORM RCDLEN(162) MBR(DEFAULT)
   ```

   Press Enter.
2. Edit the member named DEFAULT and add the following information.

   ```
   H QM4 05 F 03 E V W E R 01 03 90/12/31 09:21
   T 1110 001 000
   R
   E
   ```

   When the default form is exported later, the date and time are made current.
3. Save the member DEFAULT.
4. To create a default form object in MYLIB1, use the Create Query Management Form (CRTQMFORM) command to import the source file member you just created. At an OS/400® command line, type:

   ```
   CRTQMFORM QMFORM(MYLIB1/DEFAULT)
   SRCFILE(MYLIB1/TESTFORM) SRCMBR(DEFAULT)
   ```

Press Enter.

5. Use the Retrieve Query Management Form (RTVQMFORM) CL command to export the query management form created as a result of the CRTQMFORM command. The easiest way to do this is:
   a. Press F9 to retrieve the previous command.
   b. Replace CRT with RTV.

      The entire command is then:

      ```
      RTVQMFORM QMFORM(MYLIB1/DEFAULT)
      SRCFILE(MYLIB1/TESTFORM) SRCMBR(DEFAULT)
      ```
   c. Press Enter.

The member named DEFAULT now contains a complete form with default values for all form attributes except those set at run time. You can then edit the member DEFAULT to change field attributes and add more columns.

Defaults are provided for information that is not specified. Some defaults are provided when the form is imported. Other defaults, such as Data type and Column heading, are provided at run time and depend on the resulting data of the processed query.

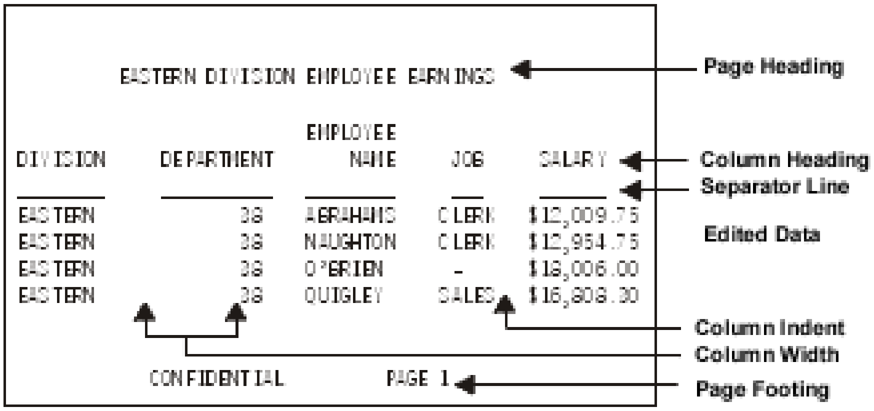
Keywords encoded into the form should be in uppercase English. Text fields (headings, footings, and final text) can be in upper and lowercase letters.

The form object fields are commonly grouped by the following functional categories:

- Break
- Column
- Final
- Options
- Page

Formatting terminology in Query Management

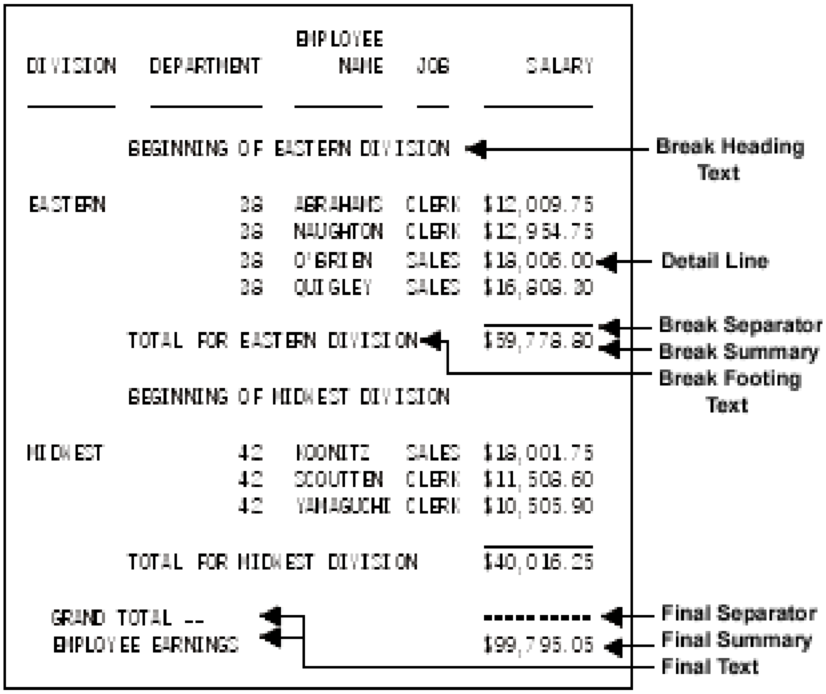
In order to understand all of the options available in the FORM, the following two figures, Figure 8 and Figure 9, show you the effect of some of the FORM options on a formatted report.



```
        EASTERN DIVISION EMPLOYEE EARNINGS

                              EMPLOYEE
DIVISION     DEPARTMENT           NAME      JOB      SALARY
________     __________     __________     ___     ______
EASTERN              38     ABRAHAMS     CLERK   $12,009.75
EASTERN              38     NAUGHTON     CLERK   $12,954.75
EASTERN              38     O'BRIEN        -     $18,006.00
EASTERN              38     QUIGLEY      SALES   $16,808.30

          CONFIDENTIAL          PAGE 1
```

Figure 8. Basic Parts of a Report

"Edited Data" is information from the database that displays according to the relevant edit code.



```
                           EMPLOYEE
DIVISION   DEPARTMENT          NAME   JOB        SALARY
________   __________   _________   ___   __________

        BEGINNING OF EASTERN DIVISION

EASTERN              38   ABRAHAMS  CLERK  $12,009.75
                     38   NAUGHTON  CLERK  $12,954.75
                     38   O'BRIEN   SALES  $18,006.00
                     38   QUIGLEY   SALES  $16,808.30
                                           __________
        TOTAL FOR EASTERN DIVISION         $59,778.80

        BEGINNING OF MIDWEST DIVISION

MIDWEST              42   KOONITZ   SALES  $18,001.75
                     42   SCOUTTEN  CLERK  $11,508.60
                     42   YAMAGUCHI CLERK  $10,505.90
                                           __________
        TOTAL FOR MIDWEST DIVISION         $40,016.25

  GRAND TOTAL --                           ----------
  EMPLOYEE EARNINGS                        $99,795.05
```

Figure 9. Basic Parts of a Report with One Level of Control Break

The FORM object contains fields that describe the report. Query Management supports up to 255 columns of information. Query Management has a limit of 32 KB of available data from the database for any one row. Defaults are provided for all the fields except “Usage.” Defaults depend on the resulting data of the processed query.

Keywords used in the FORM must be in uppercase English. Text fields (headings, footings, and final text) can be in uppercase and lowercase letters.

DBCS data in Query Management

See Appendix A, “DBCS data in Query Management” for additional information about using double-byte character set (DBCS) data in a FORM.

COLUMN fields in Query Management

The following topics are covered in COLUMN fields in Query Management:

“Data type for column in Query Management” on page 60

“Headings for column in Query Management” on page 60

“Usage for column in Query Management” on page 61

“Indent for column in Query Management” on page 62

“Width for column in Query Management” on page 63

“Datatype for column in Query Management” on page 63

COLUMN Fields

“Edit codes for column in Query Management” on page 63

“Seq for column in Query Management” on page 65

“Run-time defaults for column in Query Management” on page 65

Data type for column in Query Management

This field represents the data type for a column in the report. Possible values are NUMERIC, CHAR, GRAPHIC, and DATE/TIME. OS/400 does not support GRAPHIC and DATE/TIME. Each column in DB2 UDB for iSeries Query Management report is described by a set of field values. The values for the *n*th column in the FORM are applied to the *n*th column selected by the query with which it is used. The following sections describe the fields available for use in defining the Column fields. See Appendix A, “DBCS data in Query Management” for additional information about using double-byte character set (DBCS) data in Column fields.

Table 3 shows the defaults and possible values for the attributes on the *Column* field.

Table 3. Default Values for Column Fields

Attribute	Default	Possible Values
Column heading	Column heading in table	1 - 62 characters with up to 8 underscores
Usage	—	AVG, MIN, MAX, SUM, COUNT, BREAK1 - BREAK6, AVERAGE, MINIMUM, MAXIMUM, OMIT
Indent	2	0 to 999
Width	Depends on data type used.	1 to 32,767 SBCS
Datatype	Data type of field in table	CHAR, NUMERIC
Edit code– numeric	Edit code of field in table	E, D, I, J, K, L, P
Edit code– character	C	C, CW, CT
Seq	Column number	1 to 999
Edit code– date, time	Run time	TDY, TDM, TDD, TDYA, TOMA, TDDA, TTS, TTC, TTA, TTAN, TTU, TSI

Headings for column in Query Management

This field represents the heading for a column in the report.

A heading can be up to 62 characters long.

You can embed underscore characters in the heading and use them to indicate a new line for multiple-line headings. Query Management processes a maximum of eight (8) underscores in a heading. Leading and trailing underscores produce blank segments before and after the column headings.

For example, a column heading of “AMOUNT_LAST_INCREASE” results in the following 3-line column heading:

```
 AMOUNT
  LAST
INCREASE
```

Consecutive underscores (in any position) will introduce blank lines.

Note that the underscore rule prevents you from seeing an underscore character in a column heading. The only exception to this rule is when more than eight (8) underscores appear, in which case the extra underscores print as part of the last line of the heading.

Whenever you specify multiple-line headings, Query Management automatically centers the smaller lines within the space of the longest line. Headings for character data are automatically left-justified and headings for numeric data are automatically right-justified. Data justification takes place within the width of the column. The width specification must reflect the length of the longest segment of this field.

If the number of characters in this field is greater than the number of characters specified in width, then the field truncates to the width specified for the column.

If you do not specify a column heading, Query Management provides a run-time default. You cannot cause a column to be shown without a heading by importing a form with a blank heading unless the database definition for the column indicates it should not have a column heading.

Usage for column in Query Management

This field determines use of the column in the detail line of the formatted report result.

There can be only one *usage* specified for each column. If you want a column to have more than one usage, you must select the column multiple times in the query and define a usage code for each column in the FORM.

The usage options are:

[blank]
Column to be included in the report.

OMIT
Column to be excluded from the report.

AVERAGE, COUNT, FIRST, LAST, MAXIMUM, MINIMUM, SUM
These keywords name aggregating usages that summarize the data in a column. The result of the usage is given at a break or final summary.

Usage Code	Definition
AVERAGE (or AVG)	the average of the values in the column.
COUNT	the count of the non-null values in the column.
FIRST	the first value in the column.
LAST	the last value in the column.
MAXIMUM (or MAX)	the maximum value in the column.
MINIMUM (or MIN)	the minimum value in the column.
SUM	the sum of the non-null values in the column.

AVERAGE and SUM work only on numeric data; COUNT, FIRST, LAST, MAXIMUM, and MINIMUM work with character data as well as with numeric data.

When you compare characters using MAXIMUM and MINIMUM, the shorter string is padded with blanks and the strings are compared based on the internal binary codes. For example, the character string “ab” is greater than the character string “aaa”. There is no special processing on a character by character basis. Be aware when you use MAXIMUM and MINIMUM on applications are to be portable from one system to another that the collating sequences for the different machines may not be the same. Since EBCDIC and ASCII do not collate characters identically, using MAXIMUM or MINIMUM on different systems may produce different results.

The following rules apply to the aggregating usages AVERAGE, COUNT, FIRST, LAST, MAXIMUM, MINIMUM, and SUM.

COLUMN Fields

1. If aggregation overflow occurs, the value in the field is represented by “>>>>” for the width of the column.
2. If the aggregation cannot be displayed due to the column width being too small, the value in the field is represented by “*****” for the width of the column.

BREAK1

BREAK1 is the value used to specify a column as the first level, or highest, *control break.* A control break is the break point where the column value changes.

For example, if a set of rows of employees is ordered by department number and job title, a BREAK1 can be used to total the salaries of all the employees in the department, and a BREAK2 can be used to total the salaries by job title within department. Each time a row with a different job title is read, a BREAK2 generates and displays the appropriate data and totals. Each time a row with a different department number is read, a BREAK2 and BREAK1 generate, and both sets of appropriate data display.

Before each break summary displays, a line is placed in the report consisting of a row of hyphens (“–”) under any displayed column with an aggregation usage. You can suppress this line via an option in the “Options” part of the FORM. A blank line is normally placed after each set of break data.

The aggregation usages (AVERAGE, COUNT, FIRST, LAST, MAXIMUM, MINIMUM, SUM) may be used at control breaks. For example, summary data appears as subtotals of all columns with a usage of SUM, or an average of the columns with a usage of AVERAGE.

The data printed as part of the break is determined by the break definition, described later in this section.

The break level numbers are not required to be consecutive. In this respect, control break numbers are not absolute; you could specify control breaks 2, 4, and 6, without specifying 1, 3, and 5. However, control break text assignments continue to be absolute; text for control break 2 is always associated with control break number 2—not with the second control break.

You may assign multiple columns to the same BREAKn value. When this occurs and Query Management needs to determine control breaks, Query Management considers all columns having the same control break level as a single concatenated column. This is particularly useful when LAST_NAME and FIRST_NAME are stored as separate columns. Likewise, it is needed when MONTH, DAY, and YEAR are each separate columns.

When using a control break, the data in the column should be ordered. For the data to be in order, the SELECT that produces the report must use ORDER BY.

There is no automatic reordering of columns due to break specifications, but break text is usually displayed to the left of any summary columns. Therefore, IBM recommends using the “Seq” value to display the break columns to the left, and the aggregated columns to the right, on the report.

BREAK2

BREAK2 is the usage value used to specify a column as the second level control break. A BREAK2 automatically generates whenever either the column(s) upon which it is defined changes, or when there is a BREAK1 generated. Note that a BREAK1 causes a BREAK2, but a BREAK2 does not cause a BREAK1.

BREAK3 through BREAK6

BREAK3 through BREAK6 name control columns for breaks at levels 3 through 6.

Indent for column in Query Management

This field represents the relative location of the column within a row. Its units are the number of blank characters between the column and either:

- The right edge of the previous column.
- The left edge of the screen or paper.

You may set Indent to *n*, where 0 <= *n* <= 999. In the default format, Query Management initializes Indent to 2.

Width for column in Query Management

This is the column's output width. It specifies how many character spaces to reserve for displaying the column heading and data. Names that are wider than Width are truncated. Query Management defines the width field as numerics only. The maximum Width is 32767 single-byte characters. If the length of the value to display exceeds the width of the column, the value is either replaced with a row of asterisks (****) if it is numeric data, or truncated at the right if it is character data. The desired result may be obtained by changing Width and displaying the report again. Date, time, or timestamp data is treated like character data in a formatted report. The following also apply:

- The column heading and column detail are left aligned in the column.
- If the width of the column specified in the form is less than the formatted width of the date, time, or time stamp; the date, time, or time stamp is truncated on the right.
- If the width of the column specified in the form is greater than the formatted width of the date, time, or time stamp; the date, time, or time stamp is padded on the right with blanks.
- A null date, time, or time stamp field is formatted as a left-aligned dash (-).

If you do not specify a width value, Query Management provides a run-time default.

Datatype for column in Query Management

This field represents the type of data that is contained in the corresponding column in the queried table. The Datatype options are the following:

CHARACTER
The data in the column in the table is character.

NUMERIC
The data in the column in the table is numeric. It can be binary, packed, zoned, or floating point.

DATE, TIME, TIMEST
The data in the column is date, time, or timestamp.

GRAPHIC
The data in the column is DBCS-graphic (see Appendix A, "DBCS data in Query Management").

Edit codes for column in Query Management

Edit codes are used to format character and numeric data for display. Table 4 shows the edit codes for **date** data.

Note: The x's show where to specify the date punctuation character value. This can be any special character, including a blank, but cannot be a letter or a number. For example, a dash (-) can be used.

Table 4. Query Management CPI Date Edit Codes

Edit Code	Format	Example
TDYx	YYYYxMMxDD	TDY/ ==> 1987/01/31
TDMx	MMxDDxYYYY	TDM- ==> 01-31-1987
TDDx	DDxMMxYYYY	TDD ==> 31 01 1987
TDYAx	YYxMMxDD	TDYA/ ==> 87/01/31
TDMAx	MMxDDxYY	TDMA- ==> 01-31-87
TDDAx	DDxMMxYY	TDDA. ==> 31.01.87

COLUMN Fields

Table 5 shows the edit codes for **time** data.

Table 5. Query Management CPI Time Edit Codes

Edit Code	Format	Note	Example
TTSx	HHxMMxSS	Includes seconds	TTS. ==> 13.42.35
TTCx	HHxMMxSS	Includes seconds, 12 hr.	TTC: ==> 01:42:35
TTAx	HHxMM	Abbreviated (no seconds)	TTA, ==> 13,42
TTAN	HHMM	Abbreviated, no delimiter	TTAN ==> 1342
TTUx	HHxMM AM or PM	USA style	TTU: ==> 01:42 PM

Table 6 shows the only edit code for TIMESTAMP data. To display all of a timestamp format, except microseconds, the width field must be at least 19. If the width is less than 26, the trailing digits are truncated.

Table 6. Query Management CPI Timestamp Edit Code

Edit Code	Format	Example
TSI	yyyy-mm-dd-hh.mm.ss.nnnnnn	1987-01-21-13.42.19.123456

Below are the edit codes for **character** data. See Appendix A, “DBCS data in Query Management” for DBCS-graphic edit codes.

C makes no change in the display of a value. If the value cannot fit onto one line in the column, Query Management truncates the text according to the width of the column. C is the default for character data.

CW

makes no change in the display of a value, but if the value cannot fit on one line in the column, Query Management wraps the text according to the width of the column. That is, instead of truncating the data at the end of the column, Query Management puts as much data as possible on one line in the column and then continues the data on the next line in the column.

The CW edit code can be used on columns of mixed DBCS and single-byte character data.

CT

makes no change in the display of a value. But if the value cannot fit onto one line in the column, Query Management wraps the column according to the text in the column. That is, instead of truncating the data at the end of the column, Query Management fits as much data as possible on a line, interrupts the line when it finds a blank and continues the data on the next line. If a string of data is too long to fit into the column and does not contain a blank, Query Management wraps the data by width until the point where it finds a blank and can therefore continue wrapping by text.

The CT edit code can be used on columns of mixed DBCS and single-byte characters. Query Management interrupts the line when it finds a single-byte or double-byte blank.

Below are the edit codes for **numeric** data.

E displays numbers in scientific notation. For example, the number -1234.56789 displays as -1.234E+03. As many digits as can display are placed in the report, up to a maximum of 15. One space is always reserved for a leading sign, although it does not display for positive numbers. There is always a sign and at least two digits after the E. Up to three digits will display.

D, I, J, K, L, and P

display numbers in decimal notation with different combinations of leading zeros, negative symbols, thousands separators, currency symbols, and percent signs. Examples are in the following table.

Each code may be followed by a number (from 0 to 31) that tells how many places to allow after the decimal point. If no number is specified, zero places after the decimal point are assumed. Numbers that have more decimal places than fit into the allowed space are rounded; numbers with fewer decimal places are padded with zeros.

Figure 10 shows how the numeric edit codes format the number -1234567.885. The example assumes that

- Width is 15.
- The value of the decimal character is a period (.).
- The value of the thousand separator is a comma (,).
- Normal rounding is used (1-4 round down, 5-9 round up).
- The currency symbol for D2 is the dollar sign ($) with left position.
- The negative indicator is a minus sign (-). There is no trailing negative indicator.

  The above parameters may be established in a system or user profile, depending on the operating system. If you move your application to another environment, you should ensure that the parameters are equivalent across the environments.

```
Edit  Lead   Negative  Thousands   Currency  Percent  Display of
Code  Zeros  Sign      Separators  Symbol    Sign     -1234567.885
----  -----  --------  ----------  --------  -------  ------------

E     No     Yes       No          No        No       -1.23456789E+06

D2    No     Yes       Yes         Yes       No         -$1,234,567.89

I3    Yes    Yes       No          No        No       -0001234567.885

J2    Yes    No        No          No        No       000001234567.89

K3    No     Yes       Yes         No        No         -1,234,567.885

L2    No     Yes       No          No        No            -1234567.89

P2    No     Yes       Yes         No        Yes        -1,234,567.89%
```

Figure 10. Use of Edit Codes

If you do not specify an edit value, Query Management provides a run-time default.

Seq for column in Query Management

You can specify “Seq” to order the columns in the generated report.

The following rules apply to evaluating “Seq” values:

- Defaults to *n* for the *n*th column in the FORM.
- Any number from 1 to 999.
- Numbers need not be consecutive.
- Columns with the same “Seq” number appear in the report in the same order that they appear in the form.

Run-time defaults for column in Query Management

Query Management uses system-provided defaults for Datatype, Column Heading, Edit, and Width values when columns of data extracted by running a query need to be formatted for a report. This condition occurs when the following occur:

- You did not specify a form or you specified *SYSDFT to refer to the default form for the extracted data.
- The specified form does not contain the information needed to format the report or the form was imported with warnings about blank values or missing column table fields.

COLUMN Fields

Note: If you use the SAVE DATA AS command with a query that contains date or time columns, the default information associated with the date and time fields is the information associated with the user, **not** the default information associated with the original table column that was queried.

Heading defaults for column in Query Management

You can establish column heading defaults for file data when you define a field to the system. Use field names unless other text is specified when you use interactive data definition utility (IDDU).

When a default report is built using a form that does not contain explicit column headings, the column headings from the database are used. If:

- There are no column headings in the database, the selected name becomes the default column heading.
- The query selects the column by the column name, the column name is the default column heading.
- The query selects the column by the system column name, the system column name is the default column heading.

If the selected column has an explicit column heading that is the same as the system column name, the column name is used as the column heading.

You can define files that do not cause column headings to be defaulted. Column heading defaults for calculated data and for file fields without established defaults are taken from the set of unique column names manufactured at run time for the selected columns. These are the column names that are used to create a new file for a SAVE DATA request.

To manufacture these names, Query Management processes the selected columns in order, from first to last. The unique name for the *n*th selected column is created in the following way:

1. The column name from the file (table) definition (or SEL for a calculated field) is used as the created name if it does not match any previously created name.
2. If the matched name is too long to add to the next available number, that number is added to COL to create the name.
3. If the matched name is not too long to add to the next available number, that number is added to the column name to create the name.

Note: The first number added to a column name or COL to make the name unique is 1, the next number added is 2, and so on.

The following example shows the unique names created for a particular SELECT list.

```
  SELECT 7*WEEKS, SALARY, SALARY, SALARY/7*WEEKS, MAXBENEFIT, MAXBENEFIT
            |        |       |          |              |           |
            |        |       |          |              |           |
            |        |       |          |              |           |
           SEL    SALARY  SALARY1      SEL2       MAXBENEFIT     COL3
```

Duplicate column names are also made unique by appending a number. If the unique name would result in a column name that exceeds thirty characters, the last five characters are truncated and replaced with a five character number. For example:

```
SELECT ThisIsABigLongColumnNameExampl, ThisIsABigLongColumnNameExampl
```

results in the following column names:

```
SELECT ThisIsABigLongColumnNameExampl, ThisIsABigLongColumnNameE00001
```

If some of the columns have column heading defaults that were previously defined, the column headings in a formatted report are not necessarily unique. This can be true for the *SYSDFT form as well as a form specified by name.

Editing defaults for column in Query Management

Editing defaults are intended to be the same as those used by other data-displaying products on the system, such as Query for iSeries. Character data is unchanged or truncated (such as Query Management CPI edit code C). Scientific notation is used for floating data (such as Query Management CPI edit code E). Editing defaults usually have no corresponding Query Management CPI representation for numeric fields, and can include edit words, edit descriptions, and RPG edit codes.

File (table) data defaults are established when a field (column) is defined to the system. System-level values determine the editing for calculated data. These values come from a translatable message and are used for building a default edit description whenever one is needed.

Changeable system-level values can also change the editing applied to numeric data, for example, using the QDECFMT system value to change the editing applied for RPG edit code J.

Dates default to the closest SQL format according to the following rules:

Table 7. Default SQL formats

OS/400 format	**SQL**
*MDY	MM/DD/YYYY (USA)
*YMD	YYYY-MM-DD (ISO)
*DMY	DD.MM.YYYY (EUR)
*JUL	YYYY-MM-DD (ISO)

The date separator used is the SQL-default format shown above regardless of the job-date separator.

Times use the job time separator if the database connection is homogeneous. If the connection is heterogeneous and the job-time separator is a colon, a colon is used; otherwise, the time separator is a period.

Width defaults for column in Query Management

Width defaults are intended to provide room for everything that has to be shown in the column. For a particular column, the default is the maximum of the following:

- The length of the longest segment of the column heading.
- The edited data width (after adjustment of the raw data length by 3 if SUM Usage aggregation values have to be shown in the column).
- A length of 9 if COUNT Usage aggregation values have to be shown in the column.

Column names used as defaults for column headings are unique. When necessary to make the default names unique, Query Management adds a number as a suffix to the end of the column name. When this happens, the first occurrence of the column name remains unchanged. The number is added to all other occurrences of the name. Numbers are assigned sequentially across all column names. For example:

```
SELECT ID, ID, DEPT, JOB, ID, DEPT
```

results in default heading of

```
        ID  ID1  DEPT  JOB  ID2  DEPT3
```

When the default column names cannot be made unique, Query Management assigns COL*n* for column names. For example:

```
SELECT ABCDEFGHIJKLMNOPQR, ABC, ABCDEFGHIJKLMNOPQR
```

results in default headings of

```
        ABCDEFGHIJKLMNOPQR  ABC  COL1
```

PAGE Fields in Query Management

The following fields are used to specify headings and footings on a report. Table 8 shows the defaults and possible values for the attributes on the Page fields.

Table 8. Default Values for Page Fields

Attribute	Default	Possible Values
Blank lines before heading	0	0 to 999
Blank lines before footing	2	0 to 999
Blank lines after heading	2	0 to 999
Blank lines after footing	0	0 to 999
Alignment on heading text	CENTER	LEFT, CENTER, RIGHT
Alignment on footing text	CENTER	LEFT, CENTER, RIGHT

Blank lines before heading/footing for page fields in Query Management

These fields indicate the number of blank lines before the page heading or page footing and must be specified as numbers. An acceptable value is any number from zero to 999. The default value for the page heading is *zero (0)* and for the page footing is *two (2)*. A page eject always precedes the heading on each page. The “Blank lines before heading” field controls the number of blank lines between the heading and the top of the page. The “Blank lines before footing” field controls the number of blank lines between the report body and the first footing line.

Blank lines are included in the count of the number of lines printed on the page.

Blank lines after heading/footing for page fields in Query Management

These fields indicate the number of blank lines after the page heading or page footing. The fields are defined in Query Management as numerics only. An acceptable value is any number from zero to 999. The default value for the page heading is *two (2)* and for the page footing is *zero (0)*. The “Blank lines after heading” field controls the number of blank lines between the last heading line and the report body. The “Blank lines after footing” controls:

- The number of blank lines between the last footing line and the end of the page.
- The last footing line and the line containing the “Date and time” and/or “Page number”.

Blank lines are included in the count of the number of lines printed on the page.

“Blank lines after footing” takes precedence over “Blank lines before footing”; on a report page that has extra space left after the body of the report, extra blank lines are inserted so the “Blank lines after footing” value is the correct number of lines.

Heading text lines for page fields in Query Management

The heading text lines contain a maximum of 999 heading text lines.

Line value of heading text lines for page fields in Query Management

The Line value denotes the line positioning of the heading text lines.

For example, if you include text lines for line numbers 1 and 5, the text is displayed in the report as:

```
Text for line 1
[blank line]
[blank line]
[blank line]
Text for line 5
```

If you specify text line number 1 and then repeat the specification later, the last occurrence of line number one prevails over the first occurrence.

Align field of heading text for page fields in Query Management

The Align field controls the positioning of the page heading text within the report line. Acceptable values are:

RIGHT
Right-justify the text.

LEFT
Left-justify the text.

CENTER
Center the text.

The default value for headings is *center*.

Heading text field for page fields in Query Management

These fields allow you to enter text that appears as the page heading in the report. You can enter a maximum of 55 characters for each text line.

If the text line *n* is the biggest line with nonblank text, then *n* indicates how many lines are to be formatted. This is true even though it could result in some of the formatted lines being completely blank.

Page headings may contain four types of special variables. All variables must be coded with a leading ampersand (&) to identify them within the heading text.

&col
where *col* is a column name, system column name, or a column number. *&col* is assigned the first value on the page for the specified column.

Variable &col formats according to the edit code specification except when column wrapping is specified. Then, the edit code specification is ignored. If necessary, the edited data is truncated to the width of the column from which it was taken.

The column name is the name of the column returned from SQL. It can only be specified in uppercase. To determine the column name of a calculated column or a duplicate column, see “Heading defaults for column in Query Management” on page 66.

You must refer to the column by column number if the column name is DATE, TIME, PAGE, or the column name contains a $, #, @, or a quote.

The column number is determined by the order the columns are returned from the SQL statement query or prompted query. The column number is not determined by the order in the Sequence field. *&col* is set at each page break.

&DATE
where Query Management replaces the value with the current date.

&TIME
where Query Management replaces the value with the current time. For OS/2, the current time is in the format based on the edit code in the active profile or the system country or region code.

&PAGE
where Query Management replaces the value with the current page number. The format of the page number is a four digit number ranging from 1 to 9999 with leading zeros suppressed. After 9999, the counter wraps to 0 and continues to increase for subsequent pages *without* leading zero suppression.

PAGE Fields

When the report prints, the page heading appears at the top of each page, formatted according to the format specification. The variable *&col* formats according to the edit code specification, except when column wrapping is specified. If column wrapping is specified for the column, it is ignored when the data formats into the text.

All variables are resolved when the report is created.

Footing text lines for page fields in Query Management

The footing text contains a maximum of 999 footing text lines.

Line value of footing text lines for page fields in Query Management

The Line value denotes the line positioning of the footing text lines.

For example, if you include text lines for line numbers 1 and 5, the text is displayed in the report as:

```
Text for line 1
[blank line]
[blank line]
[blank line]
Text for line 5
```

If you specify text line number 1 and then repeat the specification later, the last occurrence of line number one prevails over the first occurrence.

Align field of the footing text for page fields in Query Management

The Align field controls the positioning of the page footing text within the report line. Acceptable values are:

RIGHT
Right-justify the text.

LEFT Left-justify the text.

CENTER
Center the text.

The default value for footings is *center*.

Footing text field for page fields in Query Management

These fields allow you to enter text for the page footing that is to appear in the report. You can enter a maximum of 55 characters for each text line. You can use the variables described above for the heading text field. When the report formats, appropriate values are substituted for the variables. The variable &col is assigned the last value on the page for the specified column. When the report prints, the page footing appears at the bottom of each page, formatted according to the format specification. The variable *&col:* formats according to the edit code specification, except when column wrapping is specified. If you specify column wrapping, it is ignored when the data formats into the text. The formatted page footing appears once at the bottom of the displayed report.

If the text line *n* is the biggest line with nonblank text, then *n* indicates how many lines are to be formatted. This is true even though it could result in some of the formatted lines being completely blank.

FINAL TEXT fields in Query Management

The following fields are used to specify the final text that appears on a report. Table 9 shows the defaults and possible values for the attributes on the Final text fields.

Table 9. Default Values for Final Text Fields

Attribute	Default	Possible Values
New page	NO	YES, NO

Table 9. Default Values for Final Text Fields (continued)

Attribute	Default	Possible Values
Put final summary at line	1	1 to 999 or NONE
Blank lines before text	0	0 to 999 or BOTTOM
Alignment for final text	RIGHT	LEFT, CENTER, RIGHT

New page for final text fields in Query Management

This field indicates whether the subsequent part of the report (final text) must begin on a separate page when printed. The default is *no*. When you specify Yes for New Page, the final text formats on a new page.

Put final summary at line for final text fields in Query Management

This field indicates whether the final summary should be in formatted form and where to vertically position it in the report final text. Acceptable values are the numbers 1 to 999, or the word NONE, where NONE indicates there is no presentation of final summary data. The default value is *one (1)*.

A value of one to 12 indicates the relative line number within the final text at which the summary data must format. This is strictly vertical placement. For horizontal placement in the line, the final summary is always formatted under the columns being summarized. If there are no column widths with aggregating usages, this value is ignored.

Blank lines before text for final text fields in Query Management

This field indicates the number of blank lines between the body of the report and the first line of final text. An acceptable value is any number from zero to 999. The default value is *zero*. You may also specify BOTTOM, which positions the final text at the bottom of the printed page. BOTTOM causes insertion of a number of blank lines, in order to position the final text immediately before the page footing text specification on the page. If there is not enough space for the final text on the current page, it is placed at the bottom of the next page.

Line field for final text lines in Query Management

This field indicates the line positioning of the final text lines.

For example, if you include text lines for line numbers 1 and 5, the text is displayed in the report as:

```
Text for line 1
[blank line]
[blank line]
[blank line]
Text for line 5
```

If you specify text line number 1 and then repeat the specification later, the last occurrence of line number one prevails over the first occurrence.

Align field for final text fields in Query Management

This field controls the positioning of the final text within the report line; it refers to alignment between the left margin and the first summary column. If the report does not contain final summary data, then the alignment refers to the entire width of the displayed or printed report. Acceptable values are:

RIGHT
Right-justify the text.

LEFT Left-justify the text.

FINAL TEXT Fields

CENTER
Center the text.

The default value is *right* for the final text. If there is no associated final text, the Align value is ignored.

Final text lines for final text fields in Query Management

There are 999 lines available for the final text. You specify what appears on these lines. A maximum of 55 characters can be entered for each text line. *&col:* is the only variable that is allowed. The variable &col is assigned the last value of the last record in the specified column. Variable &col formats according to the edit code specification except when column wrapping is specified. Then, the edit code specification is ignored. If necessary, the edited data is truncated to the width of the column from which it was taken.

If the text line *n* is the biggest line with nonblank text, then *n* indicates how many lines are to be formatted. This is true even though it could result in some of the formatted lines being completely blank.

If the *Put final summary at line* value is *m* and *m* > *n*, then there are *m* final lines formatted in the report.

BREAK fields in Query Management

You can specify information for break levels one (1) to six (6). You also change or specify the exported FORM by selecting the proper FORM field numbers. See Chapter 8, “Exported and Imported Objects in Query Management”. There are distinct field numbers for each of the break levels. You specify options for each break level in a similar manner. Each set of options is independent from the others.

Table 10 shows the defaults and possible values for the attributes in the Break fields.

Table 10. Default Values for Break Fields

Attribute	Default	Possible Values
New page for break	NO	YES, NO
New page for footing	NO	YES, NO
Repeat column heading	NO	YES, NO
Blank lines before heading	0	0 to 999
Blank lines before footing	0	0 to 999 or BOTTOM
Blank lines after heading	0	0 to 999
Blank lines after footing	1	0 to 999
Put break summary at line	1	1 to 999 or NONE
Alignment on break heading text	LEFT	LEFT, CENTER, RIGHT
Alignment on break footing text	RIGHT	LEFT, CENTER, RIGHT

New page for break/new page for footing for break field in Query Management

These fields indicate whether the subsequent part of the report begins on a new page. The default value is *no* for both. When you specify Yes for the New Page for Break field, the member lines for the break format on a new page. If you specify a break heading, it precedes the break member lines on the new page. When you specify Yes for the New Page for Footing field, the break footing formats on the next page (if a footing exists).

Repeat column heading for break field in Query Management

This field indicates whether the column headings should repeat above the member lines for a particular break level. *No* is the default value.

When paging or printing a report, the column headings always appear at the top of the screen or page. In addition to these headings, a set of headings appears at the start of the break if Yes is specified for Repeat Column Headings for that break. This happens regardless of whether there is any break heading text. However, if the break starts at the top of a printed page, only one set of column headings, the set preceding the break member line, formats.

Blank lines before heading/footing for break field in Query Management

These fields indicate the number of blank lines that appear before the break heading or break footing. If no break heading is specified, then the value for this field is the number of blank lines before the break member lines. Acceptable values are any number from zero to 999. The default is *zero* for both the heading and the footing.

The Blank Lines Before Heading field may contain a number only.

For a break footing, you may also specify BOTTOM. Applicable only to a printed report, BOTTOM causes the break footing to position at the bottom of the current page on a printed report. BOTTOM causes insertion of blank lines in order to position the text immediately before the page footing text specification on the page. This also implies that a page eject occurs, since the next line must print on the next page.

Blank lines after heading/footing for break field in Query Management

These fields indicate the number of blank lines after the break heading or break footing. If no break heading is specified, then the value of this field defaults to the number of blank lines after the break member lines. If no break footing is specified, then the blank footing is included in the blank lines after the break member lines. An acceptable value is any number from zero to 999. The default is *zero* for the heading and *one (1)* for the footing.

Put break summary at line for break field in Query Management

This field indicates whether the break summary is to format and, if it does, where to place it relative to the lines of break footing text. The value can be from 1 to 999, or NONE, with NONE indicating that no break summary information is to display for the break. The default is *one (1)*. The number used corresponds to the number of the line of break footing text with which the break summary is to display.

This placement is strictly vertical. For horizontal placement in the line, the break summary always formats under the columns being summarized. If there are no column widths with aggregating usages, this value is ignored because there are no columns to summarize.

Break heading text lines

The heading text lines field contains a maximum of 999 heading text lines.

Line value for break heading text lines in Query Management

The Line value denotes the line positioning of the heading text lines.

For example, if you include text lines for line numbers 1 and 5, the text is displayed in the report as:

```
Text for line 1
[blank line]
[blank line]
[blank line]
Text for line 5
```

If you specify text line number 1 and then repeat the specification later, the last occurrence of line number one prevails over the first occurrence.

BREAK Fields

Align field for break heading text in Query Management

This field controls the positioning of the break heading text within the report line. Acceptable values are:

RIGHT
Right-justify the text.

LEFT Left-justify the text.

CENTER
Center the text.

The default value is *left*. The alignment is based on the entire width of the displayed or printed report.

Break heading text for break heading in Query Management

Fifty-five (55) characters per line are allowed on break heading text. Only *&col:* is allowed to be used as a variable. The variable *&col:* is assigned the first value of the break group for the specified column.

Variable &col formats according to the edit code specification except when column wrapping is specified. Then, the edit code specification is ignored. If necessary, the edited data is truncated to width of the column from which it was taken.

If the text line *n* is the biggest line with nonblank text, then *n* indicates how many lines are to be formatted. This is true even though it could result in some of the formatted lines being completely blank.

Break footing text lines in Query Management

The footing text lines field contains a maximum of 999 footing text lines.

Line value for break footing text lines in Query Management

The Line value denotes the line positioning of the footing text lines.

For example, if you include text lines for line numbers 1 and 5, the text is displayed in the report as:

```
Text for line 1
[blank line]
[blank line]
[blank line]
Text for line 5
```

If you specify text line number 1 and then repeat the specification later, the last occurrence of line number one prevails over the first occurrence.

Align field for break footing text in Query Management

This field controls the positioning of the page footing text within the report line. Acceptable values are:

RIGHT
Right-justify the text.

LEFT Left-justify the text.

CENTER
Center the text.

The default value is *right*. The alignment refers to the space between the first character position on the left and the first summary column. If the report does not contain break summary data, the alignment refers to the entire width of the displayed or printed report.

Break footing text field for break footing in Query Management

Only *&col* is allowed as a variable. The variable *&col* is assigned the last value of the break group for the specified column. You can use up to 55 characters per line. Lines considered part of the break footing text include lines up to, but not including, the first blank field not followed by a nonblank field.

If the text line *n* is the biggest line with nonblank text, then *n* indicates how many lines are to be formatted. This is true even though it could result in some of the formatted lines being completely blank.

If the *Put break summary at line* value is *m* and *m* > *n*, then there are *m* break lines formatted in the report.

OPTIONS fields in Query Management

The Options fields allow you to specify various report formatting options.

Table 11 shows the defaults and possible values for the attributes on the Options fields.

Table 11. Default Values for Options Fields

Attribute	Default	Possible Values
Detail line spacing	1	1 to 4
Outlining for break columns	YES	YES, NO
Default break text	YES	YES, NO
Column-wrapped lines kept on page	YES	YES, NO
Column heading separators	YES	YES, NO
Break summary separators	YES	YES, NO
Final summary separators	YES	YES, NO

Detail line spacing for options field in Query Management

This field indicates the spacing you request between each detail line in the report. Numbers are the only valid input for this field. Acceptable values are 1, 2, 3, or 4, where 1 is single spacing, 2 is double spacing, and so on. The default value is *1*.

Outlining for break columns for options field in Query Management

If you assign a usage code for a break to one of the columns, then this field is used to determine when the value in the break column displays in the report. *Yes* is the default and displays the value in the break column only when the value changes. A No in this field displays the value in the break column on every detail line in the report.

Default break text for options field in Query Management

This field indicates whether you are requesting inclusion of the default break text in the report. The default value is *yes*. Use default break text to mark the break aggregation line. The break aggregation line is one or more asterisks for each break level; one asterisk for the highest numbered break level text, two asterisks for the next highest numbered break level text, and so on. For example, if the report has two control breaks, BREAK2 and BREAK4, Query Management generates one asterisk to mark the level-4 break and two asterisks to mark the level-2 break.

Column wrapped lines kept on a page for options field in Query Management

If you specified column wrapping for one or more columns in the report, this field is used to determine whether the wrapped columns can be split between two pages. The default for this field is *yes*. It prevents splitting wrapped columns between two pages (unless the wrapped column is longer than the page depth). A No in this field allows splitting of wrapped columns between pages.

Column heading separators for options field in Query Management

This field indicates whether the column heading separators (dash lines) appear in the report. The default value is *yes*.

Break summary separators for options field in Query Management

This field indicates whether the break summary separators (dash lines) appear in the report. The default value is *yes*. A blank separator line is generated if there are no summary columns and the value specified is yes.

Final summary separators for options field in Query Management

This field indicates whether the final summary separators (equal signs) appear in the report. The default value is *yes*. A blank separator line is generated if there are no summary columns and the value specified is yes.

Chapter 7. Callable interface in Query Management

The DB2 UDB for iSeries Query Management callable interface (CI) provides the ability for application programs to perform Query Management functions through calls to the Query Management interface. After completion of DB2 UDB for iSeries Query Management function, return code and status information is available to the calling program. The CI is supported by query management for ILEC, C, COBOL, and RPG languages.

The CI consists of the following elements:

- Query Management CI Macros

  The Query Management macro instructions are comprised of the include and macro files used when application programs that call the Query Management CI modules are compiled. They contain the declarations for the communications area structure and any constants that are required to update and access the communications area structure. They also provide a standard interface from different programming languages to Query Management CI modules. The interface provides common storage and access of program variables between the programming language and query management. One Query Management CI macro or include is provided for each language that query management supports.

  Table 12 lists the individual macro include packages available for query management. The macro include are also available in a combined package as shown in Table 13. query management:

Table 12. Macro Include Packages

Language	Library	File	Member
ILEC	QCLE	H	DSQCOMMC
COBOL	QLBL	QILBINC	DSQCOMMB
RPG	QRPG	QIRGINC	DSQCOMMR

Table 13. Combined Macro Include Package

Language	Library	File	Member
ILE C	QSYSINC	H	DSQCOMMC
OPM COBOL	QSYSINC	QLBLSRC	DSQCOMMB
OPM RPG	QSYSINC	QRPGSRC	DSQCOMMR

  Before compiling an application program that uses these includes or macros, copy the member to the default include file used by the compiler. This allows the include to be used by the application program without being qualified with the library or file, which maintains a higher degree of portability.

  You can code application programs to qualify the include with the library and file name. This ensures the program is compiled with the newest version of the include.

- Query Management

  Provides query and report writing services.

- Query Management CI Modules

  Modules provided by the interface to allow access to the function of Query Management. These modules are:

  DSQCICE
  The ILEC and C language interface module for extended parameter lists

  DSQCIC
  The ILEC and C language interface module for nonextended parameter lists

DSQCIB
The COBOL language interface module

DSQCIR
The RPG language interface module

Callable interface in Query Management description

The callable interface (CI) is an interface that programming languages can use to run Query Management commands. All Query Management commands are supported through the CI.

To run a Query Management command, a program issues a call to start communications between the program and Query Management. This call is made to Query Management supplied routine.

The calling program can issue one or more Query Management commands after the initial start call. Each Query Management command that is processed requires a call to Query Management supplied routine. Information about the processing of the call is returned to the caller in a return code at the completion of each Query Management command. Other information about the processing of the command is gathered by the CI and stored in shared variables. When control returns to the calling application, these variables are available by reference.

The program issues a call to end communication between the program and Query Management when it no longer needs to use Query Management functions. This call is made to Query Management supplied routine.

Some considerations in the above processing are:

- A call to Query Management will return to the application only when processing of the command has been completed.
- CI remains in a quiesced state when it is not processing a call.
- All communications to the application will be via return codes and variable data stored in the variable pool or in the Interface Communications Area.
- Commands must be coded in uppercase English letters.
- The length of the passed commands must be at most 256 bytes.

The following diagram shows where the CI fits into the overall scheme.

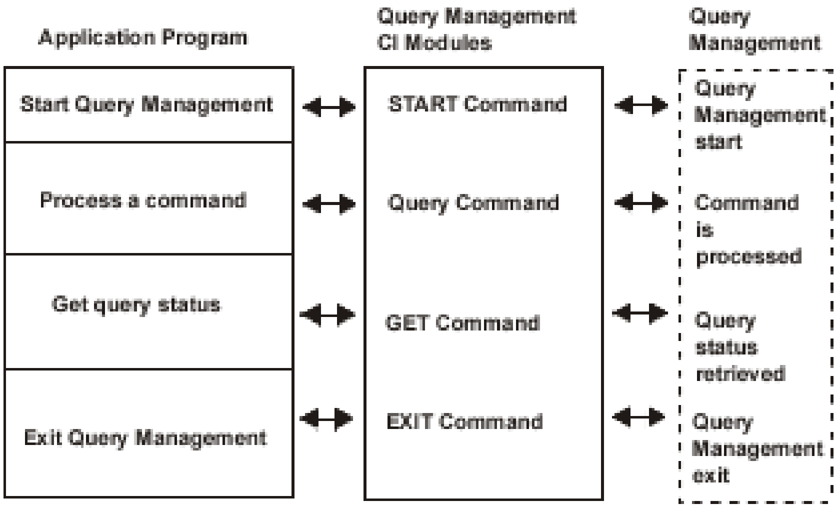


Figure 11. Callable Interface Diagram

If an application successfully processes Query Management command through the CI, the results are, in general, what they would be if the command had been processed on-line without displaying screens.

Query Management CI provides a unique interface communications macro for each language that is supported. The communications macro contains the following definitions, as appropriate:

1. Interface communications area (DSQCOMM)
2. Return codes
3. Call Interface to Query Management

Interface communications area (DSQCOMM) in Query Management CI

The Query Management CI communications area is required on all CI calls. Storage for the CI communications area is allocated by the program that is using the Query Management CI.

The START command establishes a unique instance of Query Management. As part of START command processing, the CI communications area is updated by Query Management. **The CI communications area must never be altered by the application program**. All subsequent calls after the START command must pass the address of the CI communications area that corresponds to an instance of Query Management. The user program is responsible for pointing to the correct communications area.

The CI communications area is described by the CI communications macro. There is a unique communications macro for each supported language. For applications to be portable, values must be referred to by variable name rather than by equated value because this value may be different on other systems.

The CI communications area DSQCOMM contains the following information which must **not** be altered by the calling program:

- Return Code

  Indicates the status of Query Management processing after a command is run.
- Instance Identifier

  Identifier that is established by Query Management during processing of the START command.
- Completion Message ID

Contains the message ID of the message that would have been displayed at the user terminal, if the command had been issued there.

- Query Message ID

  Contains the message ID of a query message, if the command resulted in query processing. This is the message ID of the message that would have been displayed in the job log. This message ID is different across the environments. It is provided to assist in debugging the application and should not be depended on in a portable application.
- START Command Parameter in Error

  Contains the parameter in error when START failed due to a parameter error.
- Cancel Indicator

  Indicates whether the user canceled the command processing while Query Management was running a command.
- Query Derived

  Indicates whether the query information was derived from a Query for iSeries definition.
- Form Derived

  Indicates whether the form information was derived from a Query for iSeries definition.

Return codes in the Query Management CI

Return codes are returned after each call to the Query Management CI. Return code values are described in the CI communications macro. For applications to be portable, values must be referred to by variable name rather than by equated value because this value may be different on other systems.

Return codes from the CI will include the following:

- Successful processing of the request.
- Command processed with warning condition.
- Command did not process correctly.
- Severe error: Query Management session ended for the applicable instance.

For a definition of each return code, see “C language interface in the Query Management CI” on page 87, “COBOL language interface in the Query Management CI” on page 94, and “RPG language interface in the Query Management CI” on page 100.

Return variables in the Query Management CI

When control is returned through the CI, variables will be set which contain information about the completion of Query Management command. The calling program can obtain the return variables from the variable pool.

Variables are referred to symbolically by name and are obtained from the Query Management variable pool by using the GET command.

Command message variables in the Query Management CI

After a command is initiated by an interactive user, the user sees a message on the screen indicating either a successful completion or an error during processing. This same information is available to the application through command message variables. The following command message variables are provided at the completion of each Query Management command processed through the CI:

DSQCIMNO

Contains the message number. The message number is also returned in the DSQCOMM area.

DSQCIMSG

Contains the first level message text as it would be displayed to the user interactively.

Query message variables in the Query Management CI

If an error occurs when dealing with the processing of Query Management query, a query message may be produced to help in problem analysis. For example, an error could have happened during processing of a RUN QUERY command. In this case, additional information may be provided. Query Management message variables consist of the following:

DSQCIQNO
Contains the message number. The message number is also returned in the DSQCOMM area.

DSQCIQMG
Contains the first level message text as it would be displayed to the end user interactively.

DSQCISQL
Contains the SQL return code from DB2 Universal Database for iSeries, if any.

Command syntax extension in the Query Management CI

In order for Query Management to provide variable support to high-level languages such as COBOL, Query Management must have access to the caller's program storage. (For an explanation of Query Management variable support, see "Extended variable support in the Query Management CI".) A DB2 UDB for iSeries Query Management command extension is used to support commands that require access to the caller's program storage area. The command extension is a different way to specify Query Management command options. The command extension is used for the GET, SET, and START commands because access to the user program area is required to support these commands.

Extended variable support in the Query Management CI

A variable is a named entity within Query Management which can be assigned a value. Extended variable support allows applications to define global variables within Query Management.

Variables may be used as substitution values in SQL queries and are available when using the CI. Once a variable is created, it is available to the Query Management session for the life of the session. In addition to application-defined variables, Query Management maintains a set of product variables. These variables are also available to SQL queries and the CI.

See Appendix C, "Use of quotation marks and apostrophes when setting global variables in Query Management" for examples of setting global variables.

Creating variables in the Query Management CI

Variables are created implicitly at run time, when they are first referred to during SQL query processing. They are also created explicitly, when they are set to a specific value using the SET GLOBAL command. When a variable is created implicitly, the variable value exists only during processing of the RUN QUERY command.

Referencing variables in the Query Management CI

Variables may be referred to by specifying the variable name within an SQL query or a user program via the GET GLOBAL command. When a variable name is referred to within an SQL query, the variable name must be prefixed with an ampersand (&) in order for Query Management to recognize it as a variable. For example:

```
SELECT * FROM &TNAME
```

The value &TNAME is considered Query Management variable.

Variable names in the Query Management CI

The following rules apply when you use variables in SQL queries across the callable interface.

- Names can contain the following characters:

  - **Letters**. A letter is any of the single-byte characters (A through Z, or any alphabetic character from a national alphabet).
  - **Arabic numbers** (0 through 9)
  - **Underscore** (_)
- Variable names must start with a letter (see exception for variable names used in SQL queries below).
- Names cannot be longer than 18 characters.
- Variable names that are used in SQL queries must be preceded by an ampersand (&), and the next character must be a single-byte character set letter. The ampersand does not take up one of the 30 characters allowed for the name. Be aware that the ampersand character delimits the beginning of a variable name; you cannot have more than one ampersand in a variable name because each ampersand would delimit the beginning of a distinct variable name.
- User-defined variables should not start with DSQ.

The following are valid variable names:

```
In an SQL Query              In the GET/SET Command
--------------------        -------------------------
&I_OWE_YOU                  I_OWE_YOU
&MYVAR123                   MYVAR123
&THIS_IS_A_BIG_NAM          THIS_IS_A_BIG_NAME
```

Variable values in the Query Management CI

Variable values can be character or integer values. The rules are as follows:

Character variables in the Query Management CI

- Character variable values consist of any value up to 55 characters long.
- A GET of a character variable into a smaller character field is allowed. The character string is truncated on the right after 55 characters.
- A GET of a character variable into a larger character field is allowed. The character string is left-adjusted and blank-padded. The null character at the end of a C string is not moved.
- A C null character is inserted at the end of the string if the GET was done through the C language callable interface.

Integer variables in the Query Management CI

- Integer variable values must be 4 bytes long. An attempt to SET or GET an integer with a length other then 4 bytes results in an error.
- Integer variable values are assumed to be signed.
- The value must observe SQL rules when used in an SQL query.
- An integer value is converted to a character string without leading or trailing blanks prior to substitution into the SQL statement. Do not use an integer variable if an implied result field width is needed or if you are using variable substitution while defining the result field in the SQL statement.

Defined variables in the Query Management CI

Query Management provides global variables which may be useful to user programs. The current set of Query Management variables can be used to determine the current status of the Query Management environment and particular objects. Query Management variables can’t be altered by the user, program, or procedure. A subset of these variables may be set by the user using the query command procedure which is specified on the START command. (See “START in Query Management” on page 45.)

The following variables are available in Query Management. The length given is the maximum length for the variable.

DSQAAUTH

Current connect authorization ID. This name will contain the name of the user profile under which the job is running.

Type Character

Length
10

Value -

DSQOAUTH

Default object public authority to be given to objects created through query commands. Refer to “START in Query Management” on page 45 for a description of the values.

Type Character

Length
10

Value

- *LIBCRTAUT
- *EXCLUDE
- *ALL
- *USE
- *CHANGE
- An authorization list name

DSQSNAME

Naming convention be used. Refer to “START in Query Management” on page 45 for a description of this variable.

Type Character

Length
4

Value

- *SAA
- *SYS

DSQAPRNM

Current default printer.

Type Character

Length
10

Value

- *SAME
- *JOB
- Printer Device Name

DSQCATTN

Last command cancel indicator.

Type Character

Length
3

Value

- YES
- NO

DSQCISQL
Last SQL return code.

Type Integer

Length
4

Value See the SQL Programming Concepts topic in the Information Center.

DSQSQLST
SQL state. The extended SQL return code designed for IBM relational database products.

Type Character

Length
5

Value See the appendix on SQLSTATES in the SQL Programming Concepts topic in the Information Center.

DSQAROWS
Current number of rows fetched for data.

Type Integer

Length
4

Value 0 - maximum # rows

DSQAROWC
Current data is completed.

Type Character

Length
3

Value

- YES
- NO

DSQSMODE
Current processing mode.

Type Character

Length
11

Value

- BATCH
- INTERACTIVE

DSQCONFIRM
Confirm processing default.

Type Character

Length
3

Value

- YES
- NO

DSQSCNVT
Allows the use of information derived from a query definition.

Type Character

Length
4

Value

- NO
- YES
- ONLY

DSQCIMNO
The query message ID. It is the same value that is returned in the query message line of the communications area.

Type Character

Length
8

Value -

DSQCIQNO
The message ID. It is the same value that is returned in the completion message line of the communications area.

Type Character

Length
8

Value -

DSQCIMSG
Contains the message text as it would be displayed to the user interactively.

Type Character

Length
55

Value -

DSQCIQMG
Contains the query message text as it would be displayed to the user interactively.

Type Character

Length
55

Value -

DSQSDBNM
Remote database name to which all SQL operations will be directed.

Type Character

Length
18

Value

- *CURRENT
- *NONE
- rdbname

DSQUSER

User identification to be used with a remote database.

Type Character

Length

10

Value

- *CURRENT
- username

DSQCMTLV

Specifies session commitment control level.

Type Character

Length

4

Value

- NONE
- UR
- CS
- RS (RS is changed to RR on output toDB2™ and SQL/DS; RR is changed back to RS on return to the OS/400 system.)
- RR

Commitment control in the Query Management CI

Commitment control is a means of grouping database file operations. Database changes are grouped into a single unit. That unit can be saved or removed through the commit or rollback commands. When you specify a level of commitment control, you are specifying how large a group of changes you want to work with as a single unit. Commitment control is specified using the DSQCMTLV keyword on the START command. The different types are:

- NONE—No commitment control is used.
- UR—The updated rows are locked together until the end of the transaction.
- CS—Any row that the cursor is in is locked until the cursor changes position.
- RS—All selected rows are locked until the end of the transaction.
- RR—All selected rows are locked until the end of the unit of work (UOW). This ensures that repeated reads within a unit work always give the same results.

Note: The SAVE DATA AS command always has a commitment control level of NONE.

For more information about commitment control, see the *Backup and Recovery* book.

Accessing CI with HLL programs in the Query Management

You can access the query management callable interface (CI) through the following high-level languages:

- C/400*
- COBOL/400*
- RPG/400*

C language interface in the Query Management CI

The Query Management callable interface is accessed by using normal “C” function calls. The exact description of each function call is provided in the callable interface “C” communications include file, DSQCOMMC. The communications include file, DSQCOMMC, is unique for each operating system. Query Management provides two external subroutine calls—DSQCIC and DSQCICE. DSQCIC is used to process Query Management commands that do not require access to program variables. DSQCICE is used to process commands that do require access to program variables.

The following commands must use the DSQCICE function:

START
SET GLOBAL
GET GLOBAL

All other Query Management commands must be specified using the DSQCIC function.

Example DSQCOMMC of the C interface in the Query Management CI

Figure 12 on page 88 shows the OS/400 version of the query management CI C communications macro.

C Language Interface

```
/*********************************************************************/
/*                                                                   */
/*  NAME:  dsqcommc.h                                                */
/*                                                                   */
/*  MODULE-TYPE: IBM C/400 Query Management Interface include file   */
/*                                                                   */
/*  PROCESSOR:  C                                                    */
/*                                                                   */
/*  DESCRIPTION:                                                     */
/*      This include file contains the declarations needed           */
/*      by a C application program for interfacing                   */
/*      with the query management callable interface.                */
/*      query management is the OS/400 implementation of the         */
/*      Systems Application Architecture Query Callable              */
/*      Programming Interface.                                       */
/*                                                                   */
/*  Copyright: 5728-SS1 (C) COPYRIGHT IBM CORP. 1989                 */
/*                                                                   */
/*********************************************************************/

/*********************************************************************/
/* Callable Interface Constants and Structures                       */
/*********************************************************************/

/* return code values for DSQ_RETURN_CODE                            */
#define DSQ_SUCCESS        0   /* successful running of the request  */
#define DSQ_WARNING        4   /* normal completion with warnings    */
#define DSQ_FAILURE        8   /* command did not process correctly  */
#define DSQ_SEVERE        16   /* severe error;  Query session       */
                               /* ended.                             */

/* Variable data types */
#define DSQ_VARIABLE_CHAR "CHAR"    /* unsigned character data type  */
#define DSQ_VARIABLE_FINT "FINT"    /* long integer type             */

/* Cancel indicator                                                  */
#define DSQ_CANCEL_YES    "1"       /* Yes it was canceled.          */
#define DSQ_CANCEL_NO     "0"       /* No, it was not canceled.      */

/* Derived query/form indicator                                      */
#define DSQ_DERIVED_YES   "1"       /* Yes it was derived from QRYDFN*/
#define DSQ_DERIVED_NO    "0"       /* No, it was not derived        */

/* Yes/No indicator.  This indicator can be used to test the values  */
/*   returned for the following global variables:                    */
/*      DSQCATTN - Last command cancel indicator.                    */
/*      DSQAROWC - Current data completed indicator.                 */
/*                                                                   */
#define DSQ_YES           "1"       /* Yes                           */
#define DSQ_NO            "0"       /* No                            */

/* misc defines                                                      */
#define DSQ_TRUE           1        /* indicates TRUE                */
#define DSQ_FALSE          0        /* indicates FALSE               */
#define DSQ_MATCH          0        /* match indicator               */
```

Figure 12. Example DSQCOMMC (Part 1 of 2)

```
/* define the Communication Area structure */
struct dsqcomm
  {
  unsigned long dsq_return_code;      /* function return code         */
  unsigned long dsq_instance_id;      /* instance id for this session */
  unsigned char dsq_reserve1[44]; /* reserved space -       */
                                      /* not for application use      */
  unsigned char dsq_message_id[8];  /* completion message id*/
  unsigned char dsq_q_message_id[8];     /* query message id*/
  unsigned char dsq_start_parm_error[8];       /* start parm*/
  unsigned char dsq_cancel_ind[1];   /* command canceled by */
                                         /* Control-Break (1=yes,0=no)*/
  unsigned char dsq_reserve2[17];       /* reserved space - */
                                          /*not for application use   */
  unsigned char dsq_query_derived[1];     /* query used was */
                                        /*derived from OS/400 *QRYDFN */
  unsigned char dsq_form_derived[1];/*form used was derived */
                                          /*   from OS/400 *QRYDFN    */
  unsigned int  dsq_delete_env;           /* flag used by QM to       */
                                          /*  control the QM          */
                                          /*  environment.            */
  unsigned char dsq_reserve3[924];    /* Reserve area 3     */
                                          /* application use          */
  } ;

/*********************************************************************/
/* Callable Interface External Function/Routine Definition          */
/*********************************************************************/

/* pragma definitions */
#define dsqcice DSQCICE
#define dsqcic DSQCIC
#pragma linkage(DSQCIC, OS)
#pragma linkage(DSQCICE, OS)

/* prototype for DSQCICE  */
extern void dsqcice  (
             struct dsqcomm *,      /* Communication Area            */
             signed long  *,        /* command length                */
             char *,                /* command                       */
             signed long *,         /* number of parms               */
             signed long *,         /* keyword lengths               */
             char *,                /* keywords                      */
             signed long *,         /* data lengths                  */
             void *,                /* data                          */
             char *);               /* data value type               */

/* prototype for DSQCIC   */
extern void dsqcic   (
             struct dsqcomm *,      /* Communication Area            */
             signed long  *,        /* command length                */
             char *);               /* command                       */
```

Figure 12. Example DSQCOMMC (Part 2 of 2)

C variable support in the Query Management CI

For C variables that are input character strings (including command strings and START and SET command variables), the user must pass an area that has a null value at the end. The length of the variable must also include the null value. The length function should be used to obtain the variable length that is passed to Query Management. The null value (X'00') indicates the end of a character string.

C Language Interface

For C variables that are output character strings (including values set by the GET command), Query Management moves data from Query Management storage to the user's specified storage area and sets the null indicator at the end of the string. If the character string does not fit in the user's storage area, a warning message is issued and the data is truncated on the right. A null indicator is always placed at the end of the data string.

DSQCIC function syntax for C in the Query Management CI

```
dsqcic(&communication_area,&command_length,&command_string );
```

Where:

- *communication_area* is the structure DSQCOMM.
- *command_length* is the length of command_string.

  The length is specified as a long integer.
- *command_string* is the Query Management command to be processed.

  The command string is specified as an array of character type.

DSQCICE function syntax for C in the Query Management CI

```
dsqcice (&communication_area,&command_length,&command_string,
         &number_of_parameters,&variable_length,&variable,
         &value_length,&value,&value_type);
```

Where:

- *communication_area* is the structure DSQCOMM.
- *command_length* is the length of command_string.

  The command length is specified as a long integer.
- *command_string* is a pointer to a character string which specifies the Query Management command to be processed.

  The command string is specified as an array of character type.
- *number_of_parameters* is the number of command variables.

  The number of variables is specified as a long integer.
- *variable_length* is the length of each specified variable name.

  The length of the variable name(s) is specified as a long integer type variable or variable array.
- *variable* is the Query Management variable name(s).

  The variable name string is specified as an array of unsigned character type.
- *value_length* is the length of each value associated with the variable.

  The length of the associated values is specified as a long integer type variable or variable array.
- *value* is the value associated with each variable.

  The value string is specified as an array of unsigned character type or a long integer type variable or variable array. The type is specified in the VTYPE parameter.
- *value_type* indicates the Query Management data type of the value string VALUE.

  The value type string contains one of the following values which is provided in the Query Management communications macro:

  DSQ_VARIABLE_CHAR indicates that the value string is unsigned character type.

  DSQ_VARIABLE_FINT indicates that the value string is long integer type.

Interface communications area (DSQCOMM) for C in the Query Management CI

The Query Management interface communications area is part of the communications macro DSQCOMMC. The interface communications area is described as a structure type named DSQCOMM.

The CI communications area DSQCOMM contains the following information which must ***not*** be altered by the calling program:

dsq_return_code (Unsigned long int)
Integer that indicates the status of Query Management processing after a command is run.

dsq_instance_ID (Unsigned long int)
Identifier that is established by Query Management during processing of the START command.

dsq_reserve1 (Char 44)
Reserved for future use.

dsq_message_id (Char 8)
Completion message ID.

dsq_q_message_id (Char 8)
Query message ID.

dsq_start_parm_error (Char 8)
Parameter in error when START failed due to a parameter error.

dsq_cancel_ind (Char 1)
Command cancel indicator; indicates whether the user had canceled the command processing while Query Management was running a command:

dsq_cancel_yes (Char = 1)

dsq_cancel_no (Char = 0)

dsq_reserve2 (Char 23)

dsq_query_derived (Char 1)
Indicates whether the query information used was derived from a Query for iSeries definition.

dsq_form_derived (Char 1)
Indicates whether the form information used was derived from a Query for iSeries definition.

dsq_reserve2 (Char 17)

dsq_reserve3 (Char 156).

Return codes for C language interface in the Query Management CI

Return codes are returned after each call to the Query Management CI. Return code values are described by the data interface macro. For applications to be portable, values must be referred to by variable name rather than by equated value because this value may be different on other systems.

Return code values for “dsq_return_code” are:

DSQ_SUCCESS
Successful processing of the request.

DSQ_WARNING
Normal completion with warnings.

DSQ_FAILURE
Command did not process correctly.

DSQ_SEVERE
Severe error: Query Management session ended for the applicable instance. Because the Query Management session ended, additional calls to Query Management cannot be made using this instance ID.

Sample C language query CI program in Query Management

Figure 13 is an example of a C language program written for Query Management CI.

```
 /******************************************************************/
 /* Sample Program:  DSQABFC                                       */
 /* C Version of the Query Management Callable Interface          */
 /******************************************************************/

 /******************************************************************/
 /* Include standard and string "C" functions                      */
 /******************************************************************/
 #include <string.h>
 #include <stdlib.h>

 /******************************************************************/
 /* Include and declare query interface communications area       */
 /******************************************************************/
 #include <DSQCOMMC.H>

int main()
     {

 struct dsqcomm communication_area;        /* DSQCOMM from include */

 /******************************************************************/
 /* Query interface command length and commands                    */
 /******************************************************************/
 signed long command_length;
 static char start_query_interface[] = "START";
 static char set_global_variables[] = "SET GLOBAL";
 static char run_query[] = "RUN QUERY Q1";
 static char print_report[] = "PRINT REPORT (FORM=F1";
 static char end_query_interface[] = "EXIT";

 /******************************************************************/
 /* Query command extension, number of parameters and lengths      */
 /******************************************************************/
 signed long number_of_parameters;    /* number of variables       */
 signed long keyword_lengths[10];     /* lengths of keyword names  */
 signed long data_lengths[10];        /* lengths of variable data  */

 /******************************************************************/
 /* 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[] = "DSQSCMD";
 static char start_keyword_values[] = "USERCMD1";
```

Figure 13. Sample C Program (Part 1 of 3)

```
/******************************************************************/
/* 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                                */
/******************************************************************/
    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]);
```

Figure 13. Sample C Program (Part 2 of 3)

```
  /*******************************************************************/
  /* Run a Query                                                     */
  /*******************************************************************/
      command_length = sizeof(run_query);
      dsqcic(&communication_area,&command_length,&run_query [0]);

  /*******************************************************************/
  /* Print the results of the query                                  */
  /*******************************************************************/
      command_length = sizeof(print_report);
      dsqcic(&communication_area,&command_length,&print_report[0]);

  /*******************************************************************/
  /* End the query interface session                                 */
  /*******************************************************************/
      command_length = sizeof(end_query_interface);
      dsqcic(&communication_area,&command_length,&end_query_interface[0]);
      exit(0);
 }
```

Figure 13. Sample C Program (Part 3 of 3)

COBOL language interface in the Query Management CI

The Query Management CI is accessed by using normal COBOL function calls. The exact description of each function call is provided in the Query Management COBOL communications macro DSQCOMMB. The communications macro DSQCOMMB is unique for each operating system. Query Management provides an external subroutine called DSQCIB which is used to run all Query Management commands. The parameters that are passed on the call to DSQCIB determine whether program variables are being passed. Program variables must be passed on the following Query Management commands:

START

SET GLOBAL

GET GLOBAL

The other Query Management commands do not specify program variables.

DSQCIB function syntax in the Query Management CI

```
CALL DSQCIB USING DSQCOMM, CMDLTH, CMDSTR.
```

Where:

- *DSQCOMM* is the structure DSQCOMM.
- *CMDLTH* is the length of the command string CMDSTR.

  The length is specified as an integer "PIC 9(8)" variable.
- *CMDSTR* is the Query Management command to be processed.

  The command string is specified as a character string of the length specified by CMDLTH.

DSQCIB extended function syntax for COBOL in the Query Management CI

```
CALL DSQCIB USING
            DSQCOMM CMDLTH CMDSTR
            PNUM VNLTH VNAME VLTH VALUE VTYPE.
```

Where:

- *DSQCOMM* is the structure DSQCOMM.
- *CMDLTH* is the length of the command string CMDSTR.

The length is specified as an integer "PIC 9(8)" variable.

- *CMDSTR* is the Query Management command to be processed.

  The command string is specified as a character string of the length specified by CMDLTH.
- *PNUM* is the number of command variables.

  PNUM is specified as an integer "PIC 9(8)" variable.
- *VNLTH* is the length of each specified variable name.

  The length of the variable name(s) is specified as an integer "PIC 9(8)" variable or variable array.
- *VNAME* is the Query Management variable name(s).

  The variable name string is specified as a character or a structure of characters whose length is the same as specified by VNLTH. An array of characters may be used provided all of the characters are of the same length.
- *VLTH* is the length of each value associated with the variable.

  The length of the associated values is specified as an integer "PIC 9(8)" variable or variable array.
- *VALUE* is the value associated with each variable.

  The value string is specified as a character or a structure of characters or an integer "PIC 9(8)" variable or variable array. The type is specified in the VTYPE parameter.
- *VTYPE* indicates the Query Management data type of the value string VALUE.

  The value type string contains one of the following values which is provided in the Query Management communications macro:

  DSQ-VARIABLE-CHAR indicates that value is character.

  DSQ-VARIABLE-FINT indicates that value is integer "PIC 9(8)".

Interface communications area (DSQCOMM) for COBOL in the Query Management CI

The Query Management interface communications area is part of the communications macro DSQCOMMB. The interface communications area is described as a structure named DSQCOMM.

The interface communications area DSQCOM contains the information shown in Table 14. This information must *not* be altered by the calling program.

Table 14. DSQCOM Programming Information

Variable	Type	Length (bytes)	Description
DSQRET	Binary	4 bytes	Integer that indicates the status of query management processing after a command is run.
DSQINS	Binary	4 bytes	Identifier that is established by query management when processing the START command.
DSQRES	Character	44 bytes	Reserved for future use.
DSQMSG	Character	8 bytes	Completion message ID.
DSQQMG	Character	8 bytes	Query message ID.
DSQSPE	Character	8 bytes	Parameter in error when START failed due to a parameter error.
DSQCNL	Character	1 byte	Command cancel indicator; indicates whether the user had canceled command processing while query management was running a command: DSQCLY ″VALUE 1″ DSQCLN ″VALUE 0″
DSQRS2	Character	17 bytes	Reserved for future use.

Table 14. DSQCOM Programming Information (continued)

Variable	Type	Length (bytes)	Description
DSQQDR	Character	1 byte	Query was derived from a Query for iSeries QRYDFN. DSQDRY ″VALUE 1″ DSQDRN ″VALUE 0″
DSQFDR	Character	1 byte	Form was derived from a Query for iSeries QRYDFN. DSQDRY ″VALUE 1″ DSQDRN ″VALUE 0″
DSQRS3	Character	156 bytes	Reserved for future use.
DSQRS4	Character	256 bytes	Reserved for future use.
DSQRS5	Character	256 bytes	Reserved for future use.
DSQRS6	Character	256 bytes	Reserved for future use.

Return codes for COBOL in the Query Management CI

Return codes are returned after each call to the Query Management CI. Return code values are described by the data interface macro. For applications to be portable, values must be referred to by variable name rather than by equated value because this value may be different on other systems.

Return code values for “DSQ-RETURN-CODE” are:

DSQ-SUCCESS
Successful processing of the request.

DSQ-WARNING
Normal completion with warnings.

DSQ-FAILURE
Command did not process correctly.

DSQ-SEVERE
Severe error: Query Management session ended for the applicable instance. Because the Query Management session ended, additional calls to Query Management cannot be made using this instance ID.

Example of COBOL query CI program in Query Management

Figure 14 on page 97 is an example of a COBOL language program written for the Query Management CI.

```
*****************************************************************
*   The following is a VS COBOL II version of the query
*   callable interface *** DSQABFCO **.
*****************************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  DSQABFCO.
   DATE-COMPILED.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*************************
* Copy DSQCOMMB definition - contains query interface variables
*************************
   COPY DSQCOMMB.

* Query interface commands
 01  STARTQI      PIC X(5)  VALUE "START".
 01  SETG         PIC X(10) VALUE "SET GLOBAL".
 01  QUERY        PIC X(12) VALUE "RUN QUERY Q1".
 01  REPT         PIC X(21) VALUE "PRINT REPORT (FORM=F1".
 01  ENDQI        PIC X(4)  VALUE "EXIT".
* Query command length
 01  QICLTH       PIC 9(8) USAGE IS COMP-4.
* Number of variables
 01  QIPNUM       PIC 9(8) USAGE IS COMP-4.
* Keyword variable lengths
 01  QIKLTHS.
   03   KLTHS     PIC 9(8) OCCURS 10 USAGE IS COMP-4.
* Value Lengths
 01  QIVLTHS.
   03   VLTHS     PIC 9(8) OCCURS 10 USAGE IS COMP-4.
* Start Command Keyword
 01  SNAMES.
   03  SNAME1   PIC X(7) VALUE "DSQSCMD".
* Start Command Keyword Value
 01  SVALUES.
   03  SVALUE1   PIC X(8) VALUE "USERCMD1".
* Set GLOBAL Command Variable Names to set
 01  VNAMES.
   03  VNAME1   PIC X(7) VALUE "MYVAR01".
   03  VNAME2   PIC X(5) VALUE "SHORT".
   03  VNAME3   PIC X(7) VALUE "MYVAR03".
* Variable value parameters
 01  VVALUES.
   03   VVALS     PIC 9(8) OCCURS 10 USAGE IS COMP-4.
 01   TEMP     PIC 9(8)           USAGE IS COMP-4.
```

Figure 14. Sample COBOL Program (Part 1 of 2)

```
        PROCEDURE DIVISION.
       *
       * Start a query interface session
            MOVE 0 TO QICLTH.
            INSPECT STARTQI TALLYING QICLTH FOR CHARACTERS.
            MOVE 0 TO TEMP.
            INSPECT SNAME1 TALLYING TEMP FOR CHARACTERS.
            MOVE TEMP TO KLTHS(1).
            MOVE 0 TO TEMP.
            INSPECT SVALUE1 TALLYING TEMP FOR CHARACTERS.
            MOVE TEMP TO VLTHS(1).
            MOVE 1 TO QIPNUM.
            CALL DSQCIB  USING DSQCOMM, QICLTH, STARTQI,
                               QIPNUM, QIKLTHS, SNAMES,
                               QIVLTHS, SVALUES, DSQ-VARIABLE-CHAR.
       *
       * Set numeric values into query variables using SET GLOBAL command
            MOVE 0 TO QICLTH.
            INSPECT SETG TALLYING QICLTH FOR CHARACTERS.
            MOVE 0 TO TEMP.
            INSPECT VNAME1 TALLYING TEMP FOR CHARACTERS.
            MOVE TEMP TO KLTHS(1).
            MOVE 0 TO TEMP.
            INSPECT VNAME2 TALLYING TEMP FOR CHARACTERS.
            MOVE TEMP TO KLTHS(2).
            MOVE 0 TO TEMP.
            INSPECT VNAME3 TALLYING TEMP FOR CHARACTERS.
            MOVE TEMP TO KLTHS(3).
            MOVE 4 TO VLTHS(1).
            MOVE 4 TO VLTHS(2).
            MOVE 4 TO VLTHS(3).
            MOVE 20 TO VVALS(1).
            MOVE 40 TO VVALS(2).
            MOVE 84 TO VVALS(3).
            MOVE 3 TO QIPNUM.
            CALL DSQCIB  USING DSQCOMM, QICLTH, SETG,
                               QIPNUM, QIKLTHS, VNAMES,
                               QIVLTHS, VVALUES, DSQ-VARIABLE-FINT.
       *
       * Run a Query
            MOVE 0 TO QICLTH.
            INSPECT QUERY TALLYING QICLTH FOR CHARACTERS.
            CALL DSQCIB USING DSQCOMM, QICLTH, QUERY.
       *
       * Print the results of the query
            MOVE 0 TO QICLTH.
            INSPECT REPT TALLYING QICLTH FOR CHARACTERS.
            CALL DSQCIB USING DSQCOMM, QICLTH, REPT.
       *
       * End the query interface session
            MOVE 0 TO QICLTH.
            INSPECT ENDQI TALLYING QICLTH FOR CHARACTERS.
            CALL DSQCIB USING DSQCOMM, QICLTH, ENDQI.
            STOP RUN.
```

Figure 14. Sample COBOL Program (Part 2 of 2)

Example 2 of COBOL query CI program in Query Management

Figure 15 on page 99 is an example of the Query Management CI COBOL communications macro that has been tailored to fit the OS/400 environment.

```
****************************************************************
*                                                              *
*  NAME:  DSQCOMMB                                             *
*                                                              *
*  MODULE-TYPE: IBM COBOL/400 Query Management Interface       *
*                include file                                  *
*                                                              *
*  PROCESSOR: COBOL                                            *
*                                                              *
*  DESCRIPTION:                                                *
*      This include file contains the declarations needed      *
*      by a COBOL/400 application program for interfacing      *
*      with the query management callable interface.           *
*      query management is the OS/400 implementation of the    *
*      Systems Application Architecture Query Callable         *
*      Programming Interface.                                  *
*                                                              *
*  Copyright: 5728-SS1 (C) COPYRIGHT IBM CORP. 1989            *
*                                                              *
****************************************************************

* Structure declare for communications area
  01 DSQCOMM.
     03 DSQ-RETURN-CODE      PIC 9(8) USAGE IS BINARY VALUE 0.
*                              * Function return code       *
     03 DSQ-INSTANCE-ID      PIC 9(8) USAGE IS BINARY VALUE 0.
*                              * Identifier from START cmd  *
     03 DSQ-RESERVE1         PIC X(44).
*                              * Reserved area              *
     03 DSQ-MESSAGE-ID       PIC X(8).
*                              * Completion message id      *
     03 DSQ-Q-MESSAGE-ID     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 canceled       *
*                              * 0 = Command not canceled   *
     03 DSQ-RESERVE2         PIC X(17).
*                              * Reserved space -- not for  *
*                              * application use            *
```

Figure 15. Example DSQCOMMB (Part 1 of 2)

```
     03 DSQ-QUERY-DERIVED    PIC X(1).
*                              * 1 = Query was derived from *
*                              *      OS/400 QRYDFN         *
*                              * 0 = Query was not derived  *
*                              *      from OS/400 QRYDFN    *
     03 DSQ-FORM-DERIVED     PIC X(1).
*                              * 1 = Form was derived from  *
*                              *      OS/400 QRYDFN         *
*                              * 0 = Form was not derived   *
*                              *      from OS/400 QRYDFN    *
     03 DSQ-DELETE-ENV       PIC 9(8) USAGE IS BINARY VALUE 0.
*                              * Flag used to delete env.   *
     03 DSQ-RESERVE3         PIC X(924).
*                              * Reserved space -- not for  *
*                              * application use            *


*  Return code values for DSQ-RETURN-CODE
  01 DSQ-SUCCESS             PIC 9(8) USAGE IS BINARY VALUE 0.
  01 DSQ-WARNING             PIC 9(8) USAGE IS BINARY VALUE 4.
  01 DSQ-FAILURE             PIC 9(8) USAGE IS BINARY VALUE 8.
  01 DSQ-SEVERE              PIC 9(8) USAGE IS BINARY VALUE 16.

*  Callable Interface program name
  01 DSQCIB                  PIC X(7) VALUE "QQXMAIN".

*  Values for variable type on CALL parameter
  01 DSQ-VARIABLE-CHAR       PIC X(4) VALUE "CHAR".
  01 DSQ-VARIABLE-FINT       PIC X(4) VALUE "FINT".

*  Values for query/form derived field in communications area
  01 DSQ-DERIVED-NO          PIC X(1) VALUE "0".
  01 DSQ-DERIVED-YES         PIC X(1) VALUE "1".

*  Values for the cancel indicator field in communications area
  01 DSQ-CANCEL-YES         PIC X(1) VALUE "1".
  01 DSQ-CANCEL-NO          PIC X(1) VALUE "0".

*  Yes/No indicator.  This indicator can be used
*   to test the values
*   returned for the following global variables:
*     DSQCATTN - Last command cancel indicator.
*     DSQAROWC - Current data completed indicator.
*
  01 DSQ-YES                PIC X(1) VALUE "1".
  01 DSQ-NO                 PIC X(1) VALUE "0".
```

Figure 15. Example DSQCOMMB (Part 2 of 2)

RPG language interface in the Query Management CI

The Query Management callable interface is accessed by using normal RPG function calls. The exact description of each function call is provided in the Query Management RPG communications include member DSQCOM. Query Management provides an external subroutine called DSQCIR which is used to run all Query Management commands. The parameters that are passed on the call to DSQCIR determine whether program variables are being passed. Program variables must be passed on the following Query Management commands:

START

SET GLOBAL

GET GLOBAL

The other Query Management commands do not specify program variables.

DSQCIR function syntax for RPG in the Query Management CI

```
C                     CALL DSQCIR
C                     PARM           DSQCOM
C                     PARM           CMDLTH
C                     PARM           CMDSTR
```

Where:

- *DSQCOM* is the structure DSQCOM.
- *CMDLTH* is the length of the command string CMDSTR.

  The length is specified as a 4-byte binary field.
- *CMDSTR* is the Query Management command to be processed.

  The command string is specified as a character string of the length specified by CMDLTH.

DSQCIR extended function syntax for RPG in the Query Management CI

```
C                     CALL DSQCIR
C                     PARM           DSQCOM
C                     PARM           CMDLTH
C                     PARM           CMDSTR
C                     PARM 1         PNUM
C                     PARM           KLTH
C                     PARM           KWORD
C                     PARM           VLTH
C                     PARM           VALUE
C                     PARM           VTYPE
```

Where:

- *DSQCOM* is the structure DSQCOM.
- *CMDLTH* is the length of the command string CMDSTR.

  The length is specified as a 4-byte binary field.
- *CMDSTR* is the Query Management command to be processed.
- The command string is specified as a character string of the length specified by CMDLTH.
- *PNUM* is the number of command keywords.

  PNUM is specified as a 4-byte binary field.
- *KLTH* is the length of each specified keyword.

  The length of the keyword or keywords is specified as a 4-byte binary field.
- *KWORD* is the Query Management keyword or keywords.

  The keyword string is specified as a character or a structure of characters whose length are the same as specified by KLTH. An array of characters may be used, provided all of the characters are of the same length.
- *VLTH* is the length of each value associated with the keyword.

  The length of the associated values is specified as a 4-byte binary field.
- *VALUE* is the value associated with each keyword.

  The value string is specified as a character or a structure of characters or a 4-byte binary field. The type is specified in the VTYPE parameter.
- *VTYPE* indicates the Query Management data type of the value string VALUE.

  The value type string contains one of the following values, which is provided in the Query Management communications include member:

  DSQVCH indicates that value is character.

  DSQVIN indicates that value is an integer (4-byte binary).

Interface communications area (DSQCOMMR) for RPG in the Query Management CI

The Query Management interface communications area is part of the communications include member DSQCOMMR. The interface communications area is described as a structure named DSQCOM.

The CI communications area DSQCOM contains the following information that must **not** be altered by the calling program:

DSQRET Binary (4 bytes)
Integer that indicates the status of Query Management processing after a command is run.

DSQINS Binary (4 bytes)
Identifier that is established by the Query Management during processing of the START command.

DSQRS1 Character (44 bytes)
Reserved for future use.

DSQMSG Character (8 bytes)
Completion message ID.

DSQQMG Character (8 bytes)
Query message ID.

DSQSPE Character (8 bytes)
Parameter in error when START failed due to a parameter error.

DSQCNL Character (1 byte)
Command cancel indicator; indicates whether the user had canceled command processing while Query Management was running a command:

DSQCLY "VALUE 1"

DSQCLN "VALUE 0"

DSQQDR Character (1 byte)
Indicates whether the query information used was derived from a Query for iSeries definition.

DSQDRY "VALUE 1" — Object was derived

DSQDRN "VALUE 0" — Object was not derived

DSQFDR Character (1 byte)
Indicates whether the form information used was derived from a Query for iSeries definition.

DSQDRY "VALUE 1" — Object was derived

DSQDRN "VALUE 0" — Object was not derived

DSQRS2 Character (23 bytes)
Reserved for future use.

DSQRS3 Character (156 bytes)
Reserved for future use.

Return codes for RPG in the Query Management CI

Return codes are returned after each call to the Query Management CI. Return code values are described by the data interface. For applications to be portable, values must be referred to by variable name rather than the equated value because this value may be different on other systems. **Return code** values for DSQRET are:

DSQSUC
Successful processing of the request.

DSQWAR
Normal completion with warnings.

DSQFAI
: Command did not process correctly.

DSQSEV
: Severe error: Query Management session ended for the applicable instance. Because the Query Management session ended, additional calls to Query Management cannot be made using this instance ID.

Example of RPG language query CI program in the Query Management CI

Figure 16 is an example of an RPG language program written for the Query Management CI.

```
 ****************************************************************
 *                                                              *
 *           SAMPLE RPG PROGRAM USING QUERY INTERFACE           *
 *           ----------------------------------------           *
 *                                                              *
 *  1) Include member DSQCOMMMR contains the communications     *
 *     area to be passed to the query interface.                *
 *  2) Command name lengths, command names,                     *
 *     variable name lengths, variable names,                   *
 *     variable value lengths, variable values,                 *
 *     are loaded as compile time arrays.                       *
 *  3) It is necessary to pass all interface lengths and        *
 *     numeric variable information in binary format.           *
 *                                                              *
 ****************************************************************
H
 *
 * Compile time arrays of command   name   lengths  and  values
 *                        variable  name   lengths  and  values
 *                        variable content lengths  and  values
 *
E                    CNL     1   7  9 0 CNV    25     commands
E                    VNL     1   3  9 0 VNV     7     variable names
E                    VCL     1   3  9 0 VCV     9 0   variable values
 *
I            DS
I                                    B   1  280CNL
I            DS
I                                    B   1  120VNL
I            DS
I                                    B   1  120VCL
I            DS
I                                    B   1   40BINARY
```

Figure 16. Sample RPG Program (Part 1 of 3)

```
 *
 * Pull in the communications area
 *
I/COPY DSQCOMMR
 *
 * Start a query interface session:
 *
C                     CALL DSQCIR
C                     PARM           DSQCOM           comms area
C                     PARM           CNL,1            command length
C                     PARM           CNV,1            START
C                     PARM 1         BINARY           # keywords
C                     PARM           CNL,6            keyword length
C                     PARM           CNV,6            DSQSCMD
C                     PARM           CNL,7            value length
C                     PARM           CNV,7            USERCMD1
C                     PARM DSQVCH    DATA    4        CHAR
 *
 * Set numeric values into query variables using SET GLOBAL command:
 *
C                     CALL DSQCIR
C                     PARM           DSQCOM           comms area
C                     PARM           CNL,2            command length
C                     PARM           CNV,2            SET GLOBAL
C                     PARM 3         BINARY           # variables
C                     PARM           VNL              name lengths
C                     PARM           VNV              name values
C                     PARM           VCL              variable lengths
C                     PARM           VCV              variable values
C                     PARM DSQVIN    DATA             FINT
 *
 * Run a query:
 *
C                     CALL DSQCIR
C                     PARM           DSQCOM           comms area
C                     PARM           CNL,3            command length
C                     PARM           CNV,3            RUN QUERY Q1
 *
 * Print the results of the query:
 *
C                     CALL DSQCIR
C                     PARM           DSQCOM           comms area
C                     PARM           CNL,4            command length
C                     PARM           CNV,4            PRINT REPORT
 *                                                      (FORM=F1
 * End the query interface session:
 *
C                     CALL DSQCIR
C                     PARM           DSQCOM           comms area
C                     PARM           CNL,5            command length
C                     PARM           CNV,5            EXIT
 *
C                     SETON                     LR
 *
```

Figure 16. Sample RPG Program (Part 2 of 3)

```
**  CNL/CNV           Command lengths and command values
000000005START
000000010SET GLOBAL
000000012RUN QUERY Q1
000000021PRINT REPORT (FORM=F1
000000004EXIT
000000007DSQSCMD
000000008USERCMD1
**  VNL/VNV           Variable name lengths and variable names
000000007MYVAR01
000000007MYVAR02
000000007MYVAR03
**  VCL/VCV           Variable value lengths and variable values
000000004000000020
000000004000000040
000000004000000084
```

Figure 16. Sample RPG Program (Part 3 of 3)

Example 2 of RPG language query CI program in the Query Management CI

Figure 17 is an example of the query management CI RPG communications include member. This version of the communications include member has been tailored for the OS/400 system environment.

```
     I***************************************************************
     I*                                                             *
     I*  NAME:  DSQCOMMR                                            *
     I*                                                             *
     I*  MODULE-TYPE: IBM RPG/400 Query Management Include File     *
     I*                                                             *
     I*  PROCESSOR: RPG                                             *
     I*                                                             *
     I*  DESCRIPTION:                                               *
     I*      This include file contains the declarations needed     *
     I*      by an RPG/400 application program for interfacing      *
     I*      with the query management callable interface.          *
     I*      Query Management is the OS/400 implementation of the   *
     I*      Systems Application Architecture Query Callable        *
     I*      Programming Interface.                                 *
     I*                                                             *
     I*  Copyright: 5728-SS1 (C) COPYRIGHT IBM CORP. 1989           *
     I*                                                             *
     I***************************************************************
```

Figure 17. Example DSQCOMMR (Part 1 of 3)

RPG Language Interface

```
I***************************************************************
I*               QUERY INTERFACE INCLUDE                       *
I*                                                             *
I* DSQCOM Definition, contains QUERY interface variables:      *
I*                                                             *
I*           DSQRET     - Status of QUERY processing           *
I*           DSQINS     - QUERY identifier                     *
I*           DSQRS1     - Reserved                             *
I*           DSQMSG     - Completion message-ID                *
I*           DSQQMG     - QUERY message ID                     *
I*           DSQSPE     - START fail parameter error           *
I*           DSQCNL     - Command cancel indicator             *
I*           DSQQDR     - Query was derived from OS/400 QRYDFN *
I*           DSQFDR     - Form was derived from OS/400 QRYDFN  *
I*           DSQDEN     - Environment deletion indicator       *
I*           DSQRS2     - Reserved                             *
I*           DSQRS3     - Reserved                             *
I*           DSQRS4     - Reserved                             *
I*           DSQRS5     - Reserved                             *
I*           DSQRS6     - Reserved                             *
I*                                                             *
I***************************************************************
IDSQCOM      DS     I                                    B   1   40DSQRET
I                                    B   5   80DSQINS
I                                        9  52 DSQRS1
I                                       53  60 DSQMSG
I                                       61  68 DSQQMG
I                                       69  76 DSQSPE
I                                       77  77 DSQCNL
I                                       78  94 DSQRS2
I                                       95  95 DSQQDR
I                                       96  96 DSQFDR
I                                       97 100 DSQDEN
I                                      101 356 DSQRS3
I                                      357 612 DSQRS4
I                                      613 868 DSQRS5
I                                      8691024 DSQRS6
I*
```

Figure 17. Example DSQCOMMR (Part 2 of 3)

```
I* DSQRET - DSQ return code meanings
I*                    SUCCESS       --     value  0
I*                    WARNING       --     value  4
I*                    FAILURE       --     value  8
I*                    SEVERE        --     value 16

I*
I              0                    C         DSQSUC
I              4                    C         DSQWAR
I              8                    C         DSQFAI
I              16                   C         DSQSEV
I*
I* DSQCNL - DSQ cancel indicator meanings
I*                    CANCEL YES    --     value '1'
I*                    CANCEL NO     --     value '0'
I*
I              '1'                  C         DSQCLY
I              '0'                  C         DSQCLN
I*
I* DSQQDR/DSQFDR - DSQ QRYDFN derivation indicator meanings
I*                    DERIVED YES   --     value '1'
I*                    DERIVED NO    --     value '0'
I*
I              '1'                  C         DSQDRY
I              '0'                  C         DSQDRN
I*
I* DSQYES/DSQNO  - DSQ constants for the values returned
I*                  for the following global variables:
I*                    DSQCATTN - Last command cancel indicator.
I*                    DSQAROWC - Current data completed indicator.
I*
I              '1'                  C         DSQYES
I              '0'                  C         DSQNO
I*     I* DSQCIR - Interface program call name definition
I*
I              'QQXMAIN'            C         DSQCIR
I*
I* DSQVCH - contains constant value 'CHAR'
I* DSQVIN - contains constant value 'FINT'
I*
I              'CHAR'               C         DSQVCH
I              'FINT'               C         DSQVIN
I*
I*          END OF DSQCOM QUERY INCLUDE
I************************************************************
```

Figure 17. Example DSQCOMMR (Part 3 of 3)

Using subprograms to access CI in the Query Management

You can use subprograms to access the query management callable interface (CI). Subprograms relieve you of most of the data manipulation necessary to access the CI. This section describes subprograms and how to use them in handling queries.

This section describes the listings for seven subprograms that represent the more common functions performed. Create different subprograms if you find there are other commonly used functions in your particular environment.

When using the subprograms, consider the following issues:

RPG Language Interface

- If the calling program calls the subprogram only once (or infrequently), end the subprogram when returning to the calling program. Refer to the *RPG/400 User's Guide* for more details on calling other programs.
- Once the CI is started, an instance identifier is allocated. Therefore, the data structure DSQCOM is passed from program to program. You must pass this instance identifier to the CI for each access under that session.
- If the application programs using these subprograms are run on an iSeries system different from the one that created the subprograms, object code versions of these subprograms need to be on the iSeries system running the programs.
- If the application programs using these subprograms are run on a non-iSeries system, object code versions of programs that perform the same function must be created on that system.

  Note: The code described in this chapter is written in RPG/400® language and does not necessarily compile on non-iSeries systems.
- The code provided in this chapter is written in RPG, but you can develop similar functions in other programming languages. You can also access the RPG/400 subprograms, once compiled, from a COBOL program.

The following sections describe how you can use subprograms to accomplish the commonly used query management functions. Following each description is the example subprogram created to accomplish the query management task.

START subprogram in the Query Management CI

The values for the keywords DSQSMODE, DSQSCMD, DSQSRUN, and DSQSNAME are passed to this program as a string of 132 characters. The first 33 characters represent the DSQSMODE keyword value, the next 33 characters represent the DSQSCMD keyword value, the next 33 the DSQSRUN keyword value, and the last 33 the DSQSNAME keyword value. Left-justify the keyword values you type. If a keyword value is not used, it still must be passed, but as a string of 33 blank characters.

The START subprogram reads the passed keyword values string, tests for blank values, and calculates the lengths of the values. It also strings together the start command, keywords, and keyword values with the necessary lengths and calls the programmable interface. The interface is started and the START subprogram is ended with control returned to the calling program.

```
 ****************************************************************
 *                                                              *
 *         START COMMAND CPI QUERY INTERFACE HANDLER            *
 *         -----------------------------------------            *
 *                                                              *
 *  1) Include member DSQCOMMR contains the communications      *
 *     area to be passed to the query management interface.     *
 *  2) This program handles the START CPI QM interface          *
 *     command. It reads the DSQ keywords information to be     *
 *  3) The keyword information is passed to this program in the *
 *     form of 4 values which are the 4 keyword values to be    *
 *     passed to query management. This program calculates      *
 *     the length of each keyword name and keyword value and    *
 *     strings the necessary information into arrays for        *
 *     passing to the query management callable interface.      *
 *                                                              *
 ****************************************************************
H
 *
E                    LTH     1   4  9 0 KEY     8   lengths of k/wds
E                    STA         4 33               k/wd vals passed
E                    KEL         4  9 0             keyword lengths
E                    KEN        30  1               keyword names
E                    VAL         4  9 0             value lengths
E                    VAV        81  1               value values
E                    TST        33  1               test value length
 *
I            DS
I                                    B   1   40BIN1
I                                    B   5   80BIN2
I                                    B   9  240KEL
I                                    B  25  400VAL
I/COPY BPLIB/QRPGSRC,DSQCOMMR
 *
 * receive the passed start command keyword values:
 *
C           *ENTRY    PLIST
C                     PARM           DSQCOM           comms area
C                     PARM           STA              keywords passed
 *
 * prepare keyword name lengths, names, value lengths, values
 *
C                     Z-ADD1         Y       20       initialize
C                     Z-ADD1         W       20       counters
C                     Z-ADD0         KEL              and numeric
C                     Z-ADD0         VAL              arrays
```

Figure 18. Example START Subprogram (Part 1 of 2)

```
      *
     C           V         DOUEQ4                          look at each
     C                     ADD  1         V       10       passed keyword
     C           STA,V     COMP *BLANKS                  50value & process
     C           *IN50     IFEQ '0'                        if not blank
      *
     C                     ADD  1         X       10       keyword name
     C                     MOVE LTH,V     KEL,X            lengths array
      *
     C           ' '       LOKUPKEN,Y                    60 string keyword
     C                     MOVE KEY,V     WORK1   8         name into
     C                     MOVEAWORK1     KEN,Y             names array
      *
     C                     MOVEASTA,V     TST               find keyword
     C                     Z-ADD1         Z       20        value length
     C           ' '       LOKUPTST,Z                    61 and move
     C   61                SUB  1         Z                 to keyword
     C  N61                Z-ADD33        Z                 value lengths
     C                     Z-ADDZ         VAL,X             array
      *
     C           ' '       LOKUPVAV,W                    62 string keyword
     C                     MOVE STA,V     WORK2  33         value into
     C                     MOVEAWORK2     VAV,W             values array
      *
     C                     END
     C                     END
      *
      * start the query interface session:
      *
     C                     CALL DSQCIR
     C                     PARM           DSQCOM           comms area
     C                     PARM 5         BIN1             command length
     C                     PARM 'START'   CHAR1   5        START
     C                     PARM X         BIN2             # keywords
     C                     PARM           KEL              keyword lengths
     C                     PARM           KEN              keyword names
     C                     PARM           VAL              value lengths
     C                     PARM           VAV              values
     C                     PARM DSQVCH    CHAR2   4        CHAR
      *
     C                     MOVE '1'       *INLR
      *
**  start DSQ keyword name lengths and names loaded as compile time array
000000008DSQSMODE
000000007DSQSCMD
000000007DSQSRUN
000000008DSQSNAME
```

Figure 18. Example START Subprogram (Part 2 of 2)

SETC subprogram in the Query Management CI

The SETC subprogram performs the SET GLOBAL variable function for a character value to be passed to the CI. The SETC subprogram handles one variable at a time. The variable name and value are passed to this program as two separate parameters. The name can be up to 10 characters long, and the value up to 20 characters long.

This subprogram calculates the necessary lengths, strings the information together, and calls the programmable interface. The variable is set as CHAR data type, and control then returns to the calling program.

Note: The SETC subprogram is not ended because it may be called a number of times in the session.

```
 ****************************************************************
 *                                                              *
 *           SET GLOBAL COMMAND (CHARACTER VARIABLE)            *
 *                  CPI QUERY INTERFACE HANDLER                 *
 *           ---------------------------------------            *
 *                                                              *
 *  1) Include member DSQCOMMR contains the communications      *
 *     area to be passed to the Query Interface.                *
 *  2) This program handles the SET GLOBAL Query Interface      *
 *     command for variable values to be passed to the          *
 *     interface as CHAR type.
 *  3) It reads the variable name and value, calculates the     *
 *     length of each, and passes the information to Query      *
 *     Management.                                              *
 *  4) The program handles one variable at a time, the length   *
 *     of the variable name can be a maximum of 10 characters   *
 *     and the length of the variable value can be a maximum    *
 *     of 20 characters.                                        *
 *                                                              *
 ****************************************************************
H
 *
E                    TNL        10  1               test name length
E                    TVL        20  1               test value length
 *
I              'SET GLOBAL'          C         CMD
 *
I            DS
I                                    B   1   40BIN1
I                                    B   5   80BIN2
I                                    B   9  120BIN3
I                                    B  13  160BIN4
I/COPY BPLIB/QRPGSRC,DSQCOMMR
 *
 * receive the passed variable name and value:
 *
C           *ENTRY    PLIST
C                     PARM           DSQCOM           comms area
C                     PARM           VARNAM 10        variable name
C                     PARM           VARVAL 20        variable value
 *
 * calculate the variable name length and variable value length:
 *
C                     MOVEAVARNAM    TNL
C                     Z-ADD1         X       20         X = last
C           ' '       LOKUPTNL,X                    60  non blank
C   60                SUB  1         X                  character
C  N60                Z-ADD10        X                  in name
```

Figure 19. Example SETC Subprogram (Part 1 of 2)

```
 *
C                     Z-ADD20        Y       20         if value
C           VARVAL    IFNE *BLANKS                      blank pass
C                     MOVEAVARVAL    TVL                20 blanks
C           AGAIN     TAG
C           ' '       COMP TVL,Y                    61  Y = last
C   61                SUB  1         Y                  non blank
C   61                GOTO AGAIN                        character
C                     END                               in value
 *
 * set the global variables:
 *
C                     CALL DSQCIR
C                     PARM           DSQCOM           comms area
C                     PARM 10        BIN1             command length
C                     PARM CMD       CHAR1  10        SET GLOBAL
C                     PARM 1         BIN2             # variables
C                     PARM X         BIN3             var name length
C                     PARM           VARNAM           variable name
C                     PARM Y         BIN4             var value lngth
C                     PARM           VARVAL           variable value
C                     PARM DSQVCH    CHAR2   4        CHAR
 *
C                     RETRN
```

Figure 19. Example SETC Subprogram (Part 2 of 2)

SETA subprogram in the Query Management CI

The SETA subprogram performs the SET GLOBAL variable function for a character value to be enclosed in apostrophes and then passed to the CI. This function is required when creating a query that compares a data item to a constant character value (DEPT = ’ACCT’). The variable name and value are passed to this program as two separate parameters. The name can be up to 10 characters long and the value up to 20 characters long.

This program encloses the value in apostrophes, calculates the necessary lengths, strings the information together, and calls the programmable interface. The variable is set as CHAR data type, and control then returns to the calling program.

Note: The SETA program is not ended because it may be called a number of times in the session.

```
 ****************************************************************
 *                                                              *
 *       SET GLOBAL COMMAND (APOSTROPHE ENCLOSED CHARACTER      *
 *             VARIABLE) CPI QUERY INTERFACE HANDLER            *
 *       -----------------------------------------------        *
 *                                                              *
 *  1) Include member DSQCOMMR contains the communications      *
 *     area to be passed to the query management interface.     *
 *  2) This program handles the SET GLOBAL interface            *
 *     command for variable values to be enclosed in            *
 *     apostrophes and passed to the interface as CHAR type.    *
 *  3) It reads the variable name and value, calculates the     *
 *     length of each, encloses the value in apostrophes,       *
 *     and passes the information to query management.          *
 *  4) The program handles one variable at a time, the length   *
 *     of the variable name can be a maximum of 10 characters   *
 *     and the length of the variable value can be a maximum    *
 *     of 20 characters.                                        *
 *                                                              *
 ****************************************************************
H
 *
E                    TNL        10  1               test name length
E                    TVL        22  1               test value length
 *
I              'SET GLOBAL'          C         CMD
 *
I            DS
I                                    B   1   40BIN1
I                                    B   5   80BIN2
I                                    B   9  120BIN3
I                                    B  13  160BIN4
I/COPY BPLIB/QRPGSRC,DSQCOMMR
 *
 * receive the passed variable name and value:
 *
C           *ENTRY    PLIST
C                     PARM           DSQCOM           comms area
C                     PARM           VARNAM 10        variable name
C                     PARM           VARVAL 20        variable value
 *
 * calculate the variable name length and variable value length:
 *
C                     MOVEAVARNAM    TNL
C                     Z-ADD1         X       20         X = last
C           ' '       LOKUPTNL,X                    60  non blank
C   60                SUB  1         X                  character
C  N60                Z-ADD10        X                  in name
```

Figure 20. Example SETA Subprogram (Part 1 of 2)

```
 *
C                     MOVE ''      TVL,1                  set up first
C                     MOVEAVARVAL    TVL,2                  apostrophe
C                     Z-ADD21        Y       20
C           AGAIN     TAG                                   Y = last
C           ' '       COMP TVL,Y                          61  blank
C   61                SUB  1         Y                      character
C   61                GOTO AGAIN
C                     ADD  1         Y                      set up last
C                     MOVE ''      TVL,Y                  apostrophe
 *
 * set the global variables:
 *
C                     CALL DSQCIR
C                     PARM           DSQCOM           comms area
C                     PARM 10        BIN1             command length
C                     PARM CMD       CHAR1  10        SET GLOBAL
C                     PARM 1         BIN2             # variables
C                     PARM X         BIN3             var name length
C                     PARM           VARNAM           variable name
C                     PARM Y         BIN4             var value lngth
C                     PARM           TVL              variable value
C                     PARM DSQVCH    CHAR2   4        CHAR
 *
C                     RETRN
```

Figure 20. Example SETA Subprogram (Part 2 of 2)

SETN subprogram in the Query Management CI

The SETN subprogram performs the SET GLOBAL variable function for a numeric value (nonbinary) to have a decimal point and trailing sign inserted and then passed to the CI. This function is required when creating a query that compares a numeric data item to a constant value (AMOUNT = 525.30-).

The variable name, the variable value, and the number of decimal positions are passed to this program as three separate parameters. The name can be up to 10 characters long, the value must be 15 numeric digits long, and the number of decimal places 2 numeric digits long. The value and decimal positions must be passed as standard numeric data (do not left-justify before passing). The subprogram inserts a decimal point if specified, adds a minus sign if the number is negative, calculates the necessary lengths, strings the information together, and calls the programmable interface. The variable is set as CHAR data type, and control returns to the calling program.

Note: The SETN subprogram is not ended because it may be called a number of times in the session.

```
 ****************************************************************
 *                                                              *
 *      SET GLOBAL COMMAND (NUMERIC - NON BINARY INTEGER)       *
 *                 CPI QUERY INTERFACE HANDLER                  *
 *      -------------------------------------------------       *
 *                                                              *
 *  1) Include member DSQCOMMR contains the communications      *
 *     area to be passed to the query management interface.    *
 *  2) This program handles the SET GLOBAL interface            *
 *     command for variable values to be passed to the         *
 *     interface as numeric data CHAR type.                     *
 *  3) It reads the variable name and value, calculates the    *
 *     length of each, inserts the decimal point and leading    *
 *     negative sign (if required) and passes the information   *
 *     to query management.                                     *
 *  4) The program handles one variable at a time, the length  *
 *     of the variable name can be a maximum of 10 characters   *
 *     and the length of the variable value can be a maximum    *
 *     of 15 numeric digits (plus sign and decimal point).      *
 *                                                              *
 ****************************************************************
H
 *
E                    TNL        10  1               test name length
E                    TVL        17  1               variable value
 *
I              'SET GLOBAL'          C         CMD
 *
I            DS
I                                    B   1   40BIN1
I                                    B   5   80BIN2
I                                    B   9  120BIN3
I                                    B  13  160BIN4
I/COPY BPLIB/QRPGSRC,DSQCOMMR
 *
 * receive the passed variable name and value:
 *
C           *ENTRY    PLIST
C                     PARM           DSQCOM           comms area
C                     PARM           VARNAM 10        variable name
C                     PARM           VARVAL 150       variable value
C                     PARM           VARDEC  20       decimal places
 *
 * calculate the variable name length:
 *
C                     MOVEAVARNAM    TNL
C                     Z-ADD1         X       20         X = last
C           ' '       LOKUPTNL,X                    60  non blank
C   60                SUB  1         X                  character
C  N60                Z-ADD10        X                  in name
```

Figure 21. Example SETN Subprogram (Part 1 of 2)

```
 *
 * set up the variable with decimal point and leading minus sign:
 *
C                     MOVE *BLANKS   TVL              Clear array
C                     MOVE VARVAL    VARCHA 15        Setup as alpha
C           VARVAL    COMP 0                      61  Negative value
C   61                MLLZO'8'       VARCHA           so strip sign
C*
C           VARDEC    IFEQ 0                          * Processing
C                     MOVEAVARCHA    TVL,3            * if value
C   61                MOVE '-'       TVL,2            * has no
C                     GOTO PASS                       * decimals
C                     END                             *
C*
C                     MOVEAVARCHA    TVL,2            * Processing
C   61                MOVE '-'       TVL,1            * if value
C                     Z-ADD16        Y       20       * has
C                     Z-ADD17        Z       20       * decimals
C           AGAIN     TAG                             *
C                     MOVE TVL,Y     TVL,Z            * Move each
C                     SUB  1         VARDEC           * array
C           VARDEC    IFNE 0                          * position
C                     SUB  1         Y                * over one
C                     SUB  1         Z                * place until
C                     GOTO AGAIN                      * decimal
C                     END                             * location
C                     MOVE '.'       TVL,Y            * is reached
C*
C           PASS      TAG
 *
 * set the Global Variables:
 *
C                     CALL DSQCIR
C                     PARM           DSQCOM           comms area
C                     PARM 10        BIN1             command length
C                     PARM CMD       CHAR1  10        SET GLOBAL
C                     PARM 1         BIN2             # variables
C                     PARM X         BIN3             var name length
C                     PARM           VARNAM           variable name
C                     PARM 17        BIN4             var value lngth
C                     PARM           TVL              variable value
C                     PARM DSQVCH    CHAR2   4        CHAR
 *
C                     RETRN
```

Figure 21. Example SETN Subprogram (Part 2 of 2)

RUNQ subprogram in the Query Management CI

The RUNQ subprogram activates the RUN QUERY interface. The query name and form name are passed to the program as a string of 42 characters. The first 21 characters constitute the query name, and the last 21 characters are the form name.

The query name and form name must be left-justified. If the form is not being used, positions 22 to 42 of the string must still be passed, but as blank characters.

The RUNQ subprogram reads the passed query and form names, tests for blank forms, calculates lengths, formats the RUN QUERY command, and calls the programmable interface. After the query is run, the RUNQ subprogram returns control to its calling program.

Note: The RUNQ subprogram is not ended because it may be called a number of times in the session.

```
 *****************************************************************
 *                                                               *
 *        RUN QUERY COMMAND CPI QUERY INTERFACE HANDLER          *
 *        ---------------------------------------------          *
 *                                                               *
 *  1) Include member DSQCOMMR contains the communications       *
 *     area to be passed to the query management interface.      *
 *  2) This program handles the RUN QUERY interface command.     *
 *     It reads the passed query name and form information,      *
 *     reformats it, then passes the information to query        *
 *     management.                                               *
 *                                                               *
 *****************************************************************
H
 *
E                    VAL        59  1               value to pass
 *
IRUNQ        DS
I                                        1  21 QNAM
I                                       22  42 FNAM
I            DS
I                                    B   1   40BIN1
I/COPY BPLIB/QRPGSRC,DSQCOMMR
 *
 * receive the passed run query command information:
 *
C           *ENTRY    PLIST
C                     PARM           DSQCOM           comms area
C                     PARM           RUNQ             query and form
 *
 * prepare the run query command:
 *
C                     MOVEA'RUN'     VAL              Set up RUN
C                     MOVEA'QUERY'   VAL,5            QUERY and
C                     MOVEAQNAM      VAL,11           Query name
C                     Z-ADD12        X       20       Set array index
```

Figure 22. Example RUNQ Subprogram (Part 1 of 2)

```
 *
C           FNAM      IFNE *BLANKS                Only if form
C           ' '       LOKUPVAL,X                60Find next blank
C                     ADD  1         X              leave a space &
C                     MOVEA'(FORM='  VAL,X          insert (FORM=
C           ' '       LOKUPVAL,X                60Find next blank
C                     MOVEAFNAM      VAL,X          form name
C                     END
 *
C           ' '       LOKUPVAL,X                61Find last blank
C   61                SUB  1         X              Last non blank
C  N61                Z-ADD59        X              No blanks left
 *
 * process the run query command:
 *
C                     CALL DSQCIR
C                     PARM           DSQCOM         comms area
C                     PARM X         BIN1           command length
C                     PARM           VAL            command
 *
C                     RETRN
```

Figure 22. Example RUNQ Subprogram (Part 2 of 2)

RUNP subprogram in the Query Management CI

The RUNP subprogram activates the RUN PROC interface. The procedure name is passed to the program as a string of 33 characters. The procedure name must be left-justified. The RUNP subprogram reads the passed procedure name, calculates lengths, formats the RUN PROC command, and calls the programmable interface. After the procedure is run, the RUNP subprogram returns control to the calling program.

Note: The RUNP subprogram is not ended because it may be called a number of times in the session.

```
 ****************************************************************
 *                                                              *
 *         RUN PROC COMMAND CPI QUERY INTERFACE HANDLER         *
 *         --------------------------------------------         *
 *                                                              *
 *  1) Include member DSQCOMMR contains the communications      *
 *     area to be passed to the query management interface.     *
 *  2) This program handles the RUN PROC interface command.     *
 *     It reads the passed procedure name and form information, *
 *     reformats it, calculates the length, then passes the     *
 *     information to query management.                         *
 *                                                              *
 ****************************************************************
H
 *
E                    VAL        42  1               value to pass
 *
I            DS
I                                    B   1   40BIN1
I/COPY BPLIB/QRPGSRC,DSQCOMMR
 *
 * receive the passed run procedure command information:
 *
C           *ENTRY    PLIST
C                     PARM           DSQCOM           comms area
C                     PARM           RUNP   33        procedure name
 *
 * prepare the run procedure command:
 *
C                     MOVEA'RUN'     VAL              Set up RUN
C                     MOVEA'PROC'    VAL,5            PROC and
C                     MOVEARUNP      VAL,10           Procedure name
 *
C                     Z-ADD11        X       20       Set array index
C           ' '       LOKUPVAL,X                    60Find last blank
C   60                SUB  1         X                Last non blank
C  N60                Z-ADD42        X                No blanks left
 *
 * process the run procedure command:
 *
C                     CALL DSQCIR
C                     PARM           DSQCOM           comms area
C                     PARM X         BIN1             command length
C                     PARM           VAL              command
 *
C                     RETRN
```

Figure 23. Example RUNP Subprogram

EXIT subprogram in the Query Management CI

The EXIT subprogram requires no additional parameters to the DSQCOM communications area. When called, it ends the query interface, ends itself, then returns to the program that called it.

```
 ****************************************************************
 *                                                              *
 *           EXIT COMMAND CPI QUERY INTERFACE HANDLER           *
 *           ----------------------------------------           *
 *                                                              *
 *  1) Include member DSQCOMMR contains the communications      *
 *     area to be passed to the query management interface.     *
 *  2) This program handles the EXIT interface command.         *
 *     It passes to query management the command length         *
 *     and command.                                             *
 *                                                              *
 ****************************************************************
H
 *
I            DS
I                                    B   1   40BIN1
I/COPY BPLIB/QRPGSRC,DSQCOMMR
 *
 * receive the passed communications area:
 *
C           *ENTRY    PLIST
C                     PARM           DSQCOM           comms area
 *
 * call the interface and end the session:
 *
C                     CALL DSQCIR
C                     PARM           DSQCOM           comms area
C                     PARM 4         BIN1             command length
C                     PARM 'EXIT'    DATA    4        command
 *
C                     MOVE '1'       *INLR            end program
```

Figure 24. Example EXIT Subprogram

Chapter 8. Exported and Imported Objects in Query Management

You can export certain Query Management objects to manipulate them with an editor or an application or to transport them from one environment to another. The objects can be exported by the same or another query product. You cannot export the contents of a sort sequence table associated with a *QMQRY object. The following sections describe the formats for exporting a query, procedure, and form object.

Note: Query Management queries with the attribute PROMPT are exported in the same way as queries with the attribute SQL. See *Query Manager Use*, for information about prompted queries.

General Object formats in Query Management

This section presents Query Management formats that are common across a number of systems. Other formats, specific to each Query Management object, are discussed in later sections.

Comments in externalized objects in Query Management

An object comment becomes an externalized object when it is displayed either on-line or on printed copy. An object comment is allowed in an externalized Query Management form, query, or procedure object. The comment must be specified as a V record with a field number of 1001 and a maximum length of 50. A comment longer than 50 characters will be truncated. The comment is generated in the externalized object when it is exported. The comment record, if present, must immediately follow the H record. The H record type field must be:

- F for a form.
- Q for a query.
- P for a procedure.

Follow this sequence when *importing* to determine which text description is used on the Query Management object:

- If the COMMENT= option is specified, the value for this option will be used.
- If member text is in the member, the member text will be used.
- If a comment is in the object, this comment will be used.
- If no comment exists, the text description will be blank.

Follow this sequence when *exporting* to determine which text description is written to the externalized Query Management object:

- If the COMMENT= option is specified, the value on this option will be used.
- If the COMMENT= option is not specified, the text description in the Query Management object will be used.

The H and V records are the only parts of the encoded format that are allowed for query or procedure objects. Attempting to use other record types such as T, R, E, and * will generate unpredictable results.

Comments may not span multiple lines and may not be used inside other comments. A comment begins with '/*' and ends with '*/'.

The procedures have object and user comments. An object comment in a procedure must be inside comment delimiters with no intervening blanks between the start comment and the record identifier. User comments are ignored at run time.

The following is an example of a procedure with both an object comment and some user comments:

```
/*H QM4 01 P 01 E V W E R 01 03 90/07/24 13:30 */
/*V 1001 016 SALES PROCEDURE */
'IMPORT QUERY SALES FROM QRYSRC'
'RUN QUERY SALES'  /*Total sales query*/
'PRINT REPORT'  /*Management report*/
```

The following is an example of an object comment in a query:

```
H QM4 01 Q 01 E V W E R 01 03 90/06/38 01:25
V 1001 011 SALES QUERY
SELECT SALARY FROM SALESFILE WHERE
SALARY > 50000
```

External formats in Query Management

The summary below explains the formats for objects. Respective formats for these objects are discussed in later sections.

Procedures
Panel format

SQL Queries
Panel format

Form Encoded format

Panel format in Query Management

This format consists of a number of fixed-length records containing the object as a series of text strings. The object is not formatted in any special way within the records. It remains as composed by your application or as entered from your terminal.

Objects written out in this format have the following attributes:

- Logical record length of 79 bytes
- Fixed length record format.

Encoded format in Query Management

This external format allows you to access those Query Management objects that contain more structure than the simple panel format objects. Use this format only for FORM objects.

Size of the encoded format in Query Management

The encoded format must have record lengths of up to 150 characters.

Records that make up the base encoded format in Query Management

The formats of those records that make up the base encoded format are described below. Other encoded format record types (associated with specific Query Management objects) are discussed in later sections that deal with the external formats of the individual objects.

For each record type there is a description of its purpose, actual contents, format, and a set of notes on its usage. There are also record descriptions that provide a precise and exhaustive list of the possible values for each field in the record. Some of these fields (particularly in the header record) may contain only a single value. This is indeed intentional and often signifies that other values will be allowable for the field in future releases of Query Management.

The base set of record types in the encoded format includes:

Record Type	Descriptive Name
“H”	“Header (“H”) record in Query Management” on page 123
“V”	“Value (“V”) records in Query Management” on page 125
“T”	“Table description (“T”) records in Query Management” on page 127
“R”	“Table row (“R”) records in Query Management” on page 129

Record Type	Descriptive Name
“E”	“End-of-Object (“E”) record in Query Management” on page 131
“*”	“Application data (“*”) record in Query Management” on page 131

Applications should be written to tolerate fields that are not defined here; that is, applications should ignore fields and records that are not defined or understood. Toleration of such fields will allow an application to run with an object generated by a future implementation of Query Management. This design would also allow toleration of an object containing unsupported functions.

Header (“H”) record in Query Management

The H record identifies the contents of an externalized object (an object is an externalized object when it is being displayed either on-line or on hard copy). It contains information describing the characteristics of the object as well as the file format.

The following figures summarize the contents of the header record:

```
     ....+....1....+....2....+....3....+....4....+

FORMAT:    Hdppp rr t oo f u s n a cc ii yy/mm/dd hh:mm
                |  | |  | | | | | |  |  |        |     |
                d  d d  d d d d d d  d  d        d     d

         |______________________________________|
                              |
                              |
                        fixed format

WHERE:     H         indicates this is an object Header record
           d         is the data field delimiter for this record
                     only -- a blank
           ppp       is the product identifier:
                       QM4
           rr        is the product level in which the object
                     was produced:
                       03, 04, 05, etc.
           t         is the type of object in this file:
                       F for form, Q for query, P for procedure
           oo        is the product object level at the time when
                     the given type of object was produced:
                       01, 02, 03, etc.
           f         is the format of the object in this file:
                       E for encoded format
           u         indicates the status of the object:
                       E for contains Errors
                       W for contains Warnings
                       V for Valid
           s         indicates the subset of the object included:
                       W for Whole object
```

Figure 25. Header Record Description (Part 1 of 2)

```
           n         indicates the language of the exported
                     object.
                          E           English
           a         is the action against the item:
                       R for Replace object
           cc        is the length of the control area in the
                     beginning of each following record (including
                     the 1-byte record type):
                       01 for Forms
           ii        is the length of the integer length fields
                     specified in "V" and "T" type records:
                       03 for all objects
           yy/mm/dd  date stamp
           hh:mm     time stamp

EXAMPLE:   H QM4 03 F 03 E V W E R 01 03 89/09/23 15:21

           (QM4 FORM file at "object level" 3, written in the
            encoded format, with no errors or warnings, in entirety
            in English, usable for complete replacement, with 1
            byte of control area, and 3 bytes for integer lengths)
```

Figure 25. Header Record Description (Part 2 of 2)

The following figure summarizes the location and contents of each of the fields in the header record. The field names used here were defined in the previous figure describing the entire header record. Object-specific values are noted as appropriate.

Field	Columns	Possible Values	Required on Input	Default if Blank on Input
H	01	H	yes	
d	02	(blank)	no	(not used on input)
ppp	03-05	QM4	yes	
rr	07-08	03	no	(not used on input)
t	10	F (form Q (query) P (procedure)	yes	
oo	12-13	03	no	current object level
f	15	E	yes	
u	17	E W V	no	(not used on input)
s	19	W	no	(not used on input)
n	21	E (English)	no	(not used on input)
a	23	R	yes	
cc	25-26	01 (form)	no	(not used on input)
ii	28-29	03	no	(not used on input)
yy/mm/dd	31-38	(dates)	no	(not used on input)
hh:mm	40-44	(times)	no	(not used on input)

Figure 26. Header Record Fields

Notes to Figure 26:

- The header record must be the first record in the external file.
- If the record is shorter than its fixed format length, those fields (or portions of fields) left unspecified are assumed to be blank.
- The object level ("oo") is used to denote a change in the externalized format of an object. When a particular level of Query Management changes the external format of an object, the object's level number is incremented. In this way, an application can use this number to determine the format of the object's records.
- The control area (with length "cc") is a fixed area in the beginning of each of the encoded format records (except the header record) that contains control information pertaining to the given record; the control area contains information such as the record type and a record continuation indicator.
- The subset, format, action, control area length, and integer length fields have been included in the header record for future extensions to the encoded format.
- In the future, additional fields will be added to the end of the header record.

Value ("V") records in Query Management

The V record is used to provide a value for a single nontabular field in an object (like a FORM OPTIONS field). It includes the unique field number, the field's value, and its length.

The following figure summarizes the contents of the V record:

```
FORMAT:    Vc..cdf..f l..l v..v
                     |    |    |
                     d    d    d

         |___| |___________|
           |         |
           |         |
    control area   record info

WHERE:     V         indicates this is a Value record
           c..c      is the remainder of the control area for this
                     record
                       (reserved for future use)
           d         is the data field delimiter for this record
                     only -- a blank
           f..f      is the field number
                       e.g. 1201, 1509
           l..l      indicates the length of the data value:
                       an * used instead of a numeric value
                       indicates that the data value is delimited
                       by the end of the record
           v..v      is the data value (in printable form)

EXAMPLES:  V 1511 004 NONE

             (Form field 1511 with length of 4 and value 'NONE')
```

Figure 27. Value Record Description

The following figure summarizes the location and contents of each of the fields in the V record. The field names used here were defined in the previous figure describing the entire V record. Object-specific values are noted as appropriate.

Control Area

Field	Columns	Possible Values	Required on Input	Default if Blank on Input
V	01	V	yes	
c..c	02	(x)	n/a	
x	02	(blank)	n/a	

Figure 28. Value Record Fields (Part 1 of 2)

Remainder of Record

Field	Columns	Possible Values	Required on Input	Default if Blank on Input
d	+01	(blank)	no	
f..f	+02-05	1001-9999	yes	
l..l	+07-09	* 000-999	yes	
v..v	+11-end	(data)	no	(blank)

Figure 28. Value Record Fields (Part 2 of 2)

Notes to Figure 28 on page 126:

- An omitted data value (like end-of-record), or blanks only following the “*” implicitly indicate that a null (the same as a blank) value is to be applied to this field.
- To explicitly set a field to blank, the field must have a specified positive length and a blank data value.
- Fields are set to their default values when the object is updated if
  – The specified length is zero, or
  – No length is specified.
- Query Management issues a warning when it finds a field length of zero to indicate that the default value is set for this field.
- If the specified length is shorter than the supplied data value, Query Management uses the specified length and issues a warning message.
- If the specified length is longer than the supplied data value, Query Management sets the data value without extending beyond the end of the record and issues a warning message.
- IBM is retaining the length field for future V record uses in which an explicit length may be specified (for example, to indicate significant blanks), and for the possibility of V record expansion.

Table description (“T”) records in Query Management

The T record is used to describe the content and format of the table of values that follows. The contents of a T record determine the contents of all R (row) records for this table. A T record indicates which table is being described (by its unique table number), which columns are included (by their unique field numbers), in what order they appear, and the lengths of the values in these columns.

The following figure summarizes the contents of the T record:

```
FORMAT:    Tc..cdt..t n..n m..m f..f l..l f..f l..l  ...
                     |    |    |    |    |    |    |
                     d    d    d    d    d    d    d

         |___|             |_____| |_______| ...
           |                  |        |
           |                  |        |
     control area       repeating field & length pairs


 WHERE:     T         indicates this is a Table description record
           c..c      is the remainder of the control area for this
                     record, consisting of:
                       x
                     where:
                       x     is blank (reserved for future use)
           d         is the data field delimiter for this record
                     only -- a blank
           t..t      is the table number
                       e.g. 1110, 2710
           n..n      is the number of rows ("R" records) in this
                     table:
                       an * used instead of a numeric value
                       indicates that the table consists of all
                       of the "R" records which follow
           m..m      is the number of columns (field and length
                     pairs) in this table
                       e.g. 003, 006
           f..f      is the field number for this column
                       e.g. 1113, 2712
           l..l      is the length of the data values in this
                     column
                       e.g. 005, 012

EXAMPLES:  T 1110   5   2 1112   7 1113  18

             (Form table 1110 containing 5 rows and 2 columns, with
              column 1112 of length 7, and column 1113 of length 18)
```

Figure 29. Table Record Description

The following figure summarizes the location and contents of each of the fields in the T record. The field names used here were defined in the previous figure describing the entire T record. Object-specific values are noted as appropriate.

Control Area

Field	Columns	Possible Values	Required on Input	Default if Blank on Input
T	01	T	yes	
c..c	02	(x)	n/a	
x	02	(blank)	n/a	

Figure 30. Table Record Fields (Part 1 of 2)

Remainder of Record

Field	Columns	Possible Values	Required on Input	Default if Blank on Input
d	+01	(blank)	no	
t..t	+02-05	1001-9999	yes	
n..n	+07-09	* 000-999	yes	
m..m	+11-13	000-999	yes	
f..f	+15-18 +24-27	1001-999	yes	
l..l	+20-22 +29-31 etc.	000-999	yes	

Figure 30. Table Record Fields (Part 2 of 2)

Notes to Figure 30 on page 128:

- If the number of R records following the T record does not exactly match the numeric row count specified in the T record, an error condition results.
- The number of f..f / l..l pairs is limited to the number of columns in the given table.
- The number of columns should agree with the following number of column field numbers and lengths. If not, a warning message is issued and the number of columns used is the number of field numbers/column data value lengths in the T record.
- The order of f..f/l..l pairs is arbitrary.
- All of the R records immediately following a T record (that is, those associated with a single table) must contain values of the exact lengths specified for each column in the T record. Records shorter than the implied length result in blank or blank-padded values.
- Query Management sets columns with a length of zero (or not specified) to their default values when the object is updated.
- Query Management issues a warning message for columns with a specified length of zero to indicate that it set the default value for this column.
- A table with zero rows in it (or not included in the file) has the same effect as applying columns of length zero to the table; Query Management sets all of the columns to their default values.
- 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 (“R”) records in Query Management

The R record is used to provide a set of values for a single row in the current table. It consists of an ordered list of values as described by the associated T record. An R record must exactly match the description of the positions and lengths of the data values, as specified in the T record.

The following figure summarizes the contents of the R record:

```
   FORMAT:    Rc..cdv..v v..v v..v ...
                        |    |    |
                        d    d    d

            |___||____________| ...
              |         |
              |         |
        control area  list of values

WHERE:     R        indicates this is a table Row record
           c..c     is the remainder of the control area for this
                    record, consisting of:
                    x
                    where:
                    x     is blank (reserved for future use)
           d        is the data field delimiter for this record
                    only -- a blank
           v..v     is the data value for this row and column (in
                    printable form)

EXAMPLES:  R       2 SALARY

             (Form row with first column value of '      2' with
             length 7, and second column value of 'SALARY' where
             it's assumed a length of at least 6 was given in the
             T record)
```

Figure 31. Row Record Description

The following figure summarizes the location and contents of each of the fields in the R record. The field names used here were defined in the previous figure describing the entire R record. Object-specific values are noted as appropriate.

Control Area

Field	Columns	Possible Values	Required on Input	Default if Blank on Input
R	01	R	yes	
c..c	02	(x)	n/a	
x	02	(blank)	n/a	

Figure 32. Row Record Fields (Part 1 of 2)

Remainder of Record

Field	Columns	Possible Values	Required on Input	Default if Blank on Input
d	+01	(blank)	no	
v..v	+02-xx +(xx+2)-yy +(yy+2)-zz etc.	(data)	no	(blank)

Figure 32. Row Record Fields (Part 2 of 2)

Notes to Figure 32:

- An R record must immediately follow another R record, or a T record.
- The number of v..v values must exactly 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; it is set to its default value. However, the presence of the field in the T record requires that the R record contain an extra delimiter for this field; a zero-length value results in one delimiter followed by another in the R record.

End-of-Object (“E”) record in Query Management

The E record is used to delimit the end of the object.

The following figure summarizes the contents of the E record:

```
           ....+....1....+....2....+....3....+....

  FORMAT:    Ec..c
          |___|
            |
            |
       control area

  WHERE:     E        indicates this is an End-of-object record
             c..c     is the remainder of the control area for this
                      record
                        (reserved for future use)

  EXAMPLE:   E
```

Figure 33. End-of-Object Record Description

The following figure summarizes the location and contents of each of the fields in the E record. The field names used here were defined in the previous figure describing the entire E record. Object-specific values are noted as appropriate.

Control Area

Field	Columns	Possible Values	Required on Input	Default if Blank on Input
E	01	E	yes	
c..c	02	(x)	n/a	
x	02	(blank)	n/a	

Figure 34. End-of-Object Record Fields

Notes to Figure 34:

- The E record should be the last record in the external file.
- If the record is shorter than its fixed format length, those fields (or portions of fields) left unspecified are assumed to be blank.

Application data (“*”) record in Query Management

The application data record allows you to include your own data associated with the given object in the external file. You may choose to use these as comment records to further describe the object in the file.

The following figure summarizes the contents of the application data record:

```
FORMAT:    *v..v

WHERE:     *         indicates this is an application data record
           v..v      is the data value(s) produced by an appli-
                     cation program (preferably in printable form)

EXAMPLE:   * This is the Form that groups by DEPT.

             (comment record in a Form file)
```

Figure 35. Application Data Record Description

The following figure summarizes the location and contents of each of the fields in the * record. The field names used here were defined in the previous figure describing the entire * record. Object-specific values are noted as appropriate.

Control Area

Field	Columns	Possible Values	Required on Input	Default if Blank on Input
*	01	*	yes	
v..v	02-end	data	no	(not used on input)

Figure 36. Application Data Record Fields

Notes to Figure 36:

- Application data records may appear anywhere in the external file, except ahead of the header record.
- Other than validating the format of the record, the application data record is ignored and has no effect on the input process.
- If the record is shorter than its fixed format length, those fields (or portions of fields) left unspecified are assumed to be blank.

EXPORT and IMPORT file considerations in Query Management

Following are considerations for exporting and importing files:

- Query Management allows export to and import from multiple member source files.
- You must create a source file with a record length that allows for 12 positions for the *Source sequence number* and *Date* fields required in each record.
- Query Management truncates data and generates a warning message when exporting to an existing source file if the file does not have a sufficient data length. The following situations can result in truncation:
  - Exporting an SQL query or procedure object to a file that has a data length less than 79 bytes. Query Management creates the truncation warning message whenever any part of the SQL statement or query command has been truncated.
  - Exporting a query management form object to a file that has an insufficient data length. The minimum allowed data length is 150 bytes (based on the maximum length of an exported encoded-format form record, a Columns table R record). When query management creates a file as a result of the export, it is created with a data length of 150 bytes. Therefore, a data length of 150 bytes is recommended. Query Management issues the truncation warning message when any truncation of data occurs. Since parts of the form are optional, the actual data length required may be less than 150 bytes.
- The confirmation message (when CONFIRM=YES is specified on the EXPORT command) is sent when a member of a source physical file is being replaced. It is not sent if a new member of an already existing source physical file is being created.

- Query Management ignores all columns in the file past column 79 during import of an SQL query or procedure object. A message is generated if columns are ignored.
- Query Management ignores all columns in the file past column 150 during import of a form object. A message is generated if columns are ignored.
- OS/400 does not support files with varying record lengths. Therefore, prior to or during the transfer to the iSeries system for import, the file containing the externalized form object in encoded format must be converted to fixed record format. On export of a form object, query management creates a file with fixed-length record format. The record is padded with blanks from the end of meaningful data to the end of the record. Therefore, before importing to a product that does not support fixed format externalized forms, the file must be converted to varying-length record format.
- Query Management pads the internal representation of each record with blanks up to and including position 79 when importing an SQL query or procedure object, if the input file has a data length less than 79 bytes. If the line contains an open string enclosed in quotation marks, this padding is then included within this string and can cause unexpected results.
- The source physical file that contains the externalized SQL query or procedure can be any size allowed by the iSeries system for a source physical file.

  Note: Although the source physical file can be any size allowed by the system, the results may be truncated when imported if they exceed the maximum allowed limits. For more information on import limits, see the SQL Reference.
- During IMPORT FORM, if the COLUMNS.DATATYPE field is DATE, TIME, or TIMESTAMP and the COLUMNS.USAGE is AVG or SUM, you get an incompatible usage and data type message. COLUMNS.USAGE is not used and the IMPORT ends with a warning.
- If the same applies during RUN QUERY or PRINT REPORT, you receive a message and RUN QUERY or PRINT REPORT fails. (This situation occurs if no COLUMNS.DATATYPE was specified in the externalized form when it was imported.)

Ambiguous date and time literals in Query Management

Literals for date and time that appear in an SQL statement can be of any of the SQL formats or must be a valid OS/400 date or time format. Table 15 shows the SQL time formats.

Table 15. Formats for Representations of Time Data Types

Format Name	Abbreviation	Time Format	Example
International Standards Organization	ISO	HH.MM.SS	13.30.05
IBM USA standard	USA	HH:MM am or pm	1:30 pm
IBM European standard	EUR	HH.MM.SS	13.30.05
Japanese industrial standard Christian era	JIS	HH:MM:SS	13:30:05

Table 16 shows the SQL date formats that can be used for an SQL statement.

Table 16. Formats for Representations of Date Data Types

Format Name	Abbreviation	Date Format	Example
International Standards Organization	ISO	YYYY-MM-DD	1987-10-12
IBM USA standard	USA	MM/DD/YYYY	10/12/1987
IBM European standard	EUR	DD.MM.YYYY	12.10.1987
Japanese industrial standard Christian era	JIS	YYYY-MM-DD	1987-10-12

Country or region specific date and time formats can be interpreted incorrectly. This situation occurs when the query contains global variables and the query has a different date format than the job. Because of this, Query Management recognizes when these ambiguous date and time literals are used and fails with a message that describes the error. You are told in the message how to fix the problem.

Query Management will not recognize ambiguous date and time literals in the query source when it is imported. You are responsible to appropriately document the query source to avoid ambiguity.

Note: The job date and time format or separators should not be changed after a query instance has been started. The IMPORT and RUN QUERY commands will use the date and time information that was in effect at the time of the START.

Variable-length fields in Query Management

A variable-length field consists of 2 bytes followed by the data. Maximum length is 32 740. Variable-length fields can be character (single-byte character set (SBCS) or bracketed double-byte character set (DBCS), graphic DBCS, or hexadecimal. See Appendix A, "DBCS data in Query Management" for considerations and maximum lengths for DBCS data.

When comparing fields for break processing, trailing blanks are not significant. Two strings of different length are equal if they differ only by the number of trailing blanks.

You can define a variable-length field with a maximum length of 32 740 (32 739 if null capable). You can also specify an allocated length for the field. To help improve performance, define the optimum length as the allocated length. Most records are this length or less and will be stored in fixed data storage. If a record is longer than the allocated length, it is stored in variable-length data storage. All the records are stored correctly and the extra read to auxiliary storage is only necessary for occasional long records.

Display format in Query Management

For more information on the externalized procedure and SQL query objects that use the display format, see Chapter 5, "Procedures in Query Management" and Chapter 2, "Query capability in Query Management".

Encoded format in Query Management

Query Management uses the encoded format only for the externalized form object. This section covers each record type in the encoded format as used by query management. Query Management tolerates fields that are not defined in this manual.

Note: Unrecognized fields are lost during import and are not displayed on subsequent exports.

Importing a form object in Query Management

The following rules apply when importing a form object to query management in encoded format:

- The H record must be the first record in the file.
- Record types other than H, V, T, R, and * encountered within the file before the E record are ignored.
- A warning message is issued if unknown record types are encountered.
- Records after the E record are ignored.
- The T record of the Columns table must immediately follow the header or comment V record, and must include a numeric count of the number of rows in the table (an * row count is not allowed).
- Query Management ignores the control area length (cc) field in the H record.
- Query Management assumes that the control area length value on all form object records is 01.
- Query Management uses the delimiter values specified on each of the form object records. Therefore, a nonblank delimiter is allowed.
- The delimiter value on the H record is ignored, since the H record is column-specific.
- The following fields must be in uppercase when a form object is imported:
  - Record identifiers for all records
  - The following in the header record:
    - Product identifier (QRW, QMF*, QM4, and so on)

        - Type of object (F)
        - Format of object (E)
        - Action (R)
    - Data type values (NUMERIC, CHAR) in the R records for the Columns table
    - All the form object keywords and substitution variables
- Duplicate occurrences of data values or tables override previous settings, with one exception. Query Management does not override previous settings if the new object violates the rules established for an object. For example, the number of columns provided for a form cannot be varied after the first Columns table has been processed.
- Combining the original format and the new format for representing the break information is not allowed.
- Object values not included in the input file are set to their default values.

Columns table details in Query Management

The form must contain all of the columns for the underlying data. Form and data mismatch are not detected until the form is applied at RUN or PRINT time.

When the entire Columns table is processed, unspecified fields result in the default values at run time. The default values are applied at run time and are those that were defined at file definition from the table you are querying. Export of a form that was imported with missing Columns table fields results in an externalized form that has the same Columns table fields missing. Query Management allows and uses OS/400 edit codes when the defaults are applied at run time. A warning message is generated at import and export when these fields are not specified.

Query Management allows multiple occurrences of the Columns table but does not allow subsequent occurrences of the table to alter the number of columns.

Query Management supports the import of forms that specify the data type GRAPHIC, although SQL conventions and DB2 UDB for iSeries do not support these data types. Query Management allows importing a form object containing these data types, but an attempt to apply the form results in data and form mismatch errors at run time. A warning message is generated if a form object is imported specifying an unsupported data type. Values in the Columns table that are not recognized or not valid are ignored, and default values are assumed.

Exporting a form object in Query Management

Query Management uses blank delimiters, regardless of the delimiter used on the imported object. The information that was defaulted at import time is exported for all report sections other than Columns. Query Management exports form objects using the new format to describe the break information.

Record format rules in Query Management

For Input (Import):

1. The file may consist of variable or fixed-length records.
2. The minimum allowed logical record length is 23, based upon the required header record format and contents.
3. The record type character (H, V, T, R, E, and *) must appear in the first position of every record.
4. The first “cc” bytes of all records are reserved for control information and therefore require a fixed format (“cc” varies with the object).
5. Every data object (including field numbers, lengths, and values) must be preceded and followed by precisely one delimiter character.

   There are two exceptions to this rule: 1) End-of-record counts as a delimiter, and 2) Zero-length (null) values require a pair of delimiters surrounding the “nonexistent” value.
6. All the fields that are required on input are validated.
7. Duplicate occurrences of any single data value or table override any previous settings, with one exception. Query Management does not override previous settings if the new object violates the rules

established for a particular object, such as when the number of columns provided for a form may not be varied after the first column's table has been processed.

8. Portions of an object not included in the input file are set to their default values.
9. Numeric lengths and table and field numbers may be specified with leading zeros and/or leading blanks, but may not be padded with trailing blanks (other than a single blank delimiter); they must be right justified in their positions in the record.
10. Nonnumeric lengths (an * specification) must be padded with trailing blanks to the current length of integer length values (specified in the header record).
11. Object field values shorter than the data entry field on the object panel are padded with trailing blanks.
12. Object field values longer than the data entry field on the object panel are truncated.
13. If the format of the FORM file is in error, the IMPORT is aborted. In most cases, a single message describes the error and its location in the file.
14. Any records in a file following the E record are ignored.
15. If the input file does not contain an E record, it is assumed that end-of-file implies the end of the object.

For Output (Export):

1. All table and field numbers appear as 4-digit numbers.
2. All lengths are written with leading zeros to a length of 3 digits (as specified in the header record).
3. The blank character is the data object delimiter used in all records.
4. Delimiter characters do not appear as the final character on each record.
5. All reserved fields appear with blanks.
6. If exporting to a pre-allocated data set in QMF™ under MVS™, the data set record format must be variable (V) and the logical record length of the data set must satisfy the exported object's actual maximum output logical record length. If an output data set record length is longer than this pre-allocated maximum, an error results.
7. All table columns appear in numeric field number order in the externalized form, except the column heading field, which is last.
8. An E record appears as the last record in the target file.

Specific Query Object formats in Query Management

The following sections list the Query Management objects and examples of the externalized formats.

"Externalized FORM format in Query Management"

"Externalized PROC and QUERY formats in Query Management" on page 149

Externalized FORM format in Query Management

The FORM is externalized by Query Management in the encoded format. The specification of this format was described earlier in "General Object formats in Query Management" on page 121.

The FORM objects make use of the H, V, T, R, and E records discussed in "Encoded format in Query Management" on page 122.

Deviations from the base encoded format and other special considerations specific to the form are described below.

- The input FORM must contain all of the columns for the underlying data.
- The COLUMNS table must be the first portion of the FORM specified following the header or V record (and may not be altered in size by later duplicate occurrences of this table).

- The COLUMNS table must include a numeric count of the number of rows in the TABLE (rather than the "*" row count specification).
- If the entire COLUMNS table was read in, those fields not specified are set to their default values.
- The COLUMNS.DATA_TYPE column is always written out.

Figure 37 on page 138 shows how the fields described for a FORM object are represented in an externalized form that is used to create a Query Management form (QMFORM) object. You must specify the possible field values, except for text fields, in uppercase characters. The defaults shown in Figure 37 are applied at import time, and appear in the exported form source. For any given text table:

- The 0 in the count range shown indicates the table does not need to be present in the form source when importing. If not present when importing, it will not be present in the exported form source.
- The high number in the count range shown is the highest number that can be used as a *Line* value. The lowest number is 1.
- Text line numbers do not have to be encountered in sequence to import, but are shown in sequence in the exported form source.
- Numbered lines with blank text will be added to fill any gaps before the highest numbered line imported with nonblank text.

```
Record  Table   Field                                    Count    Import
 type   number  number  Description                      range    default
------  ------  ------  -------------------------------  -------- -------
  V              1001   Object Comment
                        *************************************************
                        *                    COLUMN                     *
                        *************************************************


   T     1110           Column Fields                     1-255
                 1112     --Column data type                       -
                 1113     --Column heading                         -
                 1114     --Column usage                           -
                 1115     --Column indent                          2
                 1116     --Column width                           -
                 1117     --Column edit                            -
                 1118     --Column sequence                        n
                              (n is 5 for the 5th R record, for example)
                        *************************************************
                        *                      PAGE                     *
                        *************************************************
   V             1201   Blank lines before heading                 0
   V             1202   Blank lines after heading                  2
   T     1210           Page heading text table           0-999
                 1212     --Page heading line number               -
                 1213     --Page heading align                     CENTER
                 1214     --Page heading text                      -

   V             1301   Blank line before footing
   V             1302   Blank line after footing
   T     1310           Page footing text table           0-999
                 1312     --Page footing line number               -
                 1313     --Page footing align                     CENTER
                 1314     --Page footing text                      -
                        *************************************************
                        *                      FINAL                    *
                        *************************************************
   V             1401   New page for final text                    NO
   V             1402   Put final summary at line                  1
   V             1403   Skip lines before final text               1

   T     1410           Final text table                  0-999
                 1412     --Final text line number                 -
                 1413     --Final text align                       RIGHT
                 1414     --Final text                             -
```

Figure 37. Encoded (Externalized) FORM Field Summary (Part 1 of 2)

```
                     *************************************************
                     *                     OPTIONS                   *
                     *************************************************
V             1501   Detail line spacing                       1
V             1502   Outlining for break columns               YES
V             1503   Default break text                        YES
V             1505   Column wrapped lines kept on page         YES
V             1507   Column heading separators                 YES
V             1508   Break summary separators                  YES
V             1510   Final summary separators                  YES

                     *************************************************
                     *                      BREAK                    *
                     *************************************************
V             3080   Break level indicator                     -
V             3101   New page for break heading                NO
V             3102   Repeat column headings                    NO
V             3103   Blank lines before heading                0
V             3104   Blank lines after heading                 0
T     3110           Break heading table              0-5
V             3112     --Break heading line number             -
V             3113     --Break heading align                   LEFT
V             3114     --Break heading text                    -
V             3201   New page for break footing                NO
V             3202   Put break summary at line                 1
V             3203   Blank lines before footing                0
V             3204   Blank lines after footing                 1
T     3210           Break footing table              0-5
V             3212     --Break footing line number             -
V             3213     --Break footing align                   RIGHT
V             3214     --Break footing text                    -
```

Figure 37. Encoded (Externalized) FORM Field Summary (Part 2 of 2)

Figure 38 on page 140 is an example of an exported Query Management form. A form may be edited and subsequently imported to obtain the desired report formatting. Refer to the list of record types and field numbers to match the field numbers to specific report attributes. The * records, which are valid comment records, are used to explain the meaning of certain parts of the form. The examples given are of certain T records with R records (tables) and V records. The same interpretation applies to records of the same types in the remainder of the form.

```
H QM4 01 F 01 E V W E R 01 03 90/3/19 14:27
* The 'H' record must be the first record in the file as above.
* The columns table must immediately follow the 'H' record unless there
*  are comment records.
* The T record describes the information that follows in the R records
* The field number '1110' identifies the table as the columns table.
* |    The '005' means that there are 5 columns (R records)
* |    |  following the 'T' record.
* |    |   The '006' means that there are 6 field number, field
* |    |   |   pairs in the T record.
* |    |   |   Starting with '1112' are the field number, field
* |    |   |   pairs which describe the values in the R record.
* |    |   |   For example '1112' corresponds to 'Data type'
* |    |   |   |   (see the table of field numbers) and has a
* |    |   |   |   length of 8.
* |    |   |   |   If I wanted to change the indent for a column
* |    |   |   |     I would look in the table of field numbers
* |    |   |   |     and find that the indent identifier is '1115'.
* |    |   |   |                 Counting the field numbers, starting
* |    |   |   |                 |    with '1112', I find that '1115'
* |    |   |   |                 |    is 3rd in the series of field
* |    |   |   |                 |    length pairs and has a field
* |___ |__ |__ |______           |______  width of 6.
* |  | | | | | |      |          |      |
T 1110 005 006 1112 008 1114 007 1115 006 1116 005 1117 005 1113 040
*                 The indent value is the third field over in the
*                 |    R records and contains a 3 for every column.
*                 |    The values are left-justified and separated
*                 |_   by blank delimiters.
*                 | |
R CHAR             3      6     C     NAME
R CHAR     BREAK1  3      6     C     DEPARTMENT
R NUMERIC  SUM     3      6     L     YEARS
R NUMERIC  SUM     3      6     L     SALARY
R NUMERIC  SUM     3      6     L     COMMISSION
* A 'V' record describes a single attribute in the form.
* The '1201' field number is the blank lines before page
* |  heading attribute.
* |    It has a value length of 1.
* |    |  It has a value of 1.
* |__  |_ |_
* |  | | || |
V 1201 001 1
V 1202 001 2
```

Figure 38. Sample Externalized Form (Part 1 of 4)

```
* The following table describes the page heading text.
* The fields used are the line number, alignment, and text, respectively.
* This is the page heading text
T 1210 005 003 1212 004 1213 006 1214 055
R 1    CENTER **********************************************
R 2    CENTER **** &ID                         &DATE    ****
R 3    CENTER ****           COMPANY REPORT             ****
R 4    CENTER ****                                      ****
R 5    CENTER **********************************************
*     The '*' in place of a numeric length indicates to use the
*     |  remainder of the record for the length of the data value.
*     |_
*     |¯|
V 1301 *   1
V 1302 *   2
T 1310 003 003 1312 004 1313 006 1314 055
* This is the page footing text
R 1    CENTER XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
R 2    CENTER XXXXXXXX Internal Use Only XXXXXXXX
R 3    CENTER XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
V 1401 003 YES
V 1402 001 2
T 1410 005 003 1412 004 1413 006 1414 055
R 1    LEFT   *************************************************
R 2    LEFT   ******* &PAGE  &TIME  &DATE              ********
R 3    LEFT   *******          END OF REPORT           ********
R 4    LEFT   *******                                  ********
R 5    LEFT   *************************************************
V 1501 *   1
V 1502 003 YES
V 1503 003 YES
V 1505 003 YES
V 1507 003 YES
V 1508 003 YES
V 1510 003 YES
* The following section shows the break information using the
* new format.
*           The value in the '3080' V record indicates the break
*           |  level, which applies to all of the break information
*          _|  until the next '3080' V record is encountered.
*         |¯|  The break level in this example is 1.
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   CENTER BREAK 1 HEADING
V 3201 002 NO
V 3202 002 NO
V 3203 001 0
V 3204 001 0
T 3210 003 003 3212 004 3213 006 3214 055
```

Figure 38. Sample Externalized Form (Part 2 of 4)

```
R  1    CENTER *******************************************
R  2    CENTER **** BREAK 1 FOOTING Employee ID=&ID   ****
R  3    CENTER *******************************************
* Break level 2 information
V 3080 001 2
V 3101 003 YES
V 3102 003 YES
V 3103 001 0
V 3104 001 0
T 3110 002 003 3112 004 3113 006 3114 055
R 1    CENTER BREAK 2 HEADING
R 2    CENTER ---------------
V 3201 002 NO
V 3202 002 NO
V 3203 001 0
V 3204 001 0
T 3210 003 003 3212 004 3213 006 3214 055
R 1    CENTER *******************************************
R 2    CENTER **** BREAK 2 FOOTING    Employee ID=&1 ****
R 3    CENTER *******************************************
* Break level 3 information
V 3080 001 3
V 3101 003 YES
V 3102 003 YES
V 3103 001 0
V 3104 001 0
T 3110 002 003 3112 004 3113 006 3114 055
R 1    CENTER BREAK 3 HEADING
R 2    CENTER ---------------
V 3201 002 NO
V 3202 002 NO
V 3203 001 0
V 3204 001 0
T 3210 003 003 3212 004 3213 006 3214 055
R 1    CENTER ****************************
R 2    CENTER **** BREAK 3 FOOTING    ****
R 3    CENTER ****************************
* Break level 4 information
V 3080 001 4
V 3101 003 YES
V 3102 003 YES
V 3103 001 0
V 3104 001 0
V 3201 002 NO
V 3202 002 NO
V 3203 001 0
V 3204 001 0
T 3210 003 003 3212 004 3213 006 3214 055
```

Figure 38. Sample Externalized Form (Part 3 of 4)

```
R 1    CENTER ****************************
R 2    CENTER **** BREAK 4 FOOTING    ****
R 3    CENTER ****************************
* Break level 5 information
V 3080 001 5
V 3101 003 YES
V 3102 003 YES
V 3103 001 0
V 3104 001 0
V 3201 002 NO
V 3202 002 NO
V 3203 001 0
V 3204 001 0
* Break level 6 information
V 3080 001 6
V 3101 003 YES
V 3102 003 YES
V 3103 001 0
V 3104 001 0
V 3201 002 NO
V 3202 002 NO
V 3203 001 0
V 3204 001 0
T 3210 003 003 3212 004 3213 006 3214 055
R 1    CENTER +++++++++++++++++++++++
R 2    CENTER +++ BREAK 6 FOOTING +++
R 3    CENTER +++++++++++++++++++++++
* The 'E' record is the last record in the file. Records after
* the 'E' record will be ignored.
E
```

Figure 38. Sample Externalized Form (Part 4 of 4)

You can edit an externalized form object to change your report format. The externalized form layout is in the encoded format, which uses record types and field number identifiers to represent the form. Each field number identifier in the externalized form object represents a different attribute in the report. After making changes to the externalized form, you must import it for the changes to take effect.

Figure 39 on page 144 shows the descriptive names of the encoded form fields.

```
Record    Table      Field
  Type    Number     Number  Description
------   -------     ------  -----------
   V                  1001   Object comment
                             *************************************
                             *** Columns section of the report ***
                             *************************************


   T       1110              Column Fields
                      1112          --Column data type
                      1113          --Column heading
                      1114          --Column usage
                      1115          --Column indent
                      1116          --Column width
                      1117          --Column edit
                      1118          --Column sequence

                             *************************************
                             ***   Page section of the report  ***
                             *************************************
   V                  1201   Blank lines before heading
   V                  1202   Blank lines after heading
   T       1210              Page heading text table
                      1212           --Page heading line number
                      1213           --Page heading align
                      1214           --Page heading text

   V                  1301   Blank line before footing
   V                  1302   Blank line after footing

       T   1310              Page footing text table
                      1312           --Page footing line number
                      1313           --Page footing align
                      1314           --Page footing text
```

Figure 39. Descriptive Names of Encoded Format Form Fields (Part 1 of 2)

```
                          *************************************
                          ***  Final section of the report  ***
                          *************************************
V                  1401   New page for final text
V                  1402   Put final summary at line
V                  1403   Skip lines before final text


T      1410               Final text table
                   1412           --Final text line number
                   1413           --Final text align
                   1414           --Final text

                          ******************************************
                          ** Options fields section of the report **
                          ******************************************
V                  1501   Detail line spacing
V                  1502   Outlining for break columns
V                  1503   Default break text
V                  1505   Column wrapped lines kept on a page
V                  1507   Column heading separators
V                  1508   Break summary separators
V                  1510   Final summary separators
```

Figure 39. Descriptive Names of Encoded Format Form Fields (Part 2 of 2)

A new format exists for the break information in the encoded object. To support the forms that use the original format, Query Management supports both the original format and new format to describe the break information. An attempt to use a combination of the two formats is not allowed and results in ending the import request. All forms are exported using the new format.

Figure 40 is a description of the new format that provides for a break level indicator (V record with field number 3080) to indicate the break level. All of the break information that follows each break level indicator is applied to the break level value in the 3080 V record.

The new format uses one set of field numbers to describe the break heading and footing information, which allows for more efficient future expansion of the number of break levels supported.

```
 Record      Table      Field     Description
  Type       Number     Number
                                  *************************************
                                  * Break fields section of the report *
                                  *************************************
   V                     3080     Break level indicator
   V                     3101     New page for break heading
   V                     3102     Repeat column headings
   V                     3103     Blank lines before heading
   V                     3104     Blank lines after heading
   T         3110                 Break heading table
   V                     3112     --Break heading line number
   V                     3113     --Break heading align
   V                     3114     --Break heading text
   V                     3201     New page for break footing
   V                     3202     Put break at summary line
   V                     3203     Blank lines before footing
   V                     3204     Blank lines after footing
   T         3210                 Break footing table
   V                     3212     --Break footing line number
   V                     3213     --Break footing align
   V                     3214     --Break footing text
```

Figure 40. Preferred Format for Encoded Break Information

Figure 41 on page 147 is a description of the original format for representing the break information in the encoded object. This format uses a unique field number for each of the break attributes. This format cannot be used in combination with the new break format.

Record Type	Table Number	Field Number	Description
			**************************************
			* Break fields section of the report *
			**************************************
V		1601	Break 1: New page for heading
V		1602	Break 1: Repeat column headings
V		1603	Break 1: Blank lines before heading
V		1604	Break 1: Blank lines after heading
T	1610		Break 1: Heading table
V		1612	--Break 1: Heading line number
V		1613	--Break 1: Heading align
V		1614	--Break 1: Heading text
V		1701	Break 1: New page for break footing
V		1702	Break 1: Put break at summary line
V		1703	Break 1: Blank lines before footing
V		1704	Break 1: Blank lines after footing
T	1710		Break 1: Footing table
V		1712	--Break 1: Footing line number
V		1713	--Break 1: Footing align
V		1714	--Break 1: Footing text
V		1801	Break 2: New page for heading
V		1802	Break 2: Repeat column headings
V		1803	Break 2: Blank lines before heading
V		1804	Break 2: Blank lines after heading
T	1810		Break 2: Heading table
V		1812	--Break 2: Heading line number
V		1813	--Break 2: Heading align
V		1814	--Break 2: Heading text
V		1901	Break 2: New page for break footing
V		1902	Break 2: Put break at summary line
V		1903	Break 2: Blank lines before footing
V		1904	Break 2: Blank lines after footing
T	1910		Break 2: Footing table
V		1912	--Break 2: Footing line number
V		1913	--Break 2: Footing align
V		1914	--Break 2: Footing text
V		2001	Break 3: New page for heading
V		2002	Break 3: Repeat column headings
V		2003	Break 3: Blank lines before heading
V		2004	Break 3: Blank lines after heading
T	2010		Break 3: Heading table
V		2012	--Break 3: Heading line number
V		2013	--Break 3: Heading align
V		2014	--Break 3: Heading text

Figure 41. Original Format for Encoded Break Information (Part 1 of 3)

```
V                    2101     Break 3: New page for break footing
V                    2102     Break 3: Put break at summary line
V                    2103     Break 3: Blank lines before footing
V                    2104     Break 3: Blank lines after footing
T         2110                Break 3: Footing table
V                    2112     --Break 3: Footing line number
V                    2113     --Break 3: Footing align
V                    2114     --Break 3: Footing text

V                    2201     Break 4: New page for heading
V                    2202     Break 4: Repeat column headings
V                    2203     Break 4: Blank lines before heading
V                    2204     Break 4: Blank lines after heading
T         2210                Break 4: Heading table
V                    2212     --Break 4: Heading line number
V                    2213     --Break 4: Heading align
V                    2214     --Break 4: Heading text

V                    2301     Break 4: New page for break footing
V                    2302     Break 4: Put break at summary line
V                    2303     Break 4: Blank lines before footing
V                    2304     Break 4: Blank lines after footing
T         2310                Break 4: Footing table
V                    2312     --Break 4: Footing line number
V                    2313     --Break 4: Footing align
V                    2314     --Break 4: Footing text

V                    2401     Break 5: New page for heading
V                    2402     Break 5: Repeat column headings
V                    2403     Break 5: Blank lines before heading
V                    2404     Break 5: Blank lines after heading
T         2410                Break 5: Heading table
V                    2412     --Break 5: Heading line number
V                    2413     --Break 5: Heading align
V                    2414     --Break 5: Heading text

V                    2501     Break 5: New page for break footing
V                    2502     Break 5: Put break at summary line
V                    2503     Break 5: Blank lines before footing
V                    2504     Break 5: Blank lines after footing
T         2510                Break 5: Footing table
V                    2512     --Break 5: Footing line number
V                    2513     --Break 5: Footing align
V                    2514     --Break 5: Footing text
```

Figure 41. Original Format for Encoded Break Information (Part 2 of 3)

```
V                 2601     Break 6: New page for heading
V                 2602     Break 6: Repeat column headings
V                 2603     Break 6: Blank lines before heading
V                 2604     Break 6: Blank lines after heading
T        2610              Break 6: Heading table
V                 2612     --Break 6: Heading line number
V                 2613     --Break 6: Heading align
V                 2614     --Break 6: Heading text

V                 2701     Break 6: New page for break footing
V                 2702     Break 6: Put break at summary line
V                 2703     Break 6: Blank lines before footing
V                 2704     Break 6: Blank lines after footing
T        2710              Break 6: Footing table
V                 2712     --Break 6: Footing line number
V                 2713     --Break 6: Footing align
V                 2714     --Break 6: Footing text
```

Figure 41. Original Format for Encoded Break Information (Part 3 of 3)

Externalized PROC and QUERY formats in Query Management

The PROC and SQL QUERY objects are externalized (exported and saved) in the panel format described in “Panel format in Query Management” on page 122.

IMPORT query considerations for sort sequence in Query Management

Query Management supports sort sequence options and language identifiers when IMPORT QUERY is used to create an SQL query object.

The sort sequence and language identifiers can be specified on the IMPORT QUERY CPI command and in the query source. The value specified on the IMPORT QUERY CPI command takes precedence over values in the query source.

SRTSEQ and LANGID are two separate options on the IMPORT command. They are separate V records in the query source. Query Management allows you to specify one of the attributes on the command and one of the attributes in the source.

Error handling and warning conditions in Query Management

To support imports of queries that were externalized from other systems, Query Management is more tolerant of discrepancies in the source. When LANGID or SRTSEQ are in the source member, the following V record format errors can occur:

- The value length specified was zero or was not specified.
- The specified length is shorter than the data value.
- The specified length is longer than the data value.
- Unrecognized special values for LANGID and SRTSEQ are specified in the source.

If the query was exported from a system at a higher level that supports more special values, the following V record format error can occur:

- The language identifier is not supported.

If the query was exported from a system that supports more language identifiers, the following V record format error can occur:

- The format for a translation table name or a language identifier name is not valid.

Failing conditions in Query Management

Any errors which would be flagged as `V record format errors` are failing conditions if they are present on the command. For example, the statement IMPORT QUERY X FROM Y (SRTSEQ=*INVALID would fail for the following reasons:

- The translation table was not found.
- The library was not found when the translation table was qualified.
- No authority is granted to use the translation table specified.
- No authority is granted to the library containing the translation table.

EXPORT QUERY considerations for sort sequence in Query Management

The EXPORT QUERY CPI command has no changes. However, EXPORT QUERY processing has changed. Two new V records are exported to the query source file member. V record type 5001 is the sort sequence option. V record type 5002 is the language identifier.

```
V 5001 010 *JOB
V 5002 003 ENG
```

The new V records are placed in numeric order immediately after the V 1001 record for the comment/description. The exported information is the information used to create the query. No checking is done to ensure that a user-specified sort sequence table still exists.

If an export is done to convert a Query for iSeries *QRYDFN object into an SQL statement, Query Management exports the sort sequence and language identifier V records to the source file member. The SRTSEQ and LANGID defined in the *QRYDFN are exported in the source file for the following options:

1=Hexadecimal

4=Translation table

5=System sort sequence

If the *QRYDFN object was defined to use one of the following options:

2=Query for iSeries language

3=Define the sequence

then:

- A sort sequence of *HEX is always exported.
- No LANGID V record is exported.

Externalized query description in Query Management

Query Management supports specifying a sort sequence and a language identifier in the query. Figure 42 on page 151 shows the list of V records that can be placed in the query following the H record.

```
Record  Table   Field                                    Count    Import
 type   number  number  Description                      range    default
------  ------  ------  -------------------------------  -------- -------
  V              1001   Object Comment
  V              5001   Sort Sequence Option                       *JOBRUN
                            *JOBRUN
                            *JOB
                            *HEX
                            *LANGIDSHR
                            *LANGIDUNQ
                            *LIBL/tablename
                            *CURLIB/tablename
                            libname/tablename
  V              5002   Language Identifier                        *JOBRUN
                            *JOBRUN
                            *JOB
                            Language Identifier
```

Figure 42. Externalized Query Field Summary

For Query Management to correctly interpret these records as V records, an H record must be the first record of the source member. The V records must immediately follow the H record. Intervening blank records or records other than V are not supported. The V records may be in any order. The last record of each type is used, unless that record is in error. If the last record is in error, the previous valid V record of that type is used.

If a valid sort sequence option V record is not found, the default is used. If a valid language identifier V record is not found, the default is used. A V record warning is sent to the job log for each V record that is not valid, and the import completes with a warning.

If the SRTSEQ or LANGID parameter is specified on the IMPORT QUERY command, the command value takes precedence over the source value. The following statements are true if SRTSEQ or LANGID is specified as an option on the IMPORT QUERY CPI command:

- The V record for the corresponding option in the source member is ignored.
- The V record for the corresponding option in the source member is not verified.

If the SRTSEQ or LANGID parameter is not specified on the command or in the source, the SRTSEQ and LANGID parameters are the default, *JOBRUN.

Chapter 9. Distributed relational database architecture (DRDA) in Query Management

The DRDA function of Query Management allows an application to access multiple remote databases and perform commits and rollbacks to them in a synchronized manner.

Query Management supports two types of connection management:

- Remote Unit of Work (RUW)
- Distributed Unit of Work (DUW)

To select multiple database connections use DUW connection management. To select a connection to a single database use RUW connection management. The Query Management DSQRDBCNNMTH keyword on the START command can be used to select the connection management method to use. In addition, DSQRDBCNNMTH can be set in the query command procedure specified by the DSQSCMD keyword on the START command. For more information on the DSQRDBCNNMTH keyword, see “START in Query Management” on page 45. The connection management method can also be specified on the CL commands STRQMQRY and STRQMPRC using the RDBCNNMTH parameter.

You can specify a remote or local database name either by using the CONNECT and SET CONNECTION commands or the DSQSDBNM keyword on the START command. For more information on the DSQSDBNM keyword, see “START in Query Management” on page 45. For more information on the CONNECT and SET CONNECTION commands, see “CONNECT in Query Management” on page 24 and “SET CONNECTION in Query Management” on page 42.

Using either the CONNECT and SET CONNECTION commands or the DSQDBNM keyword on the start command will result in the application being connected to the specified database. Connection information will be associated with the query instance. If a remote database is not specified using the START or CONNECT commands, the connection information at the time of the START command shows the current server.

Note: All RUN QUERY, ERASE TABLE, and SAVE DATA AS commands will be directed to this connection. A RUN QUERY, ERASE TABLE, or SAVE DATA AS command will fail if the current connection is not the same as the connection associated with the query instance

Remote unit of work (RUW) in Query Management DRDA

With RUW, you will see no difference in the way Query Management currently connects to its remote databases. Your connection management will work as it does today. Under RUW, only one connection to a relational database is allowed. Commits and rollbacks apply to the one allowed connection.

Distributed unit of work (DUW) in Query Management DRDA

This is the default connection management method for Query Management. The DUW connection management method is much more powerful than RUW connection management. Under DUW, multiple relational database connections can be maintained. Commits and rollbacks can be performed to multiple systems in a synchronized manner.

Connection management statements in Query Management DRDA

Query Management supports the following connection statements:

- CONNECT
- COMMIT
- DISCONNECT

- RELEASE
- SET CONNECTION

Only the COMMIT command is allowed in Query Manager Queries. These commands are discussed in detail in Chapter 4, “Commands in Query Management” on page 21.

Connection management in Query Management DRDA

The CONNECT and RELEASE statements control whether a connection is in a held or released state. Released state refers to a condition when a disconnect is to occur for the connection at the next successful commit operation. A released state can be thought of as a pending disconnect. A rollback has no affect on connections. A held state means that a connection is not to be disconnected at the next commit operation. A connection is placed in the held state by the CONNECT statement. A connection is moved to the released state from the held state by the RELEASE statement. A connection in the released state cannot be returned to or placed in the held state. This means that a connection remains in a released state across unit-of-work boundaries when a rollback is issued or a commit results in a rollback.

Regardless of whether a connection is in the held or released state, a connection can also be in the current or in the dormant state. *Current state* refers to a condition when the connection is used for SQL statements that are executed while in this state. *Dormant state* refers to a condition when the connection is suspended. While the connection is in the dormant state, SQL statements use the connection only for commits and rollbacks. The SET CONNECTION and CONNECT statements change the connection for the named relational database to the current state while existing connections are placed in or remain in the dormant state. Only one connection can be in the current state at any given time. When a dormant connection becomes current in the same unit of work, all locks, cursors, and prepared statements are restored to reflect their last use when the connection was current.

The DISCONNECT statement destroys specified connections. Once a connection to a relational database is disconnected, an application must connect to that relational database again if SQL statements need to be directed to the relational database. For protected conversations, the RELEASE statement must be used.

Conversation types in Query Management DRDA

Protected

A protected conversation is used for the connection to a relational database on a remote system. A protected conversation is a conversation that uses two-phase commit protocols to ensure that, even if a failure occurs, updates made on the remote system are synchronized with updates to other remote or local resources.

Not Protected

A conversation that is not protected is used for the connection to a relational database on a remote system. Therefore, if a failure occurs, updates made on the remote system cannot be synchronized with updates to other remote or local resources.

Local

No conversation is used. The connection is to the local relational database. Two-phase commit protocols are used to ensure that, even if a failure occurs, updates made to local system are synchronized with updates to other remote or local resources.

***ARDPGM**

The connection is to be relational database accessed by an application requester driver (ARD) program. SQL requests directed to the relational database are processed by the ARD program specified on the relational database directory level.

Read-only connections in Query Management DRDA

YES

Connection is read-only. If running under commitment control, updates cannot be performed over this connection.

NO

Connection is not read-only. Updates can be performed over this connection. If an update is performed over this connection and the connection has a Conversation Type of Local or Protected, then for this unit-of-work updates are also allowed to all other connections that are not Read-Only and that have a Conversation Type of Local or Protected. Otherwise, if an update is performed over this connection and it has a Conversation Type of Not protected, then updates are only allowed over this connection for the unit-of-work.

Status in Query Management DRDA

HLD

A held state means that a connection is not to be disconnected at the next commit operation. A connection is placed in the held state by the CONNECT statement.

RLS

A released state means that a disconnect is to occur for the connection at the next successful commit operation (a rollback has no affect on connections). A connection is placed in the released state from the held state by the RELEASE statement. A connection in the released state cannot be put into the held state.

Connection management method considerations in Query Management DRDA

The connection management method affects the semantics of the Query Management CONNECT command and affects when previous connections are disconnected. When a CONNECT is performed while running under RUW Connection Management, RUW disconnects the previous connection or connections before performing the connect. When a CONNECT is performed while running under DUW Connection Management, the previous connections are not disconnected.

Using the DSQRDBCNNMTH keyword with START in Query Management DRDA

This keyword specifies which connection management method is to be used by Query Management.

Valid options for DSQRDBCNNMTH are:

***DUW**

Connections to several relational databases are allowed. This is the default value for this variable. Consecutive START or CONNECT commands to additional relational databases does not result in disconnection of previous connections. SET CONNECTION can be used to switch between connections. Read-only connections may result.

If consecutive CONNECT commands to the same database are made, they will fail. However, consecutive START commands to the same database are allowed.

***RUW**

Only one connection to a relational database is allowed. Consecutive START or CONNECT commands to additional relational databases will result in all previous connections being disconnected before a new connection is established. If consecutive START or CONNECT commands to the same database are made, there will be no change in the current connection.

Using the DSQSDBNM keyword with START in Query Management DRDA

If you use DSQSDBNM on the START command, the query instance will be connected to the remote database that you specified. This keyword indicates the remote database to which all SQL operations initiated by query management during the query instance are to be directed. If this keyword is not specified on the START command and the CONNECT command is not used, the connection associated with the query instance is the CURRENT SERVER at the time of the START command. This is an inherited connection. The connection information will be associated with this query instance. All RUN QUERY, ERASE TABLE, and SAVE DATA AS query management commands will be directed to this connection.

Valid options for DSQSDBNM are:

***CURRENT**
The instance inherits the connection associated with the CURRENT SERVER. If the remote database name (rdbname) is in the Relational Database Directory, DSQSDBNM is set to CURRENT SERVER. If the rdbname is not in the Relational Database Directory, DSQSDBNM is set to *NONE. The default value is *CURRENT.

***NONE**
The connection will be made to the local database manager. The local database does not need to be in the Relational Database Directory. If the local database name is in the Relational Database Directory, the DSQSDBNM is set to that name. If the local database name is not in the Relational Database Directory, DSQSDBNM is set to *NONE.

rdbname
The remote database name is a means of identifying a database that can be accessed using DRDA. If you specify a remote database name, you can use the DSQUSER and DSQPASSWORD keywords of the START command to specify the user identification and password for the remote database.

DRDA and activation groups in Query Management

Each query instance has an associated database connection and an associated activation group. Each activation group can have one or more connections associated with it. Query Management allows an application to manage the connections associated with the activation group that is associated with the query instance. The connections can be managed using either the RUW or the DUW connection management methods. RUW connection management method will allow one connection to be maintained and DUW connection management method will allow multiple connections to be maintained.

An application program can only use a query management instance if the activation group associated with the program is also associated with that query management instance. Activation groups cannot share a query management instance. Work done through a query management instance in one activation group, has no affect on work done by a query management instance in another activation group.

DRDA and activation group considerations in Query Management

Only application programs created with ILE C/400 may be associated with a non-default activation group. The DB2 UDB for iSeries Query Manager function is associated with the default activation group. All query management CL commands also run in the default activation group.

The application program that issues the call to the query callable programming interface to process a CONNECT command must remain in the call stack. If it does not, an implicit disconnect occurs when the application program ends. All RUN QUERY and ERASE TABLE commands are directed to the current connection that is the connection associated with the query instance.

Default activation group in Query Management DRDA

Figure 43 on page 157 shows what happens between two programs that are associated with the default activation group. Neither program is an ILE C/400 program.

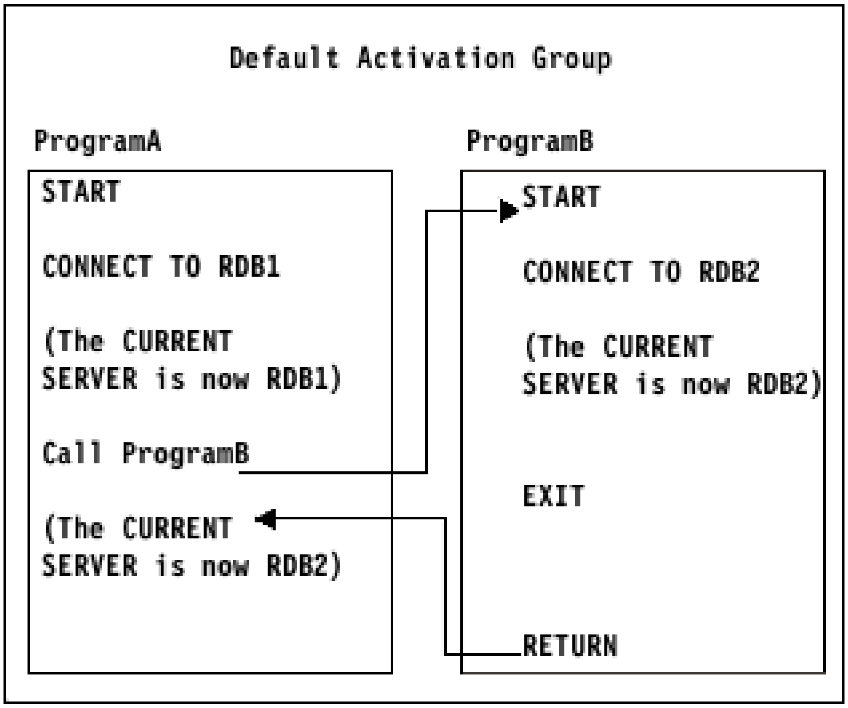


Figure 43. Programs Running in the Default Activation Group

Non-default activation group in Query Management DRDA

Figure 44 on page 158 shows the interaction of programs running in same activation group. ProgramA and ProgramB both run in activation group 2. After ProgramB returns to ProgramA, the CURRENT SERVER is set to RDB2.

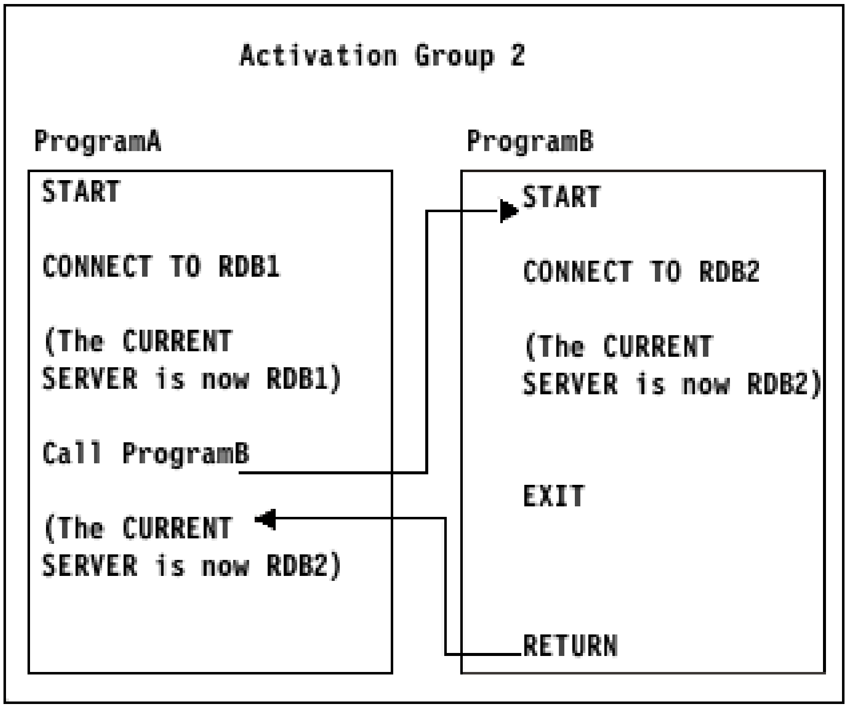


Figure 44. Programs Running in the Same Activation Group

Default and non-default activation groups in Query Management DRDA

Figure 45 on page 159 shows the interaction of programs running the default and a non-default activation group. ProgramA and ProgramB both run in different activation groups. After Program B returns to ProgramA, the CURRENT SERVER associated with ProgramA is still set to RDB1.

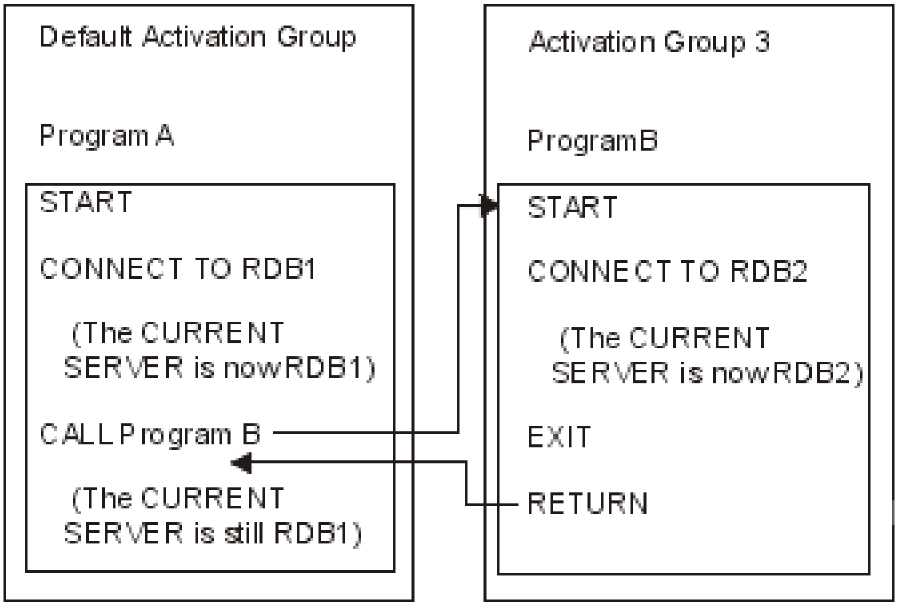


Figure 45. Programs Running in Different Activation Groups

Two non-default activation groups in Query Management DRDA

Figure 46 on page 160 shows the interaction of programs running in different non-default activation groups. ProgramA and ProgramB both run in different activation groups. After Program B returns to ProgramA, the CURRENT SERVER associated with ProgramA is still set to RDB1.

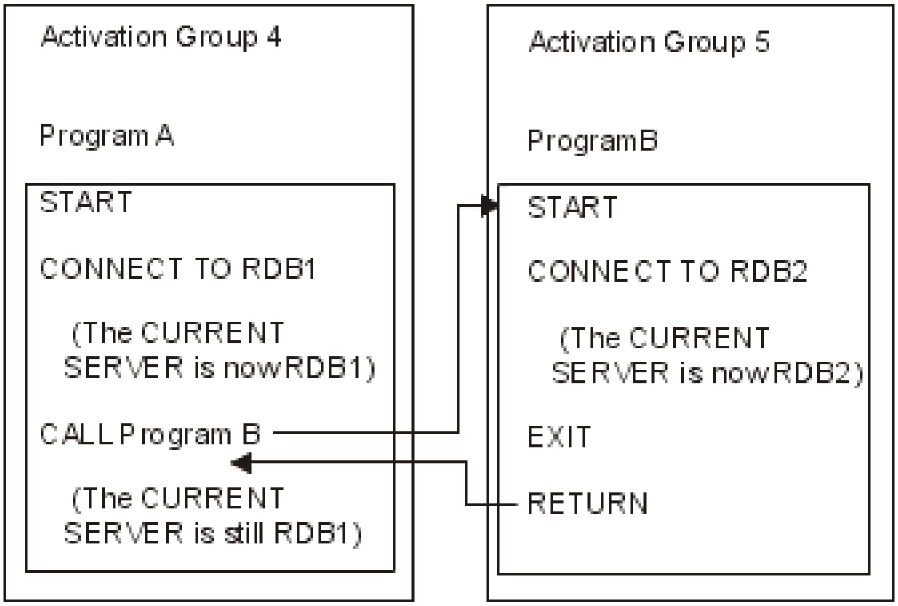


Figure 46. Programs Running in Different Non-Default Activation Groups

Command considerations with DRDA in Query Management

The following commands have special considerations when used with DRDA.

“SAVE DATA AS in Query Management DRDA”

“Other commands in Query Management DRDA”

SAVE DATA AS in Query Management DRDA

You cannot use the SAVE DATA AS command to a remote connection. However, the results of a remote query can be saved locally by switching the remote connection to a local connection before the SAVE DATA AS command. Under RUW connection management, the CONNECT RESET command can be used to establish a local connection. Under DUW connection management, if no previous local connection exists, the CONNECT RESET command can be used to establish a local connection; otherwise, the SET CONNECTION command must be used.

Other commands in Query Management DRDA

The query objects, source files, and print files referred to in the IMPORT, EXPORT, PRINT, and RUN PROC commands are always retrieved locally, regardless of whether the connection is remote. The application process does not need to be in a connectable state to run the IMPORT, EXPORT, PRINT, RUN PROC, GET, and SET commands.

Commitment control in Query Management DRDA

Commitment control is the level at which updates to the database you are working on can be done. You can specify what kind of commitment control you want your Query Management session to have.

Commitment control is specified using the DSQCMTLV keyword on the START command. The default value is NONE. Setting this keyword to a value other than NONE lets query management run all session SQL statements (using the RUN QUERY CPI command) under commitment control. You may then run a Query Management COMMIT command or COMMIT and ROLLBACK SQL statements.

Note: SAVE DATA AS cannot be run with commitment control.

The DSQCMTLV keyword can have the following values:

NONE
Indicates that commitment control is not used. This is the same as the *NONE isolation level.

UR
Specifies that only the updated rows are locked until the end of the transaction. This is the same as the *CHG isolation level.

CS
Specifies that any row that a cursor is positioned on is locked until the cursor position changes. The updated rows are locked until the end of the transaction. This is the same as the *CS isolation level.

RS
Specifies that all of the rows selected or updated are locked until the end of the transaction. This is the same as the *ALL isolation level.

RR
Specifies that all of the rows selected or updated are locked until the end of the unit of work (UOW). This ensures that repeated reads within a unit of work always give the same results. This is the same as the Serializable isolation level.

The keyword that is set for DSQCMTLV on the START command overrides any keyword value set for the DSQCMTLV variable by the query command procedure. If non-default activation groups are never used, commitment control processing will use the job level commit definition and will continue to function as it did prior to Version 2 Release 3 Modification 0 of OS/400.

ILE C/400 considerations in Query Management DRDA

When query management is running with commitment control, a commit definition is used. A process may have multiple commit definitions started. Commit definitions are either associated with the job or with a specific activation group. Application programs associated with the default activation group use the job level commit definition.

If a Query Management COMMIT command or a COMMIT SQL statement is run in the application program:

- Only the work associated with the activation group level commit definition is committed.
- All work done in the application program under commitment control will be committed.
- Work done in other application programs in the same activation group will be committed. Work done in application programs associated with other activation groups will not be committed.
- All connections in a released state will be disconnected.

Understanding commitment control for non-default activation groups in Query Management DRDA

An activation group level commit definition starts when a query instance implicitly starts if the following are true:

- The application program has started a query instance with commitment control
- A job level commit definition was not started

By starting the query instance implicitly, the job level commitment definition is used if all of the following are true:

- The application program is associated with a non-default activation group
- The application program has started a query instance with commitment control
- A job level commit definition was started
- The activation group level commit definition is not started

Running a Query Management COMMIT command or COMMIT SQL statement in the application program only commits the work that is associated with the activation group level commit definition. The work done in other application programs in the default activation group will be committed. The work done in other application programs in the default activation group will only be committed if the commit definition for the activation group started after the job level commit definition started.

Understanding commitment control for default activation groups

If the application program is associated with a default activation group, and:

- If the application program has started a query instance with commitment control
- Whether or not a job level commit definition was started starting the query instance implicitly uses the job level commit definition.

Running a Query Management COMMIT command or COMMIT SQL statement in the application program commits all the work associated with the job level commit definition. The work done in other application programs in the default activation group will be committed. The work done in application programs associated with non-default activation groups will not be committed.

For more information about commitment control and activation groups, see the Advanced Backup and Recovery Guide, SC41-8079.

Remote processing and long column names in Query Management DRDA

Running a query that attempts to select a column name > 10 characters from a table on a remote iSeries system at a release prior to Version 3 Release 1 will fail. You will receive the SQL0107 - &1 too long Maximum 10 characters error.

If a query is run against a non-iSeries system that selects columns > 10 characters, the full column name will be displayed in the report and retained if saving the data to an outfile. When saving the data to an outfile, the first ten characters of the name will be used for the system column name.

Chapter 10. Coded character set identifiers (CCSIDs) in Query Management

A CCSID is a 2-byte (unsigned) integer that uniquely identifies an encoding scheme and one or more pairs of character sets and code pages. CCSID tagged data can be converted so that it looks the same in different languages using the same character set. Data might not look the same without conversion if the code pages differ, since it could contain a hexadecimal value which looks like one character in the first language and another character in the second language.

Import CCSID processing in Query Management

When a Query Manager query or form is imported, it is marked with the CCSID of the file and no CCSID conversion is done. When a Query Manager procedure is imported and the file it is imported to does not exist, the file is created with the job CCSID. If the file does exist and the CCSID does not match the imported from file, the procedure is converted to the CCSID of the file it is imported to.

Export CCSID processing in Query Management

When a Query Manager query, form, or procedure is exported and the source file does not exist, the source file is created using the job CCSID. The source file is converted to the CCSID of the source file if the Query Manager query, form, or procedure has a different CCSID.

Print CCSID processing in Query Management

When a Query Manager query, form, or procedure is printed, it is converted to the job CCSID. When printing a report from a form that uses MIN, MAX or BREAKn, the sort sequence used to run the query may need to be converted to a different CCSID for the report to format. If the user has a job CCSID other than 65535, the sort sequence table must be converted to the job CCSID before the MIN, MAX and BREAKn processing. If the user has a job CCSID of 65535, the sort sequence table must be converted to the CCSID of the column upon which the MIN, MAX and BREAKn processing is done. The results of this conversion impacts the appearance of the printed report.

Sort sequence CCSID processing in Query Management

When using a sort sequence table for data comparisons, the sort sequence table is always converted to the CCSID of the data. If the CCSID conversion is successful, the displayed and printed reports are completed successfully. Problems may arise under the following circumstances:

- Conversion with substitution characters

  Some characters in the source CCSID may not be represented in the CCSID to which the table is being converted. In this case, substitution characters are used when the converted sort sequence table is built. Seemingly inconsistent comparisons may occur when formatted because all characters unknown to the sort sequence table have the same weight. Normal CCSID conversion of the data also converts all unknown characters to the same substitution value.
- Conversion failed

  If the sort sequence table cannot be converted into the appropriate CCSID for doing MIN, MAX, and BREAKn processing, the report formats, but the following message is sent to the job log.

  ```
  QWM1723 - Cannot convert the sort sequence for use on column ColName.
  ```

  This error is only generated for reports which require MIN, MAX or BREAKn processing. Detail values for columns which use MIN, MAX, or BREAKn, use the sort sequence without displaying the conversion error. All field and summary outputs are replaced with a row of question marks (’**?**’s) for the width of the column.

Other considerations in Query Management CCSID

When imported and exporting procedures to source files with different CCSIDs, the procedure is first converted to the job CCSID, then to the source-file CCSID. To avoid this extra conversion, copy the procedure to the other file instead of importing or exporting.

Query Management forms are not converted to the job CCSID when they appear in a report. This may make some parts of the form unrecognizable.

When a Query Manager report is displayed or printed, the data is converted to the job CCSID. When saving data using the SAVE DATA AS command, if the file does not exist it is created using the CCSID of the file where the data originated. If the data is saved to an existing file with a CCSID different from the original file, the data is converted to the CCSID of the file it is being saved to. If the data is displayed then saved, it is converted to the job CCSID when it is displayed and converted to the file CCSID when it is saved. To avoid this extra conversion, specify DISPLAY=NO on the RUN QUERY command or specify *OUTFILE for the output parameter on the STRQMQRY CL command.

Query Management queries, forms, and procedures are converted to the job CCSID when they are printed.

Chapter 11. DB2 UDB for iSeries Query Management Considerations

This chapter describes how query management interacts with other system functions and suggests some techniques to help you work with the product.

Override considerations in Query Management

You can use overrides specified by the Override Database File (OVRDBF) command to redirect a reference to a different file. The following sections discuss some considerations of how overrides are handled when query management processes different types of files.

Tables and views in Query Management

Override considerations for tables and views referred to in the Structured Query Language (SQL) statement during a RUN QUERY command are the same as those used in SQL. The following parameters are processed when you specify an override:

- TOFILE
- MBR
- SEQONLY
- LVLCHK
- INHWRT
- WAITRCD

SQL can process a member other than the first member in a query management query by specifying the desired member with the MBR keyword on the OVRDBF command prior to the RUN QUERY.

The query fails if it selects a member from a file that has an override of MBR(*ALL). For more information about using overrides in an SQL statement, see the SQL Programming Concepts topic in the Information Center.

Tables referred to by ERASE TABLE in Query Management

Overrides are ignored on the ERASE TABLE command.

Tables and views referred by SAVE DATA AS in Query Management

You can direct query management to process a file other than the table or view specified on the command by using the OVRDBF CL command. Overrides are ignored if the file specified on the TOFILE keyword of the OVRDBF command does not exist.

You can save data to a member other than the first member of the file by specifying the desired member on the MBR keyword of the OVRDBF command before issuing the SAVE DATA AS command.

If you issue a SAVE DATA AS command to a file that has an override of MBR(*ALL), the command fails.

The following parameters are processed on the SAVE DATA AS command if an override is specified:

- TOFILE
- MBR
- SEQONLY
- LVLCHK
- INHWRT
- WAITRCD

IMPORT and EXPORT source files in Query Management

Overrides on the source files referred to by an IMPORT or EXPORT command are processed.

The following parameters are processed on an IMPORT or EXPORT command if an override is specified on the source file:

- TOFILE
- MBR

An IMPORT from a source file that has an override of MBR(*ALL) is allowed. The IMPORT processes each record of each member. The members are read in the order in which they are created. The IMPORT completion message lists only the name of the first member processed during the import.

An EXPORT to a source file fails if it has an override of MBR(*ALL).

If an EXPORT refers to a file name that has an override, and the file to which the override is directed does not exist, query management creates the file. The file is named the same as the name specified on the TOFILE keyword on the OVRDBF command. For example, the following CL command was run prior to calling query management:

```
OVRDBF FILE(XYZ) TOFILE(MYLIB/MYFILE)
```

The file MYFILE in MYLIB does not exist. The following query management command results in the creation of a source physical file named MYFILE created in the library MYLIB:

```
EXPORT QUERY MYQUERY TO XYZ
```

A member name specified with an override takes precedence over a member name specified on the command. For example, the following CL command was run prior to calling query management:

```
OVRDBF FILE(XYZ) TOFILE(MYLIB/MYFILE) MBR(MEMBER2)
```

Issuing the following query management command results in the source for the query MYQUERY being put in member MEMBER2 of file MYFILE in library MYLIB:

```
EXPORT QUERY MYQUERY TO XYZ(MEMBER1)
```

Query procedures in Query Management

Overrides are not processed on files referred to as query procedures on RUN PROC, ERASE PROC, PRINT PROC, IMPORT PROC, or EXPORT PROC commands. Overrides of other files processed by query commands in procedures being run with the RUN PROC are processed. Overrides of the source files specified on IMPORT PROC and EXPORT PROC commands are processed.

Query Management cannot process overrides on any files while processing a procedure if those files have the same name as the procedure being run. This rule applies to:

- The source file on an IMPORT PROC or EXPORT PROC command if the source file has the same name as the procedure file.
- The source files on any IMPORT or EXPORT command that is run in a procedure with the same name, or to any procedure nested in a procedure of the same name.

- The file referred to on a SAVE DATA AS command if the command is run in a procedure with the same name, or to any procedure nested in a procedure of the same name.
- A file referred to by the SQL statement in a query if the RUN query is run in a procedure with the same name, or to any procedure nested in a procedure of the same name.

The following is an example of how overrides on PROC statements are processed.

- The following CL commands were run prior to calling query management:

  ```
  OVRDBF FILE(XYZ) TOFILE(MYLIB/MYFILE)
  OVRDBF FILE(ABC) TOFILE(MYLIB/MYFILE)
  ```

- The following commands result in processing the procedure ABC in MYLIB even though the above CL command overrides file ABC to the file MYFILE.

  ```
  RUN PROC MYLIB/ABC
  PRINT PROC MYLIB/ABC
  ERASE PROC MYLIB/ABC
  ```

- The following IMPORT command imports the procedure ABC in MYLIB from the source file MYFILE in MYLIB because the override was not processed for the query procedure, but it was for the source file.

  ```
  IMPORT PROC MYLIB/ABC FROM MYLIB/XYZ
  ```

- The following EXPORT command imports the procedure ABC in MYLIB to the source file ABC in MYLIB because the override was not processed for the query procedure. Because the file that was specified on the source file was the same as the query procedure, overrides were not processed.

  ```
  EXPORT PROC MYLIB/ABC(MEMBER1) TO MYLIB/ABC(MEMBER2)
  ```

- The query called QUERY1 contains the SQL statement:

  ```
  SELECT * FROM MYLIB/ABC A1, MYLIB/XYZ A2 WHERE A1.X=B1.X
  ```

  and the file MYLIB/ABC contains the command `RUN QUERY QUERY1`. The following RUN PROC command runs the query procedure ABC in MYLIB. When the query is run, the data is selected from the files ABC in MYLIB and MYFILE in MYLIB. The overrides for file ABC are not processed during the processing of the query procedure ABC.

  ```
  RUN PROC MYLIB/ABC
  ```

For more information on overrides, see Database Programming and File Management.

Miscellaneous tips and techniques in Query Management

This section describes special applications for query management functions and suggests ways to use other products and system functions to make working with query management easier. Many of the tips and techniques involve using information from Query for iSeries objects; you should be familiar with the information in Chapter 12, “Using Query for iSeries Definition Information in Query Management” before using those tips and techniques.

Printing an object in Query Management

When printing any query management object, create a source file member and edit it. Use the following instructions to complete the printing process using the member created:

- Print a query (QMQRY) object by putting the following statement in the source file member created at the start of the session:

  ```
  'PRINT QUERY libname/queryname (PRINTER= printername'
  ```

  Then run the Start Query Management Procedure (STRQMPRC) CL command against the procedure member to print the contents of the query management query.

- Print a form (QMFORM) object by putting the following statement in the source file member created at the start of the session:

  ```
  'PRINT FORM libname/formname (PRINTER= printername'
  ```

Then run the Start Query Management Procedure (STRQMPRC) CL command to print the contents of the query management form.

- Print a procedure (QMPROC) object by putting the following statement in the source file member created at the start of the session:

  ```
  'PRINT PROC libname/procedurename (PRINTER= printername'
  ```

  Then run the Start Query Management Procedure (STRQMPRC) CL command against the procedure member to print the contents of the query management procedure.
- Print a Query for iSeries QRYDFN object by putting either (or both) of the following statements in the source file member created at the start of the session:

  ```
  'PRINT QUERY libname/queryname (PRINTER= printername'
  or
  'PRINT FORM libname/formname (PRINTER= printername'
  ```

  Then run the Start Query Management Procedure (STRQMPRC) CL command with the ALWQRYDFN=YES keyword against the procedure member to print either the query or form part of the QRYDFN object.

Changing STRQMQRY defaults for QRYDFN use in Query Management

If you prefer to use the WRKQRY command to develop and maintain the query and form information used by query management, it may be convenient to use a copy of the STRQMQRY command that has been changed to use defaults that let you run a QRYDFN object just by naming it. For example, you could create the command STRQRYDFN in the current library for your job by entering the following commands:

```
CRTDUPOBJ OBJ(STRQMQRY) FROMLIB(QSYS) OBJTYPE(*CMD) TOLIB(*CURLIB)
          NEWOBJ(STRQRYDFN)

CHGCMDDFT CMD(*CURLIB/STRQRYDFN) NEWDFT('QMFORM(*QMQRY)')

CHGCMDDFT CMD(*CURLIB/STRQRYDFN) NEWDFT('ALWQRYDFN(*ONLY)')
```

To run QRY1 in the current library (*CURLIB), type `STRQRYDFN *CURLIB/QRY1`, or just `STRQRYDFN QRY1`.

Displaying information about using QRYDFN Objects in Query Management

To read about the system actions taken when there are problems deriving information from QRYDFN objects, display the query management conversion messages using the following command string:

```
DSPMSGD RANGE(QWM2301 QWM2399) DETAIL(*BASIC)
```

The messages that are displayed may contain warnings about unexpected consequences of the system action taken, or suggest ways of avoiding or minimizing problems of the sort diagnosed by the message.

Defining queries with global variables using Query for iSeries

Query for iSeries supports a data or text merge function that involves using dependent QRYDFN objects that cannot be run the same as other QRYDFN objects. These objects are different because record selection tests contain variables (dependent values). You can use query management to run these queries if you assign the correct values to these variables.

When information for a query is derived from a dependent QRYDFN object, dependent values are converted to global variables: `:T01.cusnam` becomes `&T01_CUSNAM`, for example. (You are prompted for a value for `T01_CUSNAM` if you did not specify one on the SETVAR parameter when using the STRQMQRY command to run the query.)

Use the SETVAR parameter to insert the value you want into the WHERE clause of the derived SELECT statement. You could, for example, specify the value "'Smith" and an address like "%Apt%"' for the

T01_CUSNAM variable. (See Appendix C, “Use of quotation marks and apostrophes when setting global variables in Query Management” for examples of setting global variables.)

You can create a CL program and command to improve the prompting, provide possible choices, and restrict or validate the values entered.

Using Query for iSeries to create a QMFORM for an existing QMQRY in Query Management

You can use Query for iSeries to define form information based on the system defaults if running an existing QMQRY object with the system default (*SYSDFT) form does not produce the formatting results you want. The following steps show the procedure for defining form information based on system defaults:

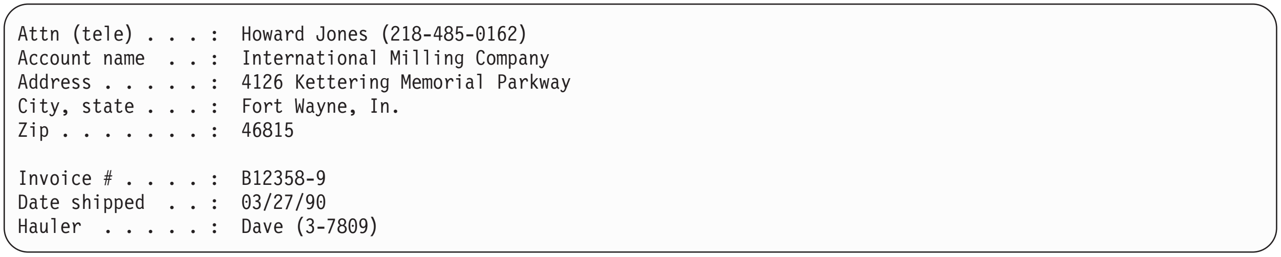
1. Run the QMQRY object and save the data in a table.
2. Use WRKQRY option 1 (Create) to define a QRYDFN object.
   a. Specify the table in which the data was saved as the file selection. (The definition of this table provides the defaults for you to override.)
   b. Consider using the following functions tolerated but not supported by Query for iSeries:
      - &field insertion variables in page text
      - &field insertion variables ended with an underscore character
      - Variables other than break field insertion variables in break or final text
      - &column# insertion variables ended with any nondigit character
   c. Save your definition work as a QRYDFN object.
3. Request query management use the form information from the new QRYDFN object when running the original query, or retrieve the form information and use it to create a QMFORM to use with the QMQRY object.

Displaying data from a single oversized record in Query Management

If you have a QRYDFN object that produces a single-record, multiple-column report that is too wide to see in column-headed format, you can create a form that lets you see the whole report in captioned format. Create the new form using the following steps:

1. Use the WRKQRY command to change the QRYDFN object:
   a. Specify 0 length for all report columns.
   b. Define page heading text with appropriately arranged captions and insert variables. Up to 3 lines are available.
   c. Use page footing text as desired.
2. Retrieve the form source, and make any desired adjustments (for example, additional page heading or footing text lines, left alignment, or spacing).
3. Create the query management form object from the adjusted source.
4. Run the query using the form created to show the complete report.

The following is an example of a displayed single-record report in captioned form:



```
Attn (tele) . . . :  Howard Jones (218-485-0162)
Account name  . . :  International Milling Company
Address . . . . . :  4126 Kettering Memorial Parkway
City, state . . . :  Fort Wayne, In.
Zip . . . . . . . :  46815

Invoice # . . . . :  B12358-9
Date shipped  . . :  03/27/90
Hauler  . . . . . :  Dave (3-7809)
```

Using Query Management or CL Commands in PDM options in Query Management

You may find it convenient to use programming development manager (PDM) to work on lists of QMQRY, QMFORM, or QRYDFN objects. Define options that run CL commands (for example, STRQMQRY or ANZQRY) after substituting library and object names selected when you type the option code beside a list entry. For example:

- Define option SQ to be:

  ```
  STRQMQRY &L/&N QMFORM(*QMQRY) ALWQRYDFN(*ONLY).
  ```

  Then use SQ to display a report using query and form information derived from a selected QRYDFN object without having to remember to override the defaults QMFORM(*SYSDFT) and ALWQRYDFN(*NO).
- Define option Z to be:

  ```
  ANZQRY   &L/&N 99.
  ```

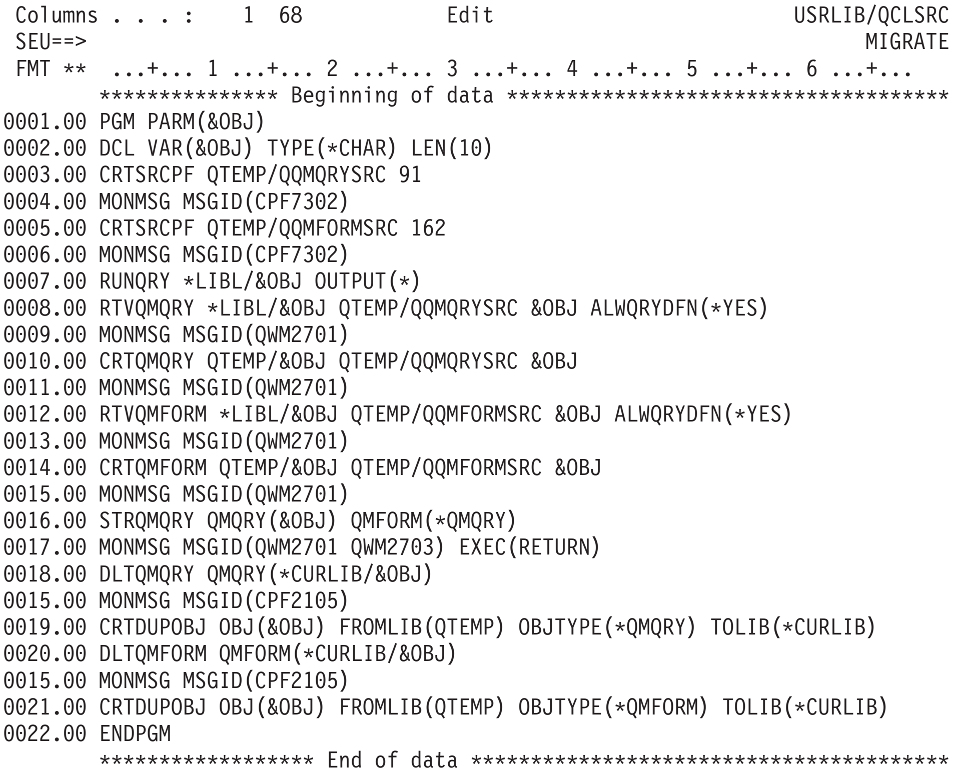
  Then type Z beside all the names in a list of QRYDFN objects to get completion messages. Check these messages to see which QRYDFN objects may need adjustment for satisfactory query management use.

You could define other options to run user-developed CL commands or to call user-developed CL programs that act on query management objects.

Creating a CL program for permanent conversion of a QRYDFN object in Query Management

You may want to create a CL program to convert QRYDFN objects to query management objects if this operation is performed frequently. Define parameters for the program to specify object names and other variables.

Figure 47 on page 171 is an example of the source for a program that converts QRYDFN information into query management objects. This program assumes *LIBL should be searched for the QRYDFN object, and that query management objects should be placed in *CURLIB. It shows the report produced from the QRYDFN object by Query for iSeries, then the report produced from converted objects by query management. If the request is not canceled, the program copies the converted objects from QTEMP to *CURLIB.



```
 Columns . . . :    1  68           Edit                         USRLIB/QCLSRC
 SEU==>                                                                MIGRATE
 FMT **  ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+...
        *************** Beginning of data *************************************
        PGM PARM(&OBJ)
        DCL VAR(&OBJ) TYPE(*CHAR) LEN(10)
        CRTSRCPF QTEMP/QQMQRYSRC 91
        MONMSG MSGID(CPF7302)
        CRTSRCPF QTEMP/QQMFORMSRC 162
        MONMSG MSGID(CPF7302)
        RUNQRY *LIBL/&OBJ OUTPUT(*)
        RTVQMQRY *LIBL/&OBJ QTEMP/QQMQRYSRC &OBJ ALWQRYDFN(*YES)
        MONMSG MSGID(QWM2701)
        CRTQMQRY QTEMP/&OBJ QTEMP/QQMQRYSRC &OBJ
        MONMSG MSGID(QWM2701)
        RTVQMFORM *LIBL/&OBJ QTEMP/QQMFORMSRC &OBJ ALWQRYDFN(*YES)
        MONMSG MSGID(QWM2701)
        CRTQMFORM QTEMP/&OBJ QTEMP/QQMFORMSRC &OBJ
        MONMSG MSGID(QWM2701)
        STRQMQRY QMQRY(&OBJ) QMFORM(*QMQRY)
        MONMSG MSGID(QWM2701 QWM2703) EXEC(RETURN)
        DLTQMQRY QMQRY(*CURLIB/&OBJ)
        MONMSG MSGID(CPF2105)
        CRTDUPOBJ OBJ(&OBJ) FROMLIB(QTEMP) OBJTYPE(*QMQRY) TOLIB(*CURLIB)
        DLTQMFORM QMFORM(*CURLIB/&OBJ)
        MONMSG MSGID(CPF2105)
        CRTDUPOBJ OBJ(&OBJ) FROMLIB(QTEMP) OBJTYPE(*QMFORM) TOLIB(*CURLIB)
        ENDPGM
        ****************** End of data ****************************************
```

Figure 47. CL Source for Permanent Conversion Program

Querying for field values in Query Management

You can create a generic query to display an ordered list of the values used in a field of a particular file. The library, file, and field names can be global variables to be set when the query is run. The following is an example of a SELECT statement that creates a generic query:

```
SELECT DISTINCT &FIELD FROM &LIBRARY/&FILE ORDER BY 1
```

Run the QMQRY object created from this statement to get the list specified in the following SETVAR parameter (this assumes you name the created QMQRY object `qryvalues`, and you have a database file named `staff`, in the library `testdata`, with a field named `dept`):

```
STRQMQRY qryvalues SETVAR((FIELD dept) (LIBRARY testdata) (FILE staff))
```

Get a subset of the values by adding a record selection test when you set the variables:

```
STRQMQRY qryvalues SETVAR((FIELD dept) (LIBRARY testdata)
         (FILE 'staff where dept > 50'))
```

View all the columns (with no records duplicated) by using an asterisk (*) when you set the variables:

```
STRQMQRY qryvalues SETVAR((FIELD '*' ) (LIBRARY testdata) (FILE staff))
```

Make it easier to specify values for the global variables by writing simple CL prompting programs and commands. Set it up so that you can get the list you want by typing:

```
q testdata/staff dept
```

This is a helpful command to use if you are using source entry utility (SEU) to edit query source and want to see which values could be used in tests. SEU permits you to request a window for entering system or user-defined commands.

Passing variable values to a query in Query Management

Global variable names are not necessarily meaningful, and a user being prompted for a value may not know what to type. You can write CL programs and commands to provide meaningful prompting and validation of typed values. Figure 48 and Figure 49 show source statements that you can use to create a program and to create a command for a query that shows an ordered list of values for a specified field in the first member of a specified file. Create a CL program from the program source, then specify it as the command processing program when the command is created from the command source.

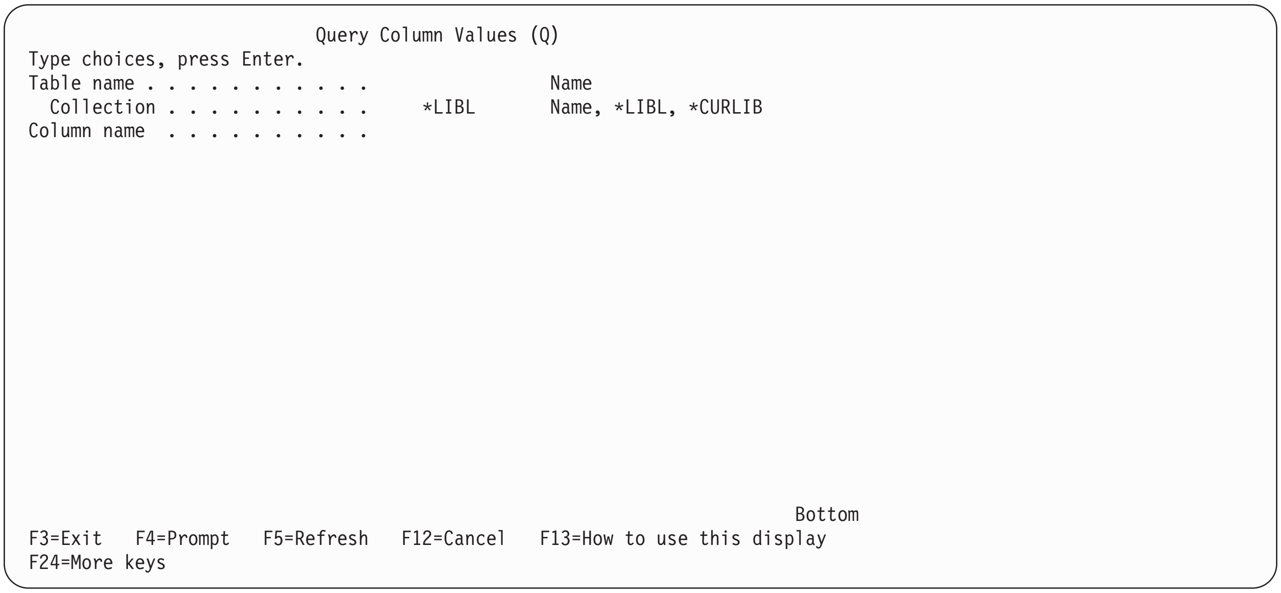
```
PGM PARM(&FILE &FIELD)
DCL VAR(&FILE) TYPE(*CHAR) LEN(20)
DCL VAR(&LIB) TYPE(*CHAR) LEN(10)
DCL VAR(&TABLE) TYPE(*CHAR) LEN(10)
DCL VAR(&FIELD) TYPE(*CHAR) LEN(10)
CHGVAR &LIB %SUBSTRING(&FILE 11 10)
CHGVAR &TABLE %SUBSTRING(&FILE 1 10)
STRQMQRY MYLIB/QRYVALUES SETVAR((LIBRARY &LIB)(FILE &TABLE)(FIELD &FIELD))
ENDPGM
```

Figure 48. CL Source for Global Variable Prompting Program

```
                                                           ******
Q:          CMD        PROMPT('Query Column Values(Q)')
            PARM       KWD(FILE) TYPE(Q1) MIN(1) MAX(1) +
                         PROMPT('Table name')
            PARM       KWD(FIELD) TYPE(*CHAR) LEN(10) +
                         PROMPT('Column name')
Q1:         QUAL       TYPE(*NAME) LEN(10) MIN(1)
            QUAL       TYPE(*NAME) LEN(10) +
                       DFT(*LIBL) +
                       SPCVAL(*LIBL (*CURLIB *CURLIB)) +
                       PROMPT('Collection')
```

Figure 49. CL Source for Global Variable Prompting Command

The following figure is a sample of a user-developed prompting display needed for passing variable values:



```
                           Query Column Values (Q)
Type choices, press Enter.
Table name . . . . . . . . . . .                 Name
  Collection . . . . . . . . . .     *LIBL       Name, *LIBL, *CURLIB
Column name  . . . . . . . . . .























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

Defining a column with no column heading in Query Management

To prevent a column from having a heading, specify *NONE in the leftmost position of the top heading line shown when working on the definition under interactive data definition utility (IDDU) or when you are using the WRKQRY command to define a Query for iSeries QRYDFN object to which query management is to be applied. You can also specify *NONE as the column heading in encoded form source. In either case, the column still has separators unless you eliminate column heading separators from the whole report, or specify *NONE for all column headings. To eliminate column heading separators, retrieve and edit the appropriate field in the form source, and create the form again.

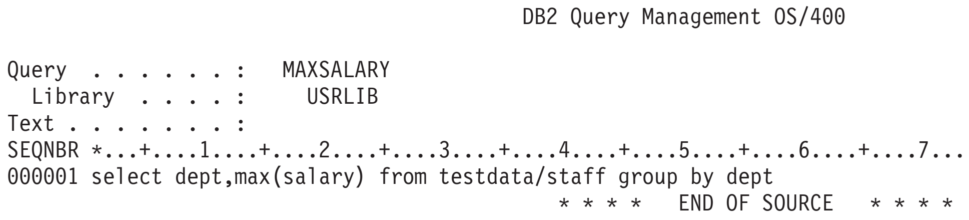
Using Query Management to format an ISQL-developed query

You can use Structured Query Language (SQL) interactively to develop a query that uses any of the supported SQL database functions. By using the Interactive Structured Query Language (ISQL) product, which exists on top of SQL, you can run SQL commands interactively. These functions include subqueries, scalar functions, GROUP BY statements, and others not available through the Query for iSeries prompted interface. The following steps describe how to get an ISQL-developed SELECT statement into a QMQRY object, and how to use Query for iSeries to define information that query management can use to format the displayed or printed output from running this QMQRY object:

1. Specify the Start Structured Query Language (STRSQL) command.
   a. Use ISQL to develop the query you want.
   b. Create a database (collection) to receive the output of this query, or use a previously created collection.
   c. Change the output device for the session to be a database file in the previously created collection. Use a name (for example, QRYPURPOSE) that describes the purpose of the query.
   d. Run the query again to create the file (table) QRYPURPOSE.
   e. Save the query session as member QRYPURPOSE. Remember the file and library names you specify so you can edit the session later for use as query management query source.
   f. Exit the ISQL session.
2. Specify the Start System Entry Utility (STRSEU) command to edit the saved session.
   a. Remove all lines other than those containing the SELECT statement that defines your query.
   b. Add any comments that are needed.
   c. Optionally replace appropriate elements in the SELECT statement with global variables.
   d. Exit SEU, saving the changed member.
3. Use the Create Query Management Query (CRTQMQRY) command to create the QMQRY object QRYPURPOSE from member QRYPURPOSE.
4. Specify the WRKQRY command and choose the *Create* option.
   a. Select file QRYPURPOSE created in the ISQL session.
   b. Specify report column formatting overrides. The defaults shown are the same as ISQL used to show the report, but not the same as query management would use if you ran QRYPURPOSE with the *SYSDFT form. If you want to use the defaults shown, make Query for iSeries treat them as overrides so that they will be saved with the QRYDFN object. (Any change to column headings causes the default to be considered overridden, even if you put back the original default value. The same is true for length, decimal positions, and numeric editing.)
   c. Use edit codes J (numeric values), J$ (currency values), and M (numeric identifiers) to define editing you can convert to Query Management edit codes incorporating any decimal position overrides you specify.
   d. Add any extra formatting you think will improve your report. You can, for example, define final text and overall summaries to appear below columns defined as aggregating scalar functions in the SELECT statement saved in QRYPURPOSE.
   e. Save your formatting choices as QRYDFN object QRYPURPOSE.

5. Optionally retrieve form source from QRYDFN QRYPURPOSE and use it to create QMFORM QRYPURPOSE.
6. Use the STRQMQRY command to run query QRYPURPOSE. Use QMFORM(*QMQRY) and, if you did not create a QMFORM from QRYDFN QRYPURPOSE, specify ALWQRYDFN(*ONLY) to force use of formatting information derived from the QRYDFN object.

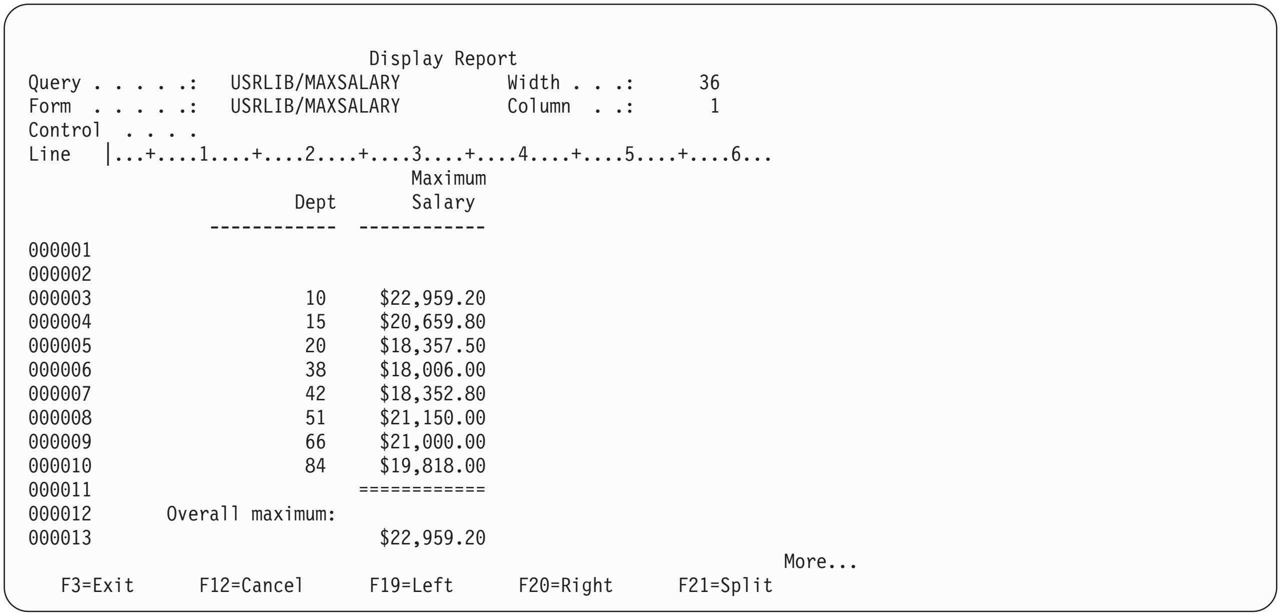
Figure 50 shows an ISQL-developed query.



```
                                                      DB2 Query Management OS/400

 Query  . . . . . . :   MAXSALARY
   Library  . . . . :     USRLIB
 Text . . . . . . . :
 SEQNBR *...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
 000001 select dept,max(salary) from testdata/staff group by dept
                                                         * * * *   END OF SOURCE   * * * *
```

Figure 50. Sample Printed ISQL-Developed QMQRY Object

The following display shows the formatted report produced from the query in Figure 50. This report was created by query management use of form information derived from a QRYDFN object created from the ISQL output file definition.



```
                                  Display Report
 Query . . . . .:   USRLIB/MAXSALARY         Width . . .:      36
 Form  . . . . .:   USRLIB/MAXSALARY         Column  . .:       1
 Control  . . . .
 Line   |...+....1....+....2....+....3....+....4....+....5....+....6...
                                        Maximum
                          Dept           Salary
                 ------------  ------------
 000001
 000002
 000003                     10     $22,959.20
 000004                     15     $20,659.80
 000005                     20     $18,357.50
 000006                     38     $18,006.00
 000007                     42     $18,352.80
 000008                     51     $21,150.00
 000009                     66     $21,000.00
 000010                     84     $19,818.00
 000011                          ============
 000012       Overall maximum:
 000013                          $22,959.20
                                                                            More...
    F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split
```

Using ISQL select report processing with referential constraints in Query Management

You should keep the following in mind when using ISQL in an enviroment using the referential constraint capabilities and features.

- If a selected table has a referential constraint, this constraint is not added to the output file when it is created through ISQL *OUTFILE.
- When the result of a SELECT statement is output to an existing file, the *OUTFILE processing may fail if the output table has a referential constraint that is violated. This error will not be detected until after the existing file has been changed.
- Check-pending errors may occur when the result of a SELECT statement is output to a dependent or parent file when an established/enabled constraint is in check-pending. Check-pending errors should be caught before the existing file is changed.

Using text insertion variables to stack captions on final summaries in Query Management

The following figure shows a final level summary report with the summary values stacked and captioned. It demonstrates that the final summary values can be kept on one page and shown in any desired order instead of being spread over multiple displays or printer files in separate columns.

```
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
       (Salary analysis for 35 employees in department 10)

  Minimum.....:$10,506
  Maximum.....:$22,959
  Average.....:$16,676
  Total......:$583,647

/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
```

Figure 51. Final Level Summary Values as Cover Page and Heading Text Insertions

The report was produced by query management from a single QRYDFN object with the following characteristics:

- Summary-only output form
- No break fields selected
- Final level summaries not suppressed
- Summaries selected
- Length 0 specified for all summary fields to be used as inserts
- Cover page and page heading text containing the desired captions and headings with summary value insert placement indicated by &#; references to output column numbers

Note: `All field widths set to 0` is diagnosed, and no report is produced, when an attempt is made to run the QRYDFN using Query for iSeries.

Using text in combination with tabular layout in Query Management

Text insertion can be used in combination with tabular layout. To produce the example that follows, record selection tests were defined for a QRYDFN object with the characteristics described in the previous tip. Record selection tests were defined to limit the output to a particular customer (specified at run time because of the use of a global variable), and MIN and COUNT functions were defined to supply the inserts for the cover page text used to define the label and for the final text.

```
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
                        (Orders Inquiry)

 Herman B. Wannamaker
 3124 Melrose Ave - Apt 35
 Gooseneck, NY  55945


        TOTAL           AVG           MAX           MIN
       Charges         Price         Price         Price
--------------  ------------  ------------  ------------
    $3,859.72        $79.54      $1024.89         $3.50

Number of transactions:  35
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
```

Figure 52. Final Level Text Insertions with Summary Table

Converting a multiple-level summary-only QRYDFN in Query Management

The following figures show summary-only reports that have both break summaries and final summaries. The query management report, which presents the information in a more readable form, was produced with the lowest-level summaries coming from SQL column functions in a query-defining QRYDFN object, and the other summaries coming from column usages in a second, form-defining QRYDFN object. Here is how it was done. A copy of the original QRYDFN was changed to collapse all levels and suppress final summaries, then saved for use as the query. Another copy was changed to suppress summaries at the lowest level, then saved for use as the form. Then STRQMQRY was applied to the query and form with the use of QRYDFN objects allowed. Note that the overall average is really an average of averages.

```
                      Salary Report Summary, 1989
                 AVERAGE       MINIMUM      MAXIMUM
Job      Years   Salary        Salary       Salary
-----   -------  ---------    ---------    ---------
CLERK        0   $12,655.98   $11,508.60   $13,504.60
             1   $10,988.00   $10,988.00   $10,988.00
             3   $12,689.78   $12,009.75   $13,369.80
             4   $12,258.50   $12,258.50   $12,258.50
             5   $12,769.35   $12,508.20   $13,030.50
             6   $12,482.95   $10,505.90   $14,460.00
             8   $14,252.75   $14,252.75   $14,252.75
                 -----------  -----------  ------------
Overall CLERK:
                 $12,585.33   $10,505.90   $14,460.00

MANAGER      5   $18,383.50   $17,506.75   $19,260.25
             6   $21,150.00   $21,150.00   $21,150.00
             7   $19,889.83   $18,352.80   $22,959.20
             9   $18,555.50   $18,555.50   $18,555.50
            10   $20,162.60   $19,818.00   $20,659.80
            12   $21,234.00   $21,234.00   $21,234.00
                 -----------  -----------  ------------
  Overall MANAGER:
                 $19,895.80   $17,506.75   $22,959.20

SALES        0   $16,808.30   $16,808.30   $16,808.30
             4   $16,858.20   $16,858.20   $16,858.20
             5   $15,454.50   $15,454.50   $15,454.50
             6   $18,488.08   $18,001.75   $19,456.50
             7   $17,333.78   $16,502.83   $17,844.00
             8   $18,171.25   $18,171.25   $18,171.25
             9   $18,674.50   $18,674.50   $18,674.50
            13   $21,000.00   $21,000.00   $21,000.00
                 -----------  -----------  -----------
Overall SALES:
                 $17,848.36   $15,454.50   $21,000.00

                 ===========  ===========  ===========
Overall:
                 $16,679.64   $10,505.90   $22,959.20
06/18/90  09:50:21
```

Figure 53. Form Usages Applied to SQL Column Functions

```
06/21/90  14:23:04  Salary Report Summary, 1989  Page    1
Job    Years       Salary
CLERK      0
           AVG $12,655.98
           MIN $11,508.60
           MAX $13,504.60

CLERK      1
           AVG $10,988.00
           MIN $10,988.00
           MAX $10,988.00

CLERK      3
           AVG $12,689.78
           MIN $12,009.75
           MAX $13,369.80

CLERK      4
           AVG $12,258.50
           MIN $12,258.50
           MAX $12,258.50

CLERK      5
           AVG $12,769.35
           MIN $12,508.20
           MAX $13,030.50

CLERK      6
           AVG $12,482.95
           MIN $10,505.90
           MAX $14,460.00

CLERK      8
           AVG $14,252.75
           MIN $14,252.75
           MAX $14,252.75

CLERK
           Overall CLERK:
           AVG $12,612.61
           MIN $10,505.90
           MAX $14,460.00

MANAGER    5
           AVG $18,383.50
           MIN $17,506.75
           MAX $19,260.25

MANAGER    6
           AVG $21,150.00
           MIN $21,150.00
           MAX $21,150.00
```

Figure 54. Report with Multiple Break Levels - Query for iSeries (Part 1 of 3)

```
06/21/90  14:23:04  Salary Report Summary, 1989  Page    2
Job    Years       Salary
MANAGER    7
           AVG $19,889.83
           MIN $18,352.80
           MAX $22,959.20
MANAGER    9
           AVG $18,555.50
           MIN $18,555.50
           MAX $18,555.50
MANAGER   10
           AVG $20,162.60
           MIN $19,818.00
           MAX $20,659.80

MANAGER   12
           AVG $21,234.00
           MIN $21,234.00
           MAX $21,234.00

MANAGER
           Overall MANAGER:
           AVG $19,805.80
           MIN $17,506.75
           MAX $22,959.20

SALES      0
           AVG $16,808.30
           MIN $16,808.30
           MAX $16,808.30

SALES      4
           AVG $16,858.20
           MIN $16,858.20
           MAX $16,858.20

SALES      5
           AVG $15,454.50
           MIN $15,454.50
           MAX $15,454.50

SALES      6
           AVG $18,488.08
           MIN $18,001.75
           MAX $19,456.50

SALES      7
           AVG $17,333.78
           MIN $16,502.83
           MAX $17,844.00
```

Figure 54. Report with Multiple Break Levels - Query for iSeries (Part 2 of 3)

```
06/21/90  14:23:04  Salary Report Summary, 1989  Page    3
Job    Years       Salary
SALES      8
           AVG $18,171.25
           MIN $18,171.25
           MAX $18,171.25

SALES      9
           AVG $18,674.50
           MIN $18,674.50
           MAX $18,674.50

SALES     13
           AVG $21,000.00
           MIN $21,000.00
           MAX $21,000.00

SALES
           Overall SALES:
           AVG $17,869.36
           MIN $15,454.50
           MAX $21,000.00

           Overall:
           AVG $16,675.64
           MIN $10,505.90
           MAX $22,959.20
* * *  E N D  O F  R E P O R T  * * *
```

Figure 54. Report with Multiple Break Levels - Query for iSeries (Part 3 of 3)

Sorting and subsetting break-level summary groups in Query Management

If you have a QRYDFN or QMQRY object that produces a break-level summary report (the SQL statement contains SQL column functions and a GROUP BY clause), you can create source for a QMQRY object that uses column function values to exclude unwanted groups and that orders the remaining groups based on the summary outcome. You can do this by editing HAVING and ORDER BY clauses in the retrieved source. The following statement produces a list of overdraft totals and counts ordered by account number, for a set of account numbers.

```
SELECT ACCTNUM, COUNT(*), SUM(OVRDRFT) FROM ACCTINFO/OVRDRFTS
GROUP BY ACCTNUM  ORDER BY ACCTNUM
```

The following statement excludes account numbers with overdraft totals within an allowed limit and orders the rest by overdraft total (descending) and number of overdrafts (ascending):

```
SELECT ACCTNUM, COUNT(*), SUM(OVRDRFT) FROM ACCTINFO/OVRDRFTS
GROUP BY ACCTNUM  HAVING SUM(OVRDRFT) > 100
ORDER BY 3 DESC, 2, ACCTNUM
```

Adding SQL function in Query Management

You can add SQL function that is not supported by Query for iSeries by changing a QRYDFN object, or by editing a Query Management source member retrieved from a QRYDFN object. The following list describes how to obtain additional SQL function:

- Use the Work with Query (WRKQRY) command to define functions that are tolerated but not supported by Query for iSeries:
  – &field insertion variables in page text
  – &field insertion variables ended with an underscore character

  - Other than break field insertion variables in break or final text
  - &column# insertion variables ended with any nondigit character.
- Use the Start Source Entry Utility (STRSEU) command to edit retrieved source to add functions that cannot be defined using the WRKQRY command
  1. Retrieve the source from a QRYDFN object or query management query (QMQRY) or form (QMFORM) object using the RTVQMQRY and RTVQMFORM CL commands with the ALWQRYDFN(*YES) parameter.
  2. Edit the source to add the desired functions (see SQL functions that can be added in Query Management, below).
  3. Use the edited source to create a QMQRY or QMFORM object that can be referred to in subsequent Query Management requests using the CRTQMQRY and CRTQMFORM CL commands.
- Use the DB2 UDB for iSeries Query Manager product to change the QMQRY or QMFORM objects to add SQL functions. For more information, see *Query Manager Use* .

SQL functions that can be added in Query Management

You can add the following functions to Query Management query source:

- DISTINCT records instead of ALL records
- Selection of all fields using an asterisk (*)
- SQL naming conventions in the FROM clause
- Column or scalar function as an SQL expression or predicate test value
- NOT as a search condition qualifier
- GROUP BY and HAVING clauses
- Use of parentheses with connectors

You can add the following functions to Query Management form source:

- Character field editing
- Different Query Management edit codes for numeric editing
- Explicit column width control
- FIRST and LAST column functions
- Ability to use more than 9 break columns
- Resequencing in form definition
- Report area spacing
- Summary value placement on a line other than line 2
- Break heading text
- Additional (more than OS/400 limits) text lines
- Text line alignment
- Control of framing (separators, default break text, or outlining, for example)

See the SQL Reference for detailed information about SQL syntax for editing query source and the encoded form layout for editing form source.

Run-time environment in Query Management

For performance reasons, Query Manager does not end its run-time environment at command completion. The result is a faster start when the next Query Manager command is issued. To accomplish this, a small amount of storage is retained. To stop the run-time environment, use the command:

```
ENDEPMENV QQXCPIENV
```

Limits to Query Management processing

Query Management may not be able to process a report in the manner that you prefer. The following sections discuss the limits to query management processing.

The Query Management command

The command string on the callable interface is limited to 256 bytes. The command string in a procedure prior to removing the quotation marks is also limited to 256 bytes.

You can specify a limit of 1000 keywords and variables on a single command. This is a combined total of the keywords or variables specified as part of the command string and keywords or variables specified through the extended interface. Duplicate occurrences of the same keyword or variable count as part of the limit.

SQL query in Query Management

The size of the SQL query statement after blanks and comments are removed is limited to 32 KB minus 1 byte.

Externalized query in Query Management

The following limits exist on the source file that makes up the externalized query:

- Data in columns past column 79 is ignored if the record width is greater than 79 bytes.
- You can specify a maximum of 211 929 lines of source text.

Externalized form in Query Management

The following limits exist on the source file that makes up the externalized form:

- Data in columns past column 150 is ignored if the record width is greater than 150 bytes.
- Since query management allows duplicate information sections in the externalized form object and allows a file to have an override of MBR(*ALL), there is almost no limit on the number of source records in an externalized form that can be handled.

Instances in Query Management

You can specify a maximum of 25 query management instances per process or job that are active at any one time.

Global variables in Query Management

A maximum of 1000 unique global variables can be set for each query management instance.

Procedure limits in Query Management

Query Management supports any file width when running and printing a query procedure. When exporting and importing a query procedure, if the source file that is the target of the command is created, it is created with a width of 79 bytes. If the target of the command already exists and has a width less than the source, data may be truncated. If the target has a width greater than the source, no data is lost, and each record is padded with blanks.

Release-to-release considerations in Query Management

A query management query with the attribute PROMPT can be used with a release prior to OS/400 Version 2 Release 2 if it only uses SQL functions that are valid on the prior release. If a query management query with the attribute of PROMPT cannot be saved for a previous release, you must use the Convert to SQL function of SQL Query Manager to convert the query to an SQL query, which can then be saved for a previous release. The query may have to be changed to run successfully.

Chapter 12. Using Query for iSeries Definition Information in Query Management

Query for iSeries definitions contain specifications for functions that are common to the following:

- Query for iSeries
- Query Management CPI Query

DB2 UDB for iSeries Query Management is able to use this information, saved in Query for iSeries definition (QRYDFN) objects, to produce reports. This chapter describes how to control DB2 UDB for iSeries Query Management use of the information contained in QRYDFN objects and what to do to get the best possible results.

Note: See "Miscellaneous tips and techniques in Query Management" on page 167 for additional information about using Query for iSeries QRYDFN objects.

Query Management is able to derive information for running queries and formatting reports from QRYDFN objects created by Query for iSeries. Conversion to query management query (QMQRY) and form (QMFORM) objects is not required. Refer to the DSQSCNVT parameter of the CPI START command for information about how to take advantage of this capability from a user-written program. Refer to the ALWQRYDFN parameter on the following CL commands for information about how to take advantage of this capability interactively or from a CL program:

- STRQMPRC - run a procedure (a stored sequence of CPI commands)
- STRQMQRY - run a query and either save the data or format a report
- RTVQMQRY - retrieve editable query management query source
- RTVQMFORM - retrieve editable query management form source

Some of the functions that can be specified and saved in a QRYDFN object cannot be transformed into query-management-supported functions, and some cannot be precisely transformed. Except for the case where a SELECT statement grows beyond 32KB in length and it is not possible to use the derived query information, query management uses the derived information and issues no warnings about the actions taken (truncation, and so on) when a transformation problem is encountered. The ANZQRY command provides analysis in the form of messages and on-line help information that suggest ways of dealing with transformation problems.

Different, and possibly unacceptable, output can be produced from derived information even when there are no transformation problems. Query Management may provide different defaults for functions that cannot be specified, or use successfully transformed choices differently. The following sections contain more information about the differences to expect when comparing query management output with that from Query for iSeries, as well as suggestions about what to do to get the best results from the use of information saved in a QRYDFN object.

Because the information saved in a QRYDFN object does not provide complete access to the query management function, and because it is less efficient to derive information from QRYDFN objects than from query management objects, many users will want to convert QRYDFN objects to the corresponding QMQRY and QMFORM objects. See "Creating DB2 UDB for iSeries Query Management Objects from QRYDFN Objects" on page 193 for an explanation of how to retrieve (export) information from QRYDFN objects and use it to create (import) query management objects.

QRYDFN Conversion in Query Management

When a query management query (QMQRY) or form (QMFORM) object is needed for command processing, DB2 UDB for iSeries Query Management ordinarily searches the library or library list for an object of that type with a name that matches the one specified. You can force DB2 UDB for iSeries Query Management either to skip this search or to search the library or library list for a Query for iSeries definition (QRYDFN) with the specified name. If a QRYDFN object is found to match the search, the query or form information required for processing is derived from this object, and appropriate messages and codes are returned to indicate that this has happened. Refer to other topics in this chapter for details about how this information is derived.

DB2 UDB for iSeries Query Management uses information from a QRYDFN object regardless of any problems encountered while deriving that information. No messages about possible defects or functional differences are generated when the QRYDFN object is used.

The derived information is discarded when the request is completed. Permanent conversion to DB2 UDB for iSeries Query Management objects can be done by retrieving the query or form source from a QRYDFN object, and then using that source to create the DB2 UDB for iSeries Query Management object of that type.

Applying DB2 UDB for iSeries Query Management to QRYDFN Objects

DB2 UDB for iSeries Query Management normally uses information only from DB2 UDB for iSeries Query Management objects. You can request that DB2 UDB for iSeries Query Management use Query for iSeries information if DB2 UDB for iSeries Query Management form or query information is not available. You can also prevent DB2 UDB for iSeries Query Management form or query information from being used.

On the START command, specify either:

- DSQSCNVT=YES or
- DSQSCNVT=ONLY

When using the following CL commands:

- STRQMPRC (Start Query Manager Procedure)
- STRQMQRY (Start Query Manager Query)
- RTVQMQRY (Retrieve Query Manager Query)
- RTVQMFORM (Retrieve Query Manager Form)

specify either ALWQRYDFN(*YES) or ALWQRYDFN(*ONLY) to set the DSQSCNVT value.

Note: Query Management resolves names specified for any QMQRY or QMFORM keyword by looking only for QRYDFN objects if the DSQSCNVT value is *ONLY.

The CPI commands shown in the following examples can be coded in a program (you code the START command) or procedure (you code the STRQMPRC command) and applied directly to QRYDFN objects:

- RUN QUERY *myqrydfn*
- RUN QUERY *myqrydfn* (FORM=*myqrydfn*
- RUN QUERY *myqrydfn* (FORM=*myqrydfn2*
- RUN QUERY *myqmqry* (FORM=*myqrydfn*
- RUN QUERY *myqrydfn* (FORM=*myqmform*
- PRINT REPORT (FORM=*myqrydfn*
- PRINT QUERY *myqrydfn*
- PRINT FORM *myqrydfn*
- EXPORT QUERY *myqrydfn*

- EXPORT FORM *myqrydfn*

Note: *myqrydfn* and *myqrydfn2* must have unique names so that query management does not find a query or form object of the same name if the DSQSCNVT value is YES instead of ONLY when it searches for an object to use.

If you do not want Query for iSeries definitions to be used during DB2 UDB for iSeries Query Management processing, allow DB2 UDB for iSeries Query Management to default to DSQSCNVT=NO on the START command or ALWQRYDFN(*NO) on the STRQMPRC, STRQMQRY, RTVQMQRY, or RTVQMFORM CL command. Another way to stop DB2 UDB for iSeries Query Management from using a QRYDFN object is to exclude the library containing the Query for iSeries definition from the library or library list that DB2 UDB for iSeries Query Management searches for the information.

You cannot directly apply DB2 UDB for iSeries Query Management to queries created in a System/36™ environment. However, you can use the Convert System/36 Query (CVTS36QRY) command to convert a System/36 query to a Query for iSeries definition. DB2 UDB for iSeries Query Management information can then be derived from the QRYDFN object converted from the System/36 query.

QRYDFN conversion considerations in Query Management

A complete conversion of all of the choices specified in the QRYDFN may not be possible. Some Query for iSeries functions are not applicable to DB2 UDB for iSeries Query Management reports and queries, and other functions, while similar, cannot be converted without loss or distortion. The information derived from a QRYDFN may be unacceptable for use as a QMQRY or QMFORM. The report or data record output produced from it can have obvious defects, or it can be so different from the report or data record output produced by Query for iSeries that it cannot be used for the same purpose. (See “Using the STRQMQRY command instead of the RUNQRY command in Query Management” on page 195 for a list of situations when the result of using the STRQMQRY command instead of the RUNQRY command will probably be unacceptable.) On the other hand, you may be able to eliminate unacceptable differences by working on the Query for iSeries definition to make simple adjustments.

Report differences in Query Management

The following figures show sample report pages contrasting a printed report produced by Query for iSeries with a printed report produced by query management from information derived from the same Query for iSeries definition. This definition was picked to show what can happen when derived information is used, and is not necessarily representative of what you can expect from most of your Query for iSeries definitions.

```
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\\/\/\/\
  08/11/90  12:25:21                                 Page 1

DEPT       SALARY         COMM      Commission
                                    as percent of salary
                                    ____________________
  10    19,260.25          .00                 .00
        20,010.00          .00                 .00
        21,234.00          .00                 .00
        22,959.20          .00                 .00

   Dept 10 summary information
   MIN  19,260.25          .00
   MAX                     .00

  15    12,258.50       110.10                 .01
        12,508.20       206.60                 .02
        16,502.83     1,152.00                 .07
        20,659.80          .00                 .00

    Dept 15 summary information
    MIN  12,258.50         .00
    MAX               1,152.00

    Year 1988 - salary and commission survey
    MIN  12,258.50         .00
    MAX               1,152.00

* * * END OF REPORT * * *

/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
```

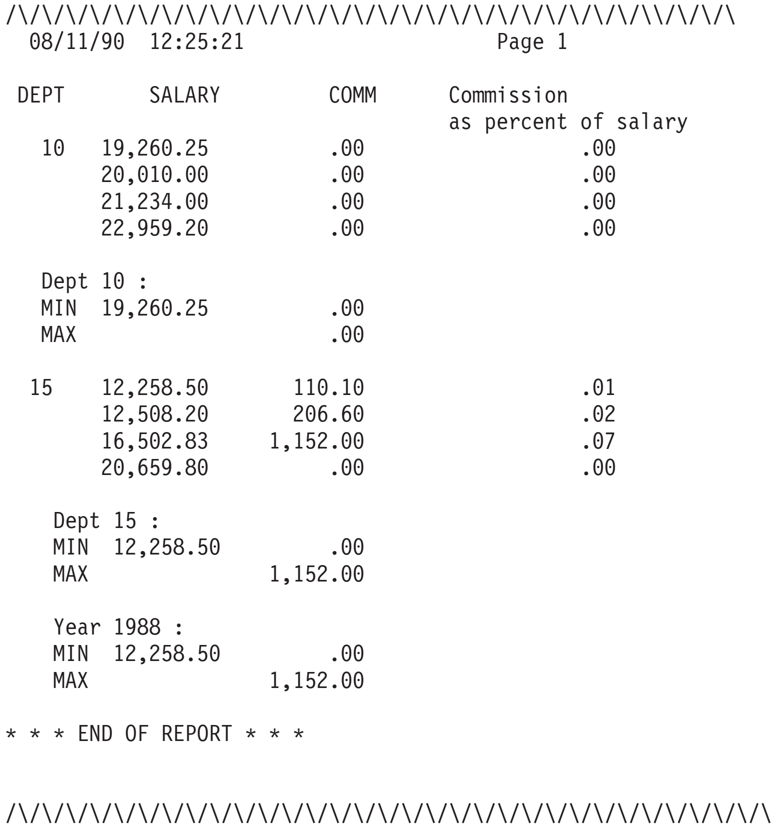
Figure 55. Query for iSeries Output before Adjustment

```
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\\/\/\/\/\/\/\/\/\/\/\
                                                    Commission
                                                as percent of salary
            MIN          MIN       MAX
 DEPT      SALARY        COMM      COMM              _______________
------    --------      -------    ------  --------------------------
  10      19,260.25      .00       .00      .000000000000000000000000
          20,010.00      .00       .00      .000000000000000000000000
          21,234.00      .00       .00      .000000000000000000000000
          22,959.20      .00       .00      .000000000000000000000000
        ------------ --------  --------
Dept 10
          19,260.25      .00       .00

   15     12,258.50    110.10    110.10    .0089815230248399070033038
          12,508.20    206.60    206.60    .0165171647399306055227770
          16,502.83  1,152.00  1,152.00    .0698062089956692276415620
          20,659.80       .00       .00    .0000000000000000000000000
        ------------ --------- ---------
Dept 15
          12,258.50       .00   1,152.00

        ============ ========= ==========
Year 19
          12,258.50       .00   1,152.00


          Page    1         *************************  IBM
08/11/90  11:58:42                                              1
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
```

Figure 56. Query Management Output before Adjustment

The following figures show sample report pages contrasting the printed output from query management with that from Query for iSeries after the following minor changes were made to the QRYDFN object:

- Space before first column overridden to 3
- Length 2 and dec pos 2 specified for the result field size
- Underscores removed from column heading text
- Break and final text condensed
- Page footing text condensed and &PAGE variable removed



```
  08/11/90  12:25:21                     Page 1

 DEPT       SALARY         COMM      Commission
                                     as percent of salary
   10   19,260.25          .00                  .00
        20,010.00          .00                  .00
        21,234.00          .00                  .00
        22,959.20          .00                  .00

   Dept 10 :
   MIN  19,260.25          .00
   MAX                     .00

  15    12,258.50       110.10                  .01
        12,508.20       206.60                  .02
        16,502.83     1,152.00                  .07
        20,659.80          .00                  .00

    Dept 15 :
    MIN  12,258.50         .00
    MAX               1,152.00

    Year 1988 :
    MIN  12,258.50         .00
    MAX               1,152.00

* * * END OF REPORT * * *
```

Figure 57. Query for iSeries Output after Adjustment

```
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\

             MIN            MIN          MAX         Commission
   Dept     SALARY          COMM         COMM   as percent of salary
-------    ---------    ---------   ----------  --------------------
     10    19,260.25         .00          .00                    .00
           20,010.00         .00          .00                    .00
           21,234.00         .00          .00                    .00
           22,959.20         .00          .00                    .00
          -----------   ---------  -----------
Dept 10:
            19,260.25        .00          .00

     15     12,258.50     110.10        110.10                  .01
           12,508.20      206.60        206.60                  .02
           16,502.83    1,152.00      1,152.00                  .07
           20,659.80         .00          .00                   .00
           -----------  ---------   -----------
Dept 15:
            12,258.50        .00      1,152.00

           ===========  =========  ============
Year 1988:
            12,258.50        .00      1,152.00


08/11/90 12:25:04                                                   1
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
```

Figure 58. Query Management Output after Adjustment

Do the following to ensure satisfactory results when using DB2 UDB for iSeries Query Management regularly to run a particular query:

1. Analyze the QRYDFN using the Analyze Query (ANZQRY) command and read all the diagnostic messages produced. Respond appropriately to ANZQRY diagnostic text warnings about actions you may need to take before using the Start Query Management Query (STRQMQRY) command.
2. Use the STRQMQRY command and inspect the output carefully. You may find undiagnosed defects or discover that some of the ANZQRY diagnostic messages can be disregarded.
3. Use the Work with Query (WRKQRY) command to display the QRYDFN object. Check the definition displays for deviations from the compatibility guidelines shown in “Applying QRYDFN option guidelines in Query Management” on page 191. Use the *Change* option to make adjustments to improve your output.
4. Check the list of Query for iSeries and DB2 UDB for iSeries Query Management differences (see “Query for iSeries and DB2 UDB for iSeries Query Management differences” on page 192) and act on those that apply.

Analyzing a QRYDFN in Query Management

Use the Analyze Query (ANZQRY) command to inspect a Query for iSeries definition to find problems that could occur when deriving information for DB2 UDB for iSeries Query Management use. The ANZQRY command returns diagnostic messages that detail the potential differences between how Query for iSeries and DB2 UDB for iSeries Query Management use information derived from the analyzed QRYDFN object.

A completion message shows the highest severity of the potential problems found. If the severity code is greater than 10, the message help may contain warnings about easily overlooked and possibly serious problems involving the system actions stated in the messages.

A low severity code does not necessarily mean there will be no serious problem when you use the derived information, nor does a high severity code mean that the QRYDFN should not be used. Consider the following when using the ANZQRY command:

- Analyze processing checks the information contained in the specified file but ignores any database file overrides in effect. There is no verification that the information saved in the QRYDFN object is still comparable to that found in the file definitions.
- Analyze processing does not predict errors that could occur during conversion, such as a SELECT statement that becomes too large.
- Analyze processing does not predict errors that could occur during run time, such as the following:
  - Structured Query Language (SQL) syntax errors (unacceptable use of expressions or decimal numbers)
  - SQL data errors (missing fields, mismatched type comparisons)
  - Excessive formatted report width
  - Inappropriate field data types for the specified usage or editing.
- Analyze processing ignores the following differences between Query for iSeries and DB2 UDB for iSeries Query Management:
  - Loss of translatable final totals text
  - Loss of leading blanks in column heading lines
  - Different alignment of oversized column headings
  - Breaking of column heading lines at underscore characters
  - No extension of break or final text into space before summary
  - No extension of break or final text past last data column
  - Different summary types on the same line
  - Different defaults for result field size
  - Different algorithms for calculation of result scale and precision
  - Different values from calculations involving data of certain types
  - Different handling of double-byte character set (DBCS) character data comparisons
  - Different handling of native-format dates (DB2 UDB for iSeries Query Management ignores OS/400 date formats like *MDY (2-digit year) as specified in the file definition and uses one of the SQL formats)
- Severity codes do not take into account the number of messages generated by the analyze processing, and can be misleading because a diagnosed problem may not be as severe as indicated. For example, no longer ignoring decimal data errors may not be a problem for a particular query, but it is diagnosed as a severity 30 problem even for a query with no numeric fields.

Inspecting the output in Query Management

In some cases, the only way to detect defects and unacceptable differences is to inspect the output. This may also be the only way to evaluate the actual severity of an ANZQRY-diagnosed problem. Use the STRQMQRY command for DB2 UDB for iSeries Query Management and the RUNQRY command for Query for iSeries to get comparable command output.

Look for the following differences when running a derived query:

- Records not coming from the *LAST member or the member specified by name
- Omission of unmatched records
- Loss of columns or different column order
- Columns added for extra summary functions
- Different record order or selection set
- Different length or precision of result field data columns
- Different values or unexpected numeric overflow in result fields
- Different values or unexpected numeric overflow of sums or averages
- Divide by zero errors for numeric calculations

Look for the following differences when displaying or printing a report using a derived form:

- Text truncation
- Unresolved text insertion variables
- Different editing
- Different report column width
- Different heading or footing alignment
- No line wrapping
- No cover page
- No page number or date and time information in page headings

Applying QRYDFN option guidelines in Query Management

You can avoid many potential problems by following definition display usage guidelines when creating or changing a Query for iSeries definition intended for DB2 UDB for iSeries Query Management use. Use the WRKQRY command to create a new QRYDFN object, or change an existing object and ensure the option fields contain values that are recommended for DB2 UDB for iSeries Query Management compatibility.

Use the following guidelines when specifying values for the following options in a QRYDFN object destined for DB2 UDB for iSeries Query Management use:

- Specify file selections.
  - Select only files with single formats.
  - Select only the *FIRST member.
  - Specify the type of join.
  - Use only matched record joining.
- Define result fields.
  - Use size overrides if an expression involves division.
  - Use size overrides only for numeric column formatting control.
  - Use an underscore character in a column heading only where you want the line to break text.
  - Do not use column headings with data alignment dependencies.
  - Do not use multiple-line figures in column headings.
  - Do not use a line for column heading separator characters.
  - Use SUBSTR (not a formatting override) to reduce character field size.
- Select and sequence fields.
  - Make specific field selections.
  - Do not select more than 255 fields.
- Select records.
  - Do not use LIKE to test a result field defined using || or SUBSTR.
  - Use dependent value syntax where a global variable is desired.
- Select sort fields.
  - Do not sort a character field unless EBCDIC sequencing is acceptable.
- Select collating sequence.
  - Select EBCDIC sequencing.
- Specify report column formatting.
  - Allow space for break or final text to the left of summaries.
  - Override the column heading if overriding the length of a file field.
  - Use an underscore character in a column heading only where you want the line to break text.
  - Do not use column headings with data alignment dependencies.
  - Do not use multiple-line figures in column headings.

  - Do not use a line for column heading separator characters.
  - Do not omit break fields.
  - Override editing when overriding numeric file field size.
- Define numeric field editing.
  - Use edit code or numeric editing choices.
- Specify edit code.
  - Use only J and M edit codes.
  - Do not use the modifier for the asterisk fill option.
  - Do not use the modifier for a floating currency symbol with M specified as the edit code.
- Describe numeric field editing.
  - Use a description that can be converted to a Query Management edit code.
  - Do not request suppression of zero values.
  - Do not request asterisk fill.
  - Do not request a single leading zero.
- Select report summary functions.
  - Do not request more than one summary function per field unless you want columns added for the extra functions.
  - Do not request a summary function for a break field unless you want a column added for that function.
- Define report breaks.
  - Do not break on a result field if you want detail records to be omitted.
  - Define only one level if you want detail records to be omitted.
- Format report breaks.
  - Do not use break text if the first report field has a summary function.
  - Use any report field name for a break text variable.
  - Define final text to replace the *Final Totals* default.
  - Omit level 0 summaries if some other level is defined and you want detail records to be omitted.
- Select output type and output form.
  - Do not define a query for producing summary-only database file output.
  - Put run-time device options in CL commands or procedures.
  - Use line spacing if desired.
  - Do not define a cover page unless the output will be final summaries only.
  - Do not use line wrapping.
  - Do not use more than 55 characters for page heading or footing text.
  - Use any report field name for a page text variable.
- Specify processing options.
  - Do not request numeric overflow truncation.
  - Do not request ignoring decimal data errors.
  - Do not request other than ignoring substitution characters.

Query for iSeries and DB2 UDB for iSeries Query Management differences

Knowing some things about the differences between Query for iSeries and query management may help you get better results from using information derived from Query for iSeries definition objects:

- The default printer form width used by query management is smaller than that used by Query for iSeries. Parallel reports may be produced for a report that Query for iSeries kept on one printer form width.
- Queries defined for files created by other queries may not be usable with files created by query management from information derived from these other queries. This is especially likely if result fields are included in the output file definitions.
- Query for iSeries cannot run a definition if an expression or value contains a decimal point delimiter that is not recognized. Query Management can run such a definition because valid decimal point delimiters are converted to the delimiter indicated by the QDECFMT system value.
- Query Management database processing truncates decimal positions after division of unscaled values. For example, the value calculated for 2/3 is 0. For this reason, conversion processing makes sure each numeric constant followed by a +, –, /, *, or (in an expression contains a decimal point delimiter. This causes the size of the expression column to include 15 digits to the right of the decimal point. Expressions involving division of unscaled numeric fields (no constants) will probably cause unexpected results. Using a numeric result field name as the length or offset for the SUBSTR function or as the argument for the DATE function may cause errors.
- Use of result field names or numbers with decimal point delimited where SQL expects positive whole numbers may cause run-preventing errors.
- Query Management database processing rules for double-byte character set (DBCS) data are different from those applied for Query for iSeries. Use of concatenated strings to test other than open strings should be avoided.
- Because form text that is too long is truncated, conversion processing compresses blank strings. Text may still be lost because query management does not allow it to extend beyond limits determined by column and summary value placement.
- Query for iSeries uses any native formatting specified for date, time, and timestamp fields, but query management uses a similar SQL format instead.
- If a QRYDFN object intended to produce summary-only output is suitably defined, query management will convert query information into an SQL SELECT statement with a GROUP BY clause and a field selection list containing grouping columns or column functions (for report break control fields and any selected summary functions). If no report break is defined, only column functions are used. If a QRYDFN object is not suitably defined for this mode of conversion, detail records will not be omitted. SQL summary conversion will cause a run-preventing error if any of the following applies:
  - Summary function other than COUNT for a result field with no file field reference
  - MIN or MAX for a character field wider than SQL allows
  - Total break fields larger than SQL allows
  - Total break and summary columns larger than SQL allows
- Summary-only output from query management cannot be added to a file created by an earlier Query for iSeries running of the same QRYDFN object. This is because Query for iSeries creates the file format with extra fields for holding level and overflow feedback information, which query management does not provide or leave room for.
- The CCSID of the QRYDFN, not the CCSID of the job, becomes the CCSID of any retrieved objects.

Note: Refer to “Conversion details in Query Management” on page 198 for complete conversion details.

Creating DB2 UDB for iSeries Query Management Objects from QRYDFN Objects

The following steps represent a typical way in which a QRYDFN could be permanently converted to DB2 UDB for iSeries Query Management form object and DB2 UDB for iSeries Query Management query object:

1. Run the ANZQRY command on the selected QRYDFN. The messages produced give you suggestions for adjusting the object before attempting to convert it to DB2 UDB for iSeries Query Management objects.
2. Create separate source file members-one for externalized queries and one for externalized forms.
3. Use the WRKQRY command to change and save the resolved QRYDFN object if there have been any changes to the file definitions referred to. You can also use this command to change the object to improve the conversion or to add functions tolerated but not supported by Query for iSeries.
4. Use CL or DB2 UDB for iSeries Query Management commands (RTVQMQRY or EXPORT QUERY, for example) to extract the usable information.
5. Edit the externalized objects if you want to add functions not available through the Query for iSeries prompted interface.
6. Use CL or DB2 UDB for iSeries Query Management commands (CRTQMQRY or IMPORT QUERY, for example) to create DB2 UDB for iSeries Query Management objects.

Figure 59 illustrates how a Query for iSeries QRYDFN is converted for use as query management query and query management form objects.

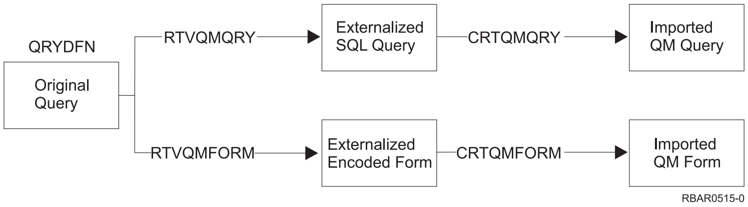
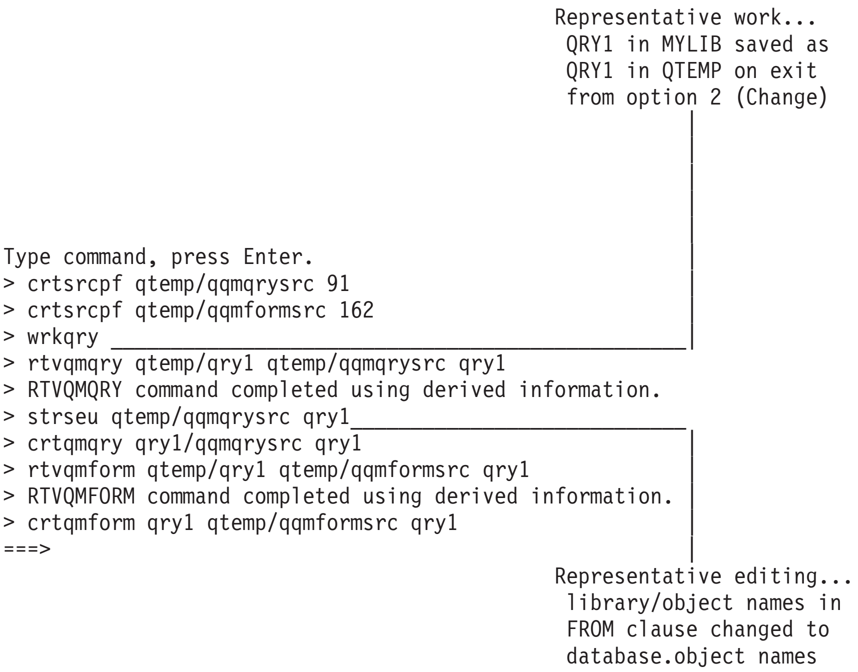


Figure 59. Conversion Data Flow

Notes:
1. The form retrieved from a QRYDFN object may have blank column entries, causing warnings when the form is used to create a QMFORM object. These warnings can be ignored.
2. The query source retrieved from a QRYDFN object is an SQL SELECT statement. The FROM clause uses system naming conventions.

Figure 60 on page 195 is an example of how you can use CL commands to convert a Query for iSeries QRYDFN object to DB2 UDB for iSeries Query Management objects.



```
                                             Representative work...
                                              QRY1 in MYLIB saved as
                                              QRY1 in QTEMP on exit
                                              from option 2 (Change)
                                                        |
Type command, press Enter.                              |
> crtsrcpf qtemp/qqmqrysrc 91                           |
> crtsrcpf qtemp/qqmformsrc 162                         |
> wrkqry _______________________________________________|
> rtvqmqry qtemp/qry1 qtemp/qqmqrysrc qry1
> RTVQMQRY command completed using derived information.
> strseu qtemp/qqmqrysrc qry1___________________________
> crtqmqry qry1/qqmqrysrc qry1                          |
> rtvqmform qtemp/qry1 qtemp/qqmformsrc qry1            |
> RTVQMFORM command completed using derived information.|
> crtqmform qry1 qtemp/qqmformsrc qry1                  |
===>                                                    |
                                             Representative editing...
                                              library/object names in
                                              FROM clause changed to
                                              database.object names
```

Figure 60. Sample CL Command Sequence for QRYDFN Conversion

See "Creating a CL program for permanent conversion of a QRYDFN object in Query Management" on page 170 for another example of using CL commands to convert Query for iSeries QRYDFN objects to DB2 UDB for iSeries Query Management objects.

Using the STRQMQRY command instead of the RUNQRY command in Query Management

Both the Start Query Management Query (STRQMQRY) and the Run Query (RUNQRY) commands can be used to produce a formatted report or database file output according to specifications in a previously created QRYDFN object. The following examples demonstrate the minimum parameter requirements for each command when the object to be run is a QRYDFN object. `RUNQRY mylib/myqrydfn STRQMQRY mylib/myqrydfn  QMFORM(*QMQRY) ALWQRYDFN(*YES)`

Note: Query Management supports alternate collating sequence tables in a Query for iSeries *QRYDFN object for both the STRQMQRY and the RUNQRY commands.

Some reasons for using the STRQMQRY command:

- The appearance of many reports is improved if STRQMQRY is used.
- STRQMQRY provides less restrictive use of QRYDFN information. Unlike RUNQRY, STRQMQRY can complete successfully using:
  - A QRYDFN object saved with errors.
  - A dependent query in batch mode.
  - Form and query information from separate objects.
  - Form information derived from a QRYDFN object that includes selection of a file that is not defined on the system.
  - Query information derived from a QRYDFN object that includes a numeric constant with a decimal point delimiter other than indicated by the QDECFMT system value.

- STRQMQRY may be faster for reporting summary-only information from large files. What makes this possible is a conversion mode (SQL summary) that uses an SQL GROUP BY clause or column functions in the derived query instead of summary function usages in the derived form. However, this more efficient way of producing summary values and omitting detail records can only be used for a QRYDFN with special characteristics. Refer to “Query for iSeries and DB2 UDB for iSeries Query Management differences” on page 192.

Using STRQMQRY instead of RUNQRY may not achieve acceptable results, or certain actions may be necessary to ensure a successful outcome. Consider the following items first:

- If STRQMQRY is used and there is a QMQRY or QMFORM named `myqrydfn` in `mylib`, information for running the query or formatting the report will be taken from that object instead of the QRYDFN object unless *ONLY is specified as the ALWQRYDFN parameter value. If *YES has to be specified to use the QRYDFN object with a query management object, one of the objects may have to be renamed or moved.
- Because the query database does not support member specification, it may be necessary to use file overrides to cause the intended member to be used instead of the *FIRST member (refer to “Override considerations in Query Management” on page 165).
- Both commands have OUTFILE parameters, but STRQMQRY has no *RUNOPT default. This means that if other than displayed output is intended, OUTFILE and related parameters must be specified for STRQMQRY because default values from the QRYDFN object are not used.
- If the QRYDFN object is a dependent query, the dependent values must be set at run-time. For the RUNQRY command, the record selection (RCDSLT) parameter must be used. This requires interactive mode, and causes the *Select records* prompt to be displayed so that the dependent values can be specified. For STRQMQRY, the dependent value names are converted to global variables, which can be set using the SETVAR parameter. The query can be run in batch mode if the SETVAR parameter is used to specify values for all the global variables. In interactive mode, STRQMQRY uses INQUIRY messages to prompt for any global variables not previously set by use of the SETVAR parameter, but these messages have no information about the value expected. You may want to keep the printed query definition on hand or write a simple STRQMQRY-based command that prompts for the value by showing possible choices.
- RUNQRY processing makes dynamic adjustments for changes made to a file definition since the query was saved; STRQMQRY processing does not. The QRYDFN object may need to be resaved before STRQMQRY is used.
- If SQL conversion mode is used to omit detail records, any COUNT summaries are converted to COUNT(*). Null values are counted. If you do not want null values to be counted, add a record selection test that excludes the records with null values (ISNOT NULL).
- If you use a character column (field) for dates-instead a date data type- you might get different results between:
  - SQL using INSERT processing.
  - SQL using INSERT processing with subselect.

  When using INSERT processing without SELECT, the date format is taken from the user’s profile. When using INSERT processing with SELECT, SQL assumes the value is for output and provides it to Query Management in the SQL format (ddmmyyyy).

  The ideal way to avoid this is to use date type columns for dates. When you must use character type columns, make sure the user profile specifies the ddmmyyyy date format.
- If the acceptability of the Query for iSeries result depends on any of the following, the Query Management result will probably be unacceptable. Parenthesized phrases indicate the potentially unacceptable system action for each item:
  - Dynamic resolution of field selections (The saved field list is used)
  - Data from a file with multiple formats (No output is produced)
  - Unmatched join with primary file (A matched join is performed)

  - Matched join with primary file (A matched join is performed)
  - Control of expression scale and precision (Defaults are used for calculations)
  - More than 255 field selections and extra function selections (Only the first 255 selections are used)
  - LIKE test of result expression with || or SUBSTR (No output is produced)
  - Nonhexadecimal collating (No alternative collating is performed)
  - Decimal position override with default numeric editing (The number of decimal positions is the default value)
  - Length override n to format n digits or characters (The number of digits or characters is the default value)
  - Length override with default numeric editing (The default length is used)
  - Length override with default column heading (The default length is used)
  - Edit override with default decimal positions (Column values are edited with 0 decimal positions)
  - Date and time numeric editing override (The run-time default is used)
  - Numeric editing override (edit code K is used)
  - Multiple summary functions for field in same column (A separate column is added for each summary function)
  - Omission of break field column (The break field column is not omitted)
  - Summary function for break field (A separate column is added to hold summary values)
  - Break text for non-SQL summary when 1st or only column has summary (No break text is displayed)
  - Final text for non-SQL summary when 1st or only column has summary (No final text is displayed)
  - Summary only output with multiple-level break summaries (Detail records are not omitted)
  - Summary only output with break and final summaries (Detail records are not omitted)
  - Summary only output with result field used for break control (Detail records are not omitted)
  - Summary only output with count of non-null values for some field (Detail records are omitted, but all records counted)
  - Column function other than COUNT for result field literal (No output is produced for SQL summary)
  - MAX column function for character field wider than SQL allows (No output is produced for SQL summary)
  - MIN column function for character field wider than SQL allows (No output is produced for SQL summary)
  - Total break fields size for GROUP BY wider than SQL allows (No output is produced for SQL summary)
  - Total break and summary columns size wider than SQL allows (No output is produced for SQL summary)
  - Line wrapping (No line wrapping is used; parallel printer files are produced)
  - Cover page text (No cover page is printed)
  - Page text wider than 55 characters (Page text is truncated)
  - Truncation of numeric overflow (Numeric overflow is rounded)
  - Ignoring of decimal data error (Errors are not ignored; data is not corrected)
  - Treating character substitution as an error (Character substitution is ignored)
  - Using object names to reference tables with names greater than 10 characters (The object name must meet system naming conventions)
- You might be able to get better results by considering the differences between Query for iSeries and DB2 UDB for iSeries Query Management listed in “Query for iSeries and DB2 UDB for iSeries Query Management differences” on page 192 and taking appropriate action. For example, because the defaults differ, you might want to ensure that the printer form width used is the same as that used by Query for iSeries.

Conversion details in Query Management

Figure 61 shows the relationship between the Work with Query displays used to prompt for query definition specifications and the corresponding sections of query management query and query management form objects. Some Structured Query Language (SQL) functions that are supported by DB2 UDB for iSeries Query Management but not represented in Figure 61 (such as sublists and certain scalar functions) are not available through the prompted interface, but are available in SQL.

```
Work with Query Display                  Query Management Object

Specify File Selections

 -  File, Library                         SELECT   FROM
 -  Member                                N/A
 -  Format                                N/A
 -  File ID                               SELECT   FROM
                                         SELECT   list, ORDER BY
                                (used as Field qualifier)
Specify Type of Join
 -  Type of join                          N/A
Specify How to Join Files
 -  Field__Test__Field                    SELECT   WHERE
Define Result Fields
 -  Field__    (if no column heading)     Column:  Column heading, Width
 -  Expression__                          SELECT   list, WHERE
                                (substituted for Field)
 -  Column heading__ (if no override)     Column:  Column heading, Width
 -  Len__                                 Column:  Edit, Width
 -  Dec                                   Column:  Edit, Width
Select and Sequence Fields
 -  Seq__Field                            SELECT   list
Select Records
 -  AND/OR__Field__Test__Value            SELECT   WHERE
Select Sort Fields
 -  Sort Prty__A/D__Field                 SELECT   ORDER BY
                                         SELECT   GROUP BY
                                         SELECT   HAVING
Select Collating Sequence
 -  Collating sequence option             N/A
 -  Table, Library                        N/A
Define Collating Sequence
 -  Sequence__Char                        N/A
Specify Report Column Formatting
 -  Field__
                                         Column:  Seq
                                         Column:  Datatype
 -  Column spacing__                      Column:  Indent
 -  Column heading__  (override only)     Column:  Column heading, Width
 -  Len__             (override only)     Column:  Usage, Edit, Width
    (0)                            (OMIT)
 -  Dec__             (override only)     Column:  Edit, Width
    (#)                            ( #)
 -  Edit              (override only)     Column:  Edit, Width
    (Untransformable numeric editing)               (K )
Define Numeric Field Editing
 -  Edit option                           N/A
Describe Numeric Field Editing
 -  Decimal point, and so on              Column:  Edit, Width
    (Transformable description)    (matched code)
```

Figure 61. Correlation between the Work with Query Display and DB2 UDB for iSeries Query Management Objects (Part 1 of 3)

```
Work with Query Display                  Query Management Object

Describe Date/Time Field Editing
 -  Date/time separator                   N/A
Specify Edit Code
 -  Edit code, Optional modifier          Column:  Edit, Width
    (J )                           (K )
    (J$)                           (D )
    (M )                           (L )
Specify Edit Word
 -  Edit word                             N/A
 -  Edit word for summary total           N/A
Select Report Summary Functions
 -  Options__Field                        Column:  Usage, Width
    (1=Total  )                    (SUM    )
    (2=Average)                    (AVERAGE)
    (3=Minimum)                    (MINIMUM)
    (4=Maximum)                    (MAXIMUM)
    (5=Count  )                    (COUNT  )
                                   (FIRST  )
                                   (LAST   )
Define Report Breaks
 -  Break level__Sort Prty__Field         Column:  Usage
    (1-6)                          (BREAK1-BREAK6)
Format Report Break (level 0)
                                         Final:   New page for final text
 -  Suppress summaries                    Final:   Put final summary at line
    (Y=Yes, N=No (if text present))(NONE, 2)
                                         Final:   Blank lines before footing
 -  Break text                            Final:   Final footing text
    (1st and only line)            (Line 1)
                                   (Lines 2-12)
Format Report Break (level 1-6)
 -  Skip to new page                      Break:   New page for heading
                                         Break:   Repeat column heading
                                         Break:   Blank lines before heading
                                         Break:   Blank lines after heading
                                         Break:   New page for footing
                                         Break:   Blank lines before footing
                                         Break:   Blank lines after footing
 -  Suppress summaries                    Break:   Put summary at line
    (Y=Yes, N=No (if text present))(NONE, 2)
                                         Break:   Break heading text
 -  Break text                            Break:   Break footing text
    (1st and only line)            (Line 1)
                                   (Lines 2-5)
Select Output Type and Output Form
 -  Output type                           N/A
 -  Form of output                        N/A
 -  Line wrapping                         N/A
```

Figure 61. Correlation between the Work with Query Display and DB2 UDB for iSeries Query Management Objects (Part 2 of 3)

```
Work with Query Display                  Query Management Object

Define Printer Output
 -  Printer                               N/A
 -  Form size                             N/A
 -  Start line                            N/A
 -  End line                              N/A
 -  Line spacing                          Options: Detail line spacing
    (1-3)                          (1-3)
                                   (4  )
                                         Options: Outlining for break columns
                                         Options: Default break text
                                         Options: Column-wrapped lines kept
                                         Options: Column heading separators
                                         Options: Break summary separators
                                         Options: Final summary separators
 -  Print definition                      N/A
Define Spooled Output
 -  Spool the output                      N/A
 -  Form type                             N/A
 -  Copies                                N/A
 -  Hold                                  N/A
Specify Cover Page
 -  Print cover page                      N/A
 -  Cover page text (up to 5 lines)       N/A
Specify Page Headings and Footings
 -  Print standard page headings
                                         Page:    Blank lines before heading
                                         Page:    Blank lines after heading
 -  Page heading                          Page:    Page heading text
    (lines 1-3)                    (Lines 1-3)
                                   (Lines 4-5)
                                         Page:    Blank lines before footing
                                         Page:    Blank lines after footing
 -  Page footing                          Page:    Page footing text
    (1st and only line)            (Line 1)
                                   (Lines 2-5)
Define Database File Output
 -  File, Library, Member                 N/A
 -  Data in file                          N/A
 -  Authority (for new file)              N/A
 -  Text (for new file)                   N/A
 -  Print definition                      N/A
Specify Processing Options
 -  Use rounding                          N/A
 -  Ignore decimal data errors            N/A
```

Figure 61. Correlation between the Work with Query Display and DB2 UDB for iSeries Query Management Objects (Part 3 of 3)

The following actions resulting from conversion may be unexpected. You should be aware of these actions to obtain the best results.

- Every character in an expression or test value is converted to uppercase unless it is in a string constant. This includes the characters in a delimited name.
- SQL reserved words used as field names are placed in quotation marks in a derived SELECT statement.
- The corresponding expression is substituted for each result field name that would otherwise appear in a SELECT list or record selection test. Substitution is recursive because result field names can be used in the expressions for other result fields. Column numbers are substituted for result field names in the ORDER BY clause.
- If any join tests exist, they are used to start the WHERE clause, and any record selection tests are added using the AND operator.

- Dependent values in record selection tests are converted to global variables. For example, the dependent value :t01.″collection″ is converted to &T01_COLLECTION.
- Valid decimal point delimiters are converted to the delimiter indicated by the Query Decimal Format (QDECFMT) system value in numeric constants in a derived SELECT statement.
- Query Management processing truncates decimal positions when dividing unscaled values. For example, the value calculated for 2 divided by 3 is 0. For this reason, the decimal point delimiter indicated by the QDECFMT system value is put at the end of every number without a decimal point delimiter (and followed by a +, –, *, (, or /) in expressions in a derived SELECT statement.
- The *Column heading* field is left blank on a column with no summary function unless the column is a result field or there is a column heading override for it. The field name is the default if no heading is defined.
- Underline characters in the heading text are not replaced with substitute characters. A single line heading such as 1_9_9_0 is converted with no changes and causes four lines of heading when the derived form information is used.

  The following actions may also occur:
  - *NONE, designating no column heading, is converted to a single underline character; or, in a summary function column, to the caption Query for iSeries uses for the values.
  - A column heading string is built by removing leading and trailing blanks from each line, up to and including the last nonblank line, and then connecting the resulting segments with single underline characters.
  - The heading for a column with a summary function is built by adding the column heading string to the caption that Query for iSeries uses for the values. If the resulting string is longer than 62 characters, it is truncated.
  - If no heading string exists to add to the summary function caption, the field name is used. Any heading defined for the field as part of a file definition is ignored.
- The character part of the Edit value is left blank if any of the following occur:
  - The column holds only COUNT summary function values.
  - No size override exists, unless the column is a result field for which a size has been specified.
  - No override editing is defined, the column is not a result field, and there is a numeric decimal positions override.
- DB2 UDB for iSeries Query Management uses the *C* edit code if the decimal positions override indicates a character, date, time, or timestamp field. The *K* edit code is used for a numeric field if a closer match to an Query Management CPI Query edit code cannot be determined for the edit override, or if no override exists and the column is for a result field.
- Only the type of override editing indicated by the *Edit* option choice is considered.
- The effect of system defaults on RPG edit code editing is disregarded. The J, J with currency symbol, and M edit codes are always converted to K, D, and L, respectively.
- Edit description choices not involved in the actual editing (a left or right currency symbol when no currency symbols are to be used, for example) are ignored when comparisons are made.
- The numeric part of the *Edit* value is left blank if the character part is blank or C. This value is also left blank if the number of decimal positions to be used cannot be determined from a decimal positions value saved as an override (with a nonzero length override) or as part of the definition of a result field.
- The Width value for a column is left blank unless the information saved in the QRYDFN object specifies the width requirements for everything that appears in the column. A column formatting length override can influence the Width, but only determines the number of digits or decimal positions formatted when nothing else in the column (such as a heading segment or count summary value 9,999,999) is wider than the edited data and the override length is smaller than the data width assumed by database processing.
- Page, break, and final text are adjusted in the following ways:
  - Strings of consecutive blanks are compressed to single blanks.

  - Fields referencing insert variables are converted to column referencing variables and special variables (such as &time, &date, &page) are converted to all uppercase characters. Strings starting with an ampersand (&) are recognized as text insert variables only if the character after the & is not a blank or a numeric digit, and if the string is ended by one of the following characters:
    - Blank
    - Slash
    - Colon
    - Dash
    - Ampersand
    - Underscore
    - DBCS shift-out character

    All selected fields are considered (in report order) when converting a field reference to a column reference.
  - If the resolved text exceeds 55 characters, it is truncated at the first blank or ampersand of the first 56 positions. If no blank or ampersand is found, the text is not used.
  - DBCS strings left open due to truncation are closed.
- The CCSID of the query definition and the formatting attributes of any literals in expressions or record selections are passed along to DB2 UDB for iSeries Query Management. This information does not come from the job under which the conversion is performed.

Chapter 13. Control language interface in Query Management

You can use Query Management management functions through the control language (CL) to create simple applications for report generation. By defining a command, a control language (CL) program, a query, and a form, you can prompt the user for any amount of information required to generate a meaningful report.

Creating QMQRY and QMFORM objects in Query Management

You can define a query (QMQRY object) using an SQL statement. Figure 62 is an example of a printed query called SALARYQ2.

```
                              DB2 Query Management OS/400

 Query . . . . . . . :   SALARYQ2
   Library . . . . . :     EXAMPLE
 Sort sequence . . . . . . . . : Sort sequence table
    Sort sequence table  . . : QASCII
       Library . . . . . . . : QSYS
 Language identifier . . . . : ENU
 Text  . . . . . . . . . . . : Sales query
 Text  . . . . . . . :   TEST QUERY
 SEQNBR |...+....1....+....2....+....3....+....4....+....5....+....6....
 000001 SELECT DEPT,NAME,ID,JOB,YEARS,SALARY,COMM
 000002        FROM TESTDATA/STAFF
 000003        WHERE DEPT = &COND1 AND SALARY < &COND2
 000004        ORDER BY DEPT,SALARY
                                 * * * * *   END OF SOURCE   * * *
```

Figure 62. Test Query SELECT Statement

Note: After a *QMQRY object is defined to use a user-defined sort sequence table, changes to the sort sequence table only take effect when the query is run if the query changed. You cannot PRINT or EXPORT the contents of the sort sequence table associated with a *QMQRY object.

For more information about creating QMQRY objects, see “Creating queries in Query Management” on page 9.

You can define a form (QMFORM object) to specify the information to be included in the report generated.

For more information about creating QMFORM objects, see “Creating forms in Query Management” on page 57.

Sample CL program for numeric variables in Query Management

The CL program in Figure 63 uses the query shown in Figure 62.

```
                                  SEU SOURCE LISTING
SOURCE FILE . . . . . . .  EXAMPLE/SOURCE
MEMBER  . . . . . . . . .  CLPGM
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...
  100 PGM PARM(&VAR1 &VAR2 &VAR3 &VAR4)
  200 DCL &VAR1 *CHAR LEN(6)
  300 DCL &VAR2 *CHAR LEN(6)
  400 DCL &VAR3 *CHAR LEN(6)
  500 DCL &VAR4 *CHAR LEN(10)
  600 STRQMQRY QMQRY(EXAMPLE/SALARYQ2) QMFORM(EXAMPLE/&VAR4) +
  700          OUTPUT(&VAR3)                             +
  800          SETVAR((COND1 &VAR1) (COND2 &VAR2))
  900 ENDPGM
                        * * * *  E N D  O F  S O U R C E  * * * *
```

Figure 63. CL Program Source File

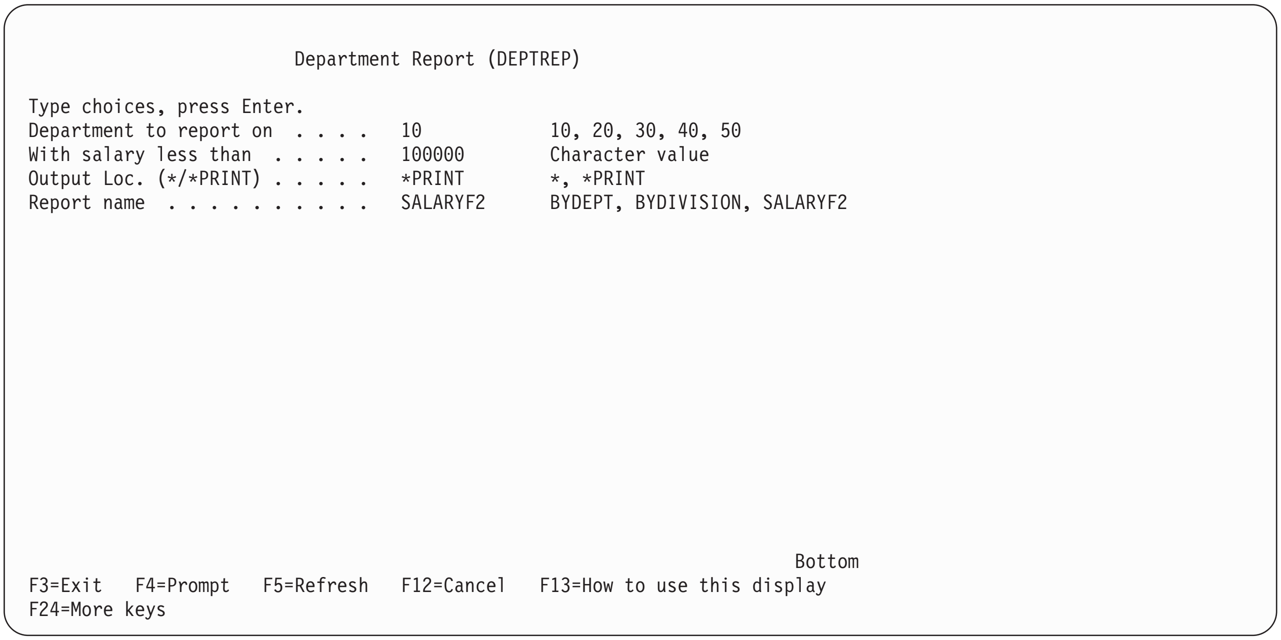
Note: Using STRQMQRY and the CL command processor with multiple variables and long SQL statements within STRQMQRY can sometimes cause unexpected results. Long SQL statements can be broken into 55 character segments and saved into multiple variables. However, you should make sure that these segments break on a character. If the statement breaks on a blank, the CL Command Processor will truncate the blanks. When the variables are concatenated, the truncated spaces are not replaced, which can cause the SQL statement to fail. However, the CL Command Process will leave blanks that occur at the beginning of a segment.

Figure 64 shows the command to run this example CL program.

```
                                  SEU SOURCE LISTING
SOURCE FILE . . . . . . .  EXAMPLE/SOURCE
MEMBER  . . . . . . . . .  DEPTREP1
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...
  200 CMD   PROMPT('Department Report')
  300             PARM      KWD(VAR1) TYPE(*CHAR) LEN(6) RSTD(*YES) +
  301                         DFT(10) VALUES(10 20 30 40 50) +
  400                         PROMPT('Department to report on')
  500             PARM      KWD(VAR2) TYPE(*CHAR) LEN(6) DFT(100000) +
  600                         PROMPT('With salary less than')
  700             PARM      KWD(VAR3) TYPE(*CHAR) LEN(6) RSTD(*YES) +
  701                         DFT(*) VALUES(* *PRINT) PROMPT('Output +
  800                         Loc. (*/*PRINT)')
  900             PARM      KWD(VAR4) TYPE(*CHAR) LEN(10) RSTD(*YES) +
  901                         DFT(SALARYF2) VALUES(BYDEPT BYDIVISION +
 1000                         SALARYF2) PROMPT('Report name')
                       * * * *  E N D  O F  S O U R C E  * * * *
```

Figure 64. CL Command Source File

When you enter the command `EXAMPLE/DEPTREP` and press F4 to prompt, the following display appears:



```
                        Department Report (DEPTREP)

Type choices, press Enter.
Department to report on  . . . .   10            10, 20, 30, 40, 50
With salary less than  . . . . .   100000        Character value
Output Loc. (*/*PRINT) . . . . .   *PRINT        *, *PRINT
Report name  . . . . . . . . . .   SALARYF2      BYDEPT, BYDIVISION, SALARYF2
















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

Press the Enter key to use the default values for this job.

The report shown in Figure 65 is displayed with the department averages for salary and years of experience for department 10.

```
     DEPT  NAME             ID  JOB       YEARS      SALARY       COMM
---------  ---------  ---------  -----  ---------  -----------  ---------
       10  DANIELS         240  MGR           5    19,260.25        .00
           LU              210  MGR          10    20,010.00        .00
           JONES           260  MGR          12    21,234.00        .00
           MOLINARE        160  MGR           7    22,959.20        .00
                                        ---------  -----------
                          Dept Avg:           9    20,865.86
```

Figure 65. CL Program Report Example

Creating QMQRY and QMFORM objects for character variables in Query Management

You define the query (QMQRY object) the same way as you do an object with numeric variables except for one thing. More quotes are required for character variables. Figure 66 is an example of a printed query called SALARYQ3.

```
                                DB2 Query Management OS/400

 Query  . . . . . . :   SALARYQ3
   Library  . . . . :     EXAMPLE
 Text . . . . . . . :   TEST QUERY
 SEQNBR |...+....1....+....2....+....3....+....4....+....5....+....6....
 000001 SELECT JOB,DEPT,NAME,ID,YEARS,SALARY,COMM
 000002        FROM TESTDATA/STAFF
 000003        WHERE JOB = &COND1 AND SALARY < &COND2
 000004        ORDER BY JOB,SALARY
                                  * * * * *   END OF SOURCE   * * *
```

Figure 66. Test Query SELECT Statement

Note: After a *QMQRY object is defined to use a user-defined sort sequence table, changes to the table only take effect when the query is run if the query changed. You cannot PRINT or EXPORT the contents of the sort sequence table associated with a *QMQRY object.

For more information about creating QMQRY objects, see “Creating queries in Query Management” on page 9.

You can define a form (QMFORM object) to specify the information to be included in the report generated.

For more information about creating QMFORM objects, see “Creating forms in Query Management” on page 57.

Sample CL program for character variables in Query Management

The CL program in Figure 67 uses the query shown in Figure 66.

```
                                         SEU SOURCE LISTING
  SOURCE FILE . . . . . . . EXAMPLE/SOURCE
  MEMBER  . . . . . . . . . CLPGM
  SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...
    100 PGM PARM(&VAR1 &VAR2 &VAR3 &VAR4)
    200 DCL &VAR1 *CHAR LEN(6)
    300 DCL &VAR2 *CHAR LEN(6)
    400 DCL &VAR3 *CHAR LEN(6)
    500 DCL &VAR4 *CHAR LEN(10)
    600 DCL &CHARJOB *CHAR LEN(10)
    700 CHGVAR    VAR(&CHARJOB) VALUE('''' *TCAT &VAR1 *TCAT '''')
    800 STRQMQRY QMQRY(EXAMPLE/SALARYQ3) QMFORM(EXAMPLE/&VAR4) +
    900          OUTPUT(&VAR3)                               +
   1000          SETVAR((COND1 &CHARJOB) (COND2 &VAR2))
   1100 ENDPGM
                                  * * * *  E N D  O F  S O U R C E  * * * *
```

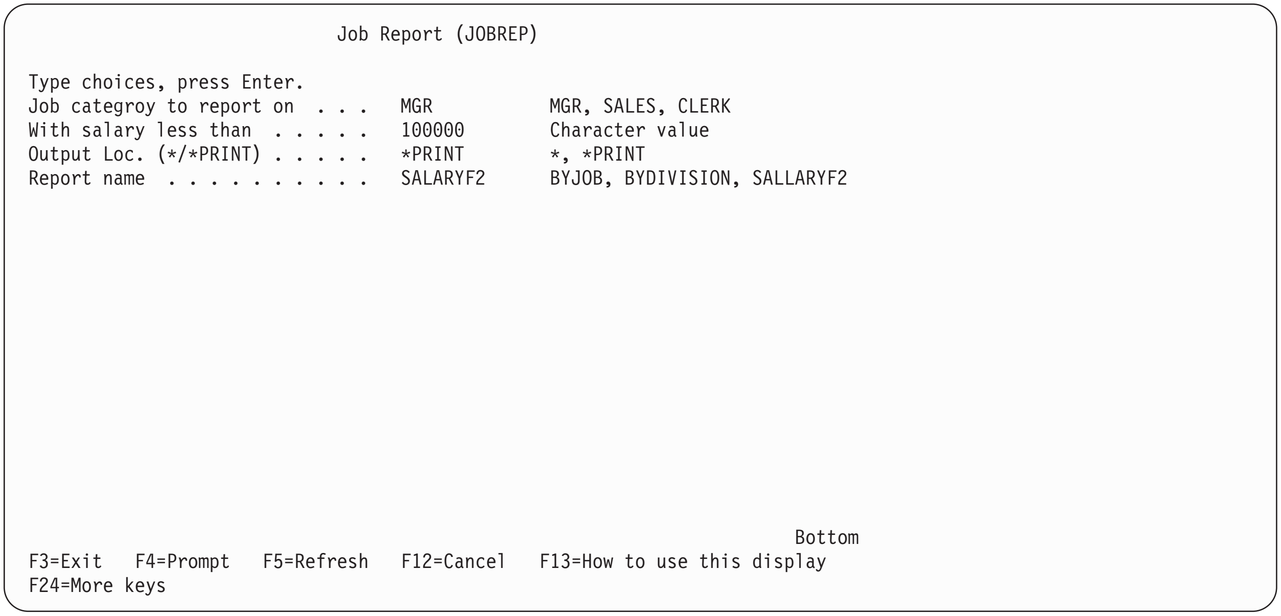
Figure 67. CL Program Source File

Figure 68 shows the command to run this example CL program.

```
                                     SEU SOURCE LISTING
SOURCE FILE . . . . . . .  EXAMPLE/SOURCE
MEMBER  . . . . . . . . .  JOBREP1
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...
  200 CMD   PROMPT('Job Report')
  300             PARM      KWD(VAR1) TYPE(*CHAR) LEN(6) RSTD(*YES) +
  301                         DFT(MGR) VALUES(MGR SALES CLERK) +
  400                         PROMPT('Job category to report on')
  500             PARM      KWD(VAR2) TYPE(*CHAR) LEN(6) DFT(100000) +
  600                         PROMPT('With salary less than')
  700             PARM      KWD(VAR3) TYPE(*CHAR) LEN(6) RSTD(*YES) +
  701                         DFT(*) VALUES(* *PRINT) PROMPT('Output +
  800                         Loc. (*/*PRINT)')
  900             PARM      KWD(VAR4) TYPE(*CHAR) LEN(10) RSTD(*YES) +
  901                         DFT(SALARYF3) VALUES(BYJOB BYDIVISION  +
 1000                         SALARYF3) PROMPT('Report name')
                            * * * *  E N D  O F  S O U R C E  * * * *
```

Figure 68. CL Command Source File

When you enter the command EXAMPLE/DEPTREP and press F4 to prompt, the following display appears:



```
                             Job Report (JOBREP)

Type choices, press Enter.
Job categroy to report on  . . .   MGR           MGR, SALES, CLERK
With salary less than  . . . . .   100000        Character value
Output Loc. (*/*PRINT) . . . . .   *PRINT        *, *PRINT
Report name  . . . . . . . . . .   SALARYF2      BYJOB, BYDIVISION, SALLARYF2













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

Press the Enter key to use the default values for this job.

The report shown in Figure 69 is displayed with the department values for salary and years of experience for job MGR.

```
                                                                         COMM
JOB       DEPT  NAME           ID   YEARS       SALARY      COMM  ---------
-----  -------  --------- -------  ------  -----------  --------        .00
MGR         38  MARENGHI      30       5    17,506.75       .00
            42  PLOTZ        100       7    18,352.80       .00
            20  SANDERS       10       7    18,357.50       .00
            66  LEA          270       9    18,555.50       .00
            10  DANIELS      240       5    19,260.25       .00
            84  QUILL        290      10    19,818.00       .00
            10  LU           210      10    20,010.00       .00
            15  HANES         50      10    20,659.80       .00
            51  FRAYE        140       6    21,150.00       .00
            10  JONES        260      12    21,234.00       .00
            10  MOLINARE     160       7    22,959.20       .00
                                   ------  -----------
                         Job Avg:      8    19,805.80
```

Figure 69. CL Program Report Example

Appendix A. DBCS data in Query Management

This appendix describes the use of double-byte character set (DBCS) data with DB2 UDB for iSeries Query Management. Using DBCS data is different in some ways from using single-byte character set (SBCS) data. In order to use DBCS data, you must have DBCS-capable hardware and software.

What is DBCS data in Query Management?

A double-byte character set is a set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by one byte (256 code points), require double-byte character sets.

Character sets that use one byte to represent each character are called single-byte character sets. Languages such as English, German, and French have single-byte character sets. In some cases, Japanese Katakana can also be represented by a single-byte character set, because the characters can be represented internally in single bytes.

References made in this chapter to *mixed* data mean that strings of both single-byte and double-byte data are contained together in one data field.

There are two types of DBCS data, *bracketed DBCS* and *graphic-DBCS*:

Bracketed-DBCS data:
Bracketed-DBCS data is DBCS data that, when stored in the database, is preceded by a one-byte shift-out (SO) character (hex'0E') and is followed by a one-byte shift-in (SI) character (hex'0F'). The SO character indicates the beginning of DBCS data and the SI character indicates the end of DBCS data. When data is mixed SBCS and DBCS, the DBCS data in the mixed string is preceded by a SO character and followed by a SI character.

Graphic-DBCS data:
Graphic-DBCS data is DBCS data that is stored in the database without being preceded and followed by SO and SI characters. Graphic-DBCS data can be fixed length or variable length.

DB2 UDB for iSeries maintains bracketed-DBCS data with a length specified as the number of *bytes*, including SO and SI characters. The maximum length of bracketed-DBCS data that can be stored in the database is 32 766.

The database maintains graphic-DBCS data with a length specified as the number of *characters*; therefore, the number of bytes used in the database for a graphic field is twice the number of characters. The maximum length of graphic-DBCS data that can be stored in the database is 16 383. For variable-length graphic-DBCS data, the maximum is 16 370.

Displayed and printed DBCS data in Query Management

You can display any DB2 UDB for iSeries Query Management object containing DBCS data whether or not you have a DBCS-capable display.

When displayed or printed, DBCS fields are surrounded by SO and SI characters. This is done whether or not the job is DBCS capable.

The default report width for bracketed-DBCS data is the length of the data in bytes, including SO and SI characters. The default report width for graphic-DBCS data is twice the number of characters plus 2 (for the SO and SI characters).

DB2 UDB for iSeries Query Management positions the SO character as the first character in the column. The SO character is not put in the indent area (see INDENT under “Using DBCS in the FORM in Query

Management" on page 212). Although this causes a misalignment when SBCS column headings are used, the alignment is correct with DBCS column headings. Figure 70 shows a report segment using DBCS data. The <> characters represent the SO and SI characters; these characters show as blanks on the printed or displayed report. This example uses DBCS alphabetic characters for the data. The first column has a DBCS column heading and the second column has an SBCS column heading.

```
<E M P L O Y E E >
<N A M E >                     JOB
----------------------------  -------------
<J O H N   S H E R M A N   > <M G R       >
<B O B   S C H R A M S K I > <C L E R K   >
<J U D I T H   B A R T A   > <S A L E S   >
```

Figure 70. Example Default Report for DBCS Data

If a printed or displayed report results in a DBCS data type field being split between print files or display panels, there are some differences in the report, depending on whether or not the job is DBCS capable. If the job is DBCS capable, SO and SI characters are added around the column segments where they are split. Figure 71 shows how a DBCS column is split between print pages when the job is DBCS capable. If the job is not DBCS capable, no SO or SI characters are added around the split column segments. In Figure 71, the report width is 46 and the PRINT REPORT width is 24. Figure 72 shows how a DBCS column is split between print pages when the job is not DBCS capable. The # character in the report represents one half of a DBCS character.

```
SPOOL FILE 1:                |SPOOL FILE 2:
                    1(1)     |                      1(2)
                             |
 <E M P L O Y E E>           |
 <N A M E>                   |           JOB
 -----------------------     | --------- --------------
 <J O H N   S H E R M >      | <A N   >  <M G R       >
 <B O B   S C H R A M >      | <S K I >  <C L E R K   >
 <J U D I T H   B A R >      | <T A   >  <S A L E S   >
 <K A R E N   R A N D >      | <      >  <S A L E S   >
```

Figure 71. Example Printed Report Split When DBCS Capable

```
SPOOL FILE 1:                |SPOOL FILE 2:
                    1(1)     |                      1(2)
                             |
 <E M P L O Y E E>           |
 <N A M E>                   |           JOB
 -----------------------     | ------- --------------
 <J O H N   S H E R M #      | #N     >  <M G R       >
 <B O B   S C H R A M #      | #K I   >  <C L E R K   >
 <J U D I T H   B A R #      | #A     >  <S A L E S   >
 <K A R E N   R A N D        |        >  <S A L E S   >
```

Figure 72. Example Printed Report Split When Not DBCS Capable

Data types used with DBCS data in Query Management

You can save DBCS data in your database if you define the columns in which you save the data as one of two basic types, CHARACTER or GRAPHIC:

- Bracketed-DBCS data must be put into CHARACTER data type columns. Bracketed-DBCS data can be mixed with SBCS data. Bracketed-DBCS constants (see the following note) must also be put into CHARACTER data type columns. Bracketed-DBCS constants consist of bracketed-DBCS data preceded and followed by the single-byte apostrophe, for example:

  ```
  '<D1D2D3>'
  ```

where D1, D2, and D3 represent double-byte characters. All entries in the column must have the same length.

- Graphic-DBCS data strings must be put into columns defined as GRAPHIC. Graphic-DBCS constants (see the following note) must also be put into GRAPHIC data type columns. Graphic-DBCS constants consist of bracketed-DBCS data preceded and followed by the single-byte apostrophe and preceded by a G, for example:

  ```
  G'<D1D2D3>'
  ```

  Graphic-DBCS data cannot be mixed with SBCS data. Either fixed-length or variable-length graphic-DBCS data can be put into columns defined as graphic.

  Note: The constants described above are those that are used in the SQL statement when selecting derived columns. Derived columns are columns that are defined as an SQL expression, for example:

  ```
  SELECT G'<A B C >' || Column1, '<D B C S >' FROM Tablename
  ```

For additional information on the maximum lengths of these columns, refer to the SQL Reference.

Using DBCS data in DB2 UDB for iSeries Query Management

The following sections explain how using DBCS data in DB2 UDB for iSeries Query Management is *different* from using SBCS data.

"Using DBCS Data in input fields in Query Management"

"Using DBCS data in queries in Query Management"

"Using DBCS in the FORM in Query Management" on page 212

"Saving DBCS data in Query Management" on page 215

"Using DBCS global variables in DB2 UDB for iSeries Query Management commands" on page 215

"Exporting DBCS data in Query Management" on page 216

"Importing DBCS data in Query Management" on page 216

"Printing DBCS reports in Query Management" on page 216

Using DBCS Data in input fields in Query Management

All DB2 UDB for iSeries Query Management input fields, except names, allow DBCS data.

Using DBCS data in queries in Query Management

The following can be in bracketed-DBCS, mixed SBCS and DBCS, or graphic-DBCS:

- Substitution values
- Delimited strings in character data type fields
- Comments

Graphic data type fields can only contain graphic-DBCS data.

SQL keywords must be in English.

Using DBCS in the FORM in Query Management

The FORM object contains specifications for formatting the data that results from running a query with a select statement. This topic describes how the FORM specifications work with DBCS data.

Bracketed-DBCS data, mixed SBCS and DBCS data, and graphic-DBCS data can be used in the FORM panels as:

- Break text
- Page text
- Final text

The FORM specifications work as follows with DBCS data:

USAGE
Specifies how to use a column. FORM usages must be SBCS characters.

INDENT
The number of blank spaces to the left of a column; this field is used to separate a column either from the column before it, or from the left margin. See "Displayed and printed DBCS data in Query Management" on page 209 for special considerations because of the DBCS SO and SI characters that appear as blank spaces on the report.

WIDTH
Specifies how wide you want the column to be. If the data type for the columns is GRAPHIC, the COLUMNS.WIDTH is interpreted as the number of graphic-DBCS characters. For any other data type, it is interpreted as the number of bytes.

If the column contains bracketed-DBCS data, the width in bytes equals the value for COLUMNS.WIDTH plus 2 (for the SO and SI characters). The maximum column width for bracketed-DBCS data is 32 766 bytes. You must allow for the extra positions in the report for SO and SI characters or undisplayable data can result. If the WIDTH is less than 4 spaces (the minimum space required to display a single DBCS character), the column is filled with asterisks (***).

If the column contains graphic data, the width in bytes is twice the COLUMNS.WIDTH plus 2 (for the SO and SI characters that are added when graphic-DBCS data is displayed or printed). The maximum WIDTH that can be specified in the FORM for graphic-DBCS data is 16 383 characters (16 370 for variable-length graphic-DBCS data). If GRAPHIC is specified for COLUMNS.DATATYPE, there is no need for you to add extra width to accommodate the SO and SI characters that are placed around the data, because this is handled by DB2 UDB for iSeries Query Management.

Note: Because DB2 UDB for iSeries Query Management does not require the COLUMNS.DATATYPE in the form, the same form may be applied to SBCS data or to DBCS data. The width of the report, in bytes, is different for each of these cases.

DATATYPE
Specifies the type of data that is contained in the corresponding column in the queried table. COLUMNS.DATATYPE for bracketed-DBCS data must be CHARACTER. COLUMNS.DATATYPE for graphic-DBCS data must be GRAPHIC.

EDIT Edit codes determine how values in a column are punctuated, if at all.

These codes must be entered on the form in single-byte characters.

- **C** is the edit code for character data type columns. It makes no change in the display of a value.
- **CW** is the edit code for character data columns to be wrapped. It makes no change in the display of a value. But if the value cannot fit on one line in the column, the text wraps according

to the width of the column. That is, instead of cutting off the data at the end of the column, as much data as possible is put on a line in the column, and then the data continues wrapping on the next line.

- **CT** is the edit code for columns of character data to be wrapped according to the column text. It makes no change in the display of a value, but if the value cannot fit on one line in the column, the column wraps according to the text in the column. That is, instead of cutting off the data at the end of the column, as much data as possible is fitted into a line; then the line interrupts at a single-byte blank and the data continues wrapping on the next line. If a string of data is too long to fit in the column and does not contain a single-byte blank, the data wraps by width until a single-byte blank is found.

  When you use the CT edit code for a column that contains mixed DBCS and SBCS data, the minimum width for the column is four (4).
- **G** is the edit code for columns of data defined as graphic type. It makes no change in the display of a value.
- **GW** is the edit code for columns of graphic data you want wrapped. It makes no change in the display of a value, but if the value cannot fit on one line in the column, DB2 UDB for iSeries Query Management wraps the text according to the width of the column. That is, instead of truncating the data at the end of the column, DB2 UDB for iSeries Query Management puts as much data as it can on a line in the column and continues wrapping the remaining data on the next line.

The sample report in Figure 73 on page 214 shows the results of displaying graphic-DBCS data using a form that has a COLUMNS.WIDTH of 1, 2, 3 and 4 for the first 4 columns. The fifth column describes the data. The column headings are SBCS data. The COLUMNS.EDIT for the first four columns is GW. The SQL statement that was run was: SELECT COL1, COL1, COL1, COL1, Desc FROM SAMPLE. The column named COL1 is defined as VARGRAPHIC(10).

```
1      22       333        4444         Description
----   ------   -------    ----------   -----------------------------
<>     <>       <>         <>           LENGTH(COL1)=0
<1 >   <1 >     <1 >       <1 >         LENGTH(COL1)=1
<2 >   <2 2 >   <2 2 >     <2 2 >       LENGTH(COL1)=2
<2 >
<3 >   <3 3 >   <3 3 3 >   <3 3 3 >     LENGTH(COL1)=3
<3 >   <3 >
<3 >
<4 >   <4 4 >   <4 4 4 >   <4 4 4 4 >   LENGTH(COL1)=4
<4 >   <4 4 >   <4 >
<4 >
<4 >
<5 >   <5 5 >   <5 5 5 >   <5 5 5 5 >   LENGTH(COL1)=5
<5 >   <5 5 >   <5 5 >     <5 >
<5 >   <5 >
<5 >
<5 >
<6 >   <6 6 >   <6 6 6 >   <6 6 6 6 >   LENGTH(COL1)=6
<6 >   <6 6 >   <6 6 6 >   <6 6 >
<6 >   <6 6 >
<6 >
<6 >
<6 >
<7 >   <7 7 >   <7 7 7 >   <7 7 7 7 >   LENGTH(COL1)=7
<7 >   <7 7 >   <7 7 7 >   <7 7 7 >
<7 >   <7 7 >   <7 >
<7 >   <7 >
<7 >
<7 >
<7 >
<8 >   <8 8 >   <8 8 8 >   <8 8 8 8 >   LENGTH(COL1)=8
<8 >   <8 8 >   <8 8 8 >   <8 8 8 8 >
<8 >   <8 8 >   <8 8 >
<8 >   <8 8 >
<8 >
<8 >
<8 >
<8 >
<9 >   <9 9 >   <9 9 9 >   <9 9 9 9 >   LENGTH(COL1)=9
<9 >   <9 9 >   <9 9 9 >   <9 9 9 9 >
<9 >   <9 9 >   <9 9 9 >   <9 >
<9 >   <9 9 >
<9 >   <9 >
<9 >
<9 >
<9 >
<9 >
```

Figure 73. Report Using GW with GRAPHIC Data

How data truncation is handled in Query Management

DB2 UDB for iSeries Query Management truncates displayed DBCS data at a field or screen boundary in a way that avoids splitting DBCS characters. Paging is necessary to view the characters on the truncated lines.

Shift-in and shift-out characters and column widths are contained in the width or the column and are not placed in the indent or margins of the report.

Saving DBCS data in Query Management

DB2 UDB for iSeries Query Management supports the following ways of saving DBCS data that is selected by means of a query:

1. DB2 UDB for iSeries Query Management command: SAVE DATA AS
2. CL command: STRQMQRY OUTPUT(*OUTFILE)

The following rules apply when using a SAVE DATA AS command to save DBCS data:

- DBCS-either columns can be saved over DBCS-either and DBCS-open columns.
- DBCS-either columns can be saved over DBCS-only columns only if all the fields in the column are bracketed-DBCS strings.
- DBCS-either columns can be saved over CHAR (non-DBCS) columns only if none of the fields in the column are bracketed-DBCS fields.
- DBCS-only columns can be saved over DBCS-only, DBCS-open and DBCS-either columns.
- DBCS-only columns cannot be saved over CHAR columns.
- DBCS-open columns can be saved over DBCS-open columns.
- DBCS-open columns can be saved over DBCS-only columns only if all the fields in the column are bracketed-DBCS fields.
- DBCS-open columns can be saved over DBCS-either columns only if all the fields in the column are either bracketed-DBCS fields or SBCS fields. No mixed strings are allowed, for example, a column containing a field with the value X'F10EF1F10F' could not be saved over a DBCS-either column (0E and 0F are the DBCS shift-out and shift-in bracket characters).
- CHAR fields can be saved over DBCS-open and DBCS-either fields.
- CHAR fields cannot be saved over DBCS-only fields.
- Graphic-DBCS data can only be saved over graphic-DBCS data. No other type of data can be saved over graphic-DBCS data.
- The length rules for fixed-length and variable-length graphic-DBCS data types are the same as the length rules for fixed-length and variable-length character data types, but when determining valid combinations, you must take into account that the length of graphic-DBCS data is counted in characters and the length of character data is counted in bytes.
  - Fixed-length graphic-DBCS data can be saved over fixed-length graphic-DBCS data if the defined length of the source is less than or equal to the defined length of the target.
  - Fixed-length graphic-DBCS data can be saved over variable-length graphic-DBCS data if the defined length of the source is less than or equal to the defined length of the target.
  - Variable-length graphic-DBCS data can be saved over variable-length graphic-DBCS data if the length of the data in the source is less than or equal to the defined length of the target.
  - Variable-length graphic-DBCS data can be saved over fixed-length graphic-DBCS data if the length of the data in the source is less than or equal to the defined length of the target.

Using DBCS global variables in DB2 UDB for iSeries Query Management commands

DB2 UDB for iSeries Query Management global variables can only be of types CHAR and INTEGER. DB2 UDB for iSeries Query Management does not support a global variable of type GRAPHIC. If you need to substitute a graphic-DBCS constant into the SQL statement, the global variable that is used should be set to be type CHAR. Because DB2 UDB for iSeries SQL only allows graphic-DBCS constants in an SQL statement if the constant is in the form:

```
G'<A B C D E >'
```

you must be sure that the character string:

```
"G'<A B C D E >'"
```

is set in the variable pool. For example, if the query to be run is:

```
    SELECT * FROM Table WHERE (ColumnName=&ComparisonValue
```

and if the column called ColumnName is a GRAPHIC type, the variable value for the global variable ComparisonValue should be a graphic-DBCS constant. The following example shows how the SET GLOBAL command could appear in the query procedure. Each SET GLOBAL command is functionally identical and is setting the global variable so that the SQL statement is built correctly.

```
   "  SET GLOBAL (ComparisonValue=G'<A B C D E >'  "
   "  SET GLOBAL (ComparisonValue=""G'<A B C D E >'""  "
   '  SET GLOBAL (ComparisonValue="G'<A B C D E >'"  '
```

In all three of these cases, the SQL statement to be run would be:

```
   SELECT * FROM Table WHERE (ColumnName=G'<A B C D E >'
```

See Appendix C, “Use of quotation marks and apostrophes when setting global variables in Query Management” for additional examples of setting global variables.

CL Commands

The following example shows how the STRQMQRY SETVAR() would be coded to successfully run the query above.

```
STRQMQRY QMQRY(QueryName) SETVAR(('ComparisonValue' 'G'<A B C D E >''')
```

Exporting DBCS data in Query Management

Data defined as graphic and variable graphic can be exported.

The data type codes for the header records of exported data are:

464 for VARGRAPHIC

468 for GRAPHIC.

Data defined as character that contains bracketed-DBCS data can be exported. Data defined as graphic can be exported.

The column width of exported data is the number of DBCS characters in the data, which is half the number of bytes used to store the data. Column data is stored in the data record exactly as it comes from the database.

Importing DBCS data in Query Management

DBCS data can be imported in queries, procedures, and forms. When importing DBCS queries and procedures this way, ensure that the record length does not exceed 79 bytes.

Printing DBCS reports in Query Management

If you are using DBCS data and the page splits, printing resumes on the second and subsequent pages of the report, at the fourth byte position from the left side of the page.

Appendix B. Example of DB2 UDB for iSeries Query Management interface

This appendix gives an example of how you can use the query management callable interface to create a report. Sample RPG and COBOL programs are provided to show how to create a procedure or program. You can work with this example on your system to assist in learning to use the DB2 UDB for iSeries Query Management callable interface. See Chapter 13, “Control language interface in Query Management” for an example of the CL interface to query management functions.

Producing a report in Query Management

Figure 74 illustrates using query management to select data and produce a report using predefined query management form (QMFORM) and query management query (QMQRY) objects. Run the SAMP1 program to produce the report.

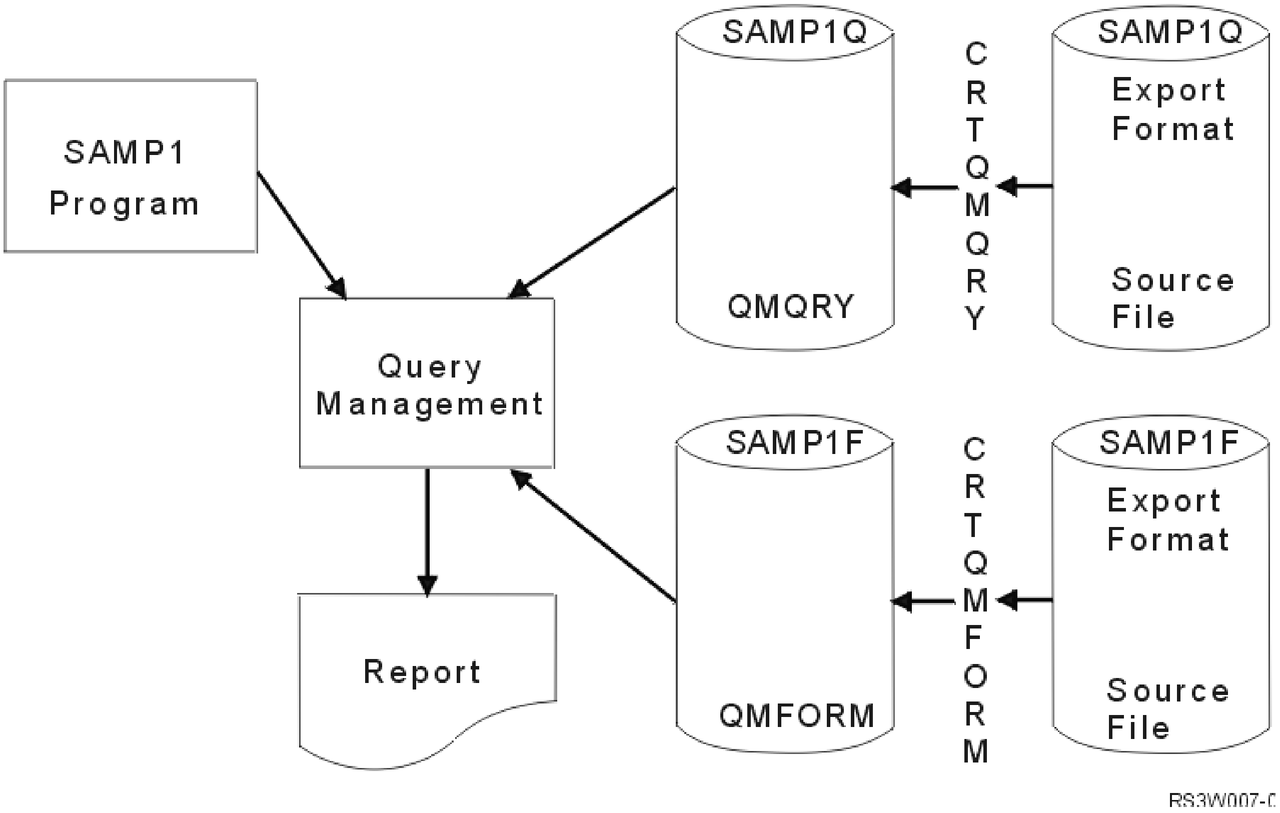


Figure 74. Overview of Using Query Management to Produce a Report

In this example, the database file WKPAY exists on the system and contains the following information: employee name, employee number, weekly hours worked, and the hourly rate of pay. Before running the SAMP1 program, create a QMQRY object and a QMFORM object on the system. Do this by first creating the source for the two objects in Query Management Query Source (QMQRYSRC) and Query Management Form Source (QMFORMSRC) files respectively. Then use the Create Query Management Query (CRTQMQRY) and Create Query Management Form (CRTQMFORM) commands to import them to QMQRY and QMFORM formats. Figure 75 on page 218 shows the data description specifications (DDS) for the WKPAY file.

```
      *
      * weekly payroll details
      *
     A                                      UNIQUE
     A          R PAYR                      TEXT('Weekly Pay Record')
     A            EMPNO          5          COLHDG('Employee' 'Number')
     A            NAME          20          COLHDG('Employee' 'Name')
     A            HOURS          5S 2       COLHDG('Hours' 'Worked')
     A            RATE           5S 2       COLHDG('Hourly' 'Rate')
     A            WKAMT          5S 2       COLHDG('Weekly' 'Pay')
     A          K EMPNO
```

Figure 75. Data Description Specifications for WKPAY File

Figure 76 shows the query used in the example:

```
    SELECT NAME, HOURS, RATE, WKAMT FROM BPLIB/WKPAY
    ORDER BY NAME
```

Figure 76. Query Source Select Statement

Note: The maximum length of this source file is 91 characters (with line number equal to 6 characters, date equal to 6 characters, and data equal to 79 characters). The SQL statement is organized to conform to OS/400 standards, since the option to use *SYS is specified in the START query command issued by the sample program.

Figure 77 shows the results of running the SAMP1 program.

```
DATE: 11/21/89           WEEKLY PAY REPORT        PAGE:    1

                           HOURS         HOURLY         PAY
EMPLOYEE NAME              WORKED        RATE          AMOUNT

ANDERSON, W                 38.50          6.23        100.00
COLLINS, R                  41.50          5.40        400.00
GREEN, C                    40.00          7.10        300.00
SMITH, T                    42.00          5.30        200.00

                                            TOTAL    1,000.00
```

Figure 77. Report Results for SAMP1

Sample programs in Query Management

The sample programs provided in RPG (Figure 78 on page 219) and COBOL (Figure 79 on page 221) perform the same functions. The programs process the WKPAY file by reading one record at a time, calculating the weekly pay amount, and then updating the file with the calculated information. When all the records are updated, query management is started with a call to the interface program QQXMAIN and passes the START command and the communications area. The START command specifies that the interface session uses system naming conventions (*SYS) and not the default (*SAA).

Perform the query by calling the callable interface and passing the RUN command. Print the results using the PRINT command. Use the EXIT command to end the interface, and end the program with control returning to the calling program.

When passing query management data to the interface, it is necessary to pass the lengths of commands and parameters in integer (binary) format. Structures have been set up in the program to allow data to be passed in this format.

A module containing the communications area is included in the program during compilation. Use this to communicate the status of operations between query management and the user program. The interface program name and standard field names for the query status are also defined in this include module. Use these names whenever required in the application program to allow for the transfer of query applications between systems.

Sample RPG program in Query Management

Figure 78 is a sample RPG program to process the WKPAY file.

```
 ****************************************************************
 *                                                              *
 *         SAMPLE 1 RPG PROGRAM USING QUERY INTERFACE           *
 *         ------------------------------------------           *
 *                                                              *
 *  1) Include member DSQCOMMR contains the communications      *
 *     area to be passed to the query management interface.     *
 *  2) The WKPAY weekly payroll details are read and the hours  *
 *     worked are multiplied by the hourly rate to calculate    *
 *     the weeks pay. The file is then updated with the weekly  *
 *     pay amount.                                              *
 *  3) Once all the records in the WKPAY file are updated then  *
 *     the interface is started and a query report              *
 *     printed using the just updated file.                     *
 *                                                              *
 ****************************************************************
H
FWKPAY   UF  E                     DISK
 *
E                    COM     1   4 25                 interface cmds
 *
I            DS
I                                    B   1   40BIN1
I                                    B   5   80BIN2
I                                    B   9  120BIN3
I                                    B  13  160BIN4
I/COPY QRPG/QIRGINC,DSQCOMMR
 *
 * Update the Weekly Pay file with weekly earnings:
 *
C           *IN50     DOUEQ'1'
C                     READ WKPAY                    50 EOF
C  N50      HOURS     MULT RATE      WKAMT     H      calculate pay
C  N50                UPDATPAYR                       update pay file
C                     END
 *                                                    ensure all
C                     FEOD WKPAY                      changes done
```

Figure 78. Sample RPG Program (Part 1 of 2)

```
      *                                                    in storage
      * Start the query interface session:
      *
     C                     CALL DSQCIR
     C                     PARM           DSQCOM           comms area
     C                     PARM 5         BIN1             command length
     C                     PARM           COM,1            START
     C                     PARM 1         BIN2             # keywords
     C                     PARM 8         BIN3             keyword length
     C                     PARM 'DSQSNAME'DATA8   8        keyword
     C                     PARM 4         BIN4             value length
     C                     PARM '*SYS'    DATA4   4        value
     C                     PARM DSQVCH    TYPE    4        CHAR
      *
      * Run the query:
      *
     C                     CALL DSQCIR
     C                     PARM           DSQCOM           comms area
     C                     PARM 16        BIN1             command length
     C                     PARM           COM,2            RUN QUERY
      *                                                      SAMP1Q
      * Print the results of the query:
      *
     C                     CALL DSQCIR
     C                     PARM           DSQCOM           comms area
     C                     PARM 25        BIN1             command length
     C                     PARM           COM,3            PRINT REPORT
      *                                                     (FORM=SAMP1F
      * End the query interface session:
      *
     C                     CALL DSQCIR
     C                     PARM           DSQCOM           comms area
     C                     PARM 4         BIN1             command length
     C                     PARM           COM,4            EXIT
      *
     C                     MOVE '1'       *INLR            end the program
      *
**    commands loaded as compile time array
START
RUN QUERY SAMP1Q
PRINT REPORT (FORM=SAMP1F
EXIT
```

Figure 78. Sample RPG Program (Part 2 of 2)

Sample COBOL program in Query Management

Figure 79 on page 221 is a sample COBOL program to process the WKPAY file.

```
  IDENTIFICATION DIVISION.
  PROGRAM-ID.     SAMP1.
  DATE-COMPILED.
  ENVIRONMENT DIVISION.
  CONFIGURATION SECTION
  SOURCE-COMPUTER.    IBM-AS400.
  OBJECT-COMPUTER.    IBM-AS400.
  INPUT-OUTPUT SECTION.
  FILE-CONTROL.
      SELECT PAY-FILE
          ASSIGN TO DISK-WKPAY
          FILE STATUS IS PAY-FILE-STATUS.
  DATA DIVISION.
  FILE SECTION.
  FD  PAY-FILE LABEL RECORDS STANDARD.
  01  PAY-REC.
      COPY DDS-PAYR OF WKPAY.
  WORKING-STORAGE SECTION.
 *****************************************************************
 *                                                               *
 *         SAMPLE 1 COBOL PROGRAM USING QUERY INTERFACE          *
 *         --------------------------------------------          *
 *                                                               *
 *  1) Include member DSQCOMMB contains the communications      *
 *     area to be passed to the query management interface.     *
 *                                                               *
 *  2) The WKPAY weekly payroll details are read and the hours  *
 *     worked are multiplied by the hourly rate to calculate    *
 *     the weeks pay. The file is then updated with the weekly  *
 *     pay amount.                                               *
 *                                                               *
 *  3) Once all the records in the WKPAY file are updated,      *
 *     the interface is started and a query report              *
 *     printed using the file just updated.                     *
 *                                                               *
 *****************************************************************

* Include the Communications area

   COPY DSQCOMMB OF QLBL-QILBINC.

 * Query Interface Commands

  77   START-CMD           PIC X(5)   VALUE "START".
  77   KEYWORD-NAME        PIC X(8)   VALUE "DSQSNAME".
  77   NAME-VALUE          PIC X(4)   VALUE "*SYS".
  77   RUN-QUERY-CMD       PIC X(16)  VALUE "RUN QUERY SAMP1Q".
  77   PRINT-CMD           PIC X(25)
                           VALUE "PRINT REPORT (FORM=SAMP1F".
  77   EXIT-CMD            PIC X(4)   VALUE "EXIT".
```

Figure 79. Sample COBOL Program (Part 1 of 3)

```
 77   ONE                 PIC 9(8) USAGE IS BINARY VALUE 1.
 77   FOUR                PIC 9(8) USAGE IS BINARY VALUE 4.
 77   FIVE                PIC 9(8) USAGE IS BINARY VALUE 5.
 77   EIGHT               PIC 9(8) USAGE IS BINARY VALUE 8.
 77   SIXTEEN             PIC 9(8) USAGE IS BINARY VALUE 16.
 77   TWENTY-FIVE         PIC 9(8) USAGE IS BINARY VALUE 25.

 77   PAY-FILE-STATUS     PIC XX.

 01   FILE-END            PIC X VALUE SPACE.
      88 END-OF-FILE      VALUE "E".

 01   FILE-ERROR-INFO.
      05  OP-NAME         PIC X(7).
      05  FILLER          PIC X(20) VALUE " ERROR ON FILE WKPAY".
      05  FILLER          PIC X(18) VALUE " - FILE STATUS IS ".
      05  STATUS-VALUE    PIC XX.

 PROCEDURE DIVISION.
 DECLARATIVES.
 FILE-ERROR SECTION.
     USE AFTER STANDARD ERROR PROCEDURE ON PAY-FILE.
 FILE-ERROR-PARA.
     MOVE PAY-FILE-STATUS TO STATUS-VALUE.
     DISPLAY "FILE PROCESSING ERROR".
     DISPLAY FILE-ERROR-INFO.
     DISPLAY "PROCESSING ENDED DUE TO FILE ERROR".
     STOP RUN.
 END DECLARATIVES.

 FILE-UPDATE SECTION.
 OPEN-FILE.
     MOVE "OPEN" TO OP-NAME.
     OPEN I-O PAY-FILE.
     PERFORM READ-PAY-FILE THRU UPDATE-PAY-FILE
          UNTIL END-OF-FILE.
 READ-PAY-FILE.
     MOVE "READ" TO OP-NAME.
     READ PAY-FILE
          AT END SET END-OF-FILE TO TRUE
          MOVE "CLOSE" TO OP-NAME
          CLOSE PAY-FILE
          PERFORM PROCESS-QUERY.
 UPDATE-PAY-FILE.
     MULTIPLE HOURS BY RATE GIVING WKAMT ROUNDED.
     MOVE "UPDATE" TO OP-NAME.
     REWRITE PAY-REC.

* Query Interface command and parameter lengths

 PROCESS-QUERY SECTION.
 START-INTERFACE.
     CALL DSQCIB USING DSQCOMM, FIVE, START-CMD,
                       ONE, EIGHT, KEYWORD-NAME,
                       FOUR, NAME-VALUE, DSQ-VARIABLE-CHAR.
```

Figure 79. Sample COBOL Program (Part 2 of 3)

```
       RUN-QUERY.
           CALL DSQCIB USING DSQCOMM, SIXTEEN, RUN-QUERY-CMD.
       PRINT-REPORT.
           CALL DSQCIB USING DSQCOMM, TWENTY-FIVE, PRINT-CMD.
       EXIT-INTERFACE.
           CALL DSQCIB USING DSQCOMM, FOUR, EXIT-CMD.
           STOP RUN.
```

Figure 79. Sample COBOL Program (Part 3 of 3)

Query and form source in Query Management

The sample RPG and COBOL programs in Figure 78 on page 219 and Figure 79 on page 221 refer to query and form source files to produce a report. Figure 80 is the query source file (SAMP1Q) referred to in the sample programs.

```
SELECT NAME, HOURS, RATE, WKAMT FROM BPLIB/WKPAY
ORDER BY NAME
```

Figure 80. Sample Query Source

Figure 81 is the form source (SAMP1F) referred to in the sample programs.

```
H QM4 01 F 01 E E   E R 01 03 89/11/20 15:51
T 1110 004 005 1112 007 1115 006 1116 005 1118 003 1113 015
R CHAR    2      30    1   Employee_Name
R NUMERIC 2      8     2   Hours_Worked
R NUMERIC 2      8     3   Hourly_Rate
R NUMERIC 2      8     4   Weekly_Pay
E
```

Figure 81. Sample Form Source

Query and form printed output in Query Management

Figure 82 is the printed output resulting from running the command PRINT QUERY SAMP1Q on the query source (SAMP1Q) referred to in the sample programs.

```
                                              DB2 UDB for iSeries Query Management

 Query  . . . . . . :   SAMP1Q
   Library  . . . . :     BPLIB
 Text . . . . . . . :   Query
 SEQNBR *...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8...
 000001 SELECT NAME, HOURS, RATE, WKAMT FROM BPLIB/WKPAY
 000002 ORDER BY NAME
                                               * * * * *   END OF SOURCE   * * * * *
```

Figure 82. Printed Output of Query Source

Figure 83 is the printed output resulting from running the command PRINT FORM SAMP1F on the form source (SAMP1F) referred to in the sample programs.

```
                                                    DB2 UDB for iSeries Query Management

Form . . . . . . . :   SAMP1F
  Library  . . . . :     BPLIB
Text . . . . . . . :   Form layout for sample 1 program
                                                         Column Information
Nbr     Heading                  Usage       Type          Indent     Width     Edit      Seq
  1     Employee_Name                        CHAR             2         30                  1
  2     Hours_Worked                         NUMERIC          2          8                  2
  3     Hourly_Rate                          NUMERIC          2          8                  3
  4     Weekly_Pay                           NUMERIC          2          8                  4
                                                          Page Information
Heading text  . . . . . . . . . . . . . . . . :   NO
Blank lines before heading  . . . . . . . . . :   0
Blank lines after heading . . . . . . . . . . :   2
Footing text  . . . . . . . . . . . . . . . . :   NO
Blank lines before footing  . . . . . . . . . :   2
Blank lines after footing . . . . . . . . . . :   0
                                                         Final Information
Final text  . . . . . . . . . . . . . . . . . :   NO
New page for final text . . . . . . . . . . . :   NO
Put final summary at line . . . . . . . . . . :   1
Blank lines before text . . . . . . . . . . . :   0
                                                         Break Information
Break number  . . . . . . . . . . . . . . . . :   1
Columns with this break number  . . . . . . . :   NONE
Heading text  . . . . . . . . . . . . . . . . :   NO
New page for heading  . . . . . . . . . . . . :   NO
Blank lines before heading  . . . . . . . . . :   0
Blank lines after heading . . . . . . . . . . :   0
Repeat column headings  . . . . . . . . . . . :   NO
Footing text  . . . . . . . . . . . . . . . . :   NO
New page for footing  . . . . . . . . . . . . :   NO
Blank lines before footing  . . . . . . . . . :   0
Blank lines after footing . . . . . . . . . . :   1
Put break summary at line . . . . . . . . . . :   1
                                                         Break Information
Break number  . . . . . . . . . . . . . . . . :   2
Columns with this break number  . . . . . . . :   NONE
Heading text  . . . . . . . . . . . . . . . . :   NO
New page for heading  . . . . . . . . . . . . :   NO
Blank lines before heading  . . . . . . . . . :   0
Blank lines after heading . . . . . . . . . . :   0
Repeat column headings  . . . . . . . . . . . :   NO
```

Figure 83. Printed Output of Form Source (Part 1 of 3)

```
                                            DB2 UDB for iSeries Query Management
Form . . . . . . . :   SAMP1F
  Library  . . . . :     BPLIB
Text . . . . . . . :   Form layout for sample 1 program
Footing text  . . . . . . . . . . . . . . . :   NO
New page for footing  . . . . . . . . . . . :   NO
Blank lines before footing  . . . . . . . . :   0
Blank lines after footing . . . . . . . . . :   1
Put break summary at line . . . . . . . . . :   1
                                                        Break Information
Break number  . . . . . . . . . . . . . . . :   3
Columns with this break number  . . . . . . :   NONE
Heading text  . . . . . . . . . . . . . . . :   NO
New page for heading  . . . . . . . . . . . :   NO
Blank lines before heading  . . . . . . . . :   0
Blank lines after heading . . . . . . . . . :   0
Repeat column headings  . . . . . . . . . . :   NO
Footing text  . . . . . . . . . . . . . . . :   NO
New page for footing  . . . . . . . . . . . :   NO
Blank lines before footing  . . . . . . . . :   0
Blank lines after footing . . . . . . . . . :   1
Put break summary at line . . . . . . . . . :   1
                                                        Break Information
Break number  . . . . . . . . . . . . . . . :   4
Columns with this break number  . . . . . . :   NONE
Heading text  . . . . . . . . . . . . . . . :   NO
New page for heading  . . . . . . . . . . . :   NO
Blank lines before heading  . . . . . . . . :   0
Blank lines after heading . . . . . . . . . :   0
Repeat column headings  . . . . . . . . . . :   NO
Footing text  . . . . . . . . . . . . . . . :   NO
New page for footing  . . . . . . . . . . . :   NO
Blank lines before footing  . . . . . . . . :   0
Blank lines after footing . . . . . . . . . :   1
Put break summary at line . . . . . . . . . :   1
                                                        Break Information
Break number  . . . . . . . . . . . . . . . :   5
Columns with this break number  . . . . . . :   NONE
Heading text  . . . . . . . . . . . . . . . :   NO
New page for heading  . . . . . . . . . . . :   NO
Blank lines before heading  . . . . . . . . :   0
Blank lines after heading . . . . . . . . . :   0
Repeat column headings  . . . . . . . . . . :   NO
Footing text  . . . . . . . . . . . . . . . :   NO
New page for footing  . . . . . . . . . . . :   NO
```

Figure 83. Printed Output of Form Source (Part 2 of 3)

```
                                            DB2 UDB for iSeries Query Management

Form . . . . . . . :   SAMP1F
  Library  . . . . :     BPLIB
Text . . . . . . . :   Form layout for sample 1 program
Blank lines before footing  . . . . . . . . . :   0
Blank lines after footing . . . . . . . . . . :   1
Put break summary at line . . . . . . . . . . :   1
                                                        Break Information
Break number  . . . . . . . . . . . . . . . . :   6
Columns with this break number  . . . . . . . :   NONE
Heading text  . . . . . . . . . . . . . . . . :   NO
New page for heading  . . . . . . . . . . . . :   NO
Blank lines before heading  . . . . . . . . . :   0
Blank lines after heading . . . . . . . . . . :   0
Repeat column headings  . . . . . . . . . . . :   NO
Footing text  . . . . . . . . . . . . . . . . :   NO
New page for footing  . . . . . . . . . . . . :   NO
Blank lines before footing  . . . . . . . . . :   0
Blank lines after footing . . . . . . . . . . :   1
Put break summary at line . . . . . . . . . . :   1
                                                        Option Information
Detail line spacing . . . . . . . . . . . . . :   1
Outlining for break columns . . . . . . . . . :   YES
Default break text  . . . . . . . . . . . . . :   YES
Column wrapped lines kept on page . . . . . . :   YES
Column heading separators . . . . . . . . . . :   YES
Break summary separators  . . . . . . . . . . :   YES
Final summary separators  . . . . . . . . . . :   YES
                * * * * *   E N D  O F  C O M P U T E R  P R I N T O U T   * * * * *
```

Figure 83. Printed Output of Form Source (Part 3 of 3)

Appendix C. Use of quotation marks and apostrophes when setting global variables in Query Management

Determining how many quotation marks or apostrophes to use when setting up global variables within procedures or programs can be difficult. The number of quotation marks and apostrophes that must be used varies greatly, depending on what feature of DB2 UDB for iSeries Query Management is being used.

The sets of quotation marks or apostrophes are necessary because of the various levels of parsing that the system has to work through. If the right number of quotation marks are not provided, the query does not run correctly.

A general rule is:

> If sets of quotation marks or apostrophes are inside of other sets of quotation marks or apostrophes, and the type used (quotation mark or apostrophe) is the same throughout, then the inside sets must be doubled to preserve them.

The following examples show how quotation marks and apostrophes are used when setting a global variable that is used as a literal in an SQL statement. The examples show what happens when an embedded quotation mark or apostrophe is in the substituted variable.

The following SQL statement, requiring a variable substitution, is used:

```
SELECT * FROM CUSTINFO
WHERE CUSTNAME=&CUSTNAME
```

The following report is being produced:

```
CUSTNAME                  CUSTID
---------------   ---------------
O'Malley                     450
```

The actual SQL query that is run, with the variable substituted, in order to produce the report is:

```
SELECT * FROM CUSTINFO
       WHERE CUSTNAME='O''Malley'
```

Query global variable pool rules for question marks and apostrophes in Query Management

The value that is set in the global variable pool for variable CUSTNAME is:

```
'O''Malley'
```

To meet SQL statement rules, the embedded apostrophe is doubled. Because CUSTNAME is a character string data type, the compare value is surrounded by apostrophes.

CL command rules for question marks and apostrophes in Query Management

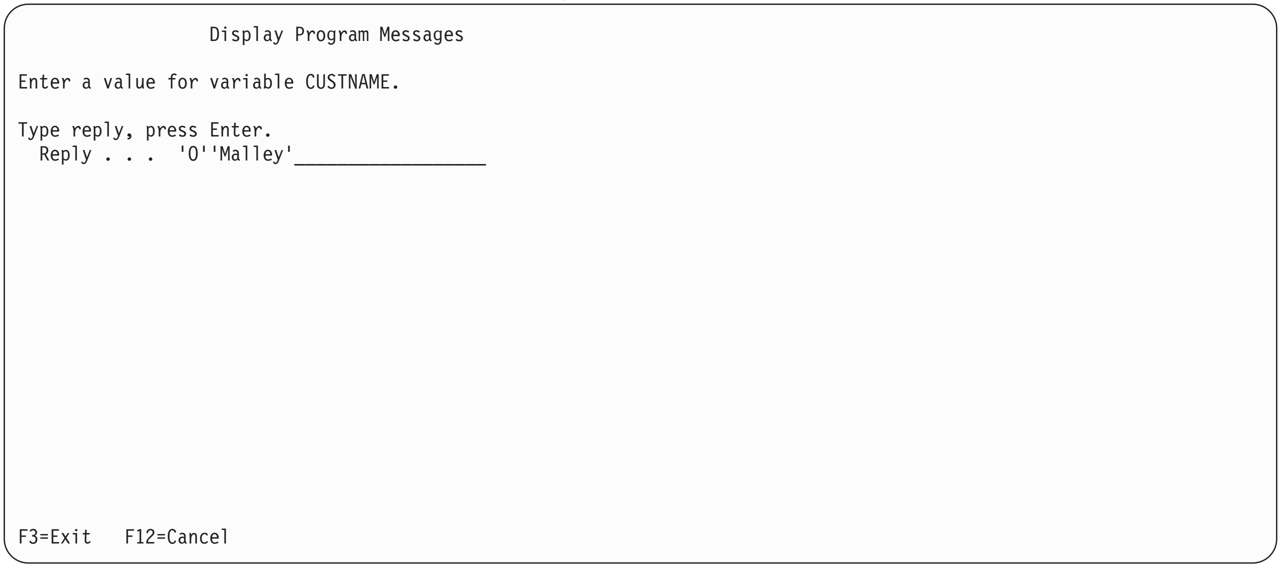
The CL command entered to run this is:

```
STRQMQRY  QMQRY(CUSTQRY)  SETVAR((CUSTNAME ' ''O''''Malley'' ')))
```

To meet CL command rules, the apostrophes embedded in the variable value is doubled and the entire string is surrounded by apostrophes.

Message prompt rules for question marks and apostrophes in Query Management

The QWM1913 message prompt occurs if the query is run interactively and the variable CUSTNAME has not been set. The value entered for the message prompt is:



```
                 Display Program Messages

Enter a value for variable CUSTNAME.

Type reply, press Enter.
  Reply . . .  'O''Malley'_________________

















F3=Exit   F12=Cancel
```

High-level language programming rules for question marks and apostrophes in Query Management

If the query command is entered through the callable interface (short version) from a high-level language program, the setup is:

```
SET GLOBAL (CUSTNAME=' ''O''''Malley'' '
```

To meet query command string variable value rules, the apostrophes embedded in the variable value are doubled and the entire string is surrounded by apostrophes. Following are high-level language examples:

- RPG

  See Figure 20 on page 113 for an example of how to setup a string.
- COBOL

  ```
  MOVE "SET GLOBAL (CUSTNAME=' ''O''''Malley'' '" TO COMMAND-STRING;
  ```
- C

  ```
  strcpy(command_string,"SET GLOBAL (CUSTNAME=' ''O''''Malley'' '");
  ```

Using a query procedure rules for question marks and apostrophes in Query Management

If the query command is entered through a query procedure, then the variable is set as follows:

```
'SET GLOBAL (CUSTNAME='' ''''O''''''''Malley'''' ''    '
```

When a SET GLOBAL command is run from a procedure, all the rules that apply to the callable interface command line also apply. To enter a command from a procedure, all of the apostrophes embedded in the command string must be doubled and the entire string must be surrounded by apostrophes.

Methods for simplification of variables in Query Management

You can use the following methods to simplify setting variables:

1. Use the extended version of the callable interface in C, COBOL or RPG to set the global variable value. The exact variable value is typed into the string. No extra quotation marks or apostrophes are needed other than those required to delimit the field or to indicate a character value.
   - COBOL

     ```
     MOVE "'O''Malley'" TO SET-VALUE;
     ```

   - C

     ```
     strcpy(set_value,"'O''Malley'");
     ```

2. Use quotation marks (″) as well as apostrophes.

   For a callable interface query command, use:

   ```
   SET GLOBAL (CUSTNAME="'O''Malley'"
   ```

   If the variable value in the command string is delimited by a quotation mark, then apostrophes imbedded in the value need not be doubled.

   ```
   "SET GLOBAL (CUSTNAME="" 'O''Malley' "" "
   ```

   When a SET GLOBAL command is done from a procedure, all the rules that apply to the callable interface command line also apply. To enter a command from a procedure, the command string must be surrounded by quotation marks (″) or apostrophes ('). If surrounded by quotation marks, all the quotation marks imbedded in the string must be doubled. If surrounded by apostrophes, all the apostrophes imbedded in the string must be doubled.

Appendix D. Examples of sort sequence in Query Management

This appendix provides sort sequence examples for multiple language environments. All example queries and reports are run against the STAFF table. The staff table contains information such as employee name and job. The staff table is as follows:

Table 17. STAFF Table

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
10	Sanders	20	Mgr	7	18357.50	0
20	Pernal	20	Sales	8	18171.25	612.45
30	Merenghi	38	MGR	5	17506.75	0
40	OBrien	38	Sales	6	18006.00	846.55
50	Hanes	15	Mgr	10	20659.80	0
60	Quigley	38	SALES	00	16808.30	650.25
70	Rothman	15	Sales	7	16502.83	1152.00
80	James	20	Clerk	0	13504.60	128.20
90	Koonitz	42	sales	6	18001.75	1386.70
100	Plotz	42	mgr	6	18352.80	0

You can use a sort sequence table to:

- Sort
- Group
- Join
- Select records
- Determine minimum and maximum values
- Cause report breaks for SBCS character data

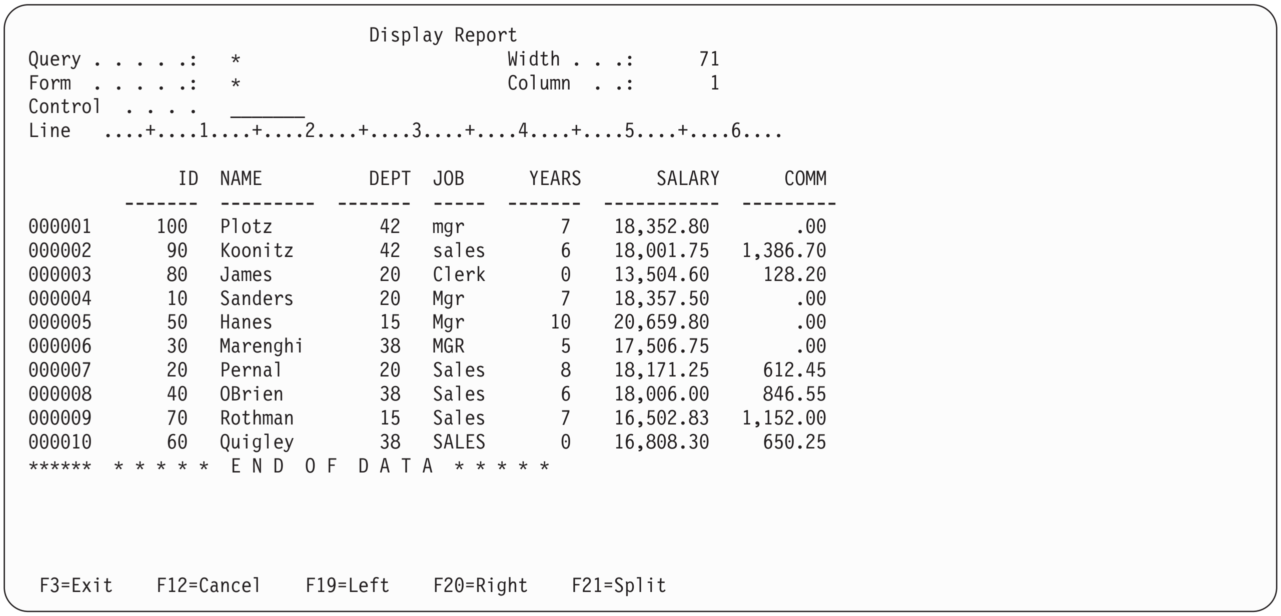
In the following examples, the queries and resulting reports used a binary character code sort sequence (*HEX), a shared-weight sort sequence (*LANGIDSHR), or a unique-weight sort sequence (*LANGIDUNQ).

Example of sort in Query Management

The following SQL statement causes a report to be sorted using the values in the JOB column of Table 17.

```
SELECT * FROM STAFF ORDER BY JOB
```

Figure 84 on page 232 shows how sorting is done with a *HEX sort sequence.

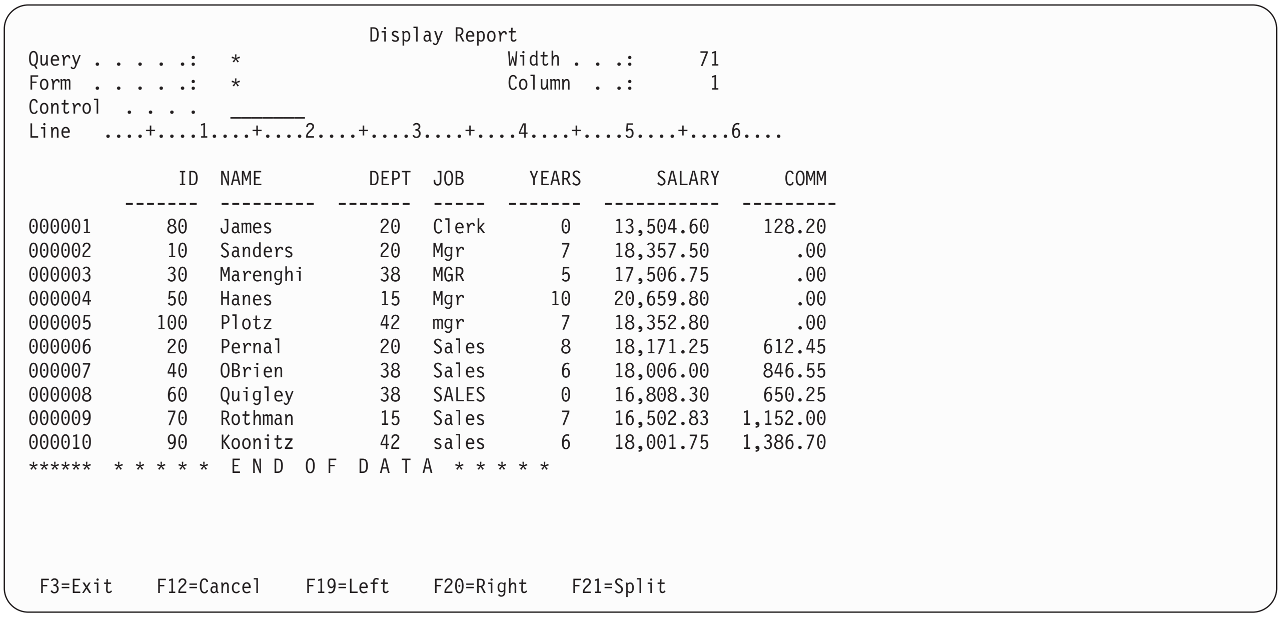


```
                               Display Report
Query . . . . .:   *                      Width . . .:      71
Form  . . . . .:   *                      Column  . .:       1
Control  . . . .   _______
Line   ....+....1....+....2....+....3....+....4....+....5....+....6....

              ID  NAME         DEPT  JOB      YEARS       SALARY      COMM
         -------  ---------  -------  -----  -------  -----------  ---------
000001      100   Plotz          42   mgr        7    18,352.80        .00
000002       90   Koonitz        42   sales      6    18,001.75   1,386.70
000003       80   James          20   Clerk      0    13,504.60     128.20
000004       10   Sanders        20   Mgr        7    18,357.50        .00
000005       50   Hanes          15   Mgr       10    20,659.80        .00
000006       30   Marenghi       38   MGR        5    17,506.75        .00
000007       20   Pernal         20   Sales      8    18,171.25     612.45
000008       40   OBrien         38   Sales      6    18,006.00     846.55
000009       70   Rothman        15   Sales      7    16,502.83   1,152.00
000010       60   Quigley        38   SALES      0    16,808.30     650.25
******  * * * * *  E N D  O F  D A T A  * * * * *




 F3=Exit    F12=Cancel    F19=Left    F20=Right    F21=Split
```

*Figure 84. SRTSEQ=*HEX. Example report showing sorting with no sort sequence*

Notice that the values in column JOB are in mixed case. There are values of **Mgr**, **MGR**, and **mgr**. The rows are sorted by the value in the column JOB, but the uppercase **MGR** is treated differently than the lowercase **mgr**. For this reason, the values of **Mgr**, **MGR** and **mgr** do not appear on adjacent rows. Also, the value **sales** is less than the value **Mgr**.

Figure 85 shows how sorting is done with a shared-weight sort sequence.

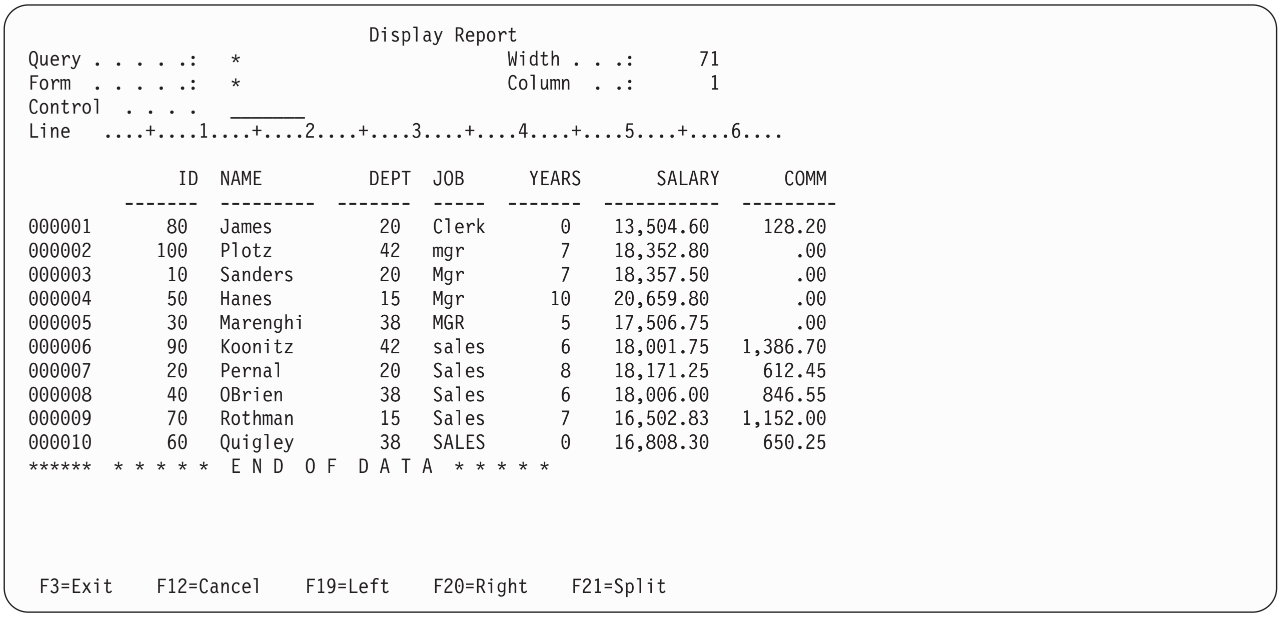


```
                               Display Report
Query . . . . .:   *                      Width . . .:      71
Form  . . . . .:   *                      Column  . .:       1
Control  . . . .   _______
Line   ....+....1....+....2....+....3....+....4....+....5....+....6....

              ID  NAME         DEPT  JOB      YEARS       SALARY      COMM
         -------  ---------  -------  -----  -------  -----------  ---------
000001       80   James          20   Clerk      0    13,504.60     128.20
000002       10   Sanders        20   Mgr        7    18,357.50        .00
000003       30   Marenghi       38   MGR        5    17,506.75        .00
000004       50   Hanes          15   Mgr       10    20,659.80        .00
000005      100   Plotz          42   mgr        7    18,352.80        .00
000006       20   Pernal         20   Sales      8    18,171.25     612.45
000007       40   OBrien         38   Sales      6    18,006.00     846.55
000008       60   Quigley        38   SALES      0    16,808.30     650.25
000009       70   Rothman        15   Sales      7    16,502.83   1,152.00
000010       90   Koonitz        42   sales      6    18,001.75   1,386.70
******  * * * * *  E N D  O F  D A T A  * * * * *




 F3=Exit    F12=Cancel    F19=Left    F20=Right    F21=Split
```

*Figure 85. SRTSEQ=*LANGIDSHR, LANGID=ENU. Example report showing sorting with a shared-weight sort sequence*

The rows are sorted by the value in the column JOB. Lowercase letters are treated the same as uppercase letters. Notice that in Figure 85, all the values (**mgr**, **Mgr** and **MGR**) are together.

Figure 86 shows how sorting is done with a unique-weight sort sequence.



```
                                Display Report
Query . . . . .:   *                     Width . . .:      71
Form  . . . . .:   *                     Column  . .:       1
Control  . . . .   _______
Line   ....+....1....+....2....+....3....+....4....+....5....+....6....

              ID  NAME          DEPT  JOB      YEARS       SALARY      COMM
         -------  ---------  -------  -----  -------  -----------  ---------
000001       80   James         20   Clerk       0    13,504.60     128.20
000002      100   Plotz         42   mgr         7    18,352.80        .00
000003       10   Sanders       20   Mgr         7    18,357.50        .00
000004       50   Hanes         15   Mgr        10    20,659.80        .00
000005       30   Marenghi      38   MGR         5    17,506.75        .00
000006       90   Koonitz       42   sales       6    18,001.75   1,386.70
000007       20   Pernal        20   Sales       8    18,171.25     612.45
000008       40   OBrien        38   Sales       6    18,006.00     846.55
000009       70   Rothman       15   Sales       7    16,502.83   1,152.00
000010       60   Quigley       38   SALES       0    16,808.30     650.25
******  * * * * *  E N D  O F  D A T A  * * * * *




 F3=Exit    F12=Cancel    F19=Left    F20=Right    F21=Split
```

*Figure 86. SRTSEQ=*LANGIDUNQ, LANGID=ENU. Example report showing sorting with a unique-weight sort sequence*

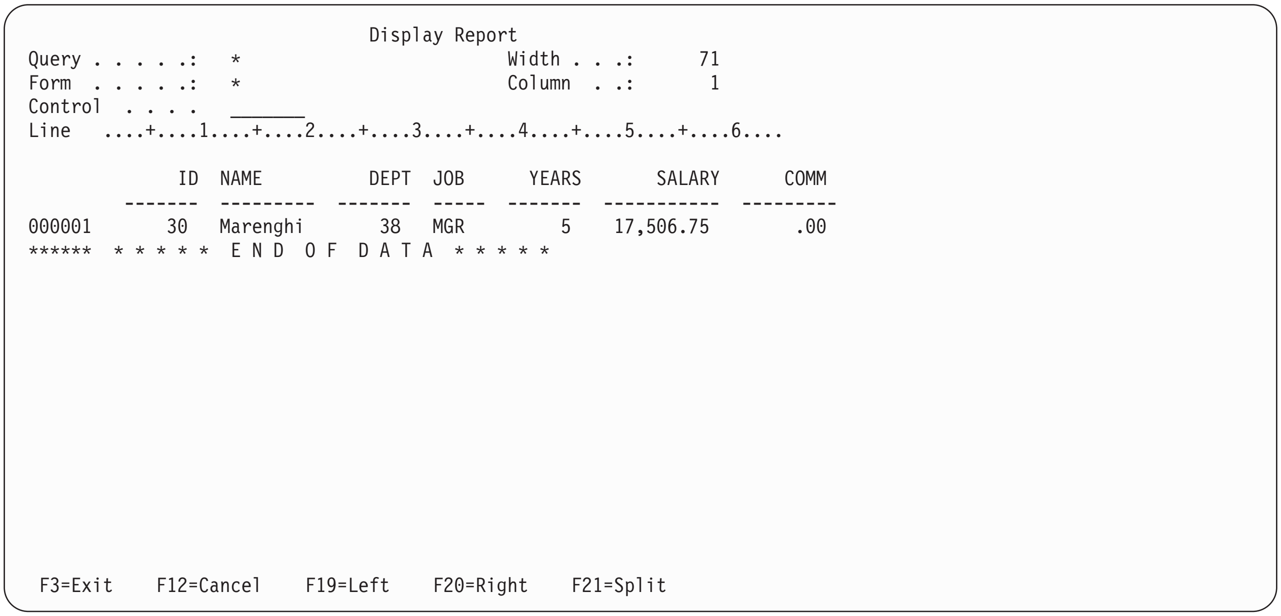
The rows are sorted by the value in the column JOB. This is the sorting that one would see in a national language dictionary. Lowercase and uppercase letters are treated as unique, but they have a weight which causes adjacent sorting of the lowercase and uppercase letters. The lowercase letters sort before the uppercase letters.

Example of record selection in Query Management

The following SQL statement causes the report to show the records that have the value **MGR** in the JOB column.

```
SELECT * FROM STAFF WHERE JOB=MGR
```

Figure 87 on page 234 shows how record selection is done with a *HEX sort sequence.

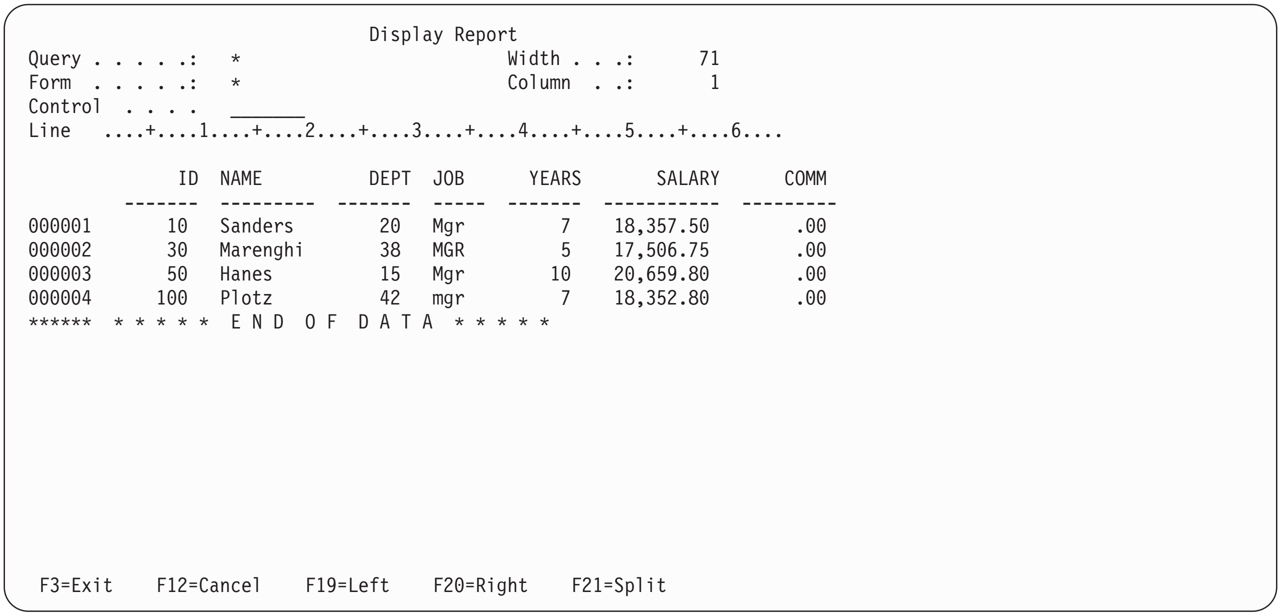


```
                              Display Report
Query . . . . .:   *                     Width . . .:      71
Form  . . . . .:   *                     Column  . .:       1
Control  . . . .   _______
Line   ....+....1....+....2....+....3....+....4....+....5....+....6....

              ID  NAME         DEPT  JOB      YEARS       SALARY      COMM
         -------  ---------  -------  -----  -------  -----------  ---------
000001        30  Marenghi       38  MGR         5    17,506.75        .00
******  * * * * *  E N D  O F  D A T A  * * * * *

 F3=Exit    F12=Cancel    F19=Left    F20=Right    F21=Split
```

*Figure 87. SRTSEQ=*HEX. Example report showing record selection with no sort sequence*

In Figure 87, the rows that match the record selection criteria for the column JOB are selected. The lowercase **mgr** is not treated the same as the uppercase **MGR**. Rows are not selected for the value **Mgr** or the value **mgr**. **MGR** is the only value for which a row is selected.

Figure 88 shows how record selection would be done with a shared-weight sort sequence.

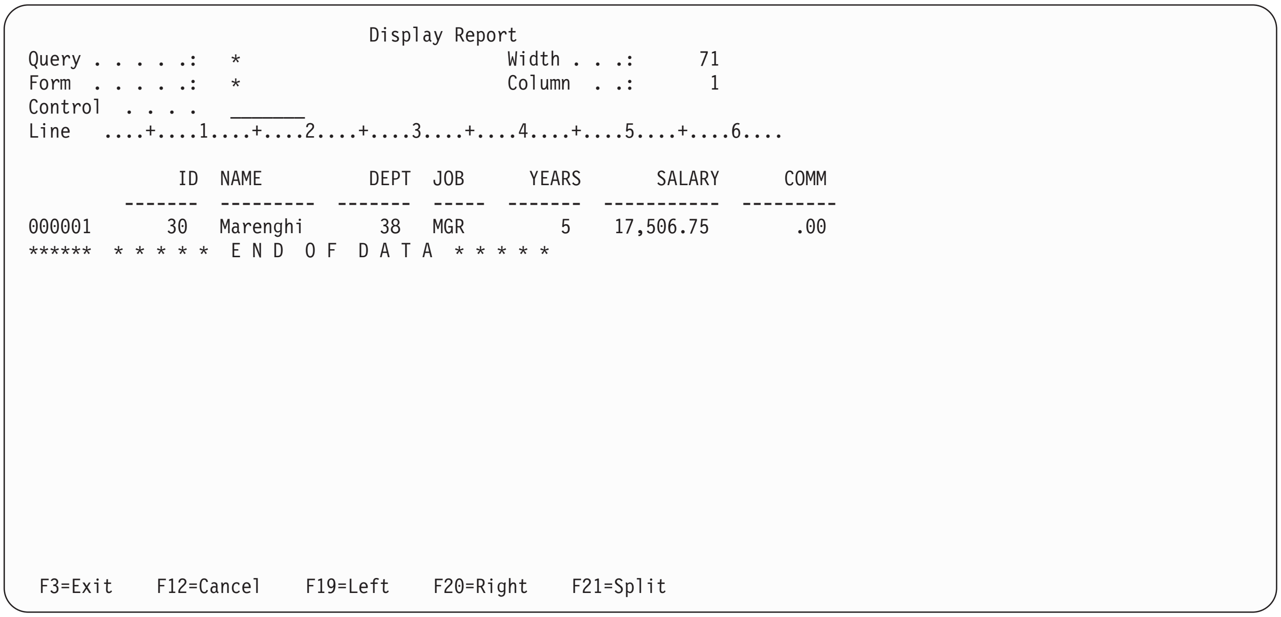


```
                              Display Report
Query . . . . .:   *                     Width . . .:      71
Form  . . . . .:   *                     Column  . .:       1
Control  . . . .   _______
Line   ....+....1....+....2....+....3....+....4....+....5....+....6....

              ID  NAME         DEPT  JOB      YEARS       SALARY      COMM
         -------  ---------  -------  -----  -------  -----------  ---------
000001        10  Sanders        20  Mgr         7    18,357.50        .00
000002        30  Marenghi       38  MGR         5    17,506.75        .00
000003        50  Hanes          15  Mgr        10    20,659.80        .00
000004       100  Plotz          42  mgr         7    18,352.80        .00
******  * * * * *  E N D  O F  D A T A  * * * * *

 F3=Exit    F12=Cancel    F19=Left    F20=Right    F21=Split
```

*Figure 88. SRTSEQ=*LANGIDSHR, LANGID=ENU. Example report showing record selection with a shared-weight sort sequence*

In Figure 88 the rows that match the record selection criteria for the column JOB are selected by treating uppercase letters the same as lowercase letters. The values **mgr**, **Mgr**, and **MGR** are all selected.

Figure 89 shows how record selection is done with a unique-weight sort sequence.



```
                                Display Report
Query . . . . .:   *                     Width . . .:      71
Form  . . . . .:   *                     Column  . .:       1
Control  . . . .   _______
Line   ....+....1....+....2....+....3....+....4....+....5....+....6....

              ID  NAME          DEPT  JOB      YEARS       SALARY      COMM
         -------  ---------  -------  -----  -------  -----------  ---------
000001       30   Marenghi       38   MGR         5    17,506.75        .00
******  * * * * *  E N D  O F  D A T A  * * * * *
```

F3=Exit   F12=Cancel   F19=Left   F20=Right   F21=Split

*Figure 89. SRTSEQ=*LANGIDUNQ, LANGID=ENU. Example report showing record selection with a unique-weight sort sequence*

In Figure 89, because the lowercase and uppercase letters are treated as unique, the lowercase **mgr** is not treated the same as uppercase **MGR**. So, **MGR** is not selected.

Example of report breaks in Query Management

The following SQL statement causes the report to be sorted using the values in the JOB column.

```
SELECT * FROM STAFF ORDER BY JOB
```

The report shown in Figure 90 on page 236 shows report breaks with a *HEX sort sequence. BREAK1 is used on the JOB column and SUM is used on the SALARY column.

```
     ID  NAME         DEPT  JOB      YEARS       SALARY       COMM
-------  ---------  -------  -----  -------  -----------  ---------
   100   Plotz          42   mgr         7    18,352.80         .00
                                             -----------
                                         *    18,352.80

    90   Koonitz        42   sales       6    18,001.75    1,386.70
                                             -----------
                                         *    18,001.75

    80   James          20   Clerk       0    13,504.60      128.20
                                             -----------
                                         *    13,504.60

    10   Sanders        20   Mgr         7    18,357.50         .00
    50   Hanes          15              10    20,659.80         .00
                                             -----------
                                         *    39,017.30

    30   Marenghi       38   MGR         5    17,506.75         .00
                                             -----------
                                         *    17,506.75

    20   Pernal         20   Sales       8    18,171.25      612.45
    40   OBrien         38               6    18,006.00      846.55
    70   Rothman        15               7    16,502.83    1,152.00
                                             -----------
                                         *    52,680.08

    60   Quigley        38   SALES       0    16,808.30      650.25
                                             -----------
                                         *    16,808.30
```

*Figure 90. SRTSEQ=*HEX. Example report showing report breaks with no sort sequence*

In Figure 90, the rows are grouped for report breaks by the value in the column JOB whether some of the values are uppercase or some are lowercase. The uppercase **MGR** is not treated the same as the mixed case **Mgr** and so is not grouped into the same break level.

Figure 91 on page 237 shows report breaks with a shared-weight sort sequence.

```
     ID  NAME          DEPT  JOB      YEARS       SALARY      COMM
-------  ---------  -------  -----  -------  -----------  ---------
     80  James           20  Clerk        0    13,504.60     128.20
                                              -----------
                                          *    13,504.60

     10  Sanders         20  Mgr          7    18,357.50        .00
     30  Marenghi        38               5    17,506.75        .00
     50  Hanes           15              10    20,659.80        .00
    100  Plotz           42               7    18,352.80        .00
                                              -----------
                                          *    74,876.85

     20  Pernal          20  Sales        8    18,171.25     612.45
     40  OBrien          38               6    18,006.00     846.55
     60  Quigley         38               0    16,808.30     650.25
     70  Rothman         15               7    16,502.83   1,152.00
     90  Koonitz         42               6    18,001.75   1,386.70
                                              -----------
                                          *    87,490.13
```

*Figure 91. SRTSEQ=*LANGIDSHR, LANGID=ENU. Example report showing report breaks with a shared-weight sort sequence*

In Figure 91, the rows are grouped for report breaks by the value in the column JOB. Lowercase letters are treated the same as uppercase letters. All the values (**mgr**, **Mgr**, and **MGR**) are grouped together in the same break level.

Figure 92 on page 238 shows report breaks with a unique-weight sort sequence.

In Figure 92 on page 238, the rows are sorted by the value in the column JOB. Lowercase and uppercase letters are treated as unique, but they have a weight which causes adjacent sorting of the lowercase and uppercase letters. The lowercase letters sort before the uppercase letters. When the rows are grouped into break levels, only the values that are the same are grouped together.

```
     ID  NAME          DEPT  JOB     YEARS       SALARY       COMM
-------  ---------  -------  -----  -------  -----------  ---------
    80   James          20   Clerk       0    13,504.60     128.20
                                             -----------
                                         *    13,504.60

   100   Plotz          42   mgr         7    18,352.80        .00
                                             -----------
                                         *    18,352.80

    10   Sanders        20   Mgr         7    18,357.50        .00
    50   Hanes          15   Mgr        10    20,659.80        .00
                                             -----------
                                         *    39,017.30

    30   Marenghi       38   MGR         5    17,506.75        .00
                                             -----------
                                         *    17,506.75

    90   Koonitz        42   sales       6    18,001.75   1,386.70
                                             -----------
                                         *    18,001.75

    20   Pernal         20   Sales       8    18,171.25     612.45
    40   OBrien         38   Sales       6    18,006.00     846.55
    70   Rothman        15   Sales       7    16,502.83   1,152.00
                                             -----------
                                         *    52,680.13

    60   Quigley        38   SALES       0    16,808.30     650.25
                                             -----------
                                         *    16,808.30
```

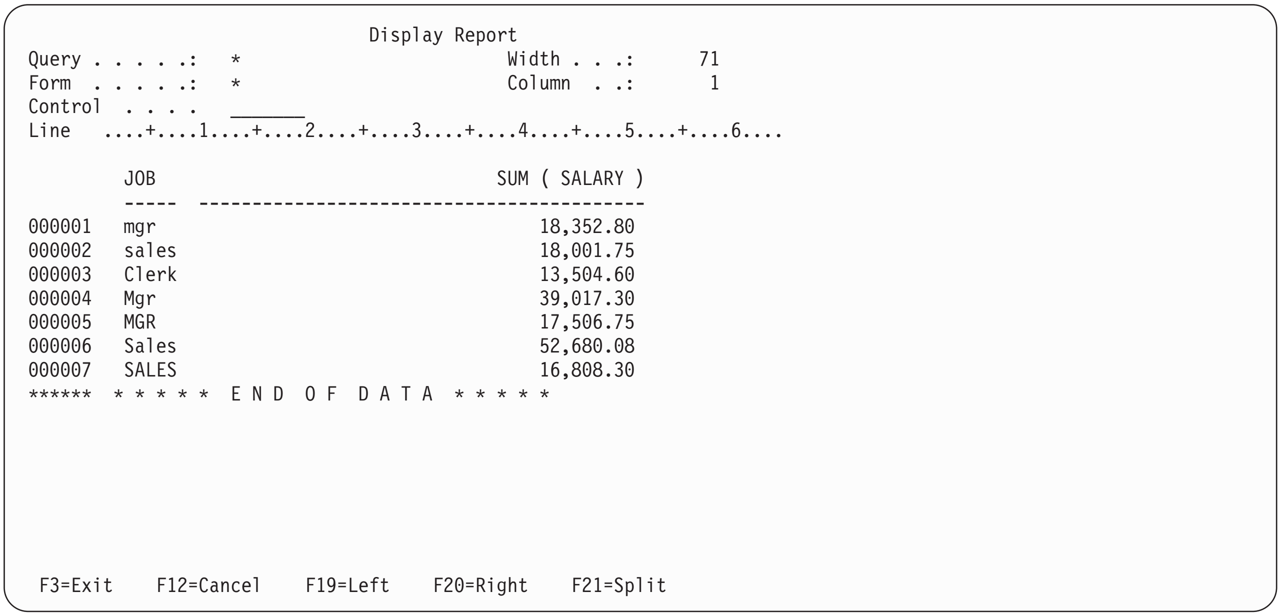
*Figure 92. SRTSEQ=*LANGIDUNQ, LANGID=ENU. Example report showing report breaks with a unique-weight sort sequence*

Example of grouping in Query Management

The following SQL statement causes the summary data to be grouped using the values in the JOB column.

```
SELECT JOB, SUM(SALARY) FROM STAFF GROUP BY JOB ORDER BY JOB
```

Figure 93 on page 239 shows query grouping with a *HEX sort sequence.

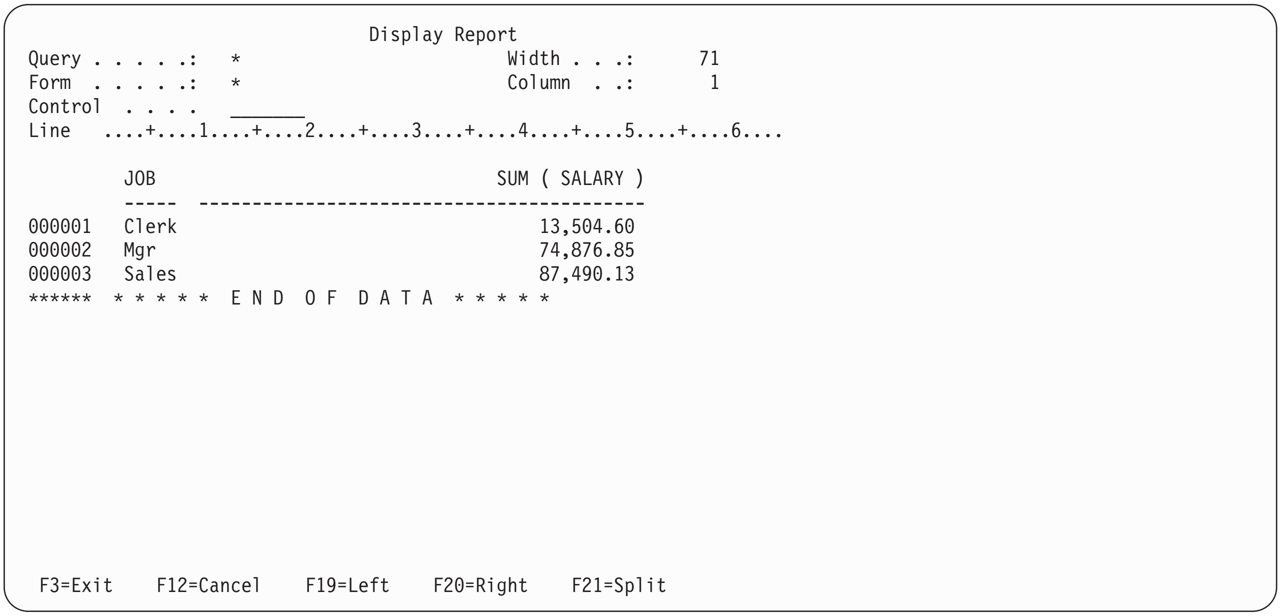


```
                              Display Report
Query . . . . .:   *                    Width . . .:      71
Form  . . . . .:   *                    Column  . .:       1
Control  . . . .   _______
Line   ....+....1....+....2....+....3....+....4....+....5....+....6....

         JOB                             SUM ( SALARY )
         -----  ------------------------------------------
000001   mgr                                  18,352.80
000002   sales                                18,001.75
000003   Clerk                                13,504.60
000004   Mgr                                  39,017.30
000005   MGR                                  17,506.75
000006   Sales                                52,680.08
000007   SALES                                16,808.30
******  * * * * *  E N D  O F  D A T A  * * * * *








 F3=Exit    F12=Cancel    F19=Left    F20=Right    F21=Split
```

*Figure 93. SRTSEQ=*HEX. Example report showing grouping with no sort sequence*

In Figure 93, the rows are grouped for report breaks by the value in the column JOB. Because **MGR** is not treated the same as **Mgr**, these values are not grouped in the same break level.

Figure 94 shows how grouping is done for a shared-weight sort sequence.

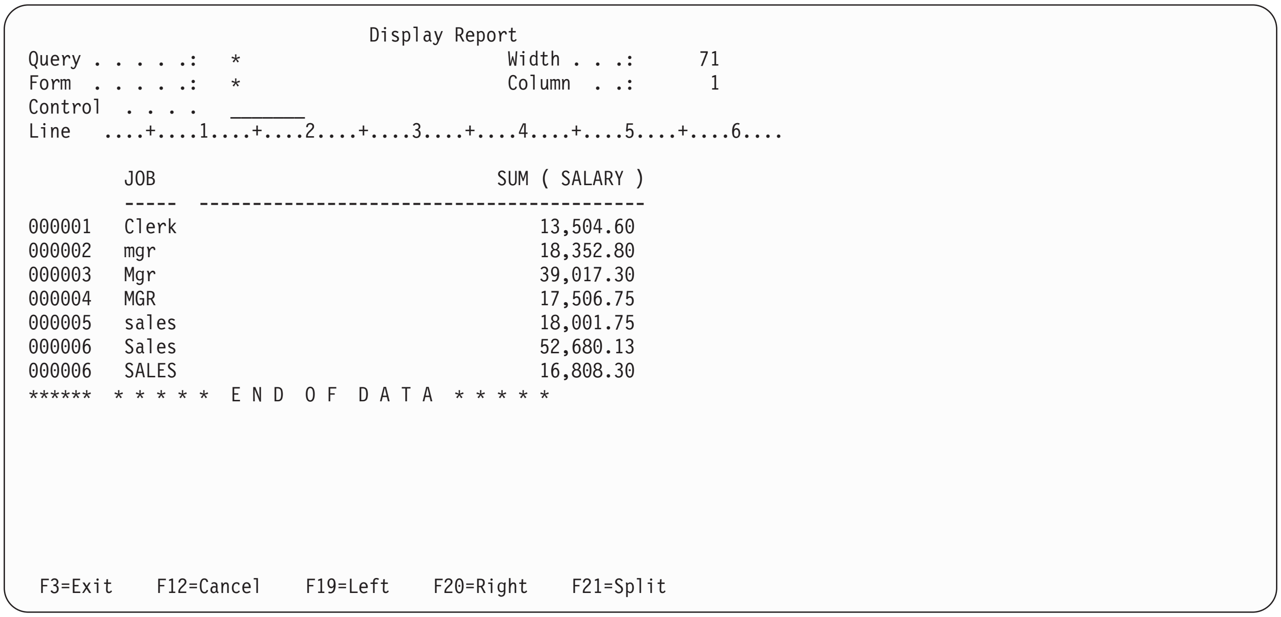


```
                              Display Report
Query . . . . .:   *                    Width . . .:      71
Form  . . . . .:   *                    Column  . .:       1
Control  . . . .   _______
Line   ....+....1....+....2....+....3....+....4....+....5....+....6....

         JOB                             SUM ( SALARY )
         -----  ------------------------------------------
000001   Clerk                                13,504.60
000002   Mgr                                  74,876.85
000003   Sales                                87,490.13
******  * * * * *  E N D  O F  D A T A  * * * * *












 F3=Exit    F12=Cancel    F19=Left    F20=Right    F21=Split
```

*Figure 94. SRTSEQ=*LANGIDSHR, LANGID=ENU. Example report showing grouping with a shared-weight sort sequence*

In Figure 94, the rows are grouped for report breaks by the value in the column JOB. Lowercase letters are treated the same as uppercase letters. All the values (**mgr**, **Mgr** and **MGR**) are grouped together.

Figure 95 shows how grouping is done with a unique-weight sort sequence.



```
                                Display Report
Query . . . . .:   *                    Width . . .:      71
Form  . . . . .:   *                    Column  . .:       1
Control  . . . .   _______
Line   ....+....1....+....2....+....3....+....4....+....5....+....6....

         JOB                                 SUM ( SALARY )
         -----  ------------------------------------------
000001   Clerk                                  13,504.60
000002   mgr                                    18,352.80
000003   Mgr                                    39,017.30
000004   MGR                                    17,506.75
000005   sales                                  18,001.75
000006   Sales                                  52,680.13
000006   SALES                                  16,808.30
******  * * * * *  E N D  O F  D A T A  * * * * *







 F3=Exit    F12=Cancel    F19=Left    F20=Right    F21=Split
```

*Figure 95. SRTSEQ=*LANGIDUNQ, LANGID=ENU. Example report showing grouping with a unique-weight sort sequence*

In Figure 95, the rows are sorted by the value in the column JOB. Lowercase and uppercase letters are treated as unique, but they have a weight which causes adjacent sorting. Lowercase letters sort before uppercase letters. For this reason, the value **mgr** appears before the value **Mgr**, and the value **Mgr** appears before the value **MGR**.

When the rows are grouped into break levels, only the values that are the same are grouped together.

Break summary use in Query Management

In Figure 93 on page 239, the rows are grouped for report breaks by the value in the column JOB. The uppercase **MGR** is not treated the same as the mixed case **Mgr**. **MGR** and **mgr** are not grouped in the the same break level.

Appendix E. Notices

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

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

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

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

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

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

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

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

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

IBM Corporation
Software Interoperability Coordinator
3605 Highway 52 N
Rochester, MN 55901-7829
U.S.A.

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which 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. 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 IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

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

Trademarks

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

C/400
DB2
Distributed Relational Database Architecture
DRDA
e (logo) IBM
IBM
iSeries
MVS
Operating System/400
OS/400
QMF
RPG/400
System/36
SQL/DS

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

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

Index

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

iSeries
DB2 Universal Database for iSeries
Query Management Programming
Version 5

Publication No. SC41-5703-05

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

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	□	□	□	□	□

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

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	□	□	□	□	□
Complete	□	□	□	□	□
Easy to find	□	□	□	□	□
Easy to understand	□	□	□	□	□
Well organized	□	□	□	□	□
Applicable to your tasks	□	□	□	□	□

Please tell us how we can improve this book:

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

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

Name

Address

Company or Organization

Phone No.

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

SC41-5703-05

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

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

POSTAGE WILL BE PAID BY ADDRESSEE

IBM CORPORATION
ATTN DEPT 542 IDCLERK
3605 HWY 52 N
ROCHESTER MN 55901-7829

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold
Along Line

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

SC41-5703-05