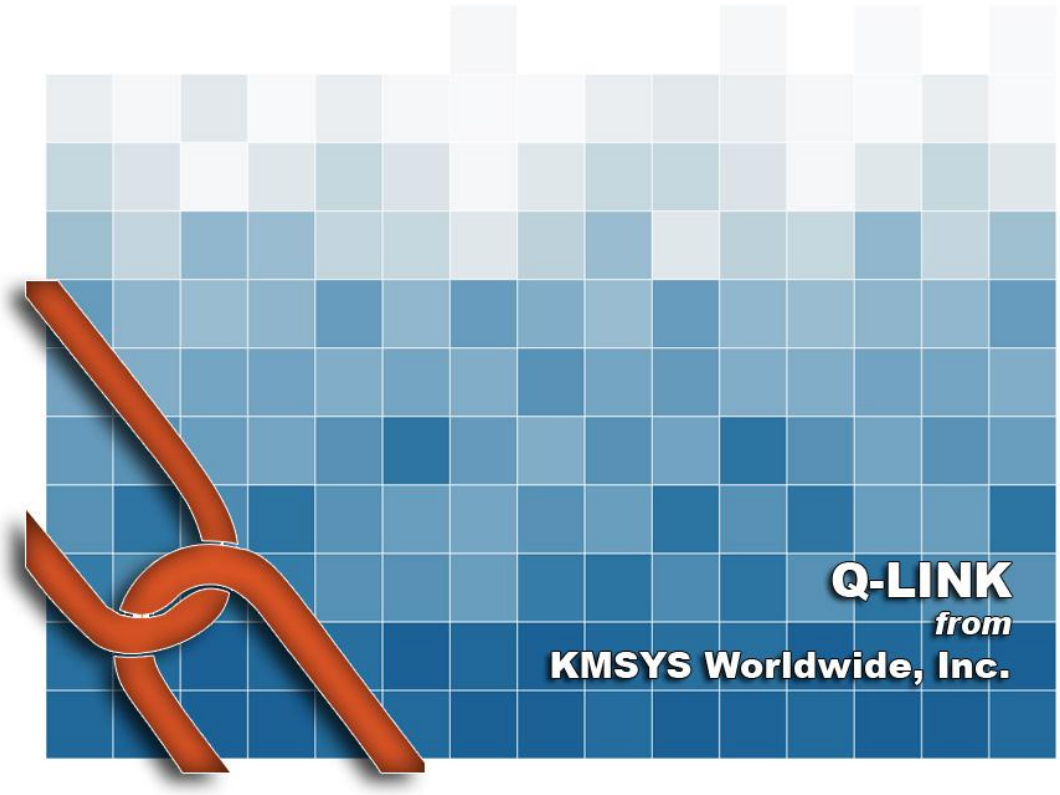


KMSYS
WORLDWIDE, INC

Application Development User Guide

10 August 2011

The information contained in this document is the latest available at the time of preparation; therefore, it may be changed without notice, and it does not represent a commitment on the part of KMSYS Worldwide, Inc. The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy this software on magnetic tape, disk, or any other medium for any purpose other than stated in the terms of the agreement, or without the express written permission of KMSYS Worldwide, Inc.

This material constitutes proprietary and confidential property of KMSYS Worldwide, Inc., having substantial monetary value and is solely the property of KMSYS Worldwide, Inc. This property is disclosed to the recipient thereof in confidence only and pursuant to the terms and conditions and for the purpose set forth in written agreements by and between KMSYS Worldwide, Inc., and the recipient of this material.

If you have any comments about the software or documentation, notify KMSYS Worldwide, Inc., in writing at the following address:

KMSYS Worldwide, Inc.
P.O. Box 669695
Marietta, Georgia 30066
U.S.A.

Technical Support (770) 635-6363 - Main Number (770) 635-6350 - Fax (770) 635-6351

Q-LINK, Release 6R6, November 1999

RESTRICTED RIGHTS LEGEND

If this Product is acquired by or for the U.S. Government, then it is provided with Restricted Rights. Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, or subparagraphs (c)(1) and (2) of the Commercial Computer Software - Restricted Rights at 48 CFR 52.227-19, or clause 18-52.227-86(d) of the NASA Supplement to the FAR, as applicable.

Table of Contents

Chapter 1: Introduction

This manual is a guide to using the Q-LINK software in conjunction with BIS as an applications development tool in lieu of other programming languages used in the development and implementation of application systems.

The following chapters will introduce the reader to the Q-LINK environment and the Q-LINK Processor, its basic structure, and modes of operations. Then, the reader will be led through the fundamentals of using Q-LINK. The remainder of the manual will discuss elementary through advanced usage techniques.

This manual will assume that the reader has a basic understanding of both BIS manual functions and BIS run writing. The reader is also expected to have minimal knowledge of some of the following: COBOL programming, PCIOS file handling, RDMS SQL and DMS 2200 database-programming techniques. Consult the appropriate Unisys Series 2200 reference manuals when more information is needed in these areas.

Throughout the manual, references will be made to the Q-LINK Programmer Reference (Programmer Reference) for detailed descriptions of items covered. The Q-LINK Programmer Reference should be available when working with this guide.

1.1 Notation Conventions

Throughout this User Guide, the following conventions will be used in presenting examples:

1. Comments shown within an example will be preceded by a less-than sign and a dash (<-). They are not intended to be part of the text, and are added only to help clarify the purpose of each step. Comments that may actually appear in text are preceded by the period-space (.) sequence; however, when a period-space sequence is used in a quoted string literal, it is NOT a comment, but a part of the literal.
2. The caret (^) character will be used to indicate the presence of a TAB character.

1.2 Key Word Abbreviation

Q-LINK provides for the abbreviation of many key words, such as command names and directives. Most examples shown in this manual will be unabbreviated; however, abbreviations will be used from time to time. A complete list of key word abbreviations will be found in the Q-LINK Programmer Reference.

It is also possible to abbreviate DMS 2200 database area, record and set names if they conform to required criteria and Q-LINK has been generated to recognize them. Consult the person responsible for the Q-LINK installation at your site, or the Q-LINK Installation Guide, for more information.

Chapter 2: The Q-LINK Operating Environment

Before using Q-LINK, it is necessary to understand how it functions with BIS and other software. The following is a general description of how the Q-LINK environment operates.

Q-LINK operates outside BIS, but under the control of BIS. Q-LINK is designed to functionally separate database access from user interface. In this environment, there are two types of programs: "requesters" and "servers". The requesters interface with the user. They collect and screen user input, then pass it to a server. The server performs the actual database access and data formatting, and returns the data to the requester for presentation to the user. In this manner, several different types of requester programs may use the same server type for database access. In addition, changes in database structure access (servers) do not affect the user interface (requesters), and changes in the user interface, such as screen formats, do not affect database access.

In the Q-LINK environment, the Q-LINK processor functions as a "server", while BIS is the "requester". Throughout the rest of this manual, Q-LINK will be referred to as the server.

The access of non-BIS databases via Q-LINK is driven by user-written BIS runs. The BIS run (the requester) handles solicitation of user input and input editing (if desired), creation of the Q-LINK request, and sending the request to the Q-LINK server. The request consists of the program required by Q-LINK to process the request and any necessary input data. The request program is written in a high-level command language designed specifically to perform database access in a form native to the non-BIS environment. The source code of the program may be sent as part of the request, or the program may be pre-compiled by Q-LINK and saved in object form. In the latter case, the request program need only contain a single directive to invoke the compiled program, followed by input data. The Q-LINK server performs the actual database access and reply formatting as instructed by the furnished program. Q-LINK then returns the results to the BIS run, and the BIS run has the data requested in the form of a Result RID.

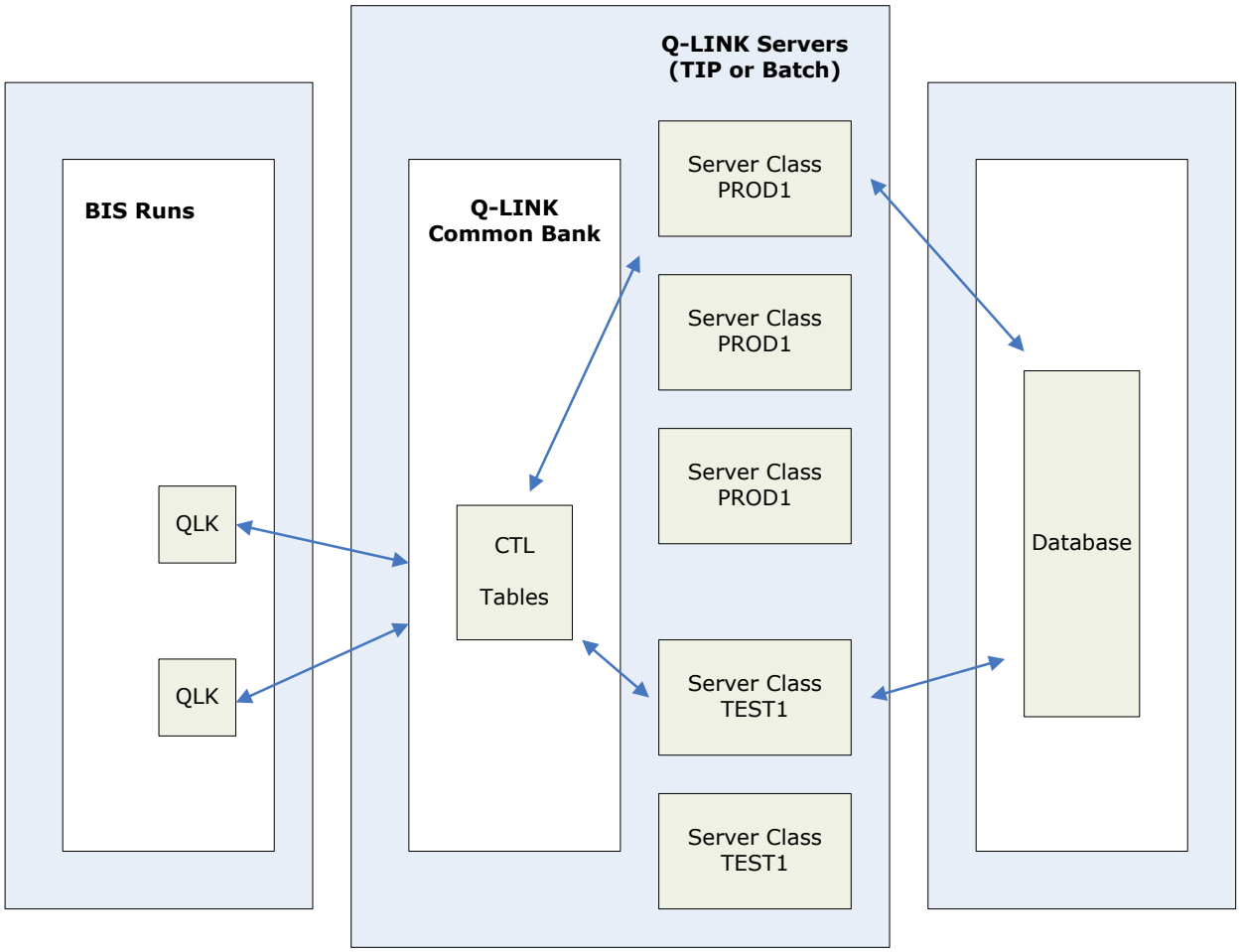
The high-level command language of Q-LINK allows access to any number of records on any path in a DMS 2200 database within the confines of an invoked subschema. In addition, Q-LINK can access any type of PCIOS file, FCSS file, and even sector-formatted mass-storage and non-PCIOS tape files. All supported database and file access can take place within the same Q-LINK request. The language also contains the commands necessary to sort and re-structure data into a form acceptable for processing in BIS. The reply delivered to BIS will be in BIS's native format, including type headers (if applicable) and tab characters.

The Q-LINK server operates as a separate run unit outside of the BIS environment. It will stand idle until a request is received from BIS. The server will be placed into an active state in order to process the request. Once the request is completed, the server will return to the idle state. There may be many server run units available at a given time. The number can be dynamically controlled. This allows the installation to make more servers available during peak load periods and fewer during slack periods. System configuration is covered in detail in the Q-LINK Installation Guide.

Q-LINK links BIS to the non-BIS environments. Data is handled on both sides in native mode; BIS data is always processed in BIS formats, and non-BIS data is always processed using native interfaces (i.e., DMS 2200, PCIOS and TIP/FCSS). This ensures that all security and integrity features available in each environment are always operable.

2.1 Servers and Server Classes

As mentioned earlier, Q-LINK run units are referred to as servers, and there may be many servers available at any given time. Servers are known to BIS by their server class name. A server class is a group of Q-LINK run units (servers) identified by the same server name. There may be many server classes running at the same time, which in turn, may have many servers in each. Server classes are used to separate Q-LINK applications. For example, a typical installation may have two server classes set up; one for testing and development, and the other for production. The class set up for test and development might have only one or two servers active, while the production class may have several more. Servers may be started and shutdown within a class as the workload increases or decreases. The startup and shutdown of servers may be accomplished either manually or automatically depending on your system's configuration (refer to the Q-LINK Installation and Operating Reference).



When a report is sent from BIS, it is routed to a particular server class. An individual server within the class will be selected by availability. Each Q-LINK server within a class will be identical. However, servers in different classes may be configured differently.

2.2 The QLK Run Function

The mechanism used to communicate with a Q-LINK server class from BIS is a special run function, called QLK, which is provided with the Q-LINK software. The BIS run, using the QLK function, need only specify which server class is to process the Q-LINK request. All routing of communications between BIS and individual Q-LINK servers is managed automatically. If no servers exist in the specified class, the QLK function returns with an appropriate error status immediately. If servers exist in the class, but all are busy, the QLK function will be queued until a server within the class is available. If many requests become queued, Operations will be automatically notified to take action, such as starting additional servers for that server class.

The QLK function is an external BIS run function; however, it operates similarly to any other BIS run function. The QLK function has the following format:

@QLK,*m1*,*t1*,*r1*,*m2*,*t2*,*server-class*[,*error-label*][,*maximum-lines*] \
[,*estimated-lines*][,*password*][,*headers-switch*][,{*server-run-id* | .*system-name*}]

Where:

m1,*t1*,*r1*	Input mode, type, RID of the request stream (the input to the Q-LINK server).
m2,*t2*	Mode and type of the created result (the output of the Q-LINK server).
server-class	Server Class designated to process the Q-LINK request stream.
error-label	Run label where control is returned if an error condition is encountered.
maximum-lines	Maximum number of reply lines to accept back into BIS.
estimated-lines	Estimated number of lines to be returned to BIS.
password	Password required to bypass the system specified maximum result lines.
headers-switch	This optional parameter may contain either a 'Y' or an 'N'. Headers associated with the reply's form type will automatically be inserted into the result by the QLK function by default. This action may be suppressed by using the 'Y' option here, thus allowing the request to produce its own header lines.
server-run-id	This optional parameter is used to direct the QLK function to access one particular server within a server class. The server's actual EXEC run-id or TIP unique run-id must be used.
system-name	If a *system-name* is specified it must be prefaced by a period (.). *System-names* are limited to five characters. Q-LINK will connect to the specified system using the DDP-PPC interface and the QLDDP processor, transfer the Q-LINK request for execution, and receive the result. Valid *system-names* are defined in the TELCON parameter file and have the label REMOTE.

Chapter 3: Q-LINK - Basic Structure

This chapter will introduce the structure of the Q-LINK Processor. A basic knowledge of the major components and operating environment of Q-LINK will aid in learning how to use all of its powerful features, effectively.

3.1 Major Program Components

The Q-LINK Processor is comprised of two major internal components: the COMMAND EDITOR and the COMMAND EXECUTOR. The COMMAND EDITOR edits each command for proper syntax, and then translates it into an internally encoded object command. If an error is detected during this process, an appropriate diagnostic message is displayed and the command is rejected. If the command passes all edits, it is stored in the object program area for later execution. The COMMAND EDITOR also controls the allocation of data variables and literals.

The COMMAND EXECUTOR interprets the encoded object command passed from the COMMAND EDITOR and actually performs the specific operation. The COMMAND EXECUTOR has been designed to do as little interpretation as possible, in order to achieve maximum throughput during program execution.

3.2 Commands vs. Directives

There are two types of control statements in a Q-LINK request program: commands and directives. The difference between a command and a directive is that a directive is always processed immediately and cannot be executed under control of a Q-LINK program. Commands are edited and placed into the internal program area before being executed.

Directives are used to set up and control the environment Q-LINK. This includes compiling and running a program, saving an object program and defining user variables, files, etc. Directives must always be entered beginning in column 1.

Commands execute the user coded Q-LINK program logic, performing data manipulation, program flow control and logic operations, and input/output. Commands must always be entered beginning after column 1.

3.3 The Q-LINK Server's Environment

Let us look at the environment in which each individual Q-LINK server is operating. There are several files used by each Q-LINK server run. Some of these files are shared with other servers, while others are unique to an individual server. The following is a brief description of the various files used by the Q-LINK server:

- The Primary Data Item Index File is automatically created as a temporary file for each individual server. It is used to hold descriptive information on every database

area, record, set, database dataname and data item in an invoked subschema. This information is used in the editing of Q-LINK commands.

- The Secondary Data Item Index File is a catalogued file that contains data item definitions built by the QINDEX processor. These definitions may be used for any data item not defined in a subschema, such as PCIOS or TIP/FCSS files. Use of these files is controlled by the INDEX directive.
- The Diagnostic Log File is a temporary file assigned each time an individual server is started. It is used to log diagnostic information during server operation. It can be retrieved into BIS and inspected via the RETLOG directive. When a Q-LINK server is in diagnostic mode (explained later) the log is automatically retrieved and appended to the reply at the completion of the request.
- The Object Request File is a program file used to store Q-LINK programs in object format. There is one default Object file shared by all active servers. There may be any number of user assigned Object files. The default Object request file is named Q$LNK*Q$LNKOBJ.
- The Source Request Library File is a program file used to store commonly used portions of source code used in Q-LINK request programs. This source code is included in a request program via the ADD directive. There is one default Source Request Library shared by all active servers. There may be any number of user assigned Source Request Library files. The default file is named Q$LNK*Q$LNKLIB.

Each of these files, and their usages, will be discussed in more depth later. In addition to the above files, there are the logical links to BIS: the request stream and the reply. These appear to the Q-LINK server logically as input (the request stream) and output (the reply) files; however, in actuality, these exist as areas of memory shared between the Q-LINK server and BIS. When we say that data is placed into the reply, we are actually moving data to the common area where BIS may then use it in building the result report.

3.4 Internal Storage Areas

When using the Q-LINK Processor, it is important to understand which internal areas are used for various operations. The internal storage of Q-LINK is organized into four major areas:

- The RECORD DELIVERY AREA (RDA) is used for all input and output for DMS 2200, PCIOS files and FCSS files. This area is not used in the transfer of data to and from BIS.
- The VARIABLE DATA STORAGE AREA is used for storage of user and reserved variables and literals.
- The OBJECT PROGRAM AREA is where commands are stored in object form before being executed.
- The Internal Table and Buffer Areas are used for internal control and storage.

3.4.1 Record Delivery Area (RDA)

The RDA is used for input and output operations for DMS 2200 database, PCIOS and TIP/FCSS files. The size of this area will vary depending upon Q-LINK generation parameters. It is recommended that the RDA be at least twice the size of any record that may be read into it. By default, all read, write, fetch, modify, etc., operations assume that the data record begins in the first position of the RDA. If multiple records are to be accessed simultaneously, data from the RDA must be moved to variable data storage prior to subsequent I/O operations. Another method of accessing multiple records simultaneously is to define alternate record areas using the "DEFINE RA" directive. This

directive allows the user to allocate alternate areas of the RDA to be used for the I/O of specific records or files. The DEFINE RA directive will be discussed in more detail later.

Since the RDA will usually be generated larger than any record that might be read into it, the upper portions of it may be used for additional user storage.

3.4.2 Variable Data Storage Area

The variable data storage area is where all user and reserved variables and literals are allocated by Q-LINK. Reserved variables are automatically defined and allocated by Q-LINK upon initialization. The names and contents of reserved variables will be found in the Q-LINK Programmer Reference. Literals are automatically allocated in this area as they are encountered by the command editor of Q-LINK. The size of this area is specified on a Q-LINK generation parameter.

3.4.3 Object Program Area

The Object Program Area is where the internally encoded object program commands are stored for execution by the command executor. The object program is not directly accessible by the user. The size of this area is specified on a Q-LINK generation parameter, and should be large enough to accommodate from 200 to 600 commands.

3.4.4 Q-LINK Internal Table and Buffer Area

This area contains several tables and buffers used internally by Q-LINK. Among these are the following:

- PCIOS and TIP/FCSS file control tables used to maintain the status of each file defined in the Q-LINK session;
- The data item index file's input buffers;
- The output buffer, which is 132 characters in length, is used to build the reply lines to be sent back to BIS;
- The sort control area for building sort key parameters and maintaining status of sort usage.

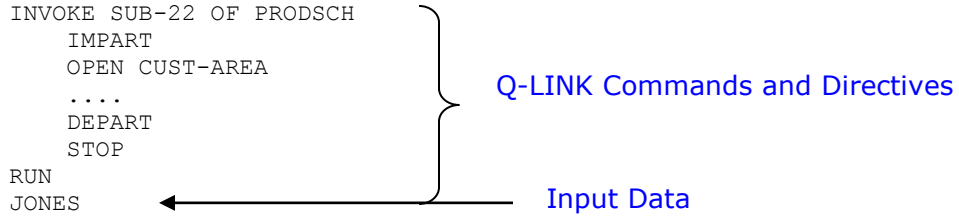
Chapter 4: Using Q-LINK

This chapter, directed to the first-time Q-LINK user, provides information on how to communicate with the Q-LINK Processor from within BIS. The chapter further explains the use of supplied run functions, commonly used commands and directives.

4.1 The Q-LINK Request Stream

Q-LINK directives, commands and input data are assembled into a BIS RID called the request stream. Lines in a request stream may be of any length allowed by BIS. The request stream may either exist as a permanent report or as a result report. The QLK function will insert the user's id, department number and station number into the request stream automatically. These will be used in automatic security checking, and are also available to the Q-LINK program as values in predefined reserved variables.

The body of the Q-LINK request stream will contain the necessary Q-LINK directives and commands to carry out the desired function. A RUN directive is used to cause the Q-LINK server to begin executing the request program. The SAVE directive can be used to cause the Q-LINK server to save the OBJECT form of the program for repeated execution of the same request. Both the RUN and SAVE directive imply a COMPILE, if necessary, for taking the appropriate action. See Chapter 6 on compiling Q-LINK Request Programs for details. A request stream may look something like this:



Any data input to be furnished to the request program from the BIS user will follow the RUN directive. Data from BIS may be appended to or inserted into the request stream RID before executing the QLK function. BIS data may also be automatically inserted into the request stream by the QLK function by placing a #INSERT directive at the point or points where another report is to be inserted. The #INSERT directive is processed by the QLK function, but the Q-LINK server only sees the resulting stream of input.

The #INSERT directive has the following format:

INSERT *mode*,*type*,*RID*[,*start-line*][,*line-quantity*]

The request stream may contain any number of #INSERT directives. Examples using the #INSERT and other directives will be discussed in detail later.

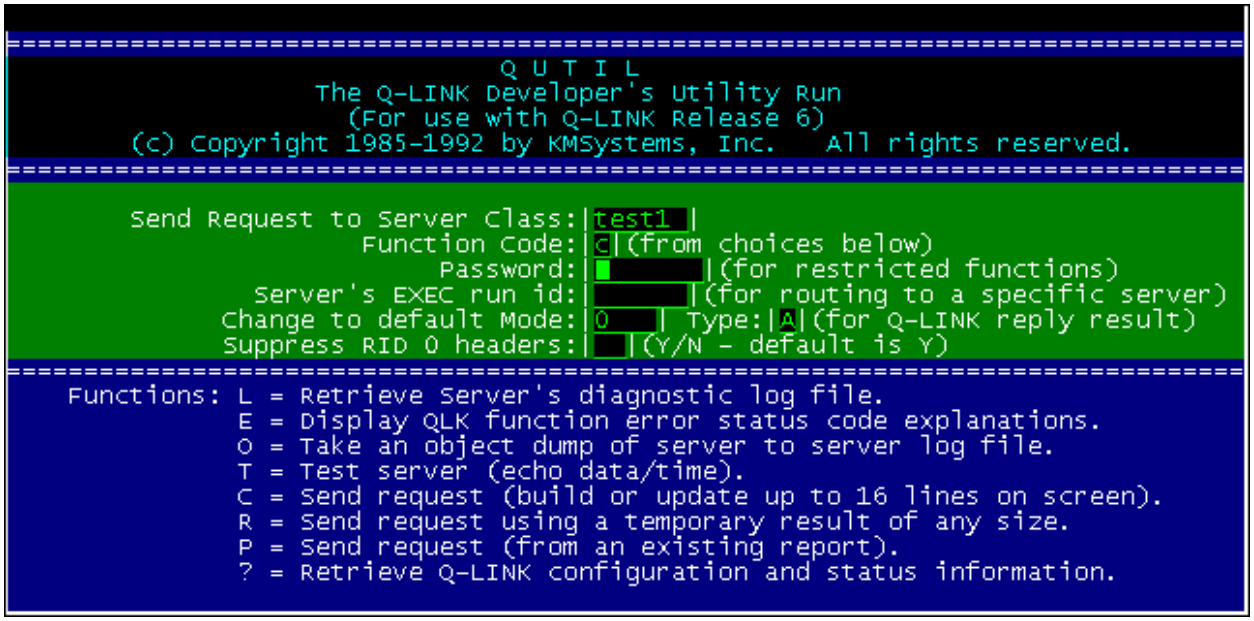
Several different groups of data lines may be collected into a request stream. In order for the Q-LINK user to determine when one group ends and another begins, the #EOF directive has been provided. When this directive is found in the request stream by the QLK function, a soft end of file is given to the Q-LINK server. The actual #EOF image is not sent to the

server. The soft end of file is detected by the Q-LINK ACCEPT command when the AT-END clause is used.

4.2 Using the QUTIL BIS Run

Since Q-LINK requests must be sent to Q-LINK servers via a BIS run function, we will need a way in which we can create a request, send it to Q-LINK, and display the reply without having to write a new BIS run each time. The QUTIL BIS run has been furnished for this purpose. QUTIL is a general-purpose run that automatically creates certain Q-LINK requests based on menu selections, or allows you to create a short request on the screen, or send a request that you created in a permanent RID earlier. To run QUTIL you will need to know the name of the Q-LINK server that you want to process your requests. Normally, your site administrator will set up one server specifically for testing and training.

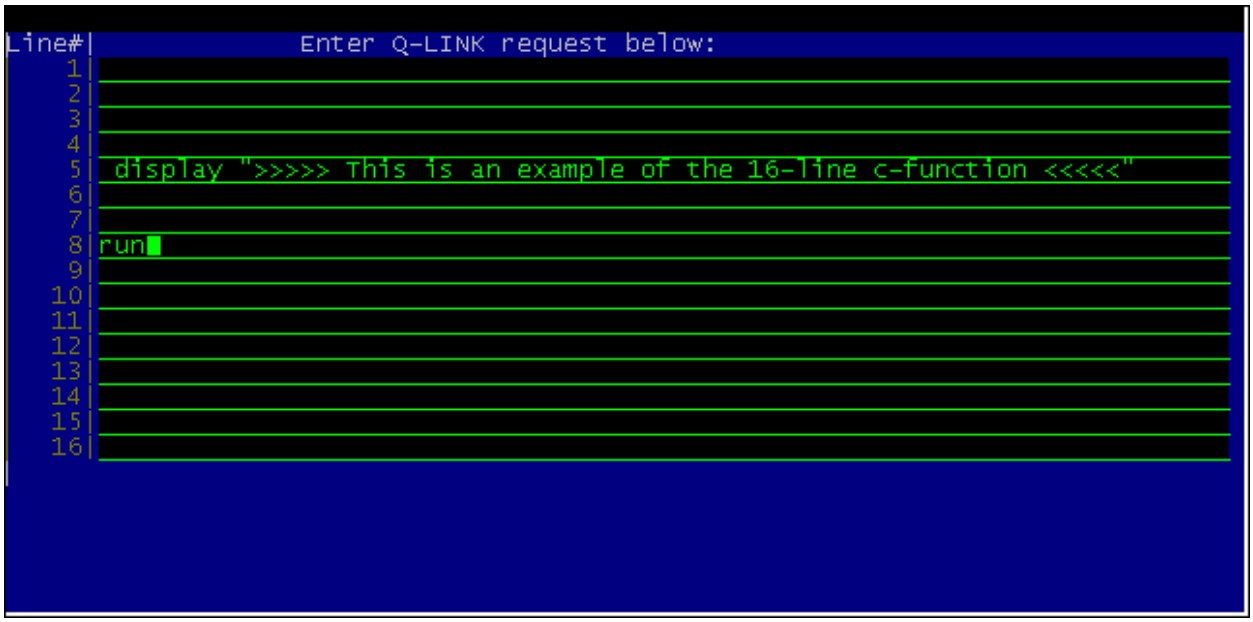
To begin, enter QUTIL as you would any other BIS run. QUTIL will then display the main menu (see below).



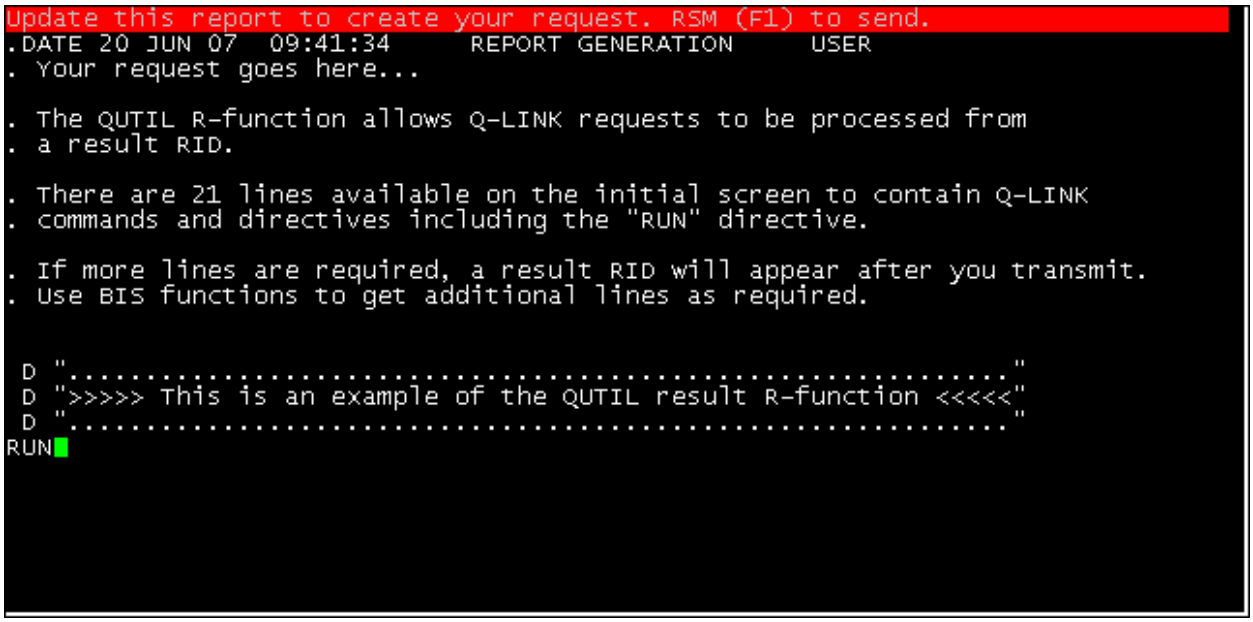
The main menu allows you to select a server class and a function. The Q-LINK requests automatically created are TEST, OBJECT DUMP, RETRIEVE LOG and SHUTDOWN. The SHUTDOWN may only be used at the direction of the system administrator with a special password. The TEST function is used to see if a server is available. It simply echoes the date and time (and gives some idea of response time). The RETRIEVE LOG function causes the Q-LINK server's diagnostic log to be retrieved into the result RID. If the server is in diagnostic mode, all source program commands and error diagnostics will be displayed on this log, making it very useful for debugging. A server set up for production may have logging suppressed for all but run time errors, while a server set up for test and development may automatically append the diagnostic log to the end of all replies.

If the C function is selected, QUTIL will display a screen containing 16 blank lines, which may be used to enter a short Q-LINK request. This is a convenient way of creating short Q-LINK requests without having to modify any permanent RIDs. When you XMIT (from the end of the last line entered), the contents of the screen will be sent to the specified Q-LINK server class.

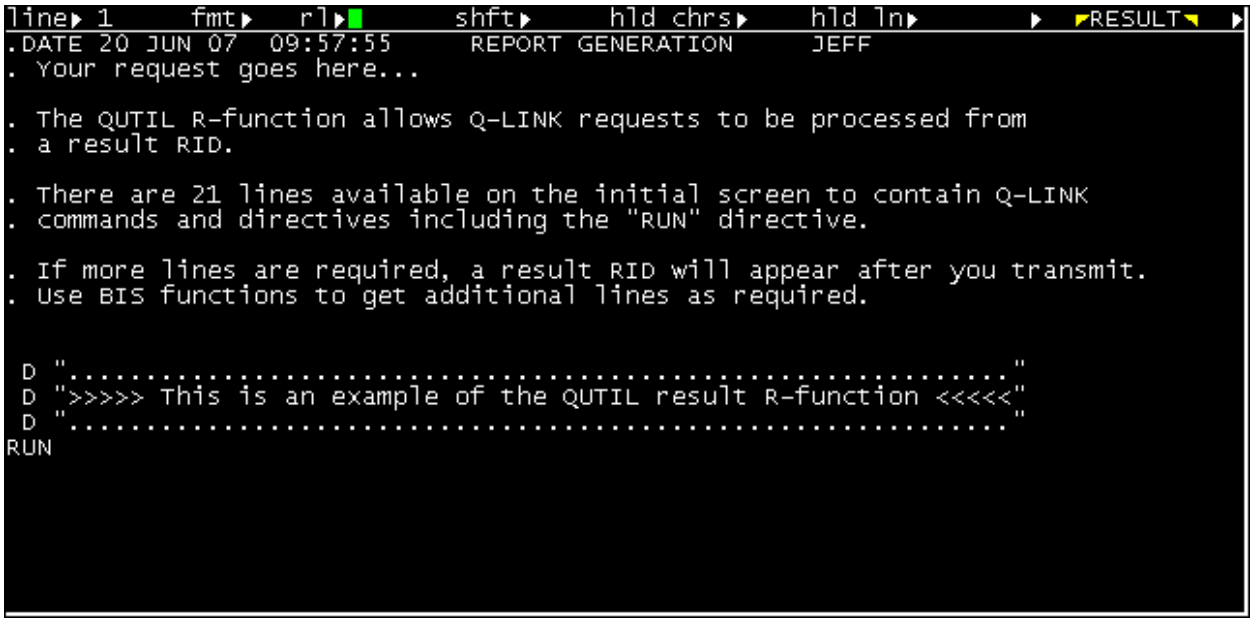
If syntax errors occur on the Q-LINK commands, an error screen will be displayed (Status 6). By depressing F1, the QUTIL run's main menu will be displayed. Then, by choosing the C function a second time, the original request will be redisplayed where the commands in error can be corrected.



If the 16 lines available through the C function are not enough for the request or if you anticipate making frequent changes to the request between executions, the R function may be used. The R function performs in a similar fashion to the C function; however, the request is built in a result RID rather than on a rigid preformatted screen.

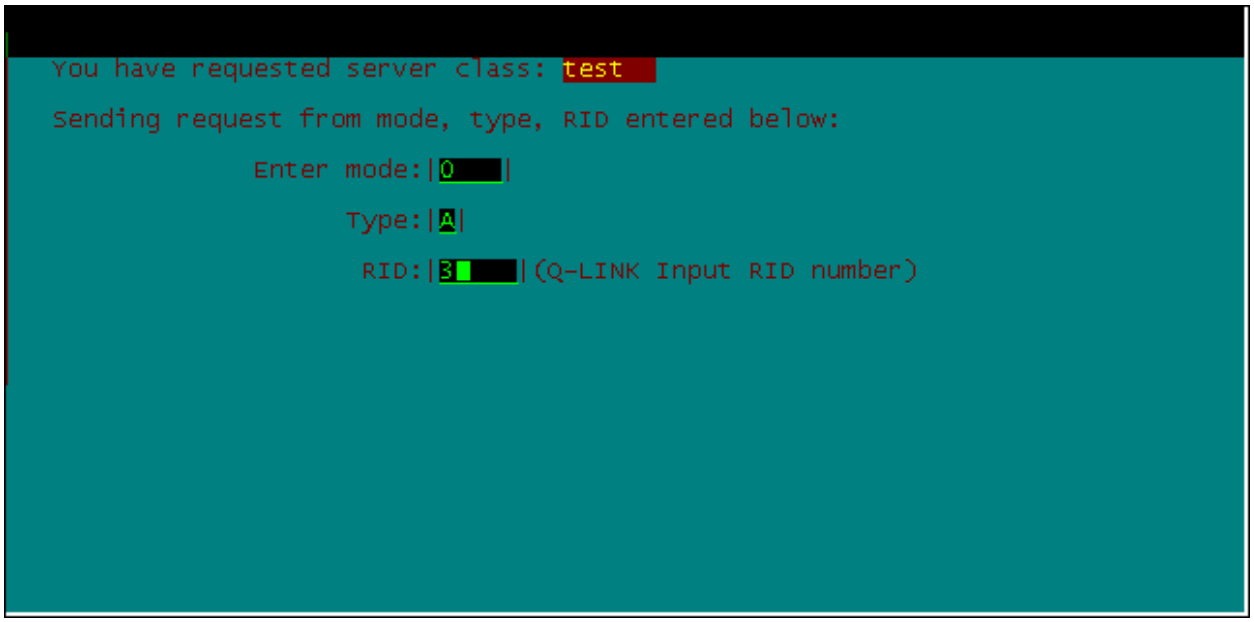


The use of a result RID allows lines to be added, changed or deleted easily without the need to re-type.



Like the C function, if syntax errors occur on the Q-LINK commands, an error screen will be displayed (Status 6). By depressing F1, the QUTIL run's main menu will be displayed. Then, by choosing the R function a second time, the original request will be re-displayed where the commands in error can be corrected.

The P function is used to send a request built in a permanent RID earlier. This will be useful when you create and test longer Q-LINK requests. When the P function is entered, QUTIL will display a screen where you may enter the mode, type and RID of the request stream you wish to send to the Q-LINK server.



The E function may be used to display descriptions of the error codes that may be returned from the QLK run function. This will be a handy quick reference.

In all cases, QUTIL only affects result RIDs. No permanent RID will be updated unless you enter a manual REPLACE or DUP function. All replies will be results, using the current work mode and form type.

To become familiar with the QUTIL run, try calling it and using the Test function. This would also be a good time to find out which server class you should use for training, and how it is configured.

4.3 A Hands-on Introduction

This section will familiarize the new user with some common Q-LINK commands and with how commands are used in simple request programs. The commands shown here will require no database or file access, and can be attempted without fear of destroying any existing data.

4.3.1 Displaying or Viewing Data

Let us first look at the commands that will be used to output data to the BIS reply: DISPLAY and EDIT.

Create the following Q-LINK Request in a permanent RID, or use the C option of QUTIL to build it on the screen:

```
1...5...10...15...20...
 DISPLAY DATE                    <- Commands MUST NOT start in col. 1.
 DISPLAY J-DAY
RUN                              <- Directives MUST start in column 1.
```

Use the QUTIL run to send the request to the Q-LINK server. Q-LINK will respond with the current date in the form YYYY/MM/DD followed by the current Julian date in the form YYYYDDD in the BIS reply. DATE and J-DAY are Q-LINK reserved variables. There are several pre-defined reserved variables set up automatically when Q-LINK is initialized. A complete list will be found in the Q-LINK Programmer Reference.

The reply RID will look something like this:

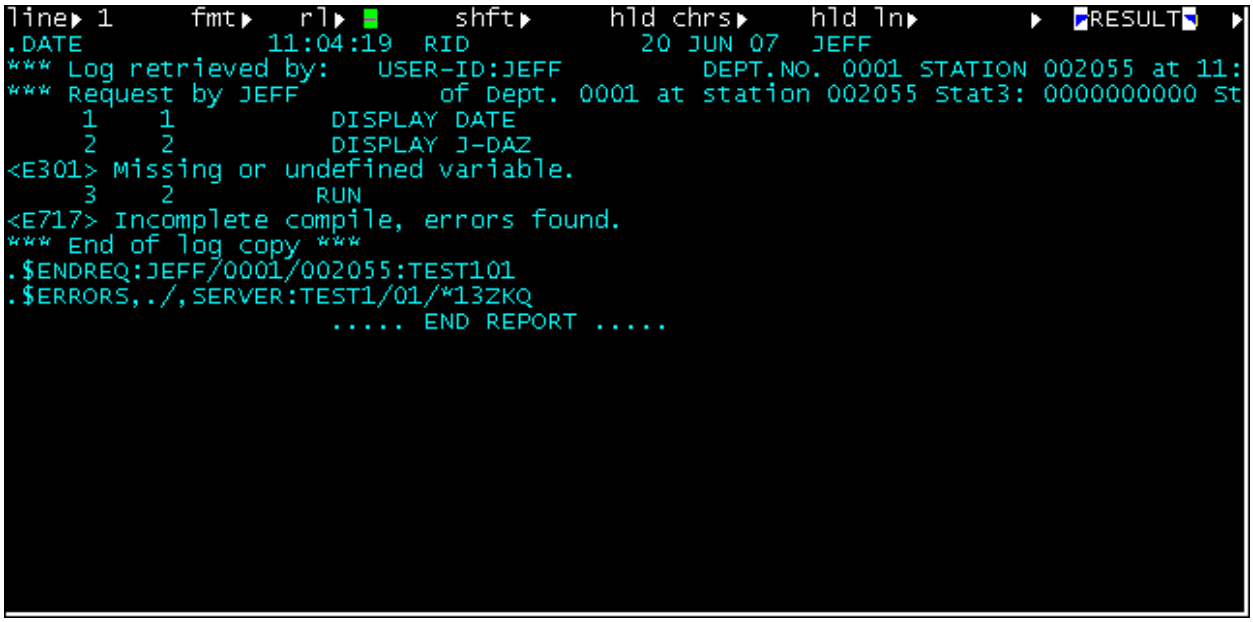
```
1994/02/15                       <— Output of DISPLAY DATE
1994046                          <— Output of DISPLAY J-DAY
```

If an error had been encountered during Q-LINK processing, BIS's reserved variable STAT1 would have been set to the appropriate error code. The QUTIL run will automatically detect this and display a message on the screen. See the Q-LINK Programmer Reference for detailed descriptions of error codes, or use the QUTIL E function for descriptions of specific codes. If the server class you are using is running in development mode, your reply will appear as follows:

```
     Stat2 =    25  Stat3 =    0
.DATE           11:02:05  RID          20 JUN 07  JEFF
2007/06/20
           2007171
*** Log retrieved by:   USER-ID:JEFF         DEPT.NO. 0001 STATION 002055 at 11:
*** Request by JEFF         of Dept. 0001 at station 002055 Stat3: 0000000000 St
     1    1          DISPLAY DATE
     2    2          DISPLAY J-DAY
     3    3         RUN
*** Compile complete ***
     4    1
     5    1
     6    1
     7    1
     8    1
     9    1
    10    1
    11    1
    12    1
    13    1
    14    1
    15    1
    16    1
*** End-of-File on request stream ***
```

This is the content of the server's diagnostic log file. The first line following your reply is from the last request processed by the Q-LINK server. The log display includes the reply followed by a listing of the program as it was edited by the Q-LINK server, as well as other diagnostic messages. The blank lines following "compile complete" are the remaining lines that were not used when the QUTIL C screen was transmitted. The last line, ".$ENDREQ", is only present when the server is in diagnostic mode. It shows the BIS user-id, department, station number, server class and specific server index used to process the request. If an error had been encountered, the QUTIL run would first have displayed an error status of 6, indicating that some sort of error was detected by the Q-LINK server. If you use the resume function (RSM or F1), you will get a reply showing a diagnostic message for each command in error and an indication that the compile failed.

Let us assume that we misspelled the word J-DAY as J-DAZ. The reply would look like this (after the resume function):



The error diagnostic identifies the error, and we were told that the compile failed. Of course, the output lines shown earlier would not be produced because Q-LINK will not attempt to execute a program found to be in error.

Note the numbers displayed in front of each command. They indicate the source line number and program counter (PC), or location within the internal program table of each command. These numbers will be important when a run time error occurs. A run time error is one that occurs during program execution. When this occurs, Q-LINK will display the PC of the command in error. All of this is covered in more detail in the chapter called, "Q-LINK Debugging".

Create and run the following request:

```
    TIME
    DISPLAY TIME +
    DISPLAY TIME-MSPM
RUN
```

The first command (TIME) in the above sequence sets the current system time of day into two reserved variables: TIME (same name as the command) and TIME-MSPM. The variable TIME is an alphanumeric string of 11 characters consisting of the time of day in the form HH:MM:SS.DDD. The variable TIME-MSPM is a numeric integer containing the time of day in milliseconds past midnight. TIME-MSPM may be used in computing elapsed time in a Q-LINK program.

The first DISPLAY entered above did not produce a separate output line because of the plus (+) sign following the name of the variable. The "+" tells Q-LINK to move the data in the variable to the reply buffer, but not to release it yet, because there is more to come. After the second DISPLAY, both the alphanumeric string and numeric integer forms of the current time of day will have been moved to the reply buffer and released. The reply RID will look like this:

```
13:52:31.070          49951070
```

The DISPLAY command places data into the reply buffer as it is stored. Another command used to output data is EDIT. The EDIT command is used to output numeric data in an edited format that may include comma and decimal placement, leading or trailing sign, zero suppression, etc.

Try this request:

```
TIME
EDIT TIME-MSPM 'ZZ,ZZZ,ZZZ.999'
RUN
```

This request should have returned the time of day in milliseconds past midnight edited by the mask furnished in the literal 'ZZ,ZZZ,ZZZ.999'. This process is similar to using the edit picture clause in COBOL. The reply should look something like this:

```
50,064,892.000
```

Note that the placement of the decimal is determined by its position in the edit mask and its implied position within the data item. In this example, we know that time is represented in milliseconds past midnight; therefore, the decimal is normally placed three positions from the right. In this case the variable (TIME-MSPM) is defined as an integer, thus its value was scaled to fit the edit mask furnished. The number of defined decimal positions in the field must be taken into consideration when editing numeric data if the implied decimal positions are different. Editing of decimal numbers will be covered in more detail later.

Look over the following request program to see some variations of the EDIT and DISPLAY commands, and the use of some other reserved variables, and then try them yourself:

```
  DATE                          .  Set the current date
  EDIT DATE-NUM '9999B99B99B' +
  EDIT J-DAY '9999/ZZ9'
  DISPLAY 'THE CURRENT DATE IS ' +
  EDIT MONTH 'ZZ' +
  DISPLAY '-' +
  EDIT DAY 'ZZ' +
  DISPLAY '-' +
  EDIT YEAR 'ZZZZ'
RUN
```

This example illustrates how output lines can be constructed using several DISPLAY and EDIT commands. Your reply should look as follows:

```
2007 02 19 2007/ 50
THE CURRENT DATE IS  2-19-2007
```

Both the DISPLAY and EDIT commands have sister commands called TRIMDISP and TRIMEDIT, respectively. TRIMDISP and TRIMEDIT function the same as their siblings with one exception: both commands will drop all non-significant leading and trailing spaces on output. As an example, replace all the DISPLAY and EDIT commands in the previous example, except those displaying literals, with TRIMDISP and TRIMEDIT commands. The resulting reply will look like this:

```
2007 02 192007/ 50
THE CURRENT DATE IS 2-19-2007
```

Notice how the output is compressed. This feature allows for very creative formatting.

4.3.2 Manipulating Data

So far, we have looked at ways to display data in the reply. Now Let us look at the SET command. The SET command is used to set the value of one item to the value of something else (another variable, a literal, the result of combining two variables, etc.). Let us try some simple forms of the SET command using the reserved variables X, Y and Z. These are three numeric integer variables automatically set up during initialization for the convenience of the Q-LINK user. The command key word “SET” is implied and may be omitted. Look over the following request, then try it yourself.

```
  SET X = 50
  Y = X * 2                     <- SET command implied.
  Z = X - Y                     <-  "     "       "
  DISPLAY X Y Z
```

```
RUN
```

This example shows how the SET command is used to manipulate numeric integer variables. Also shown is another variation of the DISPLAY command where up to four variables may be displayed at one time. This form of DISPLAY may be used to output numeric and alphanumeric string variables to the reply. The following reply should be produced:

```
                50              100             -50
```

If the DISPLAY above were replaced by a TRIMDISP, the result would look like this:

```
50100-50
```

SET may also be used to manipulate alphanumeric string data. We will use the reserved variables C-O-R and G-AREA-NAME. These two variables are used by Q-LINK when executing DMS 2200 DML commands, but are available to the user when DML commands are not being used.

Enter the following request:

```
 C-O-R = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890'
 G-AREA-NAME = C-O-R
 DISPLAY C-O-R
 DISPLAY G-AREA-NAME
 C-O-R = G-AREA-NAME
 DISPLAY C-O-R
RUN
```

The defined lengths of the variables C-O-R and G-AREA-NAME are 30 and 12 characters respectively. The literal used in the first SET is longer than C-O-R and is truncated when placed into C-O-R. Truncation also occurs when G-AREA-NAME is SET to the value of C-O-R. On the other hand, when the receiving variable is larger, the remaining character positions are space filled, as seen in the last SET command of the example. This program would produce the following reply:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ1234
ABCDEFGHIJKL
ABCDEFGHIJKL
```

4.3.3 Using the Record Delivery Area (RDA)

All data references thus far have involved the Variable Data Storage Area using predefined reserved variables. Definition of user variables will be covered later. Now, we will look briefly at basic forms of referencing the Record Delivery Area (RDA). The RDA is used for all input and output operations, but may also be used as additional user workspace.

The RDA is always referenced by starting character or byte position and length. In addition, a data type name may be used to describe what kind of data is being referenced. A detailed description of RDA referencing and data types will be found in the Q-LINK Programmer Reference. A simple example of an RDA reference would be:

```
(5,3)
```

This would refer to a three-position data item starting in position five of the RDA. The data type of the item would default to ASCII DISPLAY. When RDA references are used in Q-LINK commands, they are always preceded by the key word RDA.

Consider the following command sequence:

```
 SET RDA (5,3) = 'ABC'
 DISPLAY RDA (5,3)
```

The output line would look like this:

```
ABC
```

This form of RDA reference is called DIRECT because the user must specify the exact character positions, length and data type. Later, we will cover using data item names that

are from invoked subschemas and defined RDA fields, and how advanced RDA referencing features may be used.

The following request program will illustrate some more uses of DIRECT RDA references:

```
 RDA (1,10) = 'ABCDEFGHIJ'
 DISPLAY RDA (2,2)
 RDA (11,5) = RDA (3,5)
 DISPLAY RDA (1,15)
 X = RDA (1,1) COMP + RDA (2,1) COMP
 TRIMDISP X
RUN
```

The reply produced by the previous example:

```
BC
ABCDEFGHIJCDEFG
131
```

Notice in the example that by using a direct RDA reference, unlike variables that must be used exactly as defined, it is possible to manipulate and extract data in any position and use it as any data type.

4.3.4 Defining Data

All examples presented thus far have used either predefined reserved variables or a direct RDA reference. The DEFINE directive can be used to define user variables. There are several forms of the DEFINE directive. We will begin by defining user variables and RDA fields.

There are three types of variables: decimal numeric, floating point numeric and alphanumeric string. Decimal numeric variables may contain a positive or negative value up to 18 digits in length. Floating-point numeric variables may contain any double precision floating point value. An alphanumeric string variable may be any length.

Here are some examples of defining variables:

```
DEFINE N VAR1
DEFINE N VAR2 123
DEFINE A VAR3 'ABC'
DEFINE A VAR4 8
DEFINE N VAR5 .00
DEFINE FP VAR6 1.E-10
```

In the first line above, VAR1 will be a numeric variable as indicated by the "N". Its initial value will be set to zero. The second variable, VAR2, is also numeric; however, in this case an initial value of 123 will be assigned. Neither VAR1 nor VAR2 have any implied decimal precision. The variables VAR3 and VAR4 are both alphanumeric string variables as indicated by the "A". VAR3 will be given the initial value of "ABC", and its length will be three characters—the length is determined by the assigned literal value because no length was explicitly given. The initial value of VAR4 will be undetermined, but its length has been specified to be eight characters. VAR5 is an example of defining a decimal numeric variable with two decimal positions implied. VAR6 is an example of defining a floating-point number.

Another form of the DEFINE directive is the DEFINE RDA. With this form of DEFINE, positions within the RDA can be defined and/or redefined for various uses.

Here are some examples of RDA field definition:

```
DEFINE RDA FLD1 (1,5)
DEFINE RDA FLD2 (6,3) SN9
DEFINE RDA FLD3 (9,4) COMP
DEFINE RDA FLD3R (9,4)
DEFINE RDA FLD4 (13,6) SN9 .0000
```

The first four DEFINE RDA directives above have set up three fields within the RDA. The first field is alphanumeric display data in positions 1 thru 5, the second is signed numeric

display data in positions 6 thru 8, and the third is computational data in positions 9 thru 12. The third field has been redefined as alphanumeric display data by the fourth DEFINE RDA directive. The last DEFINE RDA directive shows how implied decimal positions are represented in an RDA definition. The ".0000" tells Q-LINK the field has four decimal positions implied; however, no actual value is assigned to the field.

After entering both the variable definitions and RDA definitions just presented, add this sequence of commands to the request RID and run it:

```
.  .  .  <- The DEFINEs go here.
 RDA FLD1 = VAR3
 RDA FLD2 = VAR2
 RDA FLD3R = VAR3
 DISPLAY 'FLD1 = ' +
 DISPLAY RDA FLD1 +
 DISPLAY ' FLD2 = ' +
 TRIMEDIT RDA FLD2 'ZZZZ' +
 DISPLAY ' FLD3 = ' +
 EDIT RDA FLD3 '9999999999+'
 VAR5 = -77.8888
 RDA FLD4 = VAR5
 DISPLAY 'VAR5 = ' +
 TRIMEDIT VAR5 '-ZZZ.99'
 DISPLAY 'FLD4 = ' +
 TRIMEDIT RDA FLD4 'ZZZ.9999'
RUN
```

The reply received after running the above request should look like this:

```
FLD1 = ABC   FLD2 = 123 FLD3 = 8741488160+
VAR5 = -77.89
FLD4 = -77.8900
```

Note: 8741488160 is the decimal equivalent of octal 101102103 or ABC. Also, notice that when VAR5 was set, the decimal value was scaled and rounded to match its definition.

The DEFINE RDA directive also provides a method of defining items as they relate to other items. This relative reference is accomplished by substituting the starting byte position with an asterisk, or another RDA item name, or a combination of both. Let us look at an example.

```
DEFINE RDA FIELD-A (1001,5)
DEFINE RDA FIELD-B (*,20)
DEFINE RDA FIELD-C (*,4) COMP
DEFINE RDA FIELD-X (FIELD-B,5)
DEFINE RDA FIELD-Y (*FIELD-A,5)
```

The asterisk is used to inform Q-LINK that the item being defined is to start immediately following the last RDA item defined. If no RDA definitions have been entered yet, position one will be used. In the above example, we can see that FIELD-B follows FIELD-A and FIELD-C follows FIELD-B. FIELD-B starts at character position 1006 and FIELD-C starts at 1026. When a previously defined RDA item name is used for the starting position, the item being defined will start at the same position. FIELD-X above starts at the same position as FIELD-B (it is a redefinition of the first five positions of FIELD-B). When the previously defined item preceded by an asterisk is used as a starting position, the item being defined starts immediately following the named item. Looking at the example once more, we can see that FIELD-Y starts immediately following FIELD-A, and in fact occupies the same positions as FIELD-X. At this point, you should have a good idea of the basic operation of the Q-LINK Processor. We will be using this type of RDA field definition more in examples later. Let us try a slightly more complicated request program. This time the request will include some user-defined variables and some logical commands.

Create the following request:

```
DEFINE N CNT                    . LOOP COUNT
```

```
DEFINE N HOWMUCH                    . INPUT FROM USER
BEGIN                                 <- This is a program label
 DISPLAY '*** STARTING ***'
 DO                                   <- Start an in-line DO block
   ACCEPT HOWMUCH AT END BREAK      .  GET USER INPUT
   CNT = 0
   IF HOWMUCH < 50
     DO COUNT WHILE CNT < HOWMUCH    <- DO procedure
   ELSE
     DISPLAY 'Sorry, too many.  50 or less please'
   ENDIF
 ENDDO
FINAL
 DISPLAY '*** END OF RUN ***'
 STOP
  .
  .  *** COUNT TO HOWMUCH            <- This is a comment
  .
COUNT PROCEDURE                      <- A PROCEDURE entry label
 CNT = CNT + 1
 DISPLAY 'CNT = ' +
 TRIMEDIT CNT 'ZZ9'
 ENDPROC
RUN
2                                    <- Data to be ACCEPTed
3
```

The above example will produce the following output:

*** STARTING ***

```
COUNT = 1
COUNT = 2
COUNT = 1
COUNT = 2
COUNT = 3
```

*** END OF RUN ***

This request program illustrates the use of comment lines. A comment is any text following a period and space sequence (unless it occurs within a literal).

Also shown in the last example is the STOP command. The STOP command is used to terminate a program just like the COBOL STOP RUN. A STOP command is automatically inserted following the last command of any program; however, if the logical end of the program is not the last command, we must tell Q-LINK when to STOP (that is why we have not needed STOP command before now). The STOP command may also be used to notify the requesting BIS run of various external system conditions by using a numeric literal or variable in the STOP command (i.e., STOP 101 or STOP MY-STATUS). The STOP value, or exit code, will be passed back to BIS and be available in the BIS reserved variable STAT3. The STOP command is a very easy and efficient way to pass an error notification back to BIS in your applications.

The last program also demonstrates several commands not discussed earlier, such as the GO, IF/ELSE/ENDIF, DO/ENDDO, ACCEPT, and the use of a Q-LINK defined procedure. Let us take some time now to discuss these commands (refer to the Q-LINK Programmer Reference for detailed information).

The ACCEPT Command is used to receive input from the request stream. The ACCEPT command is similar to the COBOL ACCEPT verb. The ACCEPT command is very important because it is the instrument by which you will supply input data from BIS for your Q-LINK request. We will see many more examples of this throughout the rest of this manual.

The DO command was used in two ways in the example. The first occurrence was an in-line DO block. An in-line DO block starts with a DO command with an optional WHILE/UNTIL

clause. The in-line DO is used to execute a block of commands repeatedly through a matching ENDDO while (or until) a specified condition is true or until a BREAK command is executed. If no WHILE/UNTIL is specified, the DO block must be exited by executing a BREAK. In its second form, the DO is used to "DO" a defined PROCEDURE, which is comparable to performing a SECTION in COBOL. A Q-LINK PROCEDURE must begin with a PROCEDURE entry label and end with an ENDPROC.

If the DO command includes an UNTIL clause, the condition in the UNTIL expression will be tested each time the ENDPROC or ENDDO is reached. If the DO includes a WHILE clause, the condition is tested upon entry (at the PROCEDURE label or the beginning DO command). The conditional expression may be any expression acceptable to the IF command, including the extended AND/OR logic. The last request example illustrates a typical DO WHILE using a Q-LINK Procedure. You may nest in-line DO blocks up to 50 levels; however, you may NOT nest a procedure within another procedure. An in-line DO block and a procedure may only be entered by executing the DO command. You may DO another procedure from within a procedure (or in-line DO) as often as you wish. Exiting a procedure with a GO prior to the ENDPROC is acceptable, but will corrupt the DO return stack and should not be used. To exit an in-line DO block or a procedure regardless of condition, use the BREAK command. In the case of a DO of a defined procedure, the BREAK causes an immediate return to the command following the DO from which the procedure was entered. When used in an in-line DO block, control will pass to the command following the matching ENDDO. There will be several examples of both forms of the DO throughout the remainder of this manual.

The IF command is very important and will be used in almost every Q-LINK request program. In the previous example, IF was used with numeric arguments. The IF command may also be used with alphanumeric string variables, RDA references, and literals. IF can also test for special values including HIGH-VALUES, LOW-VALUES, SPACES, ALPHABETIC and NUMERIC. Q-LINK uses the following special names for these special values:

$HIVALS, $LOVALS, $SPACES, $ALPHA and $NUM

The IF command may be extended by the AND, OR and ELSE, and must be terminated by an ENDIF.

The following portion of a Q-LINK request is an example of a more complex IF structure:

```
DEFINE A INPUT 4                    .  Input 4 characters
. . .
 ACCEPT INPUT AT END QUIT           .  Get data from request
 IF INPUT = $ALPHA
   DISPLAY 'Input contains all alpha characters'
   IF INPUT 'M'
     DISPLAY 'and the first digit is less than "M"'
   ENDIF
 ELSE
   IF INPUT = $NUM
     DISPLAY 'Input is all numeric.'
   ENDIF
 ENDIF
.
QUIT                                .  End of Program
. . .
```

In addition to the special values test shown above, Q-LINK also supports wildcard characters in comparisons using alphanumeric string data. For this purpose, the WILDCARD command is provided. The WILDCARD command is used to establish a wildcard character. Whenever a wildcard character is encountered during a comparison, an equal condition will result on the character position occupied by the wildcard character.

In the following example, a wildcard character is established, then used, in an IF command:

```
. . .
```

```
WILDCARD IS '?'
IF INPUT = 'A?CD'                  . Make '?' wildcard
. . .
WILDCARD IS NONE                   . No wildcard
```

If INPUT contains a string, starting with 'A' followed by any single character followed by 'CD', the IF will be true.

Chapter 5: Database and File Handling

By now, you should be comfortable with the basics of the Q-LINK Processor. In this chapter, we will begin to deal with database access, starting with DMS 2200 databases, then PCIOS files, the SORT subroutine interface, BIS DTM interface, RDMS 2200 table access, and finally, DB4 database access.

5.1 Accessing the DMS 2200 Database

The Q-LINK Processor is not very useful without data, so now it is time to examine database access. This part of the discussion will reference a 2200 database structure that might represent a typical sales/marketing enterprise. Several general types of database access will be illustrated. If possible, try the various operations on similar structures at your site.

5.1.1 Invoking a Subschema

The first step that is required to access any DMS 2200 database is to INVOKE a subschema. Q-LINK can use any ASCII COBOL subschema (do not use QLP or DMU subschemas). The simplified format of the INVOKE is:

INVOKE *subschema* [{OF | IN} *schema*] [{FILE *filename* | TIP [FILE] *file-code*}]

For the complete INVOKE syntax, see the Q-LINK Programmer Reference.

At some installations, only the subschema and schema names will be needed. The schema file used will be a default value set in the Q-LINK generation.

An example of an INVOKE directive is:

```
INVOKE SUB-SALES OF MARKETING
```

SUB-SALES is an ASCII COBOL subschema of the schema named MARKETING.

The INVOKE is a very important part of Q-LINK. It will access both the object subschema and schema in the schema file, and initialize D$WORK and S$WORK in Q-LINK. These are the parts of a program used to communicate with the DMS 2200 Data Management Routine (DMR). It also sets up a file containing information about every area, record, set, database dataname, and data element name as defined in the schema and included in the subschema. This file will be referred to as the Primary Data Item Index. The Primary Data Item Index is used by the Q-LINK Processor when editing Data Manipulation Language (DML) commands, and commands that use RDA reference by data item name.

Notice that INVOKE is a directive and will be processed immediately. It must be processed before any commands referencing database area, record, set, database datanames or data items are entered.

5.1.2 Q-LINK DML Commands

The Q-LINK DML commands used to access the DMS 2200 database parallel those used in COBOL DML programming. The formats are similar, but somewhat abbreviated. For example, in COBOL the FETCH format four would be coded as follows:

```
FETCH FIRST PART-MASTER WITHIN PARTS AREA ON ERROR . . .
```

The same command in Q-LINK would be coded as:

```
FETCH4 FIRST PART-MASTER PARTS AREA
```

Alternatively, the fully abbreviated form would be coded as:

```
F4 F PARTS-MASTER PARTS A
```

Q-LINK DML commands do not include AT END or ON ERROR clauses. Instead, the IF command is used to interrogate the DMS ERROR-NUM. In addition, certain reserved variables will automatically be altered as DML commands are executed. The reserved variable C-O-R is set automatically by all DML commands that can change record currency, and will hold the name of the last record processed by a DML command. There are also reserved variables set up to hold the IMPART-DEPART status, ERROR-NUM, current page number, record number, area key, etc. The complete list of reserved variables will be found in the Q-LINK Programmer Reference.

The next request example illustrates the use of the Data Item Index File in referencing the RDA by data item name. This is much easier than the direct RDA reference covered earlier. This type of data referencing is available whenever an INVOKE has been successfully issued. There may also be a Secondary Data Item Index file in use. This file is created by the separate QINDEX Processor for data definitions not included in a schema. See the QINDEX Reference Manual for information on the QINDEX Processor.

Let us look at some Q-LINK programs that use various DML commands. More DML examples appear in later chapters of this manual. Try fetching a record whose location mode is CALC, then display a data item from the record. The Q-LINK request sequence of commands required to fetch a CALC record would look like this:

```
INVOKE SUB-SALES IN MARKETING
 IMPART
 OPEN SALESMAN
 DBDN SALES-ANAME = 'SALESMAN'
 RDA SLSM-KEY = '10023'              <- Set key
 FETCH5 SALESMAN-REC                 <- Fetch the record
 IF ERROR-NUM NOT = 0                <- DMS error?
   DISPLAY DBERROR
 ELSE
   DISPLAY RDA SLSM-LAST-NAME        <- Display a field
 ENDIF
RUN
```

Notice that when using Q-LINK, the database dataname (DBDN) used by the CALC routine must be set to the correct value prior to issuing the FETCH. In many cases, the database dataname would be given a value in the subschema. This value would be set up in the S$PROC COBOL copy element by the SDDL Processor, and would not have to be initialized by the COBOL programmer. However, the Q-LINK Processor has access only to the object schema and subschema, and they do not have this information. Setting the key value in the RDA is similar to using the MOVE statement in COBOL to initialize the record's key in the DMCA in the program's working or common storage.

If the FETCH had not been successful, the reserved variable ERROR-NUM would have been set to the appropriate DML error number, in which case the command DISPLAY DBERROR would have been executed. DISPLAY DBERROR is a special form of the DISPLAY that outputs pre-formatted database error information to both the reply and the diagnostic log file.

The Q-LINK Processor provides a full complement of DML commands and options. Please refer to the Q-LINK Programmer Reference for more detailed information.

No limit exists to the number of database records that can be accessed by a Q-LINK server. Let us look at a request program that may retrieve information from several database records.

For this example, create a request RID containing a program similar to the one shown here (substitute names used in your schema):

```
INVOKE SUB-SALES IN MARKETING
DEF N SALESMAN-KEY
 IMPART
 OPEN ALL
 SET DBDN SLS-ANAME = 'SALESMAN'
 ACCEPT SALESMAN-KEY
 SET RDA SLS-SALESMAN-KEY = SALESMAN-KEY
 FETCH5 SALESMAN-REC
 IF ERROR-NUM = 0
   DISPLAY 'Customers of salesman #' +
   EDIT RDA SLS-SALESMAN-KEY '99999' +
   DISPLAY''+
   DISPLAY RDA SLS-SALESMAN-NAME
   FETCH4 FIRST CUSTOMER-REC SLSM-CUST SET
   DO WHILE ERROR-NUM = 0
     DISPLAY 'Customer number:' +
     EDIT RDA CUST-NUM '999999999' +
     DISPLAY ' Name:' +
     DISPLAY RDA CUST-NAME
     FETCH4 NEXT CUSTOMER-REC SLSM-CUST SET
   ENDDO
   DISPLAY '**** END OF LIST ****'
 ENDIF
 DISPLAY 'SALESMAN NOT FOUND'
```

The following reply would be expected using the above request:

```
Customers of salesman #12345 JOHN SMITH
Customer number 123450000 Name:JONES MFG. COMPANY
Customer number 239403029 Name:PITTSBURG COLOR CO.
*** END OF LIST ****
```

5.1.3 DMS 2200 Error Handling in Q-LINK

Q-LINK handles DMS 2200 errors a bit differently than COBOL/DML. There is not an ON ERROR or AT END clause used on Q-LINK DML commands. Instead, the IF statement is used to check the value of the predefined variable, ERROR-NUM, to determine if either of these conditions has occurred. Additionally, only non-fatal DMS 2200 errors will return control to a Q-LINK program. A table of non-fatal errors is maintained in Q-LINK. If an ERROR-NUM value is received and is not present in the non-fatal table, the Q-LINK request will error terminate with an automatic DML error display in both the reply and the diagnostic log file. A fatal error will also cause an automatic DEPART WITH ROLLBACK, backing out changes made since the beginning of the request or last FREE command. The non-fatal table initially contains the ERROR-NUM values 0006, 0007 and 0013. See your COBOL DML manual for the meanings of these error codes. Any other error will be considered fatal. The SET NON-FATAL command has been provided to allow the Q-LINK programmer to add or delete ERROR-NUM values in the non-fatal table as necessary. This minimizes the amount of DMS 2200 error handling required in a Q-LINK program.

An important point to remember is that while the DML DEPART command is provided in Q-LINK, it is not required. The Q-LINK processor will automatically perform a DEPART upon completion of request processing. If the request was error terminated, a DEPART with ROLLBACK will be performed.

The following is a short example, which uses the SET NON-FATAL command. In this example, the program should never get a no-find on the FETCH of a CALC record, so ERROR-NUM value 0013 will be removed from the non-fatal table. If an ERROR-NUM 0013 is encountered, Q-LINK will automatically error terminate the request.

```
DBDN CTL-ANAME = 'CONTROL'
RDA CONTROL-KEY = 'PARTS'
NON-FATAL 13 OFF                    <- Make a no-find fatal
FETCH5 CONTROL-REC                  <- Will abort if no-find
DO PROCESS-PARTS                    <- Proceed if record found
```

It is very common to set the ERROR-NUM value 0005 to non-fatal to allow checking for duplicate records in an area where DUPS are NOT ALLOWED by storing the record and testing the ERROR-NUM value. Normally, Q-LINK will abort on such a condition.

5.2 PCIOS File Handling

Most types of Processor Common Input/Output System (PCIOS) files can be handled by the Q-LINK Processor. The only exception is the interchange format. If you are not sure if a file is a PCIOS file, check the SELECT statement in any COBOL program that uses the file. If the SELECT statement reads "ASSIGN TO DISC" or "ASSIGN TO TAPE", the file is in PCIOS format. If not, it is probably one of the old COBOL file formats not supported by Q-LINK. Several sequential file types not considered PCIOS may be processed as PCIOS SEQuential files. These include any System Data File (SDF) format files and print files.

The Q-LINK processor can process several PCIOS files in a request. The number of files allowed with a Q-LINK request is a generation option. Check with your system administrator to determine the number of files allowed. PCIOS files can be opened, closed and reopened as necessary. They can also be processed concurrently with other OS 2200 data access structures.

While Q-LINK is capable of processing tape files, it is recommended that this be avoided due to the delays in awaiting tape mounts by operations, etc. Tape mounting will cause the Q-LINK server to be busy longer than a reasonable length of time for a transaction-processing environment.

5.2.1 PCIOS File Definition

Before any PCIOS file access can begin, the file must be defined to Q-LINK. To define a PCIOS file, another form of the DEFINE directive is used. The DEFINE F (File) gives the Q-LINK Processor information required to interface with the PCIOS routines. Refer to the Q-LINK Programmer Reference for detailed information on how to define PCIOS files to Q-LINK. Some examples of PCIOS file definition will be shown in the following sample programs. Additionally, before a file is opened, it must be @ASG'ed to the Q-LINK server run. The file may be assigned to the Q-LINK server run via the CSF command. All PCIOS files will be automatically closed by the Q-LINK Processor at the end of request processing. Any files assigned to the server via a CSF command will be automatically @FREEd.

5.2.2 PCIOS File Access

Let us look at a simple example, first. The following Q-LINK program produces a reply consisting of a list of all address records in the file that have a zip code between 33000 and 33999 inclusive:

```
DEFINE F ADDR SEQ 120,20          .  Define the sequential file
. . .
 CSF X '@ASG,A MIS*ADDR.  '       .  Assign it.
 IF X <> 0
   DISPLAY "Can't assign address file"
   STOP
```

```
 ENDIF
 OPEN ADDR INPUT SEQ
 DO .  *** Begin read loop
   READ ADDR AT END BREAK
   IF RDA (21,5) UN9 = 33000
   AND RDA (21,5) UN9 33999
     DISPLAY RDA (1,120)
   ENDIF
 ENDDO
 CLOSE ADDR
RUN
```

Now, let us look at a more complex PCIOS file-handling example. The following Q-LINK program uses an indexed sequential file called ABC*XYZFILE to demonstrate the use of several PCIOS file commands. The object of the program will be to retrieve the keys of all records in the range of 100000000 through 199999999. You may wish to try a similar exercise on your system.

```
DEFINE F XFILE INDEXED 120,20 (1,9)  .  DEFINE FILE
 CSF X '@ASG,A ABC*XYZFILE.'         .  ASSIGN FILE
 IF X < 0                            .  ASSIGN OK?
   DISPLAY "CAN'T ASSIGN FILE."
   FACERR X                          .  LOG REASON
   STOP                              .  QUIT
 ENDIF
 CSF X '@USE XFILE.,ABC*XYZFILE.'
 OPEN XFILE INPUT DYNAMIC            .  OPEN FILE
 RDA (1,9) = '100000000'             .  SET KEY VALUE
 START XFILE INVALID KEY NOFIND EQGT .  POSITION FILE
 DO  .  *** Begin read loop
   READNEXT XFILE AT END BREAK       .  READ UNTIL END
   IF RDA (1,9) > '199999999'        .  WITHIN RANGE?
     BREAK                           .  END OF RANGE
   ENDIF
   SET X = X + 1                     .  COUNT RECORD
   DISPLAY 'KEY VALUE = ' +          .  DISPLAY KEY
   DISPLAY RDA (1,9)
 ENDDO
 DISPLAY 'END OF SEARCH - FOUND ' +
 EDIT X 'ZZZ,ZZ9' +
 DISPLAY ' RECORDS.'
 STOP
CLOSE
 CLOSE XFILE
NOFIND
 DISPLAY 'No record at or above 100000000 found.'
 GO CLOSE
RUN
```

The reply created by the above Q-LINK program would look like this if no records with keys greater than or equal to 100000000 exist:

```
No record at or above 100000000 found
```

Otherwise, it might look like this:

```
KEY VALUE = 100020020
KEY VALUE = 140294920
KEY VALUE = 192938930
END OF SEARCH - FOUND       3 RECORDS.
```

5.2.3 PCIOS Variable Length Record Considerations

Q-LINK allows records of any length to be written within a record's defined length; however, ASCII COBOL (@ACOB) uses certain rules concerning variable length records. ASCII COBOL writes variable length records based on two things: the length of the 01-level record

definition being written (named in the COBOL WRITE statement) and/or the length of the last "OCCURS DEPENDING ON" (ODO) group in the record being written. If a record definition contains multiple ODO groups, the record size will be determined as follows: each of the ODO groups, except the last, will have its size calculated at the maximum number of occurrences in the group. The last ODO group will be examined to see how many occurrences actually exist in that group. That number will be multiplied by the entry size. As only the last ODO group is used in the calculation of the record length, calculation of the number of characters to write is relatively simple in Q-LINK. All you need to know is the length of the fixed portion of the particular record. For records containing ODO groups, all you need in addition is the number of occurrences and occurrence entry length for the last ODO group. Let us look at an example.

Assume we have an indexed sequential file with the following COBOL definition:

```
01 PROD-HIST.
    05 PROD-CODE       PIC X(5).     <- Record key.
    05 CURR-PRICE      PIC 9(5)V99.
    05 NBR-PAST        PIC 9(10) COMP.
    05 PAST-PRICES OCCURS 1 TO 20
            DEPENDING ON NBR-PAST.
        10 EFF-DATE    PIC 9(6).
        10 OLD-PRICE   PIC 9(5)V99.
```

To write the above record from Q-LINK, we would need to know the record's length up to the last (in this case only) ODO clause, and the length and number of occurrences of the ODO items. In this record, the fixed portion, PROD-CODE through NBR-PAST, is 16 characters (9(10) COMP = 4 characters), and each occurrence of PAST-PRICE is 13 characters. The formula to calculate this record's length may then be stated as 16 + (NBR-PAST * 13).

In the Q-LINK request, the commands needed to WRITE this record might look like this:

```
DEFINE F PROD-HIST INDEXED 276,10 PROD-CODE
DEFINE N PROD-LEN                 .  To calc variable rec length
. . .
PROD-LEN = RDA NBR-PAST * 13
PROD-LEN = PROD-LEN + 16
WRITE PROD-HIST PROD-LEN INVALID-KEY KEY-ERR
. . .
```

5.3 Using the SORT Interface

The Q-LINK Processor provides an easy-to-use interface to the system SORT subroutine. There are only three commands related to the SORT subroutine interface: SORT, RELEASE and RETURN. The SORT command is used to initialize the sort interface and define sort parameters. The RELEASE command releases (writes) records to the sort subroutine, and the RETURN command returns (reads) records from the sort subroutine. The sort may be entered many times within a Q-LINK program as long as the following sequence of events is used:

1. The SORT command is issued to initialize the sort subroutine and describe sort keys, record length, etc.
2. One or more records are RELEASEd to the sort subroutine.
3. One or more records are RETURNed from the sort subroutine. Attempting a RETURN if no records were RELEASEd will result in a run-time error.

The use of the sort command should be limited to the amount of data that can be sorted in memory. Q-LINK automatically allocates 22,500 words of memory when the SORT is initialized. You may optionally request up to 40,000 words of sort memory. Approximately 830 100-character records can be sorted in 22,500 words of memory, and about 1,450 100-

character records can be sorted in 40,000 words of memory. While no actual limit on the number of records that can be sorted by a Q-LINK server exists, sorting large amounts of data will affect other users. For larger sorts, the server should be started with sort files XA, XB and XC pre-assigned. Q-LINK uses the standard 2200 Sort package for processing the SORT commands.

The following is a simple example of a Q-LINK program that selects records from a DMS 2200 database, sorts the records, and lists fields from the sorted records.

```
INVOKE SUB-SALES IN MARKETING
 WILDCARD IS '?'
 IMPART
 OPEN SALESMAN
 SORT 75,75 1,5,DISP,A (40000)     .  INIT SORT
 F4 F SALESMAN-REC SALESMAN AREA
 DO SORT-IN UNTIL ERROR-NUM = 7    .  DO INPUT PROC
   DEPART
   DO SORT-OUT                     .  DO OUTPUT PROC
   STOP
.  *** Release database records to the sort
.  *** routine.....
SORT-IN PROCEDURE
 IF SLSM-KEY = '2??1'              .  WILDCARD COMPARE
   RELEASE SALESMAN-REC            .  RELEASE TO SORT
 ENDIF
 F4 N SALESMAN-REC SALESMAN AREA
 ENDPROC                           .  REPEAT
.  *** Return sorted records and list in reply......
SORT-OUT PROCEDURE
 DO                                .  In-line DO block
   RETURN AT END BREAK             .  RETURN A RECORD
   DISPLAY RDA SLSM-KEY +          .  DISPLAY REPLY LINE
   DISPLAY 6 RDA SLSM-FIRST-NAME +
   DISPLAY 27 RDA SLSM-LAST-NAME +
   EDIT 50 RDA SLSM-TOT-SALES 'Z,ZZZ.99'
 ENDDO
 ENDPROC
RUN
```

The SORT command above contains several parameters. The first two, “75,75”, indicate the minimum and maximum record size in characters. In this case, a fixed length record is being sorted. The next four parameters specify the sort key field which must be given as a direct RDA reference including the data type, followed by the ascending (A) or descending (D) indicator. There may be up to ten sort keys specified. The last parameter, “(40000)”, specifies that 40K of sort memory is to be allocated.

The length of the record being sorted must also be specified on the RELEASE command. Specifying the length is done by using the name of an invoked database record, a defined PCIOS file, or a numeric integer literal. In this program, the database record name was used. A name and integer literal may be combined if data is being appended to the sort record.

Note that the RETURN command, like the PCIOS READ, requires an AT END clause.

5.4 Accessing RDMS 2200 Tables

RDMS 2200 provides yet another file structure for maintaining data. Q-LINK can retrieve/update this data using standard RDML/SQL command formats. Let us look at a typical program that retrieves data from several RDMS 2200 tables in order to create a result RID.

In the following example, the WHERE clause shows "CUST_KEY = ACCOUNT" as part of the compound condition. This condition implies that for every occurrence selected in the CUST_ADDR_TAB, RDMS 2200 will select all rows from the ORDER_HEADER_TAB whose CUST_KEY matches a value in the ACCOUNT.

The last part of the WHERE clause matches, ORDER_KEY, in yet a third table, ORDER_LINE_TAB. However, ORDER_KEY must be qualified with the name of the table (i.e., ORDER_LINE_TAB.ORDER_KEY) since the same column name is used in both tables. The ORDER_LINE_TAB represents the individual items that make up one order of a particular customer.

The command literal portion of the DECLARE CURSOR command shown below is staged in the print buffer of Q-LINK, $PBUFF, by using the "DISPLAY ... +" convention described in Chapter 8 of this User Guide (see the "Alternate Use of the Output Buffer" subsection). Then, once the command is complete, $PBUFF is referenced on the Q-LINK RDMS command.

Once the selection criteria is established with the DECLARE CURSOR command, each row that meets the criteria is retrieved with the FETCH NEXT command. In addition, since variables ($P1 and $P2) are used on the DECLARE CURSOR, an OPEN cursor command must precede the FETCH NEXT.

5.4.1 RDMS vs. RDMS+ Command

The Q-LINK RDMS+ command can be used to continue a Q-LINK RDMS command. In the example, the RDMS 2200 FETCH NEXT command including the placeholder variables ($P1 through $P6) and the status variables (RDMS-STAT and RDMS-AUX) are placed on a single RDMS command. However, the program variables (NAME, ACCOUNT, etc.) that correspond to the placeholder variables are placed on separate RDMS+ commands. The RDMS+ command is used to continue an RDMS command. In this example, all of the program variables could have been placed on one RDMS+ command or attached to the RDMS command itself.

See the Q-LINK Programmer Reference for the rules governing the use of the RDMS and RDMS+ commands. For the rules regarding the use of RDMS 2200 RDMS/SQL commands, see Unisys Publications, UP-10094, RSA Operations and Programming Guide.

```
INIT
LISTOFF
INDEX QKMS*RDMSDATAINDX
DEF F CUST_ADDR SEQ 171,0
DEF F ORDER_HEADER SEQ 168,0
DEF F ORDER_LINE SEQ 83,0
DEF RA ORDER_HEADER AFTER CUST_ADDR
DEF RA ORDER_LINE AFTER ORDER_HEADER
.  RDMS VARS
DEF A RDMS-STAT ' '
DEF N RDMS-AUX 0
DEF A ERROR-MSG 132
DEF A STATE-VAR 2
DEF A CITY-VAR
.
 ACCEPT STATE-VAR
 SHIFT STATE-VAR TO UPPER
 ACCEPT CITY-VAR
 SHIFT CITY-VAR TO UPPER
 RDMS 'BEGIN THREAD FOR APPLICATION UDSSRC RETRIEVE ;' ;
       RDMS-STAT RDMS-AUX
 DO ERRCHK
 RDMS 'USE DEFAULT SCHEMA DEMOSCHEMA;' ;
```

```
 RDMS-STAT RDMS-AUX
 DO ERRCHK
.
 D 'DECLARE LST CURSOR SELECT NAME, ACCOUNT, ' +
 D 'ORDER_KEY, ORDER_PRICE, ITEM_DESC, UNIT_PRICE ' +
 D 'FROM CUST_ADDR_TAB, ' +
 D 'ORDER_HEADER_TAB, ORDER_LINE_TAB ' +
.
 D "WHERE STATE = $P1 AND CITY = $P2 " +
 D 'AND CUST_KEY = ACCOUNT ' +
 D 'AND ORDER_LINE_TAB.ORDER_KEY = ' +
 D 'ORDER_HEADER_TAB.ORDER_KEY;' +
.
 RDMS $PBUFF RDMS-STAT RDMS-AUX STATE-VAR CITY-VAR
 DO ERRCHK
.
 RDMS 'OPEN LST USING $P1, $P2;' ;
 RDMS-STAT RDMS-AUX STATE-VAR CITY-VAR
 DO ERRCHK
.
 DO
 RDMS 'FETCH NEXT LST INTO $P1, $P2, $P3, $P4, ';
      '$P5, $P6;' RDMS-STAT RDMS-AUX
 RDMS+ RDA NAME
 RDMS+ RDA ACCOUNT
 RDMS+ RDA ORDER_KEY
 RDMS+ RDA ORDER_PRICE
 RDMS+ RDA ITEM_DESC
 RDMS+ RDA UNIT_PRICE
 IF RDMS-STAT = '6001'
   BREAK
 ENDIF
 DO ERRCHK
 TABS ON
 D RDA NAME +
 D RDA ORDER_KEY +
 D RDA DESC +
 EDIT RDA TOTAL_PRICE 'ZZZ,ZZZ.99' +
 EDIT RDA UNIT_PRICE 'Z,ZZZ.99'
 TABS OFF
 ENDDO
 RDMS 'COMMIT WORK ;' RDMS-STAT RDMS-AUX
 DO ERRCHK
 RDMS 'END THREAD ;' RDMS-STAT RDMS-AUX
 DO ERRCHK
 STOP
ERRCHK PROCEDURE
 IF RDMS-STAT '0000'
   DISPLAY 'RDMS ERROR STATUS:' +
   DISPLAY RDMS-STAT +
   DISPLAY ' RDMS AUX INFO:' +
   DISPLAY RDMS-AUX
   DO UNTIL ERROR-MSG = $SPACES
     RDMS 'GETERROR INTO $P1 ;' ;
          RDMS-STAT RDMS-AUX ERROR-MSG
     DISPLAY '' +
     DISPLAY ERROR-MSG
   ENDDO
   STOP 9999
 ENDIF
 ENDPROC
RUN
```

Chapter 6: Compiling Q-LINK Programs

All the Q-LINK request examples shown up to this point have used the source form. Usage of the source form required the source code of the program to be sent to the Q-LINK server for editing and compiling each time. In addition, where DMS 2200 access is involved, the INVOKE processing had to be done each time. These operations can consume a great deal of time, especially when the INVOKE involves a large subschema, and can amount to several seconds before actual processing of data can take place. To eliminate this extra processing, Q-LINK has the ability to save the object form of the request. The object form is what is created when the COMPILE, RUN or SAVE directive is executed.

The SAVE directive, which implies a COMPILE, causes the Q-LINK server to save the entire object request as an omnibus element into a program file. The complete format of the SAVE directive is:

SAVE *program-name* [INTO *qualifier*filename*]

The *program-name* may be any valid element name (including version). The INTO is optional, and is used only if the object is to be saved into a file other than the default Q-LINK object program file Q$LNK*Q$LNKOBJ (this file must have been previously catalogued by your support personnel).

Once an object program has been saved, it can be recalled by using the RUN directive in the following format:

RUN [*program-name* [FROM *qualifier*filename*]]

Both the *program-name* and *qualifier*filename* have the same meaning as in the SAVE directive.

The object request includes all program commands, the data storage area, file definitions and DMS 2200 information. When an object program is invoked by a RUN directive, the entire environment is restored from the object program file and execution is begun immediately. Running object request programs is the most efficient method of using Q-LINK for a production environment.

There are some cautions to be considered when running object programs. Object programs are sensitive to changes in the configuration of the Q-LINK processor and, where DMS 2200 is involved, changes to subschemas. Like COBOL programs, Q-LINK object programs must be recompiled if the invoked subschema is changed. This can be accomplished quickly, since compilation of even large Q-LINK programs will only take a few seconds.

Let us look at the compiling process. The only difference in the request program is that the RUN directive is replaced by the SAVE directive, and any data that would have been placed after the RUN directive would not be present. Let us create a short Q-LINK program and compile it.

In this program, we will retrieve a single customer master record from a DMS 2200 database and format some basic information into the BIS reply.

```
INVOKE SUBCUST IN COMPSCHEMA
DEFINE N CUSTNO
 ACCEPT CUSTNO                    .  GET CUST NUMBER
 RDA CUSTOMER-NUMBER = CUSTNO     .  PUT KEY IN RECORD
 IMPART
 OPEN CUST-AREA
 SET DBDN CUST-ANAME = 'CUST-AREA'
 FETCH5 CUST-MASTER-REC
 IF ERROR-NUM NOT = 0
   DISPLAY "CAN'T FIND CUSTOMER MASTER RECORD."
   STOP 9999
 ENDIF
 DISPLAY '<<<< CUSTOMER ADDRESS QUERY >>>>'
 DISPLAY ' '
 EDIT RDA CUSTOMER-NUMBER '999999999' +
 DISPLAY 20 RDA CUSTOMER-NAME
 DISPLAY 20 RDA CUST-ADDR1
 DISPLAY 20 RDA CUST-ADDR2
 DISPLAY 20 RDA CUST-CITY +
 DISPLAY 50 RDA CUST-STATE +
 DISPLAY 52 RDA CUST-ZIP
 DISPLAY 20 '(' +
 DISPLAY RDA CUST-AREA-CODE +
 DISPLAY RDA ') ' +
 DISPLAY RDA CUST-PHONE +
 DISPLAY'-'+
 DISPLAY RDA CUST-PHONE-EXT
 SAVE CUSTADDR
```

This request is a good example of how quickly a simple query transaction can be developed using Q-LINK. Notice that in this example the RUN directive has been replaced by a SAVE directive. We are telling Q-LINK to save the entire environment of this program as an object element named CUSTADDR into the default object program file (Q$LNK*Q$LNKOBJ). Try sending a similar request to Q-LINK via the QUTIL run. If no errors were detected by Q-LINK, the object request will be saved. The request program will NOT be executed.

To run the saved CUSTADDR request, we need to create a request containing the following:

```
RUN CUSTADDR
102039928                          <— Customer number to find.
```

The saved request is the form that will actually be used in production. If you use QUTIL to send a saved request, you should notice improved response time over a request that has to be compiled before execution. It will also be much easier to set up the request in your BIS runs when compiled requests are used.

Since the default object program file is shared by all active Q-LINK servers, the opportunity to @PACK it will only present itself when all Q-LINK servers are shutdown. When developing a new request, it is recommended that you use your own program file for saving programs until you have completed testing. A non-default object file is only assigned to the server while it is performing a LOAD or SAVE operation.

To transfer the final object program to the default file, you may use a combination of the LOAD and SAVE directives as follows:

```
LOAD CUSTADDR FROM MY*SRC
SAVE CUSTADDR
```

The LOAD directive has the same format as SAVE. LOAD will load the object program, as does the RUN directive, but will not begin execution of the program.

Chapter 7: BIS Runs Using Q-LINK

So far, we have covered how requests can be processed using the QUTIL run, and how a request program can be compiled. Obviously, we do not want the end user to have to run QUTIL each time he needs to interface with Q-LINK. We need to set up our own BIS runs to interface with the end user and with Q-LINK. Writing BIS runs for this purpose is no more difficult than writing any other BIS run. If you are not familiar with BIS run writing at this point, it would be wise to take some time out now to learn more about this subject before continuing.

The basic BIS run will solicit some input from the user, set up a Q-LINK request, call the QLK function to interface with a Q-LINK server, and display the results to the end user. Let us write a run to invoke the compiled request programs saved in the previous chapter. The run might appear as follows:

```
@CHG V1I4 MODE1$ CHG V2I5 TYPE$    .  Get curr mode/type.
@10: .  The following is the user's input screen.
@BRK,0,1 CHG INVAR$ V3I9           .  VARIABLE FOR INPUT CUSTOMER NUMBER

           <<< CUSTOMER ADDRESS RETRIEVAL >>>

 ENTER CUSTOMER NUMBER :^0        , (JUST XMIT TO QUIT).
@BRK OUT,0,1,-0,1,4,1,1,Y,N,,I     .  Display input screen.
@IF V3 = '         ' GTO 99        .  Exit if no input
@ .  Build the query request in current result.
@BRK,0,1
RUN CUSTADDR
V3                                 .  User supplied input line.
@BRK RNM -1                        .  Rename result (request) as -1
@QLK,0,1,-1,0,1,'SERV1',90         .  Send request to Q-LINK
@DSP,0,1,-0                        .  Display reply
@GTO 10                            .  Go display input screen.
@90:RNM -1                         .  Rename reply
@CHG V2I3 STAT1 CHG V3I5 STAT2     .  err stat & line count.
               <<<<<< ERROR RETURNED >>>>>>
                      STATUS V2
        NOTIFY COORDINATOR, RESUME TO DISPLAY REPLY

@BRK,0,1 OUT,0,1,-0,2,3,1,1,Y      .  Display  error  screen.
@DSP,0,1,-1                        .  Display  error  reply
@99:GTO END
```

This run is very simple and demonstrates just how easily a complete query transaction can be written in a matter of minutes. Let us examine this run a little closer. This run would not affect any permanent RIDs, and would cause no writes to the BIS recovery tape. This is a good efficiency technique. If the user desires to save the display, he could use manual functions (REP, XR, etc.) before resuming.

The renaming of the result containing the request, just prior to the QLK function call was not necessary in this case because the entire result contained less than 50 lines. The QLK

function transfers a block of 50 lines to Q-LINK at a time. If Q-LINK generates reply lines back to the QLK function before the entire request has been transferred, the QLK function will attempt to write to the current result (which would still be open for reading additional request lines) causing a BIS error condition. As a rule, if your request is less than 50 lines, it may be sent from result -0. Otherwise, it must be renamed so it will not conflict with the reply, which must be written into result -0.

There are many variations on how the Q-LINK interface can be used in a BIS run. In this case, we demonstrated a simple query transaction. Other applications may retrieve records from a DMS 2200 database, RDMS 2200 tables and/or PCIOS files to be merged with BIS data. Another possibility would be to send an entire BIS report to Q-LINK to update an RDMS table or other file structure outside of BIS. Some other run examples will be shown in the chapter called, “How to Do It with Q-LINK”.

Chapter 8: Advanced Q-LINK Features

In this chapter, some more advanced features of Q-LINK will be discussed. These will include advanced Record Delivery Area (RDA) referencing techniques, methods of controlling print output and output formatting for BIS. Many of the techniques described in this chapter will be used in the "How to Do It with Q-LINK" chapter of this manual.

8.1 Advanced RDA Referencing

Because of the way in which the Q-LINK Processor addresses the RDA, the user can gain a great deal of power by using the advanced forms of RDA reference to perform various types of data manipulation.

8.1.1 RDA Field Name Reference

As seen in some of our earlier examples, data elements within the RDA may be referenced by name. When Q-LINK invokes a DMS subschema, it automatically builds an index file containing the schema description of every data field in the records included in the subschema. This file is referred to as the primary data item index. In addition to the primary data item index, Q-LINK is also able to use a secondary data item index file. The secondary data item index file may be created for data descriptions not available in the object schema and subschema using the QINDEX Processor. The QINDEX Processor converts COBOL file description source images into a Q-LINK data item index file. This file is then identified to Q-LINK using the INDEX directive. The data item index for a data element automatically provides the same information that you would provide in a direct RDA reference: the start position, length and data type.

In cases where the same data item name is used in several different record types, you must qualify the item name to the record in which the reference is to be made. For example, the data item name PART-NUM is used in three record types: PART-MSTR, USAGE, and PART-PRICE. If we were to use the RDA reference 'RDA PART-NUM', Q-LINK would automatically generate the reference for the PART-NUM field of the record that appears first in the subschema definition. Q-LINK does not have a mechanism to determine that the item name requires qualification. To ensure that the correct RDA positions will be used, the item reference must be qualified by the correct record name as follows:

```
RDA PART-NUM IN USAGE
```

This will generate the correct internal RDA reference.

If a data item name occurs more than once within the same record, only the first definition will be included in the data item index file. In this situation, the items may be defined within the Q-LINK request by using the DEFINE RDA directive. Additionally, record qualification may not be used in conjunction with a direct RDA reference.

8.1.2 RDA Fields

RDA data fields may be defined and/or redefined using the DEFINE RDA directive. For example, if we have a database record containing a field named PART-NUMBER in record positions 1 thru 12, and we wish to break it down into three sub-fields, we could use the DEFINE RDA as follows:

```
DEFINE RDA PART-PREFIX (PART-NUMBER,4)
DEFINE RDA PART-CAT    (*,2)
DEFINE RDA PART-SUFFIX (*,6)
. . .
DEFINE RDA DESC        (*PART-NUMBER,30)
```

We may reference the entire field by its schema-defined name (PART-NUMBER), or any of the defined subfields using the three redefinitions: PART-PREFIX, PART-CAT or PART-SUFFIX. A direct RDA reference can additionally be used to reference any other part of the field.

The RDA definition of DESC, specifies that DESC begins immediately following PART-NUMBER.

Here are some examples of referencing the PART-NUMBER:

```
DISPLAY RDA PART-NUMBER             <— Display the whole field.
DISPLAY RDA PART-CAT                <— Display positions5&6.
DISPLAY RDA (1,1)                   <— Display the 1st char.
```

8.1.3 RDA Indexing

RDA Indexing is a powerful feature allowing the user not only to handle data fields that are contained within an OCCURS clause, but to manipulate variable data strings as well.

Let us examine the use of RDA indexing in handling an OCCURS clause. RDA indexing differs from COBOL indexing or subscripting in that it is always based on character, or byte, offsets and not on the number of the occurrence. This is better explained by example.

Assume we are working with the following database record definition:

```
01  PRICE-HISTORY
    05  PH-PART-NUM            PIC X(12)
    05  PH-CURR-PRICE          PIC 9(7)V9(3)   USAGE COMP
    05  PH-NO-PRICES           PIC 9(10)       USAGE COMP
    05  PH-PREV-PRICES OCCURS 20 TIMES
        DEPENDING ON PH-NO-PRICES
        10  PH-EFF-DATE        PIC 9(6)
        10  PH-PRV-PRICE       PIC 9(7)V9(3)   USAGE COMP
```

When the subschema and schema containing this record are INVOKEd in Q-LINK, the primary data item index will contain the RDA references for all these fields as follows:

```
Field Name      Internal RDA Reference
PRICE-HISTORY   (1,220)
PH-PART-NUM     (1,12)
PH-CURR-PRICE   (13,4) COMP .000
PH-NO-PRICES    (17,4) COMP
PH-PREV-PRICES  (21,10)
PH-EFF-DATE     (21,6)
PH-PRV-PRICE    (27,4) COMP .000
```

When you reference a data field name that has not been locally defined by a DEFINE RDA, Q-LINK will actually use the internal RDA reference as shown above. Notice that there is no indication that any fields are part of an OCCURS clause, and that Q-LINK refers to the first occurrence of the fields within the OCCURRING group. To reference one of the occurring fields in a COBOL program, a statement similar to the following would be coded:

```
MOVE PH-EFF-DATE (SUB1) TO OUT-DATE.
```

SUB1 contains 1 for the first occurrence of PH-EFF-DATE, 2 for the second occurrence, and so on. In Q-LINK, RDA indexing works a bit differently:

- Any field may be indexed, not just those within an OCCURS group.
- The index must refer to the relative character offset of each occurrence of the field.

For example, using the definition above, the length in bytes of the occurring group (PH-PREV-PRICES) is 10; 6 bytes for PH-EFF-DATE, and 4 bytes for the computational field PH-PRV-PRICE. The first occurrence of PH-EFF-DATE has an offset of 0 bytes; the second is offset by 10 bytes, the third 20, etc.

An RDA index is specified as part of a Q-LINK RDA reference as follows:

RDA *RDA-reference* [:*index*]

Where *RDA-reference* may be a field name or a direct RDA reference, and may be qualified by record name. The *index* must follow a colon (:), and may be either a numeric integer literal or a numeric integer variable. If a variable is used, it may be one of the predefined variables, or a user-defined variable.

The following example is part of a Q-LINK program used to list price history information using the above record description. This Q-LINK program illustrates the use of RDA indexing to reference data fields within an OCCURS clause:

```
INVOKE .....
DEFINE N PHX                        .  USE FOR PRICE HIST INDEX
DEFINE N PCNT                       .  USE FOR PRICE HIST OCCURS COUNT
. . .
<< Database access routines, etc.  here >>
. . .
PRINT-HIST PROCEDURE
 PHX = 0                              <- The RDA index variable.
 PCNT = 0                             <- The occurs counter.
 DISPLAY 'PRICE HISTORY FOR PART NUMBER: ' +
 DISPLAY RDA PH-PART-NUM
 DO WHILE PCNT <= RDA NO-PRICES
   EDIT 11 RDA PH-EFF-DATE :PHX '99/99/99' +
   EDIT 20 RDA PH-PRV-PRICE :PHX 'Z,ZZZ,ZZZ.999'
   PHX=PHX+10                         <- Inc.  to next occurrence
   PCNT = PCNT + 1                    <- Increment occurs count
 ENDDO
 ENDPROC
```

The output of the above program would look like this:

```
PRICE HISTORY FOR PART NUMBER: 022037518721
          10/11/84      1,103.902
          07/24/81        905.200
          04/21/77        823.110
```

8.1.4 RDA Subscripting

While RDA indexing in Q-LINK is very powerful, it can be quite complicated to use, especially when one is accustomed to COBOL programming. RDA subscripting in Q-LINK works more like COBOL subscripting than does its indexing features. To use RDA subscripting, we must first define a subscript variable for each subscripted RDA data item. Unlike COBOL, a different subscript variable is required for each subscripted data item. Using the same record area definitions used in the discussion of RDA indexing, we could write the same Q-LINK request using RDA subscripting as follows:

```
INVOKE .....
DEFINE SUB PPSUB PH-PRV-PRICE .  Subscript variable
. . .
<< Database access routines, etc.  here >>
. . .
```

```
PRINT-HIST PROCEDURE
PPSUB = 0
DISPLAY 'PRICE HISTORY FOR PART NUMBER: ' +
DISPLAY RDA PH-PART-NUM
DO WHILE PPSUB <= RDA PH-NO-ENTS
PPSUB = PPSUB + 1 <--Inc.  subscript
EDIT 11 RDA PH-EFF-DATE :PPSUB '99/99/99' +
EDIT 20 RDA PH-PRV-PRICE :PPSUB 'Z,ZZZ,ZZZ.999'
ENDDO
ENDPROC
```

RDA subscripts are much easier to work with in most cases, and just as efficient. The main difference is that subscripting allows occurrences of items of fixed length to be referenced, while indexing allows item referencing to be offset by any number of characters.

RDA subscripting may also be used in referencing multi-dimension table definitions in the RDA.

Assume that the following record definition exists in the currently invoked subschema:

```
01  EXAMPLE-RECORD.
    05  DATA-FIELD1 PIC X(3).
    05  TABLE-A OCCURS 5.
        10  TABLE-B OCCURS 7.
            15  TBL-FIELD-1 PIC X(4).
            15  TBL-FIELD-2 PIC 9(5) COMP.
    05  ANOTHER-FIELD PIC X(12).
```

The following RDA subscript variables may be defined:

```
DEF SUB SUB1 TABLE-A
DEF SUB SUB2 TABLE-B :SUB1
```

Note that a subscript variable is defined for each dimension

To display the second occurrence of TBL-FIELD-1 within the 4th occurrence of TABLE-A we could code the following:

```
SUB1 = 4
SUB2 = 2
DISPLAY RDA TBL-FIELD-1 :SUB1 :SUB2
```

The COBOL program equivalent would be:

```
            MOVE 4 TO SUB1.
            MOVE 2 TO SUB2.
            DISPLAY TBL-FIELD-1 (SUB1 SUB2).
```

Because of the way in which multi-dimensional subscripting was implemented in Q-LINK, no limit (within reason) exists to the number of dimensions that can be handled.

8.1.5 Variable RDA Reference

In addition to RDA indexing, direct RDA references can be made variable in both start position and length by using two predefined reserved variables, S$ and L$, in conjunction with direct RDA references in which zero is specified for the start position and/or length. Using a variable RDA reference gives the user the ability to change an RDA reference dynamically within a Q-LINK program.

To use variable RDA referencing, substitute 0 in the direct RDA reference for the start position, the length, or both start position and length. When Q-LINK encounters a zero start position, it will use the current value of the reserved variable S$. The current value of L$ will be used when a zero length is encountered.

The following command sequences demonstrate the use of variable RDA referencing. Try it on your system:

1. Display using variable start position and variable length:

```
SET RDA (1,10) = 'ABCDEFGHIJ'        <— Put string in RDA.
```

```
SET S$ = 2                          <— Set variable start
SET L$ = 8                          <— Set variable length.
DISPLAY RDA (0,0)                   <— Display variable RDA reference.
```

The following would be displayed in the reply:

```
BCDEFGHI
```

2. Display using variable start position only:

```
SET S$ = 5                          <— Change variable start.
DISPLAY RDA (0,2)                   <— Display using variable start
                                       position only.
```

The following would be displayed in the reply:

```
EF
```

3. Display using variable length only:

```
SET L$ = 1                          <— Change variable length.
DISPLAY RDA (1,0) COMP              <— Display numeric value using
                                       variable length only.
```

The reply is the numeric value of "A":

```
65
```

Variable RDA referencing may be used in combination with RDA indexing or RDA subscripting.

The following is a more practical example of variable RDA referencing involving the SCAN command in a portion of a Q-LINK program. The object of the program is to locate and display a portion of text in the RDA enclosed within a pair of brackets ([ ]). A complete description of the SCAN command will be found in the Q-LINK Programmer Reference.

```
FIND-TEXT PROCEDURE
 X = 0
 SCAN RDA (1,80) '[' X              <— Scan for start bracket
 IF X = -1                          <— Not found??
   DISPLAY 'NO STARTING BRACKET FOUND.'
   BREAK
 ENDIF
 S$ = X + 2                         <— Set start after bracket.
 L$ = S$ - 81                       <— Set len.  for rest of scan.
 X = 0                              <— Reset for second scan.
 SCAN RDA (0,0) ']' X               <— Scan for closing bracket.
 IF X NOT = -1                      <— Found??
   L$ = X                           <— Set for text between [].
 ENDIF
 DISPLAY 'FOUND STRING = ' +
 DISPLAY RDA (0,0)                  <— Display the string.
 ENDPROC
```

Here is the result of three records read into the RDA and processed using the above program:

Input 1:	ABC[DEFG]HIJKLMNOP
Output:	FOUND STRING = DEFG
Input 2:	ABCDEFGHIJ[TEXT TO LOCATE]KLMOPQRSTUVWXYZ
Output:	FOUND STRING = TEXT TO LOCATE
Input 3:	NO BRACKETS
Output:	NO STARTING BRACKET FOUND.

Variable RDA referencing may be used in any command that uses the RDA. It may also be used in combination with an RDA Index or subscript.

8.1.6 Alternate Record Areas

All Q-LINK I/O operations assume that the object record is located beginning in position 1 of the RDA. Additionally, all RDA field references in the primary data item index created during the INVOKE are built assuming the record starts in position 1 of the RDA. Defaulting to position 1 of the RDA presents no problems as long as only one record needs to be accessed at any given time. But this becomes awkward when you need to reference two or more records concurrently. One such example is storing a VIA SET DMS 2200 record whose SET OCCURRENCE SELECTION is LOCATION MODE OF OWNER. In order to store a VIA SET record with LOCATION MODE OF OWNER, it is necessary to set the key of the owner record in the RDA, as well as the data in the object record, before executing the STORE command. Since both records are assumed to start in position 1 of the RDA, the owner's key and member's data would overlay each other, making it impossible to store the record. To avoid this type of situation, Q-LINK provides the DEFINE RA (DEFINE Record Area) directive.

The DEFINE RA directive allows the user to allocate areas within the RDA for I/O of specified records. The DEFINE RA directive applies to DMS 2200 Database records, PCIOS files, BIS DTM queue-aliases, DB4 records and SORT I/O. For a complete description of the DEFINE RA, refer to the Q-LINK Programmer Reference.

The following sequence may be used to store the record mentioned above:

```
INVOKE .......
.  The following is a definition of an alternate record
.  area for the owner record.  (Only done once in the
.  program.)
DEFINE RA INVOICE-HDR AFTER INVOICE-LINE
. . .
 RDA IL-LINE-NO = 1                .  Set data fields
 RDA IL-DESC = 'SCREW DRIVER'      .  in INVOICE-LINE
 RDA IL-QUANTITY = 2
 RDA IL-UNIT-PRICE = 1000
.   *** Set key of owner record.
 RDA IH-INVOICE-NUMBER = 'A10231'
 STORE INVOICE-LINE                .  Store member.
 . . .
```

The DEFINE RA directive in this example specifies that the INVOICE-HDR record is to be positioned in the RDA following the INVOICE-LINE record. This directive must not be entered until an INVOKE has been issued if it involves a database record. When a data item within a record in an alternate record area is referenced, Q-LINK will automatically calculate the correct internal RDA address using the item's relative offset within the record plus the record's offset within the RDA.

Another use of DEFINE RA is to extract parts of a record to create a second record. The following example is a short Q-LINK program in which all records in a database area are FETCHed, and those that meet the selection criteria are used to create a sequential file made up of a few fields from the database record:

```
INVOKE SUB-SALES IN MARKETING
DEFINE F SALES SEQ 15,50          .  Output sequential
DEFINE RA SALES AFTER SALESMAN-REC .  Record area.
DEFINE RDA S-KEY (1,5)            .  Output field
DEFINE RDA S-QUOTA (6,9) SN9 .00  .    definitions.
 IMPART
 CSF X '@ASG,UP SALES.,F///1000 '
 IF X < 0                         .  Assign error?
   DISPLAY "CAN'T ASSIGN OUTPUT."
   STOP
 ENDIF
 OPEN SALES OUTPUT SEQ
 OPEN SALESMAN AREA
 F4 F SALESMAN-REC SALESMAN A
```

```
 DO UNTIL ERROR-NUM = 7
   IF SLSM-REGION = 2              .  Select by regions 2 or 4.
   OR SLSM-REGION = 4
     .  *** MOVE FIELDS TO OUTPUT RECORD...
     RDA S-KEY OF SALES = RDA SLSM-KEY
     RDA S-QUOTA OF SALES = RDA SLSM-QUOTA
     WRITE SALES
   ENDIF
   F4 NEXT SALESMAN-REC SALESMAN A
 ENDDO
 CLOSE SALES
 DEPART
 STOP 0
RUN
```

Note that in the example, the fields in the output sequential file, S-KEY and S-QUOTA were both qualified by the defined file name (SALES). Qualifying the fields was necessary because the file's definition was not processed by the QINDEX processor. If we had created a secondary data item index file using QINDEX as shown below, and used an INDEX directive following the INVOKE, the file name qualification and RDA field definition would not have been necessary.

```
@QINDEX,I SLSDATAIDX.
FILE SALES
       01  SALES-REC.
           05  S-KEY          PIC X(5)
           05  S-QUOTA        PIC 9(7)V99.
```

Note: The leveled input (01, 05, etc.) must follow standard COBOL column placement conventions (7, 8, 12, etc.).

The data item index file, SLSDATAIDX, created by the above QINDEX runstream may be used in many request programs.

An efficiency note: Moving data from one record area to another within the RDA can be accomplished using the SET command to move item-by-item; however, it is much more efficient to use the TRANSFER command when moving an entire record.

Q-LINK Application Development User Guide 8 -11

In the following example, a record area for SORT I/O has been defined. The database record is to be transferred to the SORT record area before the release command is executed.

```
INVOKE ....
DEFINE RA SORT 200                .  SORT I/O at word 200 of RDA
. . .
LOOP
 F4 N SALESMAN-REC SALESMAN A
 IF ERROR-NUM NOT = 7
   TRANSFER SALESMAN-REC TO SORT
   RELEASE SALESMAN-REC             <— see comment *
   GO LOOP
 ENDIF
. . .
```

Note: TRANSFER only works at the record level.

* The reference to SALEMAN-REC on the RELEASE specifies the length of the record to be released, not its location. The record being released is always assumed to be in the SORT record area.

8.2 Redirecting the Output of Q-LINK

Normally, all output is directed to the reply, which is sent directly to the BIS result RID. Q-LINK output may also be redirected to an alternate file for printing on the site printer. While the DISPLAY and EDIT commands are used to format reply output, the PCONTROL command is used to control it. PCONTROL has several functions. It may be used to redirect or switch output, to cause a form feed image to be placed into the current output, or to place a special print control image in the current output for use by the operating system when the file is printed. All special print functions will only be present in the alternate output, and will not be sent back to BIS in the reply.

8.2.1 Output Switching

The print output produced by the DISPLAY, EDIT, TRIMDISP and TRIMEDIT commands will, by default, be directed to the BIS reply. One function of the PCONTROL command is to allow the user to redirect the output of these commands. The initial output mode, where output is directed to the reply, is called LOCAL.

The user may redirect output to an alternate print file in two ways. First, by using PCONTROL with the BRKPT function alone, Q-LINK will attempt to create a default alternate print file named QLK$PRTn, where n will be a number in the range 0 to 9. Q-LINK will try to create QLK$PRT0 first. If it cannot be created (already exists, etc.), the number will be incremented by one. This process will continue until print file QLK$PRT9 is reached. If Q-LINK cannot create a default print file, a run time error will occur. The second way to redirect output to an alternate print file is to use the filename option with the BRKPT function. In this case, Q-LINK will attempt to assign the specified file. If it can be assigned, print will be directed to it. If it cannot be assigned, Q-LINK will attempt to create a new file with the specified name. If Q-LINK cannot create the file, an error condition will result.

When creating print output in BRKPT mode, no DISPLAY, EDIT, TRIMDISP or TRIMEDIT command output will be put into the reply.

The ECHO function of the PCONTROL command will cause output to be directed to both an alternate print file and the BIS reply.

Output may also be sent to the system console by using the CONSOLE function of the PCONTROL command. In this mode, output lines are limited to 50 characters.

PCONTROL may be used throughout a Q-LINK request program to redirect output as necessary. Consider the following when using PCONTROL to redirect print output:

- The CONSOLE function has no effect on the current BRKPTed print file.
- If the PCONTROL ECHO function is used before a BRKPT function, Q-LINK will first attempt to create a default alternate print file as if a BRKPT function without a user specified file were executed.
- When using a default BRKPT print file, switching from BRKPT to LOCAL and back to BRKPT mode, will cause output on the alternate file to be continued. Also, user specified BRKPT files may NOT be used following a BRKPT to a default file.
- When using a user specified BRKPT print file, switching to another user specified or default file will cause the current file to be closed. If you switch back to the original file, it will be reopened as a new file and previously written output will be lost.
- Upon exiting Q-LINK, any default print files will be closed and automatically @SYMed to the current print device (see the PRINTER directive in the Q-LINK Programmer Reference). Any user specified alternate print files will be closed and @FREEed.

The following command sequence will demonstrate the PCONTROL BRKPT, CONSOLE and LOCAL functions.

```
DISPLAY DATE                    <— Start in LOCAL mode
PCONTROL CONSOLE                <— Switch to console
```

```
 DISPLAY 'OPERATIONS: PLEASE IGNORE THIS MESSAGE'
 PCONTROL BRKPT 'my*file-1'          <— BRKPT to your file
 DISPLAY 'THIS IS OUTPUT TO MY*FILE-1'
 PCONTROL BRKPT 'my*file-2'          <— BRKPT to another file
 DISPLAY 'THIS IS OUTPUT TO MY*FILE-2'
RUN                                  <— Will close remaining print files on EXIT.
```

8.2.2 Other Print Control Functions

In addition to redirecting print output, the PCONTROL command may also be used to control forms movement, add EXEC headings, set special margins, request special forms mounts, etc. These functions allow full control over the format of your printed output, but do not affect the reply RID.

The simplest form of operation in this category of PCONTROL functions is the EJECT. This function puts a form feed image into the current output. If in LOCAL mode, it has no effect.

For more advanced print output control, the PCONTROL OPTION function has been provided. This function is used to place a user-specified EXEC print control image into the current print output. Some of the more commonly used types of print control images are described in the Q-LINK Programmer Reference. Refer to the OS 2200 Executive Programmer Reference for complete documentation regarding print control options.

The following portion of a Q-LINK report program illustrates how the PCONTROL OPTION functions can be used to change the forms margins and place an EXEC page heading in the alternate print output. The main output of the request is not be sent back in the BIS reply, but instead goes directly to the alternate print file as defined:

```
. . .
.  Set up report output...
 PC BRKPT 'SLS*QLIST                    .  Start ALT. print file
 PC OPT   'M,88,3,3,8'                  .  Set for 8 lines per inch
 PC OPT   'H,,1,SALE REPORT BY REGION' .  Page heading
.  Begin report ...
. . .
```

The PCONTROL commands in the above example are shown in their abbreviated form. This sequence would be executed only once at the beginning of the request. Q-LINK will begin routing all DISPLAY, EDIT, TRIMDISP and TRIMEDIT command output to the alternate print file, SLS*QLIST, when the first PC command is executed. The next two PC commands will place print control images, which will be used by the operating system when the file is actually printed, into the alternate file. Remember that print control images output by the PCONTROL OPTION function are placed in the current print output; therefore, they must NOT be executed before the PCONTROL BRKPT function.

8.3 BIS Line Formatting

Formatting BIS lines is very easy in Q-LINK. Examples of using the DISPLAY and EDIT commands to construct print lines were illustrated earlier in this manual. Use of these commands will now be discussed in more detail.

Output from both the DISPLAY and EDIT commands is always placed in the Q-LINK reply buffer before being output as a reply line. If the plus (+) symbol has not been used in the command, the buffer is immediately outputted and cleared. If the plus symbol was used, data will be placed in the buffer and the next available column counter will be set to point to the position following the data moved to the buffer. Each time an item is DISPLAYed or EDITed, it is moved to the position in the output buffer indicated by the next available column counter. This is how a multiple item reply line is built. For further flexibility in formatting the reply line, both the DISPLAY and EDIT commands provide a column parameter, which will override the “next available column counter” maintained internally by

Q-LINK. The column parameter allows items to be output anywhere in the line without filling gaps with spaces, etc.

In order to maintain BIS/Q-LINK compatibility, the size of the reply buffer line returned to BIS is 132 characters. Anything in the remainder of the buffer will be dropped.

The following is an example of a procedure within a Q-LINK request used to format detailed report lines. In this case, the reply will go into an unformatted form type. The procedure will also handle the output of column headings desirable when not returning the reply to a formatted type:

```
FORMAT-DETAIL PROCEDURE
 LINE-COUNT = LINE-COUNT + 1       .  Count Line
.  If first line, print headings...
 IF LINE-COUNT = 1
   DISPLAY 'PART NUM DESCRIPTION UNIT';
           ' COST ON HAND INVENTORY'
   DISPLAY 58 'VALUE'
   DISPLAY ' '
 ENDIF
.  FORMAT LINE WITH CALCULATED INVENTORY VALUE...
 DISPLAY RDA PART-NUMBER +
 DISPLAY 11 RDA PART-DESCRIPTION +
 EDIT 32 RDA UNIT-COST 'Z,ZZZ.999' +
 EDIT 43 RDA QTY-ON-HAND 'Z,ZZZ,ZZZ' +
 INVENTORY-VALUE = RDA UNIT-COST * RDA QTY-ON-HAND
 EDIT 55 RDA INVENTORY-VALUE 'ZZZ,ZZZ,ZZZ.999'
 ENDPROC
. . .
```

The output of the above request would not be usable in BIS except to be viewed by the user, because there are no asterisk header lines and no tabs inserted between columns.

Creating output for a formatted BIS form type is similar to formatting output for a free form type. An important feature of Q-LINK in building output for BIS processing is automatic TAB character insertion. BIS uses the TAB character to delimit each field in a formatted form type. The BIS form type may also contain predefined column headings that are automatically added to the reply by the QLK function. Non-Q-LINK reports usually have to be processed through some sort of utility to remove report headings and insert TAB characters before being input to BIS. Q-LINK eliminates this requirement.

Automatic TAB character insertion is controlled by the TABS command. This command is used to toggle the internal TABS switch ON or OFF. When TABS is set ON, each item DISPLAYed or EDITed to the reply buffer will be preceded by a TAB character. TABS ON will therefore increase the length of the output item by one position.

Consider the following two sequences of commands:

Sequence 1:

```
DISPLAY RDA PART-NUM +
DISPLAY 12 RDA PART-DESCRIPTION
```

Sequence 2:

```
TABS ON
DISPLAY PART-NUM +
DISPLAY 12 RDA PART-DESCRIPTION
TABS OFF
```

Assume that PART-NUM is 6 characters long and PART-DESCRIPTION is 20 characters. In the first sequence above, PART-NUM will be placed beginning in column 1, and PART-DESCRIPTION in column 12. In the second sequence, TABS has been set ON; therefore, a TAB character will be placed in column 1 followed by PART-NUM, and a TAB character will be placed in column 12 preceding PART-DESCRIPTION. The following illustrates this more graphically. The caret, '^', is used to show the presence of TAB characters.

Output of sequence 1:

```
1...5...10...15...20...
123ASB     SCREW DRIVER.....          <— No TABS
```

Output of sequence 2:

```
1...5...10...15...20...
^123ASB    ^SCREW DRIVER.....         <— TABS Inserted
```

To further illustrate the formatting of lines for BIS processing, Let us assume that we need to retrieve selected customer master record data from the database and return them to a form type having the following headers:

```
*        .                                  .  AMOUNT   .  AMOUNT   .
*CUST NO . CUSTOMER NAME                    . ORDERED   .    DUE    .
*========.==================================.==========.==========.
```

TAB characters will be required in columns 1, 10, 43, 54 and 65 of the reply. The following Q-LINK request will format reply lines to match the above form. In this case, only customers who have an amount due will be selected. Records will not be retrieved in any particular sequence, as sorting will be left to the end user and BIS:

```
INVOKE AR-SUB IN MIS-SCHEMA
 IMPART
 OPEN CUST
 FETCH4 FIRST CUST-MASTER CUST AREA
 DO WHILE ERROR-NUM = 0
   IF RDA CUST-AMT-DUE <> 0
     TABS ON
     EDIT 1 RDA CUST-NUMBER '99999999' +
     DISPLAY 10 RDA CUSTOMER-NAME +
     EDIT 43 RDA CUST-TOT-ORD-AMT '-ZZZZZZ.99' +
     EDIT 54 RDA CUST-AMT-DUE '-ZZZZZZ.99' +
     DISPLAY 65 ' '
     TABS OFF
     FETCH4 NEXT CUST-MASTER CUST AREA
   ENDIF
 ENDDO
SAVE CUST-DUE
```

If the reply is returned to the form type shown above, it will be ready for BIS processing by the end user. Because the end user can further refine the reply as needed, the Q-LINK request can be kept very simple.

8.4 Alternate Use of the Output Buffer

The reply output buffer can be used to build an alphanumeric string from literals and variables, before setting an alpha variable to the string value. The string is built in the reply buffer using the DISPLAY, EDIT, TRIMDISP and TRIMEDIT commands. Data placed in the reply buffer using the EDIT or TRIMEDIT command will be formatted with the specified edit mask. After the string is built, the SET command is used in the following format to initialize the specified alpha variable to the string value:

SET VAR1 = $PBUFF

Where:

VAR1 is the user supplied alpha variable name.

$PBUFF is the key word causing Q-LINK to move the contents of the reply buffer to the specified variable.

Be careful to use the "+" option on each DISPLAY and/or EDIT command when building strings in this manner. A DISPLAY or EDIT command without the "+" will cause Q-LINK to output the string as a reply line. Q-LINK will space fill the reply buffer at the completion of the SET command.

Here is an example of how $PBUFF might be used to create a concatenated string containing a full file name from several input variables and literals:

```
TRIMDISP FILE-NAME-VAR +
DISPLAY '.' +
CSF-STRING = $PBUFF                .  Move image to CSF variable
CSF CSF-STAT CSF-STRING            .  Issue @ASG,A for file
IF CSF-STAT < 0                    .  Error assigning file?
  FACERR CSF-STAT                  .  Display facility status
  . . .
ENDIF
```

If FILE-QUAL-VAR is "ABC" and FILE-NAME-VAR is "WXYZ", the following CSF-image will be created:

```
@ASG,A ABC*WXYZ.
```

8.5 Using Prewritten Program Source

In many cases, groups of Q-LINK program commands and/or directives will be repeated in several request programs. Q-LINK provides an ADD directive for use in this situation. The ADD directive will tell the Q-LINK processor to obtain input from a symbolic element in an EXEC program file. In this way, often-used portions of Q-LINK programs can be developed and saved for use as needed. The format of the ADD directive follows:

ADD *element-name* [FROM *qualifier*filename*]

The *element-name* can be any valid symbolic element name (including version), but cannot contain all numeric characters.

Assume we have a series of RDA definitions that will be used in several request programs. They can be coded in a RID and ELTed to an EXEC program file. The following RDA definition may be in the symbolic element called Q$LNK*Q$LNKLIB.REC-DEF. Note that the file Q$LNK*Q$LNKLIB is the file Q-LINK will use as the default add file.

```
DEFINE RDA PART-NUMBER (1,5)
DEFINE RDA PART-DESC   (*,20)
DEFINE RDA PART-PRICE  (*,5)  UN9 .00
```

Whenever these definitions are needed in a request program, you can ADD them. They will appear as though they were coded where the ADD was encountered.

An ADD example:

```
INVOKE PARTSUB IN MFGSCHEMA
ADD REC-DEF                        <— Q-LINK will assume default file.
DEF RDA PART-POS-1 (1,1)
. . .
```

As Q-LINK edits the program, the added directives will be listed as if they were included in the original source input.

Chapter 9: How to Do It with Q-LINK

This chapter contains several examples of techniques for using Q-LINK to accomplish tasks encountered in applications development. This is by no means a complete collection of Q-LINK applications. Someone will always find a new or better way. KMSYS Worldwide welcomes user contributions or comments.

9.1 Processing BIS Reports in Q-LINK

This section contains four examples of applications in which an entire BIS report is sent to a Q-LINK server for processing. In the first example, the Q-LINK request will add data to each line of a report and send it back to BIS. This first example will use the #INSERT/ACCEPT covered earlier in this guide. In the second example, the same task will be accomplished using the BIS DTM interface. In the third example, each line of the report sent to Q-LINK will be used to apply an update to database records. Finally, the fourth example will use a Q-LINK server to sort a large BIS RID, thus reducing the impact on BIS system performance.

9.1.1 EXAMPLE 1: Merging DMS 2200 Data into a BIS Report

The objective of this Q-LINK example is to add part descriptions obtained from a DMS 2200 database to an existing BIS report. The original report is developed within BIS, but the column reserved for part description is left blank because this information does not exist in the BIS application. The run will pass the original report to Q-LINK where part names will be inserted into each line. The completed report will be returned to BIS as the current result in the same form type as the original.

The Q-LINK request will be listed first:

```
INVOKE INVENTORYSUB IN MFGSCHEMA
DEF A IN-REC 132                    .  Input variable from BIS.
.  The following define the BIS line in the RDA.
DEF RDA BIS-REC   (201,132)         .  The whole line
DEF RDA TAB1      (BIS-REC,1)
DEF RDA PARTNO    (*,5)
DEF RDA TAB2      (*,1)
DEF RDA PARTNAME  (*,25)
 IMPART
 OPEN PART-AREA
 DBDN PART-ANAME = 'PART-AREA'
 DO                                 .  *** Begin ACCEPT loop
   ACCEPT IN-REC AT END BREAK       .  Read a line
   RDA BIS-REC = IN-REC             .  Move input to RDA
   RDA PM-PART-KEY = RDA PARTNO
   FETCH5 PART-MASTER
   IF ERROR-NUM NOT = 0
     RDA PARTNAME = '**UNKNOWN**'
   ELSE
```

```
      RDA PARTNAME = RDA PM-PART-NAME
    ENDIF
    .  *** Put completed line into reply
    DISPLAY RDA BIS-REC
  ENDDO
SAVE GET-P-NAMES
```

Notice that this program will be compiled before being used in the following BIS run.

The BIS run would look like this:

```
@ . Report is in mode 20, type B.  User supplies RID.
@10:CHG INVAR$ V1I4                  .  Input variable for RID number.
                    <<< GET PART NAMES >>>

                 ENTER RID OF ORIGINAL REPORT:^6    ,
                      (JUST XMIT TO QUIT)
@BRK,0,1 OUT,20,B,-0,1,4,1,1,Y,N,,I  .  Display screen
@IF V1 = 0 GTO END                   .  No input, quit
@ .  Make a result containing Q-LINK request stream...
RUN GET-P-NAMES
#INSERT 20,B,V1                      .  Inserts RID into request stream
@BRK,0,1 RNM -1                      .  Rename current result to -1
@QLK,0,1,-1,20,B,'SERV1',90          .  Sent to Q-LINK Server
@ .
@DSP,20,B,-0                         .  Display completed rept.  User may save it.
@GTO 10                              .  Go ask for another.
@ .
@ .  *********** Q-LINK Error processing *************
@90:RNM -1                           .  Rename reply to keep it.
@BRK,0,1 CHG V5I2 STAT1              .  Get Q-LINK error status
  **** Q-LINK ERROR: STATUS V5 NOTIFY CO-ORDINATOR
       USE F1 (RSM) TO VIEW REPLY.
@OUT,20,1,-0,2,1,1,1,Y               .  Display error screen
@DSP,0,1,-1                          .  Display what we got when error occurred.
```

The original report might look like this before processing by Q-LINK (only the first five report columns are shown):

```
*PART .                        .  UNIT  .QUANTITY.  EXT   .
*NUMB . DESCRIPTION            .  PRICE .ON  HAND.  PRICE .
*=====.========================.========.========.========.=======
^20012^                        ^   45.50^      10^  455.00^ . . .
^20900^                        ^   15.90^      10^  159.00^
^21001^                        ^   90.00^       5^  450.00^
. . .
```

The result after Q-LINK processing (header lines will automatically be replaced when the result is created):

```
*PART .                        .  UNIT  .QUANTITY.  EXT   .
*NUMB . DESCRIPTION            .  PRICE .ON  HAND.  PRICE .
*=====.========================.========.========.========.=======
^20012^BLACK BOX TYPE B        ^   45.50^      10^  455.00^ . . .
^20900^GREEN BOX TYPE T90      ^   15.90^      10^  159.00^
^21001^**UNKNOWN**             ^   90.00^       5^  450.00^
. . .
```

The response time for operations as the above will be quite good. Typically, a 200 line report could be processed in about 2 to 3 seconds.

9.1.2 EXAMPLE 2: Accessing BIS RIDs via DTM

Another way to access BIS data is via the BIS DTM interface. Instead of using the #INSERT directive shown in the previous example, we will use the DTM interface to accomplish the same task; however, the only data passed to the Q-LINK program from the run will be the RID number, which is supplied through a BIS variable. The RID number will be used to set

a data item in the DTM parameter block which will help identify which RID is to be retrieved through the DTM interface. Each row will be read using the Q-LINK DTM READ. The completed report will be returned to BIS as the current result in the same form type as the original. For more information on the DTM interface, see the Q-LINK Programmer Reference.

> It should be noted that #INSERT/ACCEPT as shown in the previous example is a faster method for accessing BIS data.

Before a DEFINE F directive can be issued, the parameter block must be defined in the Q-LINK program or session. A predefined parameter block is provided with each installation of Q-LINK. It is normally located in the element DTM-PARM-BLK in the file SYSLIB*QLINK-1.

> Consult the person responsible for installing Q-LINK in order to determine the actual file name of this file on your system.

This definition can be included in the Q-LINK program by simply issuing the following ADD directive:

```
ADD DTM-PARM-BLK FROM SYS$LIB$*QLINK-1
```

The Q-LINK directives included will appear as follows:

```
.
.         DTM interface parameter block for Q-LINK
.
def rda param-block             (*,168)
def rda pb-dest-queue           (param-block,12) A9
def rda pb-userid               (*,12)           A9
def rda pb-dept                 (*,4)            UN9
def rda pb-password             (*,6)            A9
def rda pb-filler1              (*,2)
def rda pb-mode                 (*,12)           A9
def rda pb-type                 (*,1)            A9
def rda pb-filler2              (*,3)
def rda pb-rid                  (*,4)            A9
def rda pb-start-line           (*,4)            UN9
def rda pb-xfer-lines           (*,4)            UN9
def rda pb-run-name             (*,12)           A9
def rda pb-status               (*,1)            UB9
def rda pb-filler3              (*,3)
def rda pb-err-code             (*,8)            A9
def rda pb-err-message          (*,80)           A9
.                        168 character positions total .
```

The following program does not retrieve all the lines from the RID during one OPEN. A maximum of 2000 lines is read during each OPEN due to the BIS Transfer Queue (MTQ) limit placed on any program using the DTM interface:

```
.      DTM INTERFACE PARAMETER BLOCK FOR Q-LINK
.
ADD DTM-PARM-BLK FROM SYS$LIB$*QLINK-1
.
INVOKE INVENTORYSUB IN MFGSCHEMA   .  Using a default
                                   .    schema file
DEF F BIS-RID BIS PARAM-BLOCK
DEF RA BIS-RID 51                  .  BIS input beyond PARAM-BLOCK
DEF RA PART-MASTER AFTER BIS-RID   .  DMS 2200 input beyond BIS input
.
.  Define 1st 2 columns of BIS RID
.
DEF RDA BIS-REC           (201,132)
DEF RDA TAB1              (BIS-REC,1)
```

```
DEF RDA PARTNO            (*,5)    .  Key for DMS access
DEF RDA TAB2              (*,1)
DEF RDA PARTNAME          (*,25)   .  Updated w/DMS field .
DEF A PV-RID 4                     .  Input variable from BIS .
 RDA PB-DEST-QUEUE = 'MAPPER'
 RDA PB-USERID = 'TEACH'
 RDA PB-DEPT = 7
 RDA PB-PASSWORD = ''
 RDA PB-RUN-NAME = 'QLINK$DTM'
 RDA PB-MODE = '20'
 RDA PB-TYPE = 'B'
 ACCEPT PV-RID
 RDA PB-RID = PV-RID
 Z = 1
 RDA PB-XFER-LINES = 2000          .  Limit # lines due to MTQ limit.
 IMPART
 OPEN PART-AREA
 DBDN PART-ANAME = 'PART-AREA'
.
 DO                                .  DO for multiple OPENs
   RDA PB-START-LINE = Z
   OPEN BIS-RID INPUT SEQ
   IF RDA PB-STATUS <> 0
     D 'OPEN ERROR: ' +
     D RDA PB-STATUS
     D RDA PB-ERR-CODE
     D RDA PB-ERR-MESSAGE
     D RDA BIS-REC
     STOP 96
   ENDIF
.
   DO                              .  READ limited to 2000 lines per OPEN
     READ BIS-RID AT END BREAK
     IF RDA PB-STATUS <> 0
       D 'READ STATUS: ' +
       D RDA PB-STATUS
       D RDA PB-ERR-CODE
       D RDA PB-ERR-MESSAGE
       D RDA BIS-REC
       STOP 97
     ELSE
       IF RDA BIS-REC = '.DTERR***'
         D '*** FATAL ERROR IN BIS DTM RUN: '
         D RDA BIS-REC
         STOP 99
       ENDIF
     ENDIF
     RDA PM-PART-KEY = RDA PARTNO
     FETCH5 PART-MASTER
     IF ERROR-NUM <> 0
       RDA PARTNAME = '**UNKNOWN**'
     ELSE
       RDA PARTNAME = RDA PM-PART-NAME
     ENDIF
     DISPLAY RDA BIS-REC
     X = X + 1
   ENDDO
   IF X = 0                        .  If no lines returned during this OPEN,
     BREAK                         .  BREAK out of this DO and end request.
   ENDIF
   X = 0
   IF RDA PB-STATUS = 1
     CLOSE BIS-RID
```

```
      IF RDA PB-STATUS <> 0
        D 'CLOSE ERROR: ' +
        D RDA PB-STATUS
        D RDA PB-ERR-CODE
        D RDA PB-ERR-MESSAGE
        D RDA BIS-REC
        STOP 98
      ENDIF
    ENDIF
    Z = Z + 2000                    .  Set var. to get next 2000 lines
  ENDDO                             .  SAVE GET-P-NAMES .
```

The BIS run would appear as follows:

```
@ . Report is in mode 20, type B.  User supplies RID.
@10:CHG INVAR$ V1I4                 .  Input variable for RID number.
                    <<< GET PART NAMES >>>

                 ENTER RID OF ORIGINAL REPORT:^6   ,
                      (JUST XMIT TO QUIT)
@BRK,0,1 OUT,20,B,-0,1,4,1,1,Y,N,,I .  Display screen
@IF V1 = 0 GTO END                  .  No input, quit
@ .  Make a result containing the Q-LINK request stream...
RUN GET-P-NAMES
V1                                  .  Supply RID # to request stream
@BRK,0,1 RNM -1                     .  Rename current result to -1
@QLK,0,1,-1,20,B,'SERV1',90         .  Sent to Q-LINK Server
@.
@DSP,20,B,-0                        .  Display completed report.  User may save it.
@GTO 10                             .  Go ask for another.
@.
@ .  *********** Q-LINK Error processing *************
@90:RNM -1                          .  Rename reply to keep it.
@BRK,0,1 CHG V5I2 STAT1             .  Get Q-LINK error status
  **** Q-LINK ERROR: STATUS V5 NOTIFY CO-ORDINATOR
       USE F1 (RSM) TO VIEW REPLY.
@OUT,20,1,-0,2,1,1,1,Y              .  Display error screen
@DSP,0,1,-1                         .  Display what we got when error occurred.
```

The original report might look like this before processing by Q-LINK (only the first five report columns are shown):

```
*PART .                         .  UNIT  .QUANTITY.  EXT   .
*NUMB . DESCRIPTION             .  PRICE .ON  HAND.  PRICE .
*=====.=========================.========.========.========.=======
^20012^                         ^   45.50^      10^  455.00^ . . .
^20900^                         ^   15.90^      10^  159.00^
^21001^                         ^   90.00^       5^  450.00^
. . .
```

The result after Q-LINK processing (header lines will automatically be replaced when the result is created):

```
*PART .                         .  UNIT  .QUANTITY.  EXT   .
*NUMB . DESCRIPTION             .  PRICE .ON  HAND.  PRICE .
*=====.=========================.========.========.========.=======
^20012^BLACK BOX TYPE B         ^   45.50^      10^  455.00^ . . .
^20900^GREEN BOX TYPE T90       ^   15.90^      10^  159.00^
^21001^**UNKNOWN**              ^   90.00^       5^  450.00^
. . .
```

9.1.3 EXAMPLE 3: Applying Changes to the Database

In this example, a BIS report is used to collect input entered by data entry personnel. After data is entered, a run will be used to send the entire report to a Q-LINK server to update a DMS 2200 database. Each line of the report contains a part number and a new unit price

left zero filled. The Q-LINK request program will locate the part, update the unit price and display whether or not it was successful for each line processed. The input report is in a formatted type, the reply from Q-LINK will be unformatted.

This is the Q-LINK request:

```
INVOKE INVENTORYSUB IN MFGSCHEMA
DEF A IN-REC 20
DEF RDA BIS-REC    (201,20)
DEF RDA TAB1       (BIS-REC,1)
DEF RDA PART-NO    (*,5)
DEF RDA TAB2       (*,1)
DEF RDA NEW-PRICE  (*,8) MAPNUM    .  BIS numeric data
 IMPART
 OPEN PART-AREA UPDATE
 SET DBDN PART-ANAME = 'PART-AREA'
 DO
   ACCEPT IN-REC AT END BREAK
   RDA BIS-REC = IN-REC
   RDA PM-PART-KEY = RDA PART-NO
   FETCH5 PART-MASTER
   IF ERROR-NUM = 0
     RDA PM-UNIT-PRICE = RDA NEW-PRICE
     MODIFY PART-MASTER
     DISPLAY 'PART:' +
     DISPLAY RDA PM-PART-KEY +
     DISPLAY ' NEW UNIT PRICE:' +
     EDIT RDA PM-UNIT-PRICE 'ZZ,ZZZ.99-'
   ELSE
     DISPLAY 'PART:' +
     DISPLAY RDA PM-PART-KEY +
     DISPLAY ' NOT FOUND.'
   ENDIF
 ENDDO
SAVE PM-UNIT-PR
```

The BIS run that would be used in this example would look almost identical to the run shown in example 1. The input and output reports would look like this:

Input report:

```
*PART . UNIT  .
*NUM. . PRICE .
*=====.=======.
^20012^  35.50^
^21001^  93.00^
^34501^  66.50^
^60900^ 196.00^
. . .
```

The output result from Q-LINK:

```
PART:20012 NEW UNIT PRICE:     35.50
PART:21001 NEW UNIT PRICE:     93.00
PART:34501 NEW UNIT PRICE:     66.50
PART:60900 NOT FOUND.
```

The BIS run could be made more sophisticated to match the reply to the original report so that data entry personnel could correct and resubmit the remaining updates. Many processes can be carried out in BIS runs before and after interfacing with Q-LINK.

9.1.4 EXAMPLE 4: Extending a BIS Run - No Database Access

While BIS runs can be extremely powerful, they can often get very complex and somewhat inefficient, especially when manipulating large amounts of data. One answer to overcoming some of these problems is to transfer some of the BIS run processing to Q-LINK. Using

Q-LINK does not necessarily mean that DMS 2200 or PCIOS files must be involved. The Q-LINK procedural language allows for much easier data manipulation as well as being more self-documenting. In addition, since Q-LINK procedures are pre-compiled, they run much more efficiently than BIS runs.

Let us examine a very simple example of Q-LINK extending the power of a BIS run. Assume that we have a very large RID — in excess of 2000 lines — and we must sort it on two columns. In addition, a numeric field that is currently left zero filled is to be changed to left justified and space filled. These changes are not that difficult in BIS, but due to the volume, can influence system performance considerably. The following is a portion of a BIS run showing how a Q-LINK request, containing Q-LINK commands and data, is built and processed.

Assume report to be processed is in result -2.

```
. . .
@ . Create request stream in current result
DEF A BISIN 132
DEF RDA NUM-FIELD-IN (12,5) MAPNUM     .  Field to convert
DEF RDA NUM-FIELD-OUT (12,5)           .  Redefinition for
.  after convert
 SORT 132,132 2,5,DISP,A 12,5,UN9,D
.  *** Accept lines from BIS and release to SORT
 DO
   ACCEPT BISIN AT END BREAK
   RDA (1,132) = BISIN
   RELEASE 132
 ENDO
.  *** Return line from SORT, convert and send back to
.  *** BIS
 DO
   RETURN AT END BREAK
   TRIMEDIT RDA NUM-FIELD-IN 'ZZZZZ' + .  Convert
   RDA NUM-FIELD-OUT = $PBUFF          .  Put into BIS rec
   DISPLAY RDA (1,132)                 .  Write back to BIS
 ENDDO
RUN
#INSERT 20,B,-2,4                      .  Input RID added here
@BRK,0,1 RNM -1
@QLK,0,1,-1,20,B,PRODA,199             .  Call Q-LINK
. . .
```

The sorted report with the converted field is now in result -0.

In this example, the Q-LINK program was built in-line within the BIS run. The Q-LINK procedure could have been compiled and saved separately gaining a bit more efficiency.

The example was chosen for clarity, but numerous applications — limited only by the programmer's imagination — exist.

9.2 Tables in Q-LINK

The Q-LINK Processor does not allow definition of array variables; however, arrays or tables can be set up and manipulated in the RDA. The following example should aid in developing table-handling logic.

9.2.1 Example 1: A Table of Accumulators

In this example, as database records are read, a table of accumulators will be built to accumulate totals by sales region, and thus avoid having to sort the input. As each record is read, the program will try to find a region entry in the RDA table. If an entry does not

exist yet, one is created. Once all input records have been processed, only the table entries will be sorted and output as a report.

```
INVOKE SUB-SALES IN MARKETING
.  *** TABLE DEFINITIONS
DEFINE RDA RGN-TABLE (1001,6)          .  ONE TABLE ENTRY
DEFINE RDA RGN-CODE (RGN-TABLE,2)      .  REGION CODE
DEFINE RDA RGN-TOT (*,4) COMP .00      .  REGION TOTS
DEFINE SUB RGNX RGN-TABLE              .  Subscript for table
.  *** SORT RECORD DEFINITIONS
DEFINE RDA SRT-RGN (1,2)
DEFINE RDA SRT-TOT (*,4) COMP .00
.
 IMPART
 OPEN SALESMAN
 F4 F SALESMAN-REC SALESMAN A
 DO WHILE ERROR-NUM = 0
   RGNX = 1
   DO  .  Accumulate sales totals by region in table.
     IF RDA RGN-CODE :RGNX = $LOVALS   .  Not in use?
       RDA RGN-CODE :RGNX = RDA SLSM-REGION
       RDA RGN-TOT :RGNX = RDA SLSM-TOTAL
       BREAK
     ENDIF
     IF RDA RGN-CODE :RGNX = SLSM-REGION
  .  *** Add amount from record to table entry ****
       RDA RGN-TOT :RGNX = ;
         RDA RGN-TOT :RGNX + RDA SLSM-TOT
       BREAK
     ENDIF
     RGNX = RGNX + 1
   ENDDO
   F4 N SALESMAN-REC SALESMAN A
 ENDDO                                 .  End of WHILE ERROR-NUM = 0
.
.  Sort table entries by region code and list the totals
.
 SORT 6,6 1,2,DISP,A                   .  Init sort by region
 RGNX = 1
 DO WHILE RDA RGN-CODE :RGNX NOT = $LOVALS
   RDA SRT-RGN = RDA RGN-CODE :RGNX    .  Entry to sort
   RDA SRT-TOT = RDA RGN-TOT :RGNX
   RELEASE 6
   RGNX = RGNX + 1                     .  Increment to next entry
 ENDDO
 DO                                    .  Begin SORT return loop
   RETURN AT END BREAK
   DISPLAY 'REGION CODE: ' +
   DISPLAY RDA SRT-RGN +
   DISPLAY ' TOTAL SALES: ' +
   EDIT RDA RDG-TOT COMP 'Z,ZZZ,ZZZ.99-'
 ENDDO
RUN                                    .  (or SAVE)
```

In this example, the table-handling procedures use $LOVALS to determine the end of the table. The RDA is always set to binary zeros during Q-LINK initialization. The name $LOVALS is a special value test for binary zeros. If unsure of the current value of the RDA, the program may have included the following statement at the beginning to set the area to binary zeros:

```
RDA (1001,3000) = $LOVALS
```

The reply produced by this request would look something like this:

```
REGION CODE: 01 TOTAL SALES:   901,129.90
```

```
REGION CODE: 02 TOTAL SALES:       100.00-
REGION CODE: 04 TOTAL SALES: 1,102,203.00
. . .
```

9.2.2 Example 2: Table Lookup

In this example, a simple table lookup will be done. A REGION code in the input record will be translated to the name of the region for output on a report. Only those portions of the program dealing with the table will be shown.

```
.  The following are RDA definitions for the region
.  table: .
DEF RDA TAB-ENT (1001,12)              .  Region code & name
DEF RDA TAB-CODE (TAB-END,2) UN9       .  Region code
DEF RDA TAB-NAME (*,10)                .  Region name
DEF SUB TX TAB-ENT                     .  Subscript for table
DEFINE N TMAX 6                        .  Table limit
. . .
. The following loads the region table from an
. alphanumeric-string literal:
.
LOAD-TAB PROCEDURE
 RDA (1001,100) = ;                    .  This is all one literal.
 '01WEST COAST';                       .  It is entered this way
 '02NORTH EAST';                       .  for clarity.
 '03SOUTH WEST';
 '04MIDWEST   ';
 '05CANADA    ';
 '06EUROPE    '
. . .
.  Translates the numeric region code to region name:
.
 TX = 1
 DO WHILE TX NOT > TMAX
   IF RDA TAB-CODE :TX = RDA SLSM-REGION
     REGION-NAME = TAB-NAME :TX
     BREAK
   ELSE
     TX = TX + 1
   ENDIF
 ENDDO
 IF TX > TMAX
   DISPLAY '*UNKNOWN*'
 ELSE
   DISPLAY REGION-NAME
 ENDIF
. . .
```

When setting up tables in the RDA, it is important to be aware of the maximum area available in the RDA. In this example, the table was only 72 characters, and required an RDA of at least 1072 characters in length.

Chapter 10: Q-LINK Debugging

The Q-LINK processor provides several debugging aids including a diagnostic log file, a command trace that can be invoked on request, and a formatted object dump. This chapter will explain how these debugging aids may be used.

10.1 The Q-LINK Diagnostic Log

Each individual Q-LINK server has assigned to it a temporary diagnostic log file. The amount of information written to the log file depends on the mode of the Q-LINK server. This mode is set by a processor option when the server is initially started. If the server class is configured to be in diagnostic mode, Q-LINK will automatically close the log file at completion of each request and copy its contents to the reply. In diagnostic mode, the developer can see all the results of the request immediately. In the case of a request program compile, the compile listing can be saved in a permanent RID or printed, so that it will be available if a problem occurs in the future production use of the request.

When a server class is configured as production mode, only limited logging will take place. The production mode log will contain a record of the start and end of each request, and when a run time error occurs, an object dump will be written to the diagnostic log. A production mode server can be switched to diagnostic mode for the execution of one request by inserting a DIAG directive as the first image in the request stream. The server will remain in diagnostic mode until the end of the request stream has been encountered.

Diagnostic log files can be retrieved into BIS by sending the RETLOG directive to the server class. Because requests are normally routed to the first available server within a server class, you must supply the desired server's EXEC run-id on the QLK function call. If you do not specify run-id, the log of the first available server in the class will be retrieved.

10.2 Command Trace

A helpful feature of Q-LINK is its command trace. Command trace can be turned on and off anywhere in your program logic using the TRACE ON/OFF Command. When turned on, the trace will write the PC (program counter) of each command as it is executed to the diagnostic log file. Using this information with the compile listing of the request program, you can trace program logic. The following is a short example of the diagnostic log reply from a compile and run with the TRACE ON command:

```
The date and time is 2006/08/09 - 10:42:33.384
It's not December yet.
End of demo....bye
*** Log retrieved by: ......
*** Request by ......
      1   1         $$REQ   JDOE             0104    00000482
      2   1             TRACE ON   .  Turn on command trace.
      3   2             DATE
      4   3             TIME
```

```
        5   4             DISPLAY 'The date and time is ' +
        6   5             DISPLAY DATE +
        7   6             DISPLAY'-'+
        8   7             DISPLAY TIME
        9   8             IF MONTH = 12
       10   9                 DISPLAY 'This is December!'
       11  10             ELSE
       12  11                 DISPLAY 'It''s not December yet.'
       13  12             ENDIF
       14  12             DISPLAY 'End of demo....bye'
       15  13         RUN
*** End of command input ***
*** Compile complete ***
< TRACE turned ON >
***TRACE PC = 0002
***TRACE PC = 0003
***TRACE PC = 0004
***TRACE PC = 0005
***TRACE PC = 0006
***TRACE PC = 0007
***TRACE PC = 0008
***TRACE PC = 0011
***TRACE PC = 0012
***TRACE PC = 0013
*** EOF on request file ***
*** End of log copy ***
.$ENDREQ:BOB/0104/000482:DEV00
```

10.3 Q-LINK Object Dumps

When a run-time error is encountered in a Q-LINK request, the processor automatically produces a formatted object dump before terminating. This dump was originally included as a debugging aid in the actual development of the Q-LINK Processor. It is still used for this purpose; however, you will find this dump very helpful in debugging your Q-LINK applications. The dump can also be produced by using the OBJECT directive.

Refer to the figure containing the Q-LINK Object dump example while going over the following description. The first two sections of the dump will be very useful to the Q-LINK user. The DATA STORAGE INDEX (DSI) contains the names and DATA STORAGE AREA (DSA) locations of all variables, numeric literals and local RDA definitions. There are two numbers following the item's name: one for DSA INDEX and one for WORD NUMBER. The index is used internally for DSA addressing. The word number is used for convenience in locating values stored in variables. At the end of each DSI dump line, is the item type: V for variable, L for literal, etc. Type V will be of interest to the Q-LINK user.

Following the DATA STORAGE INDEX dump is the DATA STORAGE AREA (DSA) dump. The DSA is where you will be able to find the contents of all variables. You will also find numeric and alphanumeric string literals stored here. Other items stored in this area include sort parameters, indexed sequential file key definitions, etc. These entries will not be indexed in the DSI.

To locate a variable in the DSA, first find it by name in the DSI. The WORD number found in the DSI indicates the variable's location in the DSA. The DSA word number is displayed at the left of the DSA dump. The format of a numeric variable is two contiguous words. The first quarter word is reserved for future use with the actual value of the variable stored in the remaining seven quarter words (this is large enough to accommodate the maximum numeric value allowed in COBOL). The format of an alpha variable consists of a two word entry followed by enough words (in multiples of two) to accommodate the alphanumeric string length. The two words preceding the alphanumeric string data are made up of a one

quarter word special value indicator followed by seven quarter words that contain the character length of the alpha numeric string.

If PCIOS files were defined when the dump was taken, the next section of the dump will be a listing of the defined PCIOS files with their current usage and access modes, and the alternate record area word offset. No PCIOS files were in use when the dump shown below was taken.

Next will be a list of the DMS 2200 non-fatal ERROR-NUM table. This table contains a list of DMS 2200 ERROR-NUM values that will NOT cause Q-LINK to automatically error. These codes must be handled by the Q-LINK user when they occur.

If an INVOKE directive was processed as part of the request, a list of all areas, records, sets and database datanames will be next on the dump listing. If the request being processed was in object form, this list will not be shown. This list is obtained from the Q-LINK primary data item index file, not from Q-LINK internal storage. The schema and subschema code of each item will be shown here. For records, the record length in words and alternate area word offset will be displayed. For database datanames, the word length and dataname type code are displayed. This list may be helpful when determining what is included in the invoked subschema.

The last part of the dump output will be the contents of the RDA. This is where values of the last record read, or RDA tables, etc. may be found. The RDA is empty in the example below.

This is an example of a Q-LINK Object dump taken after a run time DMS 2200 DML error:

```
Doing a FETCH without OPEN to get object dump.
     1    1          INVOKE DEMOSUB IN DEMOSCHEMA FILE DMS*SCHABS

** Invoke complete for MT4 DMR **
     2    1              IMPART
     3    2              DISPLAY 'Doing a FETCH without OPEN to get object dump.'
     4    3              FETCH3 FIRST DEMO-ORD AREA
     5    4          .   *** WILL NOT GET THIS FAR....
     6    4              DISPLAY 'Done.........'
     7    5          RUN
*** Compile complete ***
(( Fatal DML error code ))
(( encountered! ))
ERROR-STATUS:   000301 ERR.NUM.:0001  DATE-TIME:092189  151730 COMMAND SEQ.  :
000003            IMPART-DEPART:1
ERROR-AREA:     DEMO-ORD
ERROR-RECORD:
ERROR-SET:
<E891> ** Q-Machine fatal run time error; instruction start at PC 3

*** Start of Data Storage Index ****
ERROR-NUM                        INDEX-00001 WORD-00001 DEC-00000 TYPE-00000-V
C-AKEY                           INDEX-00002 WORD-00003 DEC-00000 TYPE-00000-V
C-PAGE                           INDEX-00003 WORD-00005 DEC-00000 TYPE-00000-V
C-REC                            INDEX-00004 WORD-00007 DEC-00000 TYPE-00000-V
G-AKEY                           INDEX-00005 WORD-00009 DEC-00000 TYPE-00000-V
G-PAGE                           INDEX-00006 WORD-00011 DEC-00000 TYPE-00000-V
G-REC                            INDEX-00007 WORD-00013 DEC-00000 TYPE-00000-V
A30                              INDEX-00008 WORD-00015 DEC-00032 TYPE-00001-V
C-O-R                            INDEX-00013 WORD-00025 DEC-00032 TYPE-00001-V
G-RECORD-NAME                    INDEX-00018 WORD-00035 DEC-00032 TYPE-00001-V
J-DAY                            INDEX-00023 WORD-00045 DEC-00000 TYPE-00000-V
YEAR                             INDEX-00024 WORD-00047 DEC-00000 TYPE-00000-V
MONTH                            INDEX-00025 WORD-00049 DEC-00000 TYPE-00000-V
DAY                              INDEX-00026 WORD-00051 DEC-00000 TYPE-00000-V
DATE                             INDEX-00027 WORD-00053 DEC-00032 TYPE-00001-V
```

```
X                                 INDEX-00030 WORD-00059 DEC-00000 TYPE-00000-V
Y                                 INDEX-00031 WORD-00061 DEC-00000 TYPE-00000-V
Z                                 INDEX-00032 WORD-00063 DEC-00000 TYPE-00000-V
C-AREA-NAME                       INDEX-00033 WORD-00065 DEC-00032 TYPE-00001-V
G-AREA-NAME                       INDEX-00036 WORD-00071 DEC-00032 TYPE-00001-V
IMPART-DEPART                     INDEX-00039 WORD-00077 DEC-00000 TYPE-00000-V
TIME-MSPM                         INDEX-00040 WORD-00079 DEC-00000 TYPE-00000-V
TIME                              INDEX-00041 WORD-00081 DEC-00032 TYPE-00001-V
C-DBK                             INDEX-00044 WORD-00087 DEC-00000 TYPE-00000-V
H-DBK                             INDEX-00045 WORD-00089 DEC-00000 TYPE-00000-V
DATE-NUM                          INDEX-00046 WORD-00091 DEC-00000 TYPE-00000-V
S$                                INDEX-00047 WORD-00093 DEC-00000 TYPE-00000-V
L$                                INDEX-00048 WORD-00095 DEC-00000 TYPE-00000-V
REC$LEN                           INDEX-00049 WORD-00097 DEC-00000 TYPE-00000-V
RB-CODE                           INDEX-00050 WORD-00099 DEC-00000 TYPE-00000-V
A8                                INDEX-00051 WORD-00101 DEC-00032 TYPE-00001-V
$TAB                              INDEX-00054 WORD-00107 DEC-00032 TYPE-00001-V
USER$                             INDEX-00056 WORD-00111 DEC-00032 TYPE-00001-V
SERVER$                           INDEX-00059 WORD-00117 DEC-00032 TYPE-00001-V
DEPN$                             INDEX-00061 WORD-00121 DEC-00000 TYPE-00000-V
STNUM$                            INDEX-00062 WORD-00123 DEC-00000 TYPE-00000-V
*** Start of Data Storage Area ***
0001 000000000000 000000000001 000000000000 000000000000 ??????????????????
0005 000000000000 000000000000 000000000000 000000000000 ??????????????????
0009 000000000000 000000000000 000000000000 000000000000 ??????????????????
0013 000000000000 000000000000 040000000000 000000000036 ???????? ???????
0017 040040040040 040040040040 040040040040 040040040040
0021 040040040040 040040040040 040040040040 040040040040
0025 040000000000 000000000036 040040040040 040040040040  ???????
0029 040040040040 040040040040 040040040040 040040040040
0033 040040040040 040040040040 040000000000 000000000036          ???????
0037 040040040040 040040040040 040040040040 040040040040
0041 040040040040 040040040040 040040040040 040040040040
0045 000000000000 000000256260 000000000000 000000000131 ??????????????????Y
0049 000000000000 000000000011 000000000000 000000000025 ??????????????????
0053 040000000000 000000000010 070071057060 071057062061  ???????89/09/21
0057 040040040040 040040040040 000000000000 000000000000         ????????
0061 000000000000 000000000000 000000000000 000000000000 ??????????????????
0065 040000000000 000000000014 040040040040 040040040040  ???????
0069 040040040040 040040040040 040000000000 000000000014          ???????
0073 040040040040 040040040040 040040040040 040040040040
0077 000000000000 000000000001 000000000000 000321767466 ??????????????????
0081 040000000000 000000000014 061065072061 067072062065  ???????15:17:25
0085 056071064062 040040040040 000000000000 000000000000 .942    ????????
0089 000000000000 000000000000 000000000000 000003314051 ??????????????????)
0093 000000000000 000000000000 000000000000 000000000000 ??????????????????
0097 000000000000 000000000000 000000000000 000000000000 ??????????????????
0101 040000000000 000000000010 040040040040 040040040040  ???????
0105 040040040040 040040040040 040000000000 000000000001          ???????
0109 011040040040 040040040040 040000000000 000000000014 ?        ???????
0113 111121103117 117122104040 040040040040 040040040040 IQCOORD
0117 040000000000 000000000006 124105123124 061040040040  ???????TEST1
0121 000000000000 000000000014 000000000000 000000001631 ??????????????????
0125 000000000000 000000000056 104157151156 147040141040 ???????.Doing a
0129 106105124103 110040167151 164150157165 164040117120 FETCH without OP
0133 105116040164 157040147145 164040157142 152145143164 EN to get object
0137 040144165155 160056040040 040040040040               dump.

*** INVOKE tables follow ***
AREA---DEMO-ADDR           SCODE=00001 SSCODE=00001
AREA---DEMO-CKEY           SCODE=00003 SSCODE=00003
AREA---DEMO-INDX           SCODE=00002 SSCODE=00002
AREA---DEMO-ORD            SCODE=00004 SSCODE=00004
```

```
RECORD--CUST-ADDR-REC       SCODE=00004 SSCODE=00002 RLEN=00045 ROFF=00000
RECORD--CUST-KEY-REC        SCODE=00002 SSCODE=00001 RLEN=00012 ROFF=00000
RECORD--ORDER-COMMENT-REC   SCODE=00013 SSCODE=00004 RLEN=00021 ROFF=00000
RECORD--ORDER-HEADER-REC    SCODE=00011 SSCODE=00003 RLEN=00043 ROFF=00000
RECORD--ORDER-LINE-REC      SCODE=00014 SSCODE=00005 RLEN=00018 ROFF=00000
SET---CUST-KEY-ADDR-SET   SCODE=00005 SSCODE=00001
SET---CUST-ORD            SCODE=00027 SSCODE=00004
SET---ORDH-CMT            SCODE=00008 SSCODE=00003
SET---ORDH-LINE           SCODE=00007 SSCODE=00002
DBDN---AREA-DEMO-CKEY      TYPE=00001, LEN=00003, SCODE=00002, SSCODE=00002
DBDN---AREA-DEMO-ORD       TYPE=00001, LEN=00003, SCODE=00003, SSCODE=00003
DBDN---SH-MISC-ANAME       TYPE=00001, LEN=00003, SCODE=00001, SSCODE=00001

*** Non-fatal ERROR-NUM table ***
ERROR-NUM = 6
ERROR-NUM = 7
ERROR-NUM = 13
**** Dump of RDA follows:
0001 000000000000                                                ????
**** Remainder of RDA is unused ****
** You have not DEPARTed from DMS 2200 - departing with NO rollback.  **
*** End of log copy ***
.$ENDREQ:IQCOORD/0012/000921:TEST102
.$ERRORS,./,SERVER:TEST1/02/*000BC .$PC=0003
```

If you would like to help us make our documentation better, please take a few moments to complete this form and return it to KMSYS Worldwide. We are always looking for ways to improve our products and your feedback will help us reach our goal.

Name _____

Company _____

Address _____

_____

City _____ State/Province _____

Country _____ Zip/Mail Code _____

Document Name _____ OS Level _____

KMSYS Worldwide Product _____ Level _____

Please rate the documentation on a scale of 1 to 5:

	5	4	3	2	1	
Complete	❑	❑	❑	❑	❑	Incomplete
Accurate	❑	❑	❑	❑	❑	Inaccurate
Usable	❑	❑	❑	❑	❑	Unusable
Readable	❑	❑	❑	❑	❑	Unreadable
Understandable	❑	❑	❑	❑	❑	Unintelligible
Attractive	❑	❑	❑	❑	❑	Unattractive
Excellent	❑	❑	❑	❑	❑	Poor

What information did you expect to find that was omitted?

_____

Is more information needed? ❑ Yes ❑ No. If yes, on what topic?

_____

Did you find factual errors in the documentation? ❑ Yes ❑ No. If yes, please give page number and description of the error.

_____

If the documentation is difficult to understand, please specify page number and problem.

_____

Is the documentation intimidating? ❑ Yes ❑ No.

_____

Are the manuals: ❑ Too long? ❑ Too short? ❑ About the right length?

_____

Other suggestions or comments? (Use back of form if necessary.)

_____

(Additional Comments)

. . . . . Fold along dotted line. . . . . .

KMSYS
WORLDWIDE, INC
P.O. Box 669695
Marietta, GA 30066
U.S.A.

Attn: Technical Documentation Section