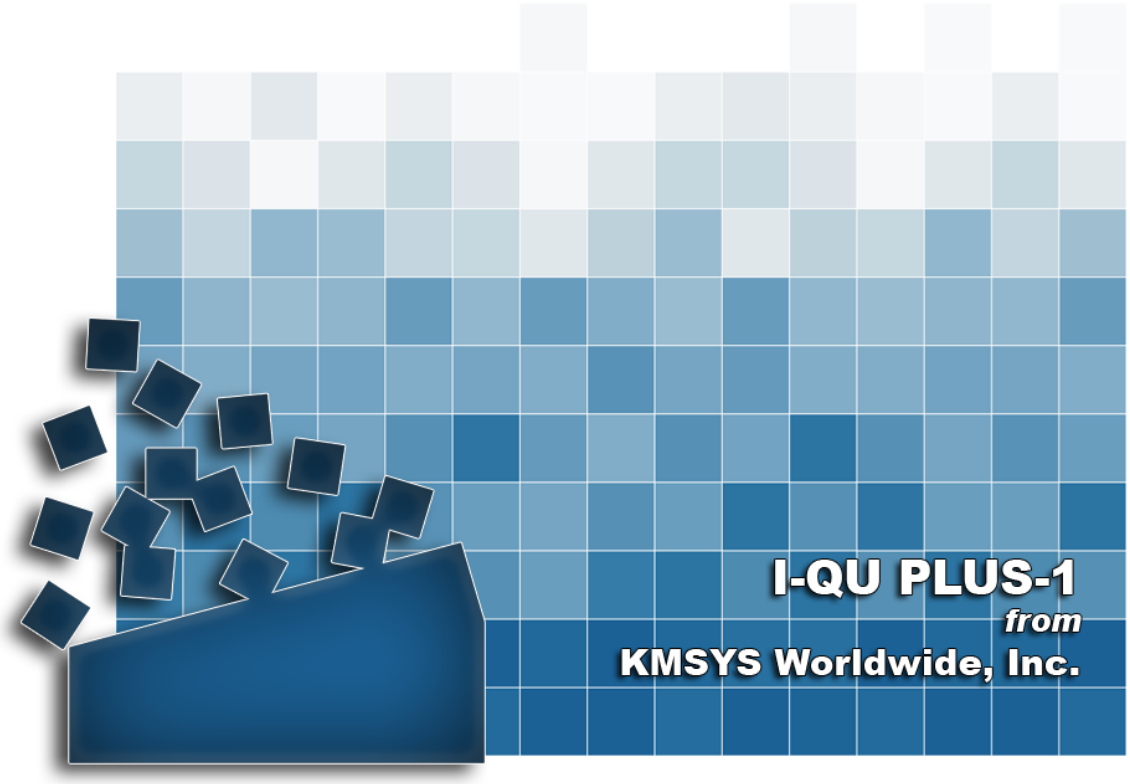

KMSYS
WORLDWIDE, INC

Application Development User Guide

12 March 2012

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

I-QU PLUS-1 Release 11R6, November 1999

Table of Contents

Chapter 1: Introduction

I-QU PLUS-1 is a command language processor that may be used to access and update DMS 2200 hierarchical databases, PCIOS files, TIP/FCSS files, RDMS 2200 relational databases, BIS (formerly, MAPPER) reports via the DTM interface, and DB4 databases. Using the simple but powerful command language in I-QU PLUS-1, the user may interactively peruse a database or file record by record, examine BIS reports line by line or create full function I-QU PLUS-1 programs quickly and easily. Basics of the command language can be mastered in hours, and within just a few days users can make use of the more advanced features.

This manual is a guide to using the I-QU PLUS-1 Processor as an applications development tool in the areas of COBOL program prototyping and debugging, and in using the I-QU PLUS-1 Processor in lieu of other programming languages in the actual development and implementation of application systems.

The following section will introduce the reader to the I-QU PLUS-1 Processor, its basic structure and modes of operations. Then, the reader will be led through the fundamentals of using I-QU PLUS-1. The remainder of the manual will discuss elementary and advanced user techniques.

This manual will assume that the reader has a basic understanding of COBOL programming, PCIOS file types, DMS 2200 database programming techniques, and other file systems utilized within. Consult the appropriate Unisys 2200 reference manuals if more information is needed in these areas.

Throughout the manual, references will be made to the I-QU PLUS-1 Programmer Reference (Programmer Reference) for detailed descriptions of items covered. The I-QU PLUS-1 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

I-QU PLUS-1 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 I-QU PLUS-1 Programmer Reference.

It is also possible to abbreviate DMS 2200 database area, record and set names if they conform to required criteria and I-QU PLUS-1 has been generated to recognize them. Consult the person responsible for the I-QU PLUS-1 installation at your site, or the Q-LINK Installation Guide, for more information.

Chapter 2: Basic Structure

This chapter will introduce the structure of the I-QU PLUS-1 Processor. A basic knowledge of the major components and operating modes of I-QU PLUS-1 will aid in learning how to use all its powerful features, effectively.

2.1 Major Program Components

The I-QU PLUS-1 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 passed to either the COMMAND EXECUTOR for immediate execution (conversational mode), or stored in the object program area for later execution (input mode). 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.

2.2 Modes of Operation

I-QU PLUS-1 commands may be entered in one of two modes: CONVERSATIONAL or INPUT. In CONVERSATIONAL mode, each command is edited and passed to the executor immediately. Command results are displayed to the user. In the case of DML commands, the DMS 2200 status is displayed. For IF commands, the resulting TRUE or FALSE condition will be displayed. When entering commands in INPUT mode, each command is edited immediately, and then stored in the object program area for later execution. When all commands have been entered, the user may enter the RUN directive, which will cause the command executor to resolve all program labels, procedure names, and IF/ENDIF pairs. If the I-QU PLUS-1 Processor is unable to resolve any of these, or an error was detected during input and the error flag has not been CLEARed, the object program will not be executed. Otherwise, the command executor will begin executing the object program. All I-QU PLUS-1 Directives will be processed immediately in either CONVERSATIONAL or INPUT mode.

2.3 Commands vs. Directives

There are two types of control statements in the I-QU PLUS-1 Processor: 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 an I-QU PLUS-1 program. Commands are edited and placed into the internal program area before being executed. In conversational mode, the program area never contains more than one command, which is always executed immediately.

Directives are used to set up and control the I-QU PLUS-1 environment. This includes setting the processing mode to INPUT or CONVERSATIONAL, clearing the error switch, 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 I-QU PLUS-1 program logic; in other words, performing data manipulation, program flow control, logic operations, and input/output. In INPUT mode, commands must always be entered beginning after column 1; however, in CONVERSATIONAL mode, they must always begin in column 1.

2.4 Processor Environment

Let us look at the environment in which each individual execution of the I-QU PLUS-1 processor is operating. There are several files used by each I-QU PLUS-1 processor. Some of these files are shared with other I-QU PLUS-1 users, while others are unique to a particular execution of I-QU PLUS-1. The following is a brief description of the various files used by the I-QU PLUS-1 processor:

- The Primary Data Item Index File is automatically created as a temporary file for each individual user. It is used to hold descriptive information on every database area, record, set, database data name and data item in an invoked subschema. This information is used in the editing of I-QU PLUS-1 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 Object Request File is a program file used to store I-QU PLUS-1 programs in object format. There is one default object file shared by all active users. There may be any number of user assigned object files. The default object request file is named I$QU*I$QUOBJ.
- The Source Library File is a program file used to store commonly used portions of I-QU PLUS-1 program source code. This source code is included in a program via the ADD directive. There is one default source library shared by all active users. There may be any number of user assigned source library files. The default file is named I$QU*I$QULIB.

Each of these files, and their usages, will be discussed in more depth later.

2.5 1 Internal Storage Areas

When using the I-QU PLUS-1 Processor, it is important to understand which internal areas are used for various operations. I-QU PLUS-1 internal storage is organized into four major areas:

- The RECORD DELIVERY AREA (RDA) is used for all data input and output for DMS 2200, PCIOS and TIP/FCSS files, BIS report lines, RDMS 2200 table items, and DB4 buffers.
- 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.

2.5.1 Record Delivery Area (RDA)

The RDA is used for input and output operations for DMS 2200 database records, PCIOS and TIP/FCSS files, BIS reports lines, RDMS 2200 database table items, and DB4 database buffers. The size of this area will vary depending upon I-QU PLUS-1 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, files, etc. 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.

2.5.2 Variable Data Storage Area

The variable data storage area is where all user and reserved variables and literals are allocated by I-QU PLUS-1. Reserved variables are automatically defined and allocated by I-QU PLUS-1 upon initialization. The names and contents of reserved variables will be found in the I-QU PLUS-1 Programmer Reference. Literals are automatically allocated in this area as they are encountered by the I-QU PLUS-1 command editor. The size of this area is specified on an I-QU PLUS-1 generation parameter.

2.5.3 Object Program Area

The Object Program Area is where the internally encoded object program (a single command in conversational mode) is 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 an I-QU PLUS-1 generation parameter, and should be large enough to accommodate from 300 to 600 commands.

2.5.4 Internal Table and Buffer Area

This area contains several tables and buffers used internally by I-QU PLUS-1. Among these are the following:

- File control tables used to maintain the status of each file defined in the I-QU PLUS-1 session;
- The data item index file input buffer;
- The output print buffer, which is the same size as the RDA;
- The sort control area for building sort key parameters and maintaining status of sort usage.

Chapter 3: Using the I-QU PLUS-1 Processor

This chapter, directed to the first-time I-QU PLUS-1 user, will present information on the I-QU PLUS-1 Processor and the use of common commands and directives in CONVERSATIONAL mode. An introduction to I-QU PLUS-1 programming will also be presented.

3.1 Calling the Processor

I-QU PLUS-1 is called as a processor, not by an @XQT statement. The processor call format is as follows:

@processor-name,[options] [password]

The processor-name will vary depending on the installation mode and processor name parameter given when the I-QU PLUS-1 processor was built. In this manual, we will assume that the processor's name is IQU.

There are several options available. These are discussed in the I-QU PLUS-1 Programmer Reference. Generally, no options are needed.

The password is only needed if certain security parameters were specified when I-QU PLUS-1 was generated. Again, check with the person responsible for installation for this information.

Here is a typical example of how the I-QU PLUS-1 processor might be called:

@IQU,C

The C-Option sets the initial mode to CONVERSATIONAL. Upon entry, I-QU PLUS-1 will display a copyright message and the date and time it was generated, followed by the message “Initial Mode is CONVERSATIONAL” or “Initial Mode is INPUT”. If the initial mode is INPUT, enter the directive CONV (this will put I-QU PLUS-1 in conversational mode) before proceeding with the exercises shown in this section. If the initial mode is CONVERSATIONAL, or after entering the CONV directive, the following prompt will be displayed:

▶Command:▶

At this point, the I-QU PLUS-1 Processor has completed internal initialization and is ready to accept user commands and directives in CONVERSATIONAL mode. Remember that in CONVERSATIONAL mode, all commands will be edited and executed immediately.

To exit the I-QU PLUS-1 Processor, enter the EXIT directive at the command prompt.

When in CONVERSATIONAL mode, the prompt “Command:” will be displayed after each command is executed. For example:

```
▶Command:▶DISPLAY 'Hello'    <— User's input.
▶Hello                       <— Result of execution.
▶Command:▶                   <— Conversational prompt.
```

3.2 A Hands-on Introduction

This section will familiarize the new user with some common I-QU PLUS-1 commands and with how they are used in CONVERSATIONAL mode. The commands shown here will require no database or file access, and can be attempted without fear of destroying any existing data.

3.2.1 Displaying or Viewing Data

Let us first look at the commands that will be used to output data to the display screen or print file — DISPLAY and EDIT.

Enter the following:

```
▶Command:▶DISPLAY DATE
```

I-QU PLUS-1 will respond with the current date in the form YYYY/MM/DD. DATE is an I-QU PLUS-1 reserved variable. There are several predefined reserved variables set up automatically when I-QU PLUS-1 is initialized. A complete list will be found in the I-QU PLUS-1 Programmer Reference.

Enter the following sequence of commands:

```
▶Command:▶TIME
▶Command:▶DISPLAY TIME +
▶Command:▶DISPLAY TIME-MSPM
```

After the first command (TIME) in the above sequence, there was no response. The TIME command simply 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 alpha 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 an I-QU PLUS-1 program.

The first DISPLAY entered above did not produce a response because of the plus (+) sign used following the name of the variable. The "+" tells I-QU PLUS-1 to move the data in the variable to the print buffer, but not to print yet, because there is more to come. After the second DISPLAY, both the alpha string and numeric forms of the current time of day will have been moved to the print buffer and displayed. The output line should look something like this:

```
▶09:57:07.226           35827226
```

The DISPLAY command displays data as it is stored. Another command used to display data is EDIT. The EDIT command is used to display numeric data in an edited format which may include comma and decimal placement, zero suppression, etc.

Enter the following:

```
▶Command:▶EDIT TIME-MSPM 'ZZ,ZZZ,ZZZ.999'
```

This command displayed the time of day in milliseconds past midnight edited by the mask furnished in the literal 'ZZZ,ZZZ.999'. This process is similar to using the edit picture clause in COBOL. The output should look like this:

```
▶ 35,827,226.000
```

Note that the placement of the decimal is determined by the number of implied decimal positions in the item being edited. In this example, we know that time is represented in milliseconds past midnight. The variable TIME-MSPM does not have any implied decimal positions; therefore, the value was scaled accordingly. Handling of decimal values in data will be covered in more detail later.

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

```
▶Command:▶EDIT DATE-NUM '9999B99B99B' +
▶Command:▶EDIT J-DAY '9999/ZZ9'
▶2007 02 21 2007/ 52
▶Command:▶DISPLAY 'The current date is' +
▶Command:▶EDIT 21 MONTH 'ZZ' +       <— '21' is a start column position.
▶Command:▶DISPLAY '-' +
▶Command:▶EDIT DAY 'ZZ' +
▶Command:▶DISPLAY '-' +
▶Command:▶EDIT YEAR 'ZZZZ'
▶The current date is  2-21-2007
```

While not normally done in CONVERSATIONAL mode, this example illustrates how output lines can be constructed using the DISPLAY and EDIT commands. Both the DISPLAY and EDIT commands have sister commands called TRIMDISP and TRIMEDIT. TRIMDISP and TRIMEDIT functions the same as

their relatives 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 with TRIMDISP and TRIMEDIT commands.

The TRIMed output lines will look like this:

```
▶2007 02 212007/ 52
▶The current date is 2-21-2007
```

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

Another way to view data is with the DUMP command. The DUMP command is used to display data in its octal representation. When DUMPing variables, it is important to remember that in addition to the data stored in the variable, certain control information will also be shown. The following is an example of DUMPing the reserved variable DATE:

```
▶Command:▶DUMP DATE                    <— DUMP the contents of DATE
```

You will get a response similar to this:

```
▶0053    040000000000 000000000012 062060060067 057060062057  ???????2007/02/
▶0057    062061040040                                        21
```

The first two octal words contain the character length (in octal, 12 is 10 decimal characters) of the variable. The third, fourth and fifth words are the actual contents of DATE. The portion of the display to the far right is the ASCII representation of the four octal words on the left. The question marks (?) in the ASCII portion of the display are used in place of unprintable ASCII characters (i.e., those outside the range 32 through 127).

3.2.2 Manipulating Data

So far we have looked at ways to display data. 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 I-QU PLUS-1 user. The command key word SET may be omitted.

Look over the following sequence, and then try it yourself.

```
▶Command:▶SET X = 50
▶Command:▶Y = X * 2
▶Command:▶Z = X - Y
▶Command:▶DISPLAY X Y Z
▶                 50                100               -50
```

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 display numeric and alpha string variables.

SET may also be used to manipulate alpha string data. We will use the reserved variables C-O-R and G-AREA-NAME. These two variables are used by I-QU PLUS-1 when executing DMS 2200 DML commands, but are available to the user when DML commands are not being used.

Enter the following:

```
▶Command:▶C-O-R = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890'
▶Command:▶G-AREA-NAME = C-O-R
▶Command:▶DISPLAY C-O-R
▶ABCDEFGHIJKLMNOPQRSTUVWXYZ1234
▶Command:▶DISPLAY G-AREA-NAME
▶ABCDEFGHIJKL
▶Command:▶C-O-T = G-AREA-NAME
▶Command:▶DISPLAY C-O-T
▶ABCDEFGHIJKL
```

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.

3.2.3 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 I-QU PLUS-1 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 I-QU PLUS-1 commands, they are always preceded by the key word RDA. Consider the following command sequence:

```
▶Command:▶SET RDA (5,3) = 'ABC'
▶Command:▶DISPLAY RDA (5,3)
▶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 sequence will illustrate some more uses of DIRECT RDA references:

```
▶Command:▶RDA (1,10) = 'ABCDEFGHIJ'
▶Command:▶DISPLAY RDA (2,2)
▶BC
▶Command:▶RDA (11,5) = RDA (3,5)
▶Command:▶DISPLAY RDA (1,15)
▶ABCDEFGHIJCDEFG
▶Command:▶X = RDA (1,1) COMP + RDA (2,1) COMP
▶Command:▶TRIMDISP X
▶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.

The DUMP command shown earlier may also be used to show octal representations of data in the RDA. Unlike the DUMP of a variable, which will display control information along with the variable contents, only data is shown when DUMPing the RDA. The area of the RDA to be DUMPed may be specified as absolute word positions, by an invoked database record name, or as a defined file name.

Here is an example of using the DUMP command to display the contents of the first two words of the RDA after executing the previous sequence of commands:

```
▶Command:▶DUMP 1,2                  <— Dump two words starting at word one.
▶101102103104 105106107110 ABCDEFGH
```

3.2.4 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 alpha 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 alpha string variable may be any length.

Here are some examples of defining variables:

```
▶Command:▶DEFINE N VAR1
▶Command:▶DEFINE N VAR2 123
▶Command:▶DEFINE A VAR3 'ABCD'
▶Command:▶DEFINE A VAR4 8
▶Command:▶DEFINE N VAR5 .00
▶Command:▶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 alpha string variables as indicated by the “A”. VAR3 will be given the initial value of “ABCD”, and its length will be four 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:

```
▶Command:▶DEFINE RDA FLD1 (1,5)
▶Command:▶DEFINE RDA FLD2 (6,3) SN9
▶Command:▶DEFINE RDA FLD3 (9,4) COMP
▶Command:▶DEFINE RDA FLD3R (9,4)
▶Command:▶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 I-QU PLUS-1 the field has four decimal positions implied; however, no actual value is assigned to the field.

Enter both the variable definitions and RDA definitions just presented and then try this sequence of commands:

```
▶Command:▶RDA FLD1 = VAR3
▶Command:▶RDA FLD2 = VAR2
▶Command:▶RDA FLD3R = VAR3
▶Command:▶DISPLAY 'FLD1 = ' +
▶Command:▶DISPLAY RDA FLD1 +
▶Command:▶DISPLAY ' FLD2 = ' +
▶Command:▶TRIMEDIT RDA FLD2 'ZZZZ' +
▶Command:▶DISPLAY ' FLD3 = ' +
▶Command:▶EDIT RDA FLD3 '9999999999+'
```

The information displayed should appear as follows:

```
▶FLD1 = ABCD FLD2 = 123 FLD3 = 8741488196+
```

Note: 8741488196 is the decimal equivalent of octal 101102103104 or ABCD.

Continue with the following set of commands:

```
▶Command:▶VAR5 = -77.8888
▶Command:▶RDA FLD4 = VAR5
▶Command:▶DISPLAY 'VAR5 = ' +
▶Command:▶TRIMEDIT VAR5 '-ZZZ.99' +
▶Command:▶DISPLAY ' FLD4 = ' +
▶Command:▶TRIMEDIT RDA FLD4 '-zzz.9999'
```

The sequence above should produce:

```
▶VAR5 = -77.89 FLD4 = -77.8900
```

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 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.

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

The asterisk is used to inform I-QU PLUS-1 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 is in fact in the same positions as FIELD-B.

At this point, you should have a good idea how the I-QU PLUS-1 processor operates in CONVERSATIONAL mode. The next step will be to try some short I-QU PLUS-1 programs.

3.3 An Introduction to I-QU PLUS-1 Programming

To get some hands-on experience with I-QU PLUS-1 programming, first call the I-QU PLUS-1 processor. If the Initial Mode is CONVERSATIONAL, get into INPUT mode by entering INPUT at the “Command:” prompt. I-QU PLUS-1 will respond with “Switched to INPUT Mode”.

The I-QU PLUS-1 Processor is now ready to accept, edit and store commands INPUT by the user. Remember that commands will not be executed as they are entered. Also remember that directives will be processed immediately (i.e., DEFINEs).

There is a difference in how commands are entered in INPUT mode. Commands cannot be entered in position one. The only entries that may be input starting in position one are directives (always entered in position one) and program or procedure labels. We will cover labels later.

Enter the following program:

```
    X = 1
    Y = 2
    DISPLAY X Y
    Z = X + Y
    DISPLAY 'The value of Z is ' +
    DISPLAY Z
    STOP
RUN
```

There are several points to be mentioned here. First, notice that the command was echoed and preceded by two numbers. The first number is the sequential number of the input statement. The second number is the program counter (PC) and indicates the command position in the internal object program table. The PC value will be displayed when certain errors are encountered and will be useful in debugging. Second, none of the SET or DISPLAY commands were executed until the RUN directive was entered. The STOP command caused the execution of commands to stop and the processor to be returned to CONVERSATIONAL mode.

Let us try a slightly more complicated program. This time the program will include some user defined variables and some logical commands.

Enter the following program:

```
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 'COUNT = ' +
  TRIMEDIT CNT 'ZZ9'
  ENDPROC
RUN
2                                   <— Data to be ACCEPTed
3
@EOF
```

It may be difficult to type this many command lines without making mistakes. You may wish to create a symbolic element (use @ED, @CTS, etc.) containing the I-QU PLUS-1 program. Then call the I-QU PLUS-1 Processor, get into INPUT mode, and @ADD the program.

Note the Program Counter is not incremented by the DEFINEs. This is because DEFINEs are directives and are not part of the object program. The ENDIFs do not occupy space in the program either. They only set control information in the program table.

The above example will produce the following output:

```
*** STARTING ***
▶COUNT = 1
▶COUNT = 2
▶COUNT = 1
▶COUNT = 2
▶COUNT = 3
▶*** END OF RUN ***
```

This program illustrates the use of comment lines. A comment is any text following a period-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 all programs; however, if the logical end of the program is not the last command we must tell I-QU PLUS-1 when to STOP (that is why we have not needed a STOP command before now).

The last program also demonstrates several commands not discussed earlier, such as the GO, IF/ELSE/ENDIF, DO/ENDDO, ACCEPT, and the use of an I-QU PLUS-1 defined procedure. Let us take some time now to discuss these commands (refer to the I-QU PLUS-1 Programmer Reference for detailed information).

The ACCEPT Command is used to receive input from the runstream (the terminal when running in demand mode), or from the system console. This is similar to the COBOL ACCEPT verb with the exception that the I-QU PLUS-1 command may include a prompt.

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. An I-QU PLUS-1 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 program example illustrates a typical DO WHILE using an I-QU PLUS-1 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 will work, 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 I-QU PLUS-1 program. In this case, 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. I-QU PLUS-1 uses the following 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 an I-QU PLUS-1 program is an example of a more complex IF structure:

```
DEFINE A INPUT 4 .  4 Chars.  for input
...
  ACCEPT INPUT AT END QUIT .  Get data from user PROGRAM
  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, I-QU PLUS-1 also supports wildcard characters in comparisons using alpha string data. For this purpose, the WILDCARD command is provided. The WILDCARD command is used to establish a wildcard character. When 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 '?'                    .  Make '?' wildcard
IF INPUT = 'A?CD'
.  .  .
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 4: Database and PCIOS File Handling

By now you should be fairly comfortable with both the CONVERSATIONAL and INPUT mode operation of the I-QU PLUS-1 Processor. In this chapter, we will begin to deal with database access, starting with DMS 2200 databases, then PCIOS files, and finally, the SORT subroutine interface.

4.1 Accessing the DMS 2200 Database

The I-QU PLUS-1 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.

4.1.1 Invoking a Subschema

Once the I-QU PLUS-1 Processor has been called, the first step that is required to access any DMS 2200 database is to INVOKE a subschema. I-QU PLUS-1 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 I-QU PLUS-1 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 I-QU PLUS-1 configuration. 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 I-QU PLUS-1. It will access both the object subschema and schema in the schema file, and initialize D$WORK and S$WORK in I-QU PLUS-1. 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 I-QU PLUS-1 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.

4.1.2 I-QU PLUS-1 DML Commands

The I-QU PLUS-1 DML commands required 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 I-QU PLUS-1 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
```

I-QU PLUS-1 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 I-QU PLUS-1 Programmer Reference.

DML commands entered in CONVERSATIONAL mode always respond with the ERROR-STATUS, ERROR-NUM, and the CURRENT PAGE and RECORD number. In this manner, the user is always immediately aware of the result of each DML command.

The following DML commands are illustrated with the conversational responses from I-QU PLUS-1. Try the same sequence on your system, substituting area, record and set names for those used in your subschema.

```
▶Command:▶INVOKE SUB-SALES IN MARKETING
▶ ** Invoke Complete for MT1 **
▶Command:▶IMPART
▶
▶Error Status = 000000 Error-Num = 0000 Current page/rec 000000/00000
▶Command:▶OPEN SALESMAN
▶
▶Error Status = 000000 Error-Num = 0000 Current page/rec 000000/00000
▶Command:▶FETCH3 FIRST SALESMAN AREA
▶
▶Error Status = 000000 Error-Num = 0000 Current page/rec 000001/00001
▶Command:▶DISPLAY C-O-R
▶SALESMAN-MASTER-REC                <— The current record name
▶Command:▶DISPLAY RDA SLSM-LAST-NAME
▶SMITH
▶Command:▶DEPART
▶
▶Error Status = 000000 Error-Num = 0000 Current page/rec 000001/00001
▶Command:▶DISPLAY IMPART-DEPART
▶ 2                                 <— The Impart-Depart Switch
```

Using the above sequence of commands, substituting area and record names from your schema, you should have FETCHed the first record in one of your areas. You have probably noticed that the DISPLAY of the reserved variable C-O-R contains the name of the current record type. The reserved variable C-O-R is set automatically by all DML commands that can change record currency. There are also reserved variables set up to hold the impart-depart status, error-num, current page number, record number, area key, etc.

The next DISPLAY command 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 more detailed information on the QINDEX Processor.

Try fetching a record whose location mode is CALC. The sequence of commands required to fetch a CALC record would look like this (shown here without I-QU PLUS-1 automatic responses):

```
IMPART
OPEN SALESMAN
DBDN SALES-ANAME = 'SALESMAN'          <— Database Dataname for CALC
RDA SLSM-KEY = '10023'                 <— Set key value in the record
FETCH5 SALESMAN-REC                    <— Fetch the record
DISPLAY RDA SLSM-LAST-NAME             <— Display a field
```

```
FREE                                    <— Release Currency Lock.
```

The IMPART and OPEN commands above would not have been needed had we not DEPARTed in the previous example. Notice that when using I-QU PLUS-1, 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 I-QU PLUS-1 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 ERROR-STATUS and ERROR-NUM values would have been immediately known to the user by the automatic display of the status line. The DML FREE command is not necessary, but is a good habit to get into when working with an on-line system. It will allow other users to access the record while you are looking at it in the RDA. If you plan to MODIFY the record, do not FREE it.

Let us assume the CALC record fetched in the above example is the owner of a set. In the following examples, some variations of VIA SET access will be shown.

```
FETCH3 NEXT SLSM-CUST SET               <— Next record of set
F3 L SLSM-CUST S                        <— Last of set
F3 O SLSM-CUST S                        <— Owner of set
```

In the next sequence, the user would be looking for all New York customers for the SALESMAN fetched earlier.

```
RDA CUS-ST = 'NY'                       <— Set a key in the record
F6 CUSTOMER-REC SLSM-CUST USING CUS-ST <— Fetch the 1st of set with matching key
D RDA CUS-NAME                          <— Display field
F7 CUSTOMER-REC SLSM-CUST USING CUS-ST <— Fetch next of set with DUP key
D RDA CUS-NAME                          <— Display a field
```

So far, we have fetched records and displayed data fields. MODIFYing and STORing records is just as easy. The following sequence illustrates how a single record may be fetched, corrected, and modified. In this case, the I-QU PLUS-1 automatic responses will be shown.

```
▶Command:▶RDA SLSM-KEY = '12300'
▶Command:▶F5 SALESMAN-REC
▶
▶Error Status = 000313 Error-Num = 0013 Current page/rec 000000/00000
```

The above fetch received an error because the key was entered incorrectly to show how DML errors are handled in CONVERSATIONAL mode.

The key can be corrected and processing continued in CONVERSATIONAL mode as shown below:

```
▶Command:▶SET RDA SLSM-KEY = '12030'   <— Correct key value
▶Command:▶F5 SALESMAN-REC              <— Fetch the record
▶
▶Error Status = 000000 Error-Num = 0000 Current page/rec 000120/00003
▶Command:▶D RDA SLSM-LAST-NAME         <— Field to correct
▶SMTIH
▶Command:▶RDA SLSM-LAST-NAME = 'SMITH' <— Enter corr.  spelling
▶Command:▶MODIFY SALESMAN-REC          <— Modify the record
▶
▶Error Status = 000000 Error-Num = 0000 Current page/rec 000120/00003
▶Command:▶FREE                         <— Release locks
▶
▶Error Status = 000000 Error-Num = 0000 Current page/rec 000120/00003
```

We can store a record by continuing with the following sequence:

```
RDA SALESMAN-REC = $SPACES              <— Set rec to spaces
RDA SLSM-KEY = '12345'                  <— Set key field
RDA SLSM-LAST-NAME = 'JONES'            <— Set data fields...
RDA SLSM-FIRST-NAME = 'SAM'
RDA SLSM-QUOTA = 9000
```

```
RDA SLSM-REGION = 22
STORE SALESMAN-REC                    <— Store new record
FREE                                  <— Release Locks
```

The FREE command makes all changes to the database permanent, as well as releasing locks to allow others to access the modified pages. It may be more desirable to make several modifications before issuing a FREE or DEPART. In this way a DEPART with the ROLLBACK option may be used to undo changes if necessary.

There are many more DML commands and options available in the I-QU PLUS-1 Processor. Please refer to the I-QU PLUS-1 Programmer Reference for more detailed information.

4.1.3 DML Commands in an I-QU PLUS-1 Program

The previous section demonstrated the use of several DMS 2200 DML commands used in CONVERSATIONAL mode. The same commands may be entered in INPUT mode to be executed as an I-QU PLUS-1 program. DML commands executing in an I-QU PLUS-1 program will not echo the status each time they are executed. Latter sections of the manual will contain many examples of DML commands in I-QU PLUS-1 programs; however, some basic examples will be shown here.

For this example, create an element containing the program shown (substitute names used in your schema). Notice that only directives (INVOKE, RUN, etc.) and labels begin in column one.

```
INVOKE SUB-SALES IN MARKETING
  IMPART
  OPEN ALL
  FETCH4 FIRST SALESMAN-REC SALESMAN AREA
  GO CHECK
LOOP
  FETCH4 NEXT SALESMAN-REC SALESMAN AREA
  IF ERROR-NUM = 7 .  End of area?
    DISPLAY "Found and deleted " +
    TRIMEDIT X 'Z,ZZ9' +
    DISPLAY ' salesman that do not have customers.'
    DEPART
    STOP EXIT
  ENDIF
CHECK .  If salesman is not an owner of set, delete it.
  IF NOT OWNER SLSM-CUST
    SET X = X + 1
    DISPLAY 'Deleted salesman number ' +
    DISPLAY RDA SLSM-KEY
    DELETE SALESMAN-REC
  ENDIF
  GO LOOP
RUN
```

For purposes of demonstration, you would probably omit the DELETE SALESMAN-REC command. This is a typical I-QU PLUS-1 database maintenance program. It will check all SALESMAN-REC records in the area for participation as an owner of the SLSM-CUST set. Any SALESMAN-REC records that do not own any CUSTOMER-REC records via the SLSM-CUST set will be counted and deleted.

This example program may be run by calling the I-QU PLUS-1 Processor, then doing an @ADD of the element containing the program. As the program is read by I-QU PLUS-1, it will be listed, showing the program counter (PC). Since the @ADDed element contains the RUN directive, execution will begin immediately, unless there is an error in the program.

When the RUN directive is encountered, the following output would be expected:

```
>>> End of Command Input <<<
Deleted salesman number 12345
Deleted salesman number 23940
Deleted salesman number 40282
Found and deleted 3 salesmen that do not have customers
```

Because the EXIT option was used on the STOP command, I-QU PLUS-1 was automatically exited upon completion of processing.

4.1.4 DMS 2200 Error Handling in I-QU PLUS-1 Programs

I-QU PLUS-1 handles DMS 2200 errors a bit differently than COBOL/DML. There is not an ON ERROR or AT END clause used on I-QU PLUS-1 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 an I-QU PLUS-1 program. A table of non-fatal errors is maintained in I-QU PLUS-1. If an ERROR-NUM value is received and is not present in the non-fatal table, I-QU PLUS-1 will perform a runtime controlled abort (INPUT mode only) forcing DMS 2200 to rollback any changes made since the beginning of the session or last FREE command. The non-fatal table initially contains the ERROR-NUM values 0006, 0007 and 0013. Any other error will be considered fatal. The SET NON-FATAL command has been provided to allow the I-QU PLUS-1 programmer to add or delete ERROR-NUM values in the non-fatal table as necessary thereby minimizing the amount of DMS 2200 error handling required in an I-QU PLUS-1 program.

An important point to remember is that while the DML DEPART command is provided in I-QU PLUS-1, it is not required. The I-QU PLUS-1 processor will automatically perform a DEPART upon exiting. If a DMS error occurred during execution, a DEPART with ROLLBACK will be performed.

The following is a short example of using 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.

```
...
 DBDN CTL-ANAME = 'CONTROL'
 RDA CONTROL-KEY = 'PART'
 NON-FATAL 13 OFF                  <- Make 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, I-QU PLUS-1 will terminate on such a condition.

4.2 PCIOS File Handling

Most types of Processor Common Input/Output System (PCIOS) files can be handled by the I-QU PLUS-1 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 it. 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 I-QU PLUS-1. 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 I-QU PLUS-1 processor can process several PCIOS files in a session, depending on the limit set in the configuration when the product was built. PCIOS files can be opened, closed and reopened at will, and can be handled concurrently with other OS 2200 data access structures.

4.2.1 PCIOS File Definition

Before any PCIOS file access can begin, the file must be defined to I-QU PLUS-1. To define a PCIOS file, another form of the DEFINE directive is used. The DEFINE F (File) gives the I-QU PLUS-1 Processor information required to interface with the PCIOS routines. Refer to the I-QU PLUS-1 Programmer Reference for detailed information on how to define PCIOS files to I-QU PLUS-1. Some examples of PCIOS file definition will be shown in the following sample programs.

4.2.2 Reading and Writing PCIOS Files

To get some hands-on experience with PCIOS files, Let us define and create a sequential file. The following will assume CONVERSATIONAL mode:

```
DEFINE F TEST SEQ 80,20        <— Seq file with 80 character records, blocked 20.
CSF X '@ASG,T TEST.'           <— Use CSF to assign file.
OPEN TEST OUTPUT SEQ           <— OPEN with usage & access
RDA (1,80) = 'REC 1'           <— Put some data in the RDA
WRITE TEST                     <— Write the record
RDA (1,80) = 'REC 2'           <— Repeat ...
WRITE TEST
...
CLOSE TEST                     <— Close the file
```

That is all it takes to create a sequential file. Continue with the following to read it:

```
OPEN TEST INPUT SEQ            <— Note the usage mode INPUT
READ TEST                      <— Read one record
DISPLAY RDA (1,80)             <— Display its contents
.
.  .  .  repeat the READ and DISPLAY until end-of-file.
```

At end-of-file, the message “END OF FILE ON TEST” will be displayed. To begin reading the file again, CLOSE it, then OPEN it for INPUT.

An Indexed Sequential file can be handled almost as easily. Let us assume that a COBOL program was used to create an ISAM file containing records 120 characters in length with a BLOCK CONTAINS 20 records. The record key is in positions 1 thru 9 of the record.

The following sequence would be used to DEFINE the indexed sequential file to I-QU PLUS-1:

```
DEFINE F XYZFILE INDEXED 120 20 (1,9)
```

The record key is specified by (1,9); the key begins in position one and is nine characters long. The record key may also be a named RDA reference either if the field has been defined to I-QU PLUS-1 by a DEFINE RDA directive, or if the field is included in the data item index file.

If we wanted to locate and examine one record in this file, the following commands might be used:

```
OPEN XYZFILE INPUT DYNAMIC <— Allow both sequential and random processing.
RDA (1,9) = 'ABCDEFGHI' <— Set record key value
READ XYZFILE <— Read the record
```

If a record with the desired key does not exist, I-QU PLUS-1 will display “INVALID KEY ON FILE XYZFILE”; otherwise, I-QU PLUS-1 will only display the “Command:” prompt.

4.2.3 PCIOS Files in an I-QU PLUS-1 Program

All the PCIOS file examples shown thus far have been in CONVERSATIONAL mode. In CONVERSATIONAL mode, PCIOS end-of-file or invalid-key conditions are displayed to the user immediately. When using PCIOS file commands in an I-QU PLUS-1 program, however, some additional command syntax is required to handle these conditions.

The following I-QU PLUS-1 program uses the indexed sequential file XYZFILE from the previous example to demonstrate the use of both the AT END and INVALID KEY clauses. The object of the program will be to list the keys of all records in the range of 100000000 thru 199999999.

```
DEFINE F XYZFILE INDEXED 120,20 (1,9)
.
  OPEN XYZFILE INPUT DYNAMIC
  RDA (1,9) = '100000000'               .  Set key value
  START XYZFILE INVALID KEY NOFIND EQGT .  Pos. to rec.
  DO                                    .  Start DO
    READNEXT XYZFILE AT END BREAK
    IF RDA (1,9) > '199999999'
      BREAK                             .  End of range
    ENDIF
    X = X + 1                           .  Count record
    DISPLAY 'Key value = ' +
```

```
    DISPLAY RDA (1,9)
  ENDDO
FINAL
  DISPLAY 'End of search - found ' +
  TRIMEDIT X 'ZZZ,ZZ9' +
  DISPLAY ' records.'
  STOP EXIT
NOFIND
  DISPLAY 'No record at or above 100000000 found.'
  STOP EXIT
RUN
```

The output of the above 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.
```

4.2.4 PCIOS Variable Length Record Considerations

I-QU PLUS-1 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. Basically 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 I-QU PLUS-1. 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 I-QU PLUS-1, 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 I-QU PLUS-1 program, the commands needed to WRITE this record might look like this:

```
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.
...
```

4.3 Using the Sort Interface

The I-QU PLUS-1 Processor provides an easy-to-use interface to the system SORT subroutine. This interface is only available in I-QU PLUS-1 programs (INPUT mode). 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 an I-QU PLUS-1 program as long as the following sequence of events is used:

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

I-QU PLUS-1 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 an I-QU PLUS-1 program exists, sorting large amounts of data will have an impact on other users. For larger sorts, the program should pre-assign sort files XA, XB and XC. The I-QU PLUS-1 processor uses the standard 2200 Sort package for processing the SORT commands.

The following is a simple example of an I-QU PLUS-1 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
  .  *** Release database records to the sort
  .  *** routine.....
  DO UNTIL ERROR-NUM = 7
  IF SLSM-KEY = '2??1'             .  Wildcard compare
    RELEASE SALESMAN-REC           .  Release to SORT
  ENDIF
  F4 N SALESMAN-REC SALESMAN AREA
  ENDDO                            .  Repeat
  DEPART
  PCONTROL BRKPT                   .  Alt print
  PCONTROL OPTION 'H,,1,SALESMAN LIST'
  .  *** Return sorted records and list in reply......
  DO                               .  Do output proc
    RETURN AT END BREAK            .  Return a rec
    DISPLAY RDA SLSM-KEY +         .  Display data
    DISPLAY 6 RDA SLSM-FIRST-NAME +
    DISPLAY 27 RDA SLSM-LAST-NAME +
    EDIT 50 RDA SLSM-TOT-SALES 'Z,ZZZ.99'
  ENDDO
  STOP EXIT
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.

4.4 Accessing RDMS 2200 Tables

RDMS 2200 provides yet another file structure for maintaining data. I-QU PLUS-1 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 report.

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, $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 I-QU PLUS-1 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. Also, since varibles ($P1 and $P2) are used on the DECLARE CURSOR, an OPEN cursor command must precede the FETCH NEXT.

4.4.1 RDMS vs. RDMS+ Command

The I-QU PLUS-1 RDMS+ command can be used to continue a I-QU PLUS-1 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 I-QU PLUS-1 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 5: Compiling I-QU PLUS-1 Programs

All the I-QU PLUS-1 programs examples shown up to this point have used the source form which means that the source code of each program was implicitly compiled when the RUN directive was first issued. 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 takes place. To eliminate this extra processing, I-QU PLUS-1 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 I-QU PLUS-1 program to save the entire object program 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 I-QU PLUS-1 object program file I$QU*I$QUOBJ (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 program 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 programs is the most efficient method of using I-QU PLUS-1 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 I-QU PLUS-1 processor and, where DMS 2200 is involved, changes to subschemas. Like COBOL programs, I-QU PLUS-1 object programs must be recompiled if the INVOKEd subschema is changed. Recompiling can be accomplished quickly, since compilation of even large I-QU PLUS-1 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 I-QU PLUS-1 program and compile it.

In this program, we will retrieve a single customer master record from a DMS 2200 database and print some basic information.

```
INVOKE SUBCUST IN COMPSCHEMA
DEFINE N CUSTNO
DEFINE A ANSWER ' '
  IMPART
  OPEN CUST-AREA
  SET DBDN CUST-ANAME = 'CUST-AREA'
BEGIN
  ACCEPT CUSTNO "Enter Customer Number:"
  CUSTOMER-NUMBER = CUSTNO .  Put key in record
  FETCH5 CUST-MASTER-REC
```

```
  IF ERROR-NUM NOT = 0
    DISPLAY "CAN'T FIND CUSTOMER MASTER RECORD."
  ELSE
    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
  ENDIF
  ANSWER = $SPACES
  DO WHILE ANSWER <> 'Y'
      AND ANSWER <> 'N'
    ACCEPT ANSWER 'Do you want to do another (Y/N)?'
    SHIFT ANSWER TO UPPER
  ENDDO
  IF ANSWER = 'Y'
    GO BEGIN
  ENDIF
  DEPART
  STOP EXIT
SAVE CUSTADDR
```

This program is a good example of how quickly a simple query transaction can be developed using I-QU PLUS-1. Notice that in this example the RUN directive has been replaced by a SAVE directive. We are telling I-QU PLUS-1 to save the entire environment of this program as an object element named CUSTADDR into the default object program file (I$QU*I$QUOBJ). Try entering a similar program in I-QU PLUS-1. If no errors are detected by I-QU PLUS-1, the object program will be saved. The program will NOT be executed.

To run the saved CUSTADDR program, we need to enter the following:

```
▶RUN CUSTADDR
▶Enter Customer Number: ▶102039928
```

The object program is the form of program that will actually be used in production.

When developing a new object program, it is recommended that you use your own program file for saving programs until you have completed testing.

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 6: Advanced Features

In this section, we will discuss some of the I-QU PLUS-1 features that are more advanced. These will include advanced Record Delivery Area (RDA) referencing techniques, methods of controlling print output and output formatting for reports and BIS. Many of the techniques described in this section will be used in the "How to Do It with I-QU PLUS-1" section of this manual.

6.1 Advanced RDA Referencing

Because of the way in which the I-QU PLUS-1 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.

6.1.1 RDA Field Name Reference

As seen in some of our earlier examples, data elements within the RDA may be referenced by name. When I-QU PLUS-1 invokes a DMS subschema, it automatically builds an index file containing the schema description of every data field in records included in the subschema. This file is referred to as the primary data item index. In addition to the primary data item index, I-QU PLUS-1 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 (FD) source images into an I-QU PLUS-1 data item index file. This file is then invoked by 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, PART-PRICE. If we were to use the RDA reference 'RDA PART-NUM', I-QU PLUS-1 would automatically generate the reference for the PART-NUM field of the record that appears first in the subschema definition. I-QU PLUS-1 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 qualification will generate the correct internal RDA reference. Qualification of items by record name also serves a second purpose involving the use of alternate record areas, which will be discussed later.

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 I-QU PLUS-1 session by using the DEFINE RDA directive. Additionally, record qualification may not be used in conjunction with a direct RDA reference.

6.1.2 RDA Fields

An earlier example demonstrated how the same character positions may be referenced as either alpha display or computational. In the next example, we will set up a sequence of terminal control characters which, when displayed, will clear the terminal screen. Since these control characters cannot be entered

directly from the keyboard, we will set them up in the RDA as a four-byte (one-word) computational field, then display them as an alpha display field.

Try the following:

```
RDA (1,4) UB9 = 3650369101
DISPLAY RDA (1,4)
```

3650369101 is the decimal representation of the ASCII control sequence: ESC,"e", ESC, "M". The DISPLAY of those characters will cause the screen to be cleared. Clearing the terminal screen may also be accomplished by executing the CLEARSCREEN command.

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 PART-DESC (*PART-NUMBER,22)
```

We may reference the entire field by its schema-defined name PART-NUMBER, or by any of the three subfields defined above. A direct RDA reference can additionally be used to reference any other part of the field. The last definition above specifies that PART-DESC is a 22-character field starting immediately after PART-NUMBER.

Here are some examples referencing the PART-NUMBER:

```
DISPLAY RDA PART-NUMBER             <— Display whole field.
DISPLAY RDA PART-CAT                <— Display pos.  5 & 6.
DISPLAY RDA (1,1)                   <— Display the 1st char.
```

6.1.3 RDA Indexing

RDA Indexing is a powerful feature that allows the user great flexibility in manipulating RDA data. For a preferred method of handling data elements with OCCURS clauses, such as table items, see "RDA Subscripting."

Let us examine the use of RDA indexing as one method of 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. RDA indexing is better explained by example. Assume we are working with the following database record definition from the schema:

```
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 I-QU PLUS-1, the primary data item index will contain the implicit RDA references for each field 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
```

These implicit references are what I-QU PLUS-1 will actually use when you reference a data field name that has not been locally (explicitly) defined by a DEFINE RDA. Notice that there is no indication that any fields are part of an OCCURS clause, and that I-QU PLUS-1 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.
```

Where SUB1 contains 1 for the first occurrence of PH-EFF-DATE, 2 for the second occurrence, and so on.

In I-QU PLUS-1, RDA indexing works a bit differently:

1. Any field may be indexed, not just those within an OCCURS group.
2. 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 an I-QU PLUS-1 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 is part of an I-QU PLUS-1 program used to list price history information using the above record description. This I-QU PLUS-1 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 procedure 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
```

6.1.4 RDA Subscripting

While the RDA indexing of I-QU PLUS-1 is very powerful, it can be quite complicated to use, especially when one is accustomed to COBOL programming. RDA subscripting in I-QU PLUS-1 works more closely to COBOL subscripting than does the indexing feature in I-QU PLUS-1. 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 I-QU PLUS-1 request using RDA subscripting as follows:

```
INVOKE .....
```

```
DEFINE SUB PPSUB PH-PREV-PRICES    .  Subscript variable
  ...
  << Database access routines, etc.  here >>
  ...
PRINT-HIST PROCEDURE
  PPSUB = 1                          <— Init subscript.
  DISPLAY 'PRICE HISTORY FOR PART NUMBER: ' +
  DISPLAY PH-PART-NUM
  DO WHILE PPSUB <= RDA PH-NO-PRICES
    EDIT 11 RDA PH-EFF-DATE :PPSUB '99/99/99' +
    EDIT 20 RDA PH-PRV-PRICE :PPSUB 'Z,ZZZ,ZZZ.999'
    PPSUB = PPSUB + 1                <— Increment subscript
  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-dimensional 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 fourth 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 I-QU PLUS-1, there is no finite limit to the number of dimensions that can be handled.

6.1.5 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. This gives the user the ability to change an RDA reference dynamically within an I-QU PLUS-1 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 I-QU PLUS-1 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 sequence will demonstrate the use of variable RDA referencing. Try it on your system:

```
▶Command:▶RDA (1,10) = 'ABCDEFGHIJ'  <— Put a string in RDA.
▶Command:▶S$=2                       <— Set variable start pos.
▶Command:▶L$=8                       <— Set variable length.
```

```
▶Command:▶DISPLAY RDA (0,0)          <— Using variable RDA reference.
▶BCDEFGHI
▶Command:▶SET S$ = 5                 <— Change variable start.
▶Command:▶DISPLAY RDA (0,2)          <— Display using only variable start.
▶EF
▶Command:▶L$=1                       <— Change variable length.
▶Command:▶TRIMDISP RDA (1,0) COMP    <— Display numeric value of "A".
▶65
```

Variable RDA referencing may be used in combination with RDA indexing. This means that in addition to varying the RDA reference start position and length as shown here, you may further modify the resulting reference by adding an RDA index. Such an RDA reference may look like this:

```
RDA (1,0) :X
```

In this case, the start position is modified by the index, X, and the length is obtained from L$.

The following is a more practical example of variable RDA referencing involving the SCAN command in a portion of an I-QU PLUS-1 program. The object of the procedure 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 I-QU PLUS-1 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 var after pos of [
  L$ = 81 - S$                       <— Var len for rest of scan
  X = 0                              <— Reset for second scan
  SCAN RDA (0,0) ']' X               <— Scan for ending 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 procedure:

```
Input 1: ABC[DEFG]HIJKLMNOP
Output: FOUND STRING = DEFG

Input 2: ABCDEFGHIJ[TEXT TO LOCATE]KLMNOPQRSTUVWXYZ
Output: FOUND STRING = TEXT TO LOCATE

Input 3: NO BRACKETS
Output: NO STARTING BRACKET FOUND.
```

6.1.6 Alternate Record Areas

All DMS 2200 database, PCIOS file, RDMS 2200 database, BIS DTM and SORT I/O 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 becomes awkward when, for example, an attempt is made to store a VIA SET 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, I-QU PLUS-1 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 including DMS 2200 database records, PCIOS files, RDMS 2200 table rows, BIS DTM report lines and SORT records. For a complete description of the DEFINE RA, refer to the I-QU PLUS-1 Programmer Reference.

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

```
INVOKE .......
```

The following defines 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 in INVOICE-LINE
RDA IL-DESC = 'SCREW DRIVER'
RDA IL-QUANTITY = 2
RDA IL-UNIT-PRICE = 1000
RDA IH-INVOICE-NUMBER = 'A10231'
.  *** Set key of owner record.
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, I-QU PLUS-1 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 I-QU PLUS-1 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 SEQ file
DEFINE RA SALES AFTER SALESMAN-REC   .  Rec area
.
DEFINE RDA S-KEY (1,5)               .  Output rec key
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 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.

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 (*) below
    GO LOOP
  ENDIF
...
```

Note: TRANSFER only works at the record level.

* The reference to SALESMAN-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.

6.2 Controlling Print Output

The I-QU PLUS-1 Processor can output to the user's PRINT$ (the terminal in demand mode), alternate print files or the system console. While the DISPLAY and EDIT commands are used to format print output, the PCONTROL command is used to control it. PCONTROL has several functions. It may be used to redirect or switch print output, to cause a form feed image to be placed into the current print output, or to place a special print control image in the current print output for use by the operating system when the file is printed.

6.2.1 Print Output Switching

The print output produced by the DISPLAY, EDIT, TRIMDISP, TRIMEDIT and DUMP commands will, by default, be directed to PRINT$. PRINT$ is the terminal screen when running I-QU PLUS-1 from a demand terminal; it is the primary print output when running I-QU PLUS-1 in a batch runstream. One function of the PCONTROL command is to allow the user to redirect the output of the DISPLAY, EDIT, TRIMDISP, TRIMEDIT or DUMP commands. The initial print mode, where output is directed to PRINT$, is called LOCAL. In LOCAL mode, the print line length is 76 characters. This line length prevents display images from wrapping to two lines on the terminal screen. If running in batch mode or if the B-option is used when entering I-QU PLUS-1, the printed line length will be 132 characters.

The user may redirect print output to an alternate print file in two ways. First, by using PCONTROL with the BRKPT function alone, I-QU PLUS-1 will attempt to create a default alternate print file named I$QUPRINTn, where n will be a number in the range 0

to 9. I-QU PLUS-1 will try to create I$QUPRINT0 first. If it cannot be created (already exists, etc.), the number will be incremented by one. This process will continue until print file I$QUPRINT9 is reached. If I-QU PLUS-1 cannot create a default print file, an error message will be displayed. I-QU PLUS-1 will terminate if this happens while running an I-QU PLUS-1 Program. The second way to redirect print to an alternate print file is to use the file-name option with the BRKPT function. In this case, I-QU PLUS-1 will attempt to assign the specified file. If it can be assigned, print will be directed to it. If it cannot be assigned, I-QU PLUS-1 will attempt to create a new file with the specified name. If I-QU PLUS-1 cannot create the file, an error condition will result.

When creating print output in BRKPT mode, the output line will be 132 characters (33 words) in length. In addition, no DISPLAY, EDIT, TRIMDISP, TRIMEDIT or DUMP command output will be displayed at the terminal.

The ECHO function of the PCONTROL command will cause output to be directed to both an alternate print file and the terminal. In this case, the output line length is the same as in LOCAL mode.

Output of the DISPLAY, EDIT, TRIMDISP and TRIMEDIT commands 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 an I-QU PLUS-1 program to redirect print as necessary. Consider the following when using PCONTROL to direct print output:

1. The CONSOLE function has no effect on the current BRKPTed print file.
2. If the PCONTROL ECHO function is used before a BRKPT function, I-QU PLUS-1 will first attempt to create a default alternate print file as if a BRKPT function without a user specified file were executed.
3. 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. In addition, user specified BRKPT files following a BRKPT to a default file will close the default file before opening the alternate file.
4. 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.
5. Upon exiting I-QU PLUS-1, any default print files will be closed and automatically @SYMed to the current print device (see the PRINTER directive in the I-QU PLUS-1 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. Try it yourself and observe the various I-QU PLUS-1 responses.

```
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'
EXIT                                  <— Will close remaining print files.
```

6.2.2 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.

The simplest form of operation in this category of PCONTROL functions is the EJECT. This function puts a form feed image into the current print 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 I-QU PLUS-1 Programmer Reference. Refer to the OS/2200 Executive Programmer Reference for complete documentation regarding print control options.

The following portion of an I-QU PLUS-1 report-writing program illustrates how the PCONTROL OPTION functions would be used to change the forms margins and place an EXEC page heading in the alternate print output:

```
.  Set up report output...
PC BRKPT 'SLS*IQLIST'                . 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 report. I-QU PLUS-1 will begin routing all DISPLAY, EDIT, TRIMDISP, TRIMEDIT and DUMP command output to the alternate print file SLS*IQLIST, when the first PC command is executed. The next two PC commands will place print control images into the alternate file, which will be used by the operating system when the file is actually printed. 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.

6.3 Formatting Output

The following section will describe various techniques used for formatting print output, and print file output that will be retrieved into BIS.

6.3.1 Print Line Formatting

Formatting print lines is very easy in I-QU PLUS-1. Examples of using the DISPLAY, EDIT, TRIMDISP and TRIMEDIT commands to construct print lines were illustrated earlier in this manual. Use of these commands will now be discussed in more detail.

Output from the DISPLAY, EDIT, TRIMDISP and TRIMEDIT commands is always placed first in the print buffer of I-QU PLUS-1. If the plus (+) symbol has not been used in the command, the buffer is immediately output and cleared. If the plus symbol was used, data will be placed into the print buffer and the next available column counter will be set to point to the position following the data in the print buffer. Each time an item is DISPLAYed or EDITed, it is moved to the position in the print buffer indicated by the next available column counter, which is how a multiple item print line is built. For further flexibility in formatting the print line, the DISPLAY, EDIT, TRIMDISP and TRIMEDIT commands provide a column parameter, which will override the “next available column counter” maintained internally by I-QU PLUS-1. The column parameter allows items to be output anywhere in the print line without filling gaps with spaces, etc.

Try the following sequence in CONVERSATIONAL mode on your system:

```
DISPLAY 10 DATE
DISPLAY 'COL1' +
DISPLAY 40 'COL40' +
DISPLAY 20 'COL20'                  <— Print between last two DISPLAYs
X = 1234
TRIMDISP 10 'The value of X is ' +
TRIMEDIT X 'Z,ZZZ'
```

In the last part of the sequence above, the TRIMDISP of the literal is started in column 10 and the output of the TRIMEDIT command immediately follows the literal. The next available column counter is always set to the end of the last item displayed plus 1 position. The TRIMDISP and TRIMEDIT are used whenever it is desirable to eliminate any leading or trailing spaces on output items. However, when a string literal contains leading and/or trailing spaces, they will not be suppressed. If X = 234 in the above example, the concatenated results would appear as follows:

```
The value of X is 234
```

Only the leading spaces resulting from the TRIMEDIT of X will be suppressed.

The size of the print buffer in I-QU PLUS-1 equal to the size of the RDA (the default is 4000 characters); however, if a line of more than 132 characters (76 in CONVERSATIONAL mode) is built in the print buffer, I-QU PLUS-1 will break it into multiple lines when it is finally released.

The following is an example of a procedure within an I-QU PLUS-1 program used to format detailed report lines. The procedure will also handle printing column headings:

```
FORMAT-DETAIL PROCEDURE
  LINE-COUNT = LINE-COUNT + 1
  .  If page full (55 LINES) skip to new page and
  .  print headings...
  IF LINE-COUNT = 55
    PC EJECT
    LINE-COUNT = 0
    DISPLAY 'PART NUM.  DESCRIPTION            UNIT';
    ' COST      ON HAND      INVENTORY'
    DISPLAY 58 'VALUE'
    DISPLAY ' '
  ENDIF
  .  Format detail 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' +
  SET INVENTORY-VALUE = RDA UNIT-COST * RDA QTY-ON-HAND
  EDIT    55 RDA INVENTORY-VALUE 'ZZZ,ZZZ,ZZZ.999'
  ENDPROC
```

6.3.2 Formatting for BIS

Formatting output for BIS is similar to formatting output for printing. It is simplified in that no heading or page control is necessary. An important feature of I-QU PLUS-1 in building output for BIS is automatic TAB character insertion. BIS uses the TAB character to delimit each field in a formatted TYPE. The BIS TYPE also contains predefined column headings. Non-I-QU PLUS-1 reports usually have to be processed through some sort of utility to remove report headings and insert TAB characters before being input to BIS. I-QU PLUS-1 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 will be preceded by a TAB character. The tab character will 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 six 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 “^” is used to show the presence of TAB characters.

Output of sequence 1:

```
1...5...10...15...20...
123ASB     SCREW DRIVER.....
```

Output of sequence 2:

```
1...5...10...15...20...
^123ASB    ^SCREW DRIVER.....
```

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 in preparation for delivery 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 I-QU PLUS-1 program 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
  PCONTROL BRKPT 'CUST*TO-BIS'
  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
    ENDIF
    FETCH4 NEXT CUST-MASTER CUST AREA
  ENDDO
  PCONTROL LOCAL
SAVE CUST-DUE
```

If the output is sent to the form type shown above, it will be ready to be retrieved into BIS for processing by the end user. Because the end user can further refine the report as needed using BIS functions, the I-QU PLUS-1 program can be kept very simple.

6.3.3 Miscellaneous Print Buffer Use - $PBUFF

The program 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 I-QU PLUS-1 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 I-QU PLUS-1 to output the string as a reply line. I-QU PLUS-1 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:

```
DEF A CSF-STRING 80               .  For the CSF image
DEF N CSF-STAT                    .  Facility status
DEF A FILE-QUAL-VAR 12
```

```
DEF A FILE-NAME-VAR 12
  ...
  ACCEPT FILE-QUAL-VAR
  ACCEPT FILE-NAME-VAR
  DISPLAY '@ASG,A ' +
  TRIMDISP FILE-QUAL-VAR +          .  Build the CSF image
  DISPLAY '*' +                     .  In the print buffer
  TRIMDISP FILE-NAME-VAR +
  DISPLAY '.' +
  CSF-STRING = $PBUFF               .  Move image to CSF var
  CSF CSF-STAT CSF-STRING           .  Issue @ASG,A for file
  IF CSF-STAT < 0                   .  Error assigning file?
    FACERR CSF-STAT                 .  Display fac 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.
```

6.4 Using Prewritten Program Source

In many cases, groups of I-QU PLUS-1 program commands and/or directives will be repeated in several programs. I-QU PLUS-1 provides an ADD directive for use in this situation. The ADD directive will tell the I-QU PLUS-1 processor to obtain input from a symbolic element in an EXEC program file. In this way, often-used portions of I-QU PLUS-1 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 programs. The following RDA definition may be in the symbolic element called I$QU*I$QULIB.REC-DEF. Note that the file I$QU*I$QULIB is the file I-QU PLUS-1 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 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                         <— Assume default source library.
DEF RDA PART-POS-1 (1,1)
...
```

As I-QU PLUS-1 edits the program, the added directives will be listed as if they were included in the original source input.

Chapter 7: How to Do It with I-QU PLUS-1

This section contains examples of techniques for using the I-QU PLUS-1 Processor to accomplish many tasks encountered in applications development. This is by no means a complete collection of I-QU PLUS-1 uses in applications development. Someone will always find a new or better way. KMSYS Worldwide welcomes user contributions or comments.

7.1 Populate a Database or File

One common use of the I-QU PLUS-1 Processor is in the creation and maintenance of test data. I-QU PLUS-1 is well suited to this purpose because the logic used closely parallels that used in a COBOL program. The following two examples show how I-QU PLUS-1 may be used to load, or update, a test or production database. The examples deal with DMS 2200 databases; however, the same logic is easily applied to PCIOS files.

7.1.1 Example 1: Load Data from an Existing File

In this example, I-QU PLUS-1 will be used to load data into the database from an existing PCIOS sequential file. The file may have been produced by another application, or by using a text editor such as @ED or @CTS.

The following is the COBOL definition of the input sequential file:

```
01 SALESMAN-INFO.
    05 SALESMAN-NUM            PIC 9(5).
    05 FULL-NAME.
        10 LAST-NAME           PIC X(15).
        10 FIRST-NAME          PIC X(15).
        10 REGION              PIC 9.
        10 QUOTA               PIC 9(7)V99.
```

The following is the definition of the database record to be loaded:

```
01 SALESMAN-REC.
    05 SLSM-KEY                PIC 9(5)
    05 SLSM-REGION             PIC 99
    05 SLSM-LAST-NAME          PIC X(20)
    05 SLSM-FIRST-NAME         PIC X(15)
    05 SLSM-QUOTA              PIC 9(10)V99 USAGE COMP
    05 SLSM-TOT                PIC 9(10)V99 USAGE COMP
    05 SLSM-DIVISION           PIC 99
```

The format of the two records is different; therefore, each field will be handled separately. The SLSM-DIVISION and SLSM-TOT fields do not exist in the input file and will be set to a constant value during the load. We will assume that the definition of the input file has been processed through the QINDEX processor to create a secondary data item index file to be used in this run.

Here is the entire runstream needed to load SALESMAN-REC records:

```
@RUN,D SLSBLD,,SALES
@ASG,A SALESFILE.                    .  Input sequential file
@IQU,I
INVOKE SUB-SALES IN MARKETING
```

```
INDEX SLS-INDEX                       .  Use secondary data item index
.
DEFINE F SALESFILE SEQ 45,100         .  Input file definition
DEFINE RA SALESFILE AFTER SALESMAN-REC  .  for input
.
  NON-FATAL 05 ON                     .  Make dup error non-fatal
  IMPART
  OPEN SALESMAN LOAD
  DBDN SALES-ANAME = 'SALESMAN'
  OPEN SALESFILE INPUT SEQ
  DO
    READ SALESFILE AT END BREAK
    .
    .  Move data fields from input to database record....
    .
    RDA SLSM-KEY = RDA SALESMAN-NUM
    RDA SLSM-LAST-NAME = RDA LAST-NAME
    RDA SLSM-FIRST-NAME = RDA FIRST-NAME
    RDA SLSM-REGION = RDA REGION
    RDA SLSM-QUOTE = RDA QUOTA
    RDA SLSM-DIVISION = 20            .  Constant to division
    RDA SLSM-TOT = 0                  .  Zero total
    .
    STORE SALESMAN-REC
    IF ERROR-NUM = 5                  .  Dup key ??
      DISPLAY 'Duplicate key on following input:' +
      DISPLAY RDA SALESMAN-INFO :SX   .  Display rec.
    ELSE
      X = X + 1                       .  Use X for rec count
    ENDIF
  ENDDO
  DEPART
  DISPLAY 'Loaded ' +
  TRIMEDIT X 'ZZ,ZZ9' +
  DISPLAY ' SALESMAN-REC records.'
  STOP EXIT
RUN
@FIN
```

7.1.2 Example 2: Interactive Update

This example will show an interactive I-QU PLUS-1 program used to add or replace data records in the area loaded in the last example. This runstream would be @ADDed from a demand terminal whenever records need to be added to the area.

```
@IQU,I                                .  Enter I-QU PLUS-1 in input mode
INVOKE SUB-SALES IN MARKETING
.  Following are used to accept input from user.
DEFINE N NFLD                         .  Use for integer num fields
DEFINE N NFLD-DEC2 .00                .  Use for decimal num fields
DEFINE A AFLD 20                      .  Use for alpha fields
.
DEFINE N UPDATE-SW                    .  Update or add switch
.
  CLEARSCREEN
  IMPART
  OPEN SALESMAN UPDATE
  DBDN SALES-ANAME

IN-LOOP
  ACCEPT AFLD 'Enter @EOF to quit, anything' ;
              ' else to continue' AT END FINAL

. Get key field...
```

```
  ACCEPT NFLD 'Enter SLSM-KEY:'
  RDA SLSM-KEY = NFLD
  FETCH5 SALESMAN-REC
  IF ERROR-NUM = 0                    .  Record exists?
    ACCEPT AFLD 'Record exists -replace (Y/N)' ;
      AT END FINAL
    IF AFLD = 'Y'
      UPDATE-SW = 1                   .  Record will be modified
    ELSE
      DISPLAY 'Skipping record.'
      GO IN-LOOP

    ENDIF
  ELSE
    SET UPDATE-SW = 0                 .  Record will be added
  ENDIF
  . Get remaining data fields into the RDA...
  ACCEPT AFLD 'Enter SLSM-LAST-NAME:' AT END FINAL
  RDA SLSM-LAST-NAME = AFLD
  ACCEPT AFLD 'Enter SLSM-FIRST-NAME:' AT END FINAL
  RDA SLSM-FIRST-NAME = AFLD
  ACCEPT NFLD 'Enter SLSM-REGION:' AT END FINAL
  RDA SLSM-REGION = NFLD
  ACCEPT NFLD-DEC2 'Enter SLSM-QUOTA:' AT END FINAL
  RDA SLSM-QUOTES = NFLD
  ACCEPT NFLD 'Enter SLSM-DIVISION:' AT END FINAL
  RDA SLSM-DIVISION = NFLD
  ACCEPT NFLD-DEC2 'Enter SLSM-TOT:' AT END FINAL
  RDA SLSM-TOT = NFLD
  . Store or modify the record...
  IF UPDATE-SW = 1
    MODIFY SALESMAN-REC
    DISPLAY '*** Record replaced ***'
  ELSE
    STORE SALESMAN-REC
    DISPLAY '*** Record added ***'
  ENDIF
  GO IN-LOOP
 .
FINAL
  DEPART
  CLEARSCREEN
  DISPLAY '**** END OF SALESMAN ADD/REPLACE ****'
  STOP EXIT
RUN
```

7.2 Writing Reports

Writing reports from DMS 2200 databases, PCIOS files and/or other file systems using I-QU PLUS-1 is similar to the same operation in COBOL, with the major difference being the amount of code required. While I-QU PLUS-1 and COBOL programs will be logically equivalent, I-QU PLUS-1 requires approximately 75% less code, and a comparable savings in development time, than the same program written in COBOL.

The following examples will demonstrate data selection and extraction, sorting, totaling and report formatting techniques.

7.2.1 Example 1: Sort, List and Total

This example will sort and list, with totals, all SALESMAN-REC records in the SALESMAN area. Records will be sorted in ascending order by sales representative name within region. Sales and quotas will be subtotaled by region and overall. The output report will be printed on special 8 1/2 X 11 forms.

```
@IQU,I
INVOKE SUB-SALES IN MARKETING
DEFINE N SALES-TOT-1
DEFINE N QUOTA-TOT-1
DEFINE N SALES-TOT-2
DEFINE N QUOTA-TOT-2
DEFINE N LINE-CT
DEFINE N H-REGION
. Init SORT to sort by region and last name..
  SORT 51,51 6,2,UN9,A 8,20,DISP,A
  IMPART
  OPEN SALESMAN
  F4 F SALESMAN-REC SALESMAN A
  .
  DO WHILE ERROR-NUM = 0
  .
  . Sort input proc - Fetches all SALESMAN-REC recs. and
  . releases them to the sort. If data selection was
  . required, it would be done here.
  .
    RELEASE SALESMAN-REC
    F4 N SALESMAN-REC SALESMAN A
  ENDDO
  DEPART
  . *** Set up print file first...
  PCONTROL BRKPT 'SALESRPT'
  PCONTROL OPTION 'S,FORM-8X11'
  PCONTROL OPTION ;
  'H,,1,SALES REPORT BY SALESMAN NAME WITHIN';
  ' REGION'
  DO
  .
  . This procedure outputs region totals, then rolls them into
  . overall totals for printing at the end of the report.
    RETURN AT END BREAK
    IF X = 0 . IF X = record count, 1st return
      H-REGION = RDA SLSM-REGION
      DO HEAD . First page header
    ELSE
      IF H-REGION NOT = RDA SLSM-REGION  .  Cntrl break?
        DO REGION-TOT
        H-REGION = RDA SLSM-REGION
      ENDIF
    ENDIF
    X = X + 1
    . Accumulate totals...
    SALES-TOT-1 = SALES-TOT-1 + RDA SLSM-TOT
    QUOTA-TOT-1 = QUOTA-TOT-1 + RDA SLSM-QUOTA
    . Format detail line...
    DISPLAY SLSM-LAST-NAME +
    DISPLAY SLSM-FIRST-NAME +
    EDIT 36 SLSM-TOT 'Z,ZZZ,ZZZ.99' +
    EDIT 50 SLSM-QUOTA 'Z,ZZZ,ZZZ.99'
    LINE-CT = LINE-CT + 1
    IF LINE-CT > 55
      DO HEAD
    ENDIF
  ENDDO
  . Print final totals.
  DISPLAY ' '
  DISPLAY ' *** OVERALL TOTALS' +
  EDIT 36 SALES-TOT-2 'Z,ZZZ,ZZZ.99' +
  EDIT 50 QUOTA-TOT-2 'Z,ZZZ,ZZZ.99'
```

```
  PCONTROL EJECT . Skip to new page
  DISPLAY 22 'TOTAL SALESMAN RECORDS PROCESSED = ' +
  EDIT X 'Z,ZZZ,ZZZ.99'              .  Print record count
  STOP EXIT
.
. Page head routine - There will be an automatic page header with the
. title, date and page number on the page, due to the PCONTROL OPTION
. command used earlier. This routine adds the region code and column
. headings.
HEAD PROCEDURE
  PCONTROL EJECT                    .  Form feed to new page
  LINE-CT = 0                       .  Clear line count
  DISPLAY 'REGION CODE: ' +         .  Region header line
  DISPLAY RDA SLSM-REGION
  DISPLAY ' '
  DISPLAY 'SALESMAN NAME (LAST,FIRST) TOTAL';
  ' SALES SALES QUOTA'              .  Column headings
  DISPLAY
  ENDPROC

REGION-TOT PROCEDURE
  DISPLAY ' '
  DISPLAY '*** REGION TOTAL' +
  EDIT 36 SALES-TOT-1 'Z,ZZZ,ZZZ.99' +
  EDIT 50 QUOTA-TOT-1 'Z,ZZZ,ZZZ.99'
  SALES-TOT-2 = SALES-TOT-2 + SALES-TOT-1  .  Roll totals
  QUOTA-TOT-2 = QUOTA-TOT-2 + QUOTA-TOT-1
  SALES-TOT-1 = 0                          .  Zero region totals
  QUOTA-TOT-1 = 0
  ENDPROC
```

The following is an example of the output produced by the above program:

```
SALES REPORT BY SALESMAN NAME WITHIN REGION   091185   PAGE      23
REGION CODE: 34

SALESMAN NAME (LAST,FIRST)          TOTAL SALES    SALES QUOTA

BLACK               JOHN             102,100.39      91,400.00
CAMPBELL            LAWRENCE          10,123.00       9,000.00
CLEMMONS            ROBERT            12,902.20      14,500.00

*** REGION TOTALS                     125,125.59     114,900.00

*** OVERALL TOTALS                  1,203,920.65   1,400,450.00
```

7.2.2 Example 2: Format a Report for Use in BIS

In this example, data from a PCIOS sequential file will be combined with data from a DMS 2200 database, and formatted for use in BIS. The output can then be transported to BIS with a BIS function (for examples using the BIS DTM Interface, see the next subsection). The PCIOS file contains sales history information by customer. The customer's name is not included in the file and must therefore be obtained from the DMS 2200 database. The formatted output will include the customer's account number, name, order number, product code and price paid. Only product codes A100 through B200 will be selected. The input file definition has been processed by the QINDEX processor to create a secondary data item index named SALES*CUSTHIST-IDX.

```
@ASG,A SALES*CUSTHIST.
@IQU,I
INVOKE SUB-CUST OF ORDER-ENTRY
DEFINE F CUSTHIST SEQ 120,50
DEFINE RA CUSTOMER-REC AFTER CUSTHIST
INDEX SALES*CUSTHIST-IDX          .  Secondary Data Item Index
.
BEGIN
  IMPART
  OPEN CUSTOMERS
  OPEN CUSTHIST INPUT SEQ
  DBDN CUST-ANAME = 'CUSTOMERS'
  PCONTROL BRKPT 'SALES*BISIN'
  .
  DO
    READ CUSTHIST AT END BREAK
    .  Check for selection criteria...
    IF RDA PRODUCT-CODE >= 'A100'
      AND RDA PRODUCT-CODE <= 'B200'
      .  Get customer's name from database...
      RDA CUST-KEY :CX = RDA CUSTNO
      FETCH5 CUSTOMER-REC
      IF ERROR-NUM = 13
        RDA CUST-NAME = '*NO MASTER ON FILE*'
      ENDIF
      .  Format output...
      TABS ON                     .  Start auto tabs
      DISPLAY RDA CUST-KEY +
      DISPLAY 8 RDA CUST-NAME +
      EDIT 30 RDA ORDER-NUM '9B999' +
      DISPLAY 36 RDA PRODUCT-CODE ' +
      EDIT 41 RDA PRICE-PAID '-ZZ,ZZZ.99'
      TABS OFF                    .  Turn off auto tabs
      X = X + 1
    ENDIF
  ENDDO
  .
```

```
  PCONTROL LOCAL
  DEPART
  CLOSE CUSTHIST
  DISPLAY 'End of BIS extract.  Output ' +
  TRIMEDIT X 'ZZ,ZZZ' +
  DISPLAY ' customer history records.'
  STOP EXIT
RUN
```

The output print file would be formatted for BIS as follows (the '^' is used to indicate TAB character placement):

```
1...5...10...15...20...25...30...35...40...45...50...
^120342^ABC PRODUCTS, INC    ^1 292^A101^ 11,245.95
^120560^WILLIAMS PRINT CO.   ^2 393^B190^   -230.90
^120560^WILLIAMS PRINT CO.   ^2 393^B120^  1,459.45
^120002^*NO MASTER ON FILE*  ^1 201^A299^    505.77
```

7.3 Processing BIS Reports via the DTM Interface

This subsection contains three examples in which an entire BIS report is processed by I-QU PLUS-1. Each example uses the BIS DTM interface. The first example will create a PCIOS file from the data contained in a BIS report. The second example will read a BIS report and display each line returned at the DEMAND terminal (Note: special consideration will be given to converting the BIS tab characters before displaying each line at the terminal). Finally, the third example will read a report, update a field (column) and send the results back to a second BIS RID.

7.3.1 Example 1: Create a PCIOS File from BIS Data

BIS Reports, or RIDs as they are commonly called, can be accessed with I-QU PLUS-1 as easily as accessing sequential PCIOS files. In fact, the commands used to access these reports are, for the most part, identical to the commands used to access PCIOS files. They are OPEN, CLOSE, READ and WRITE.

A DEFINE F directive is used to provide I-QU PLUS-1 information required to interface with BIS's Data Transfer Module (DTM). To begin with, the directive defines a name called a “queue-alias” which is used on other I-QU PLUS-1 DTM commands. The queue-alias is unique to I-QU PLUS-1 and is normally associated with a particular RID opened to the I-QU PLUS-1 session.

In addition to the queue-alias, the DEFINE F directive specifies the location of a parameter block which contains a list of fields in the RDA that are used to pass critical information to DTM necessary to effect the transfer. Refer to the I-QU PLUS-1 Programmer Reference for detailed information on how to define queue-alias “files” to I-QU PLUS-1. An example of queue-alias definition will be shown in the following sample programs.

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

> Consult the person responsible for installing I-QU PLUS-1 in order to determine the actual file name of this file on your system.

This definition can be included in the I-QU PLUS-1 program by simply issuing the following ADD directive:

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

The I-QU PLUS-1 directives included will appear as follows:

```
.
.         DTM interface parameter block for I-QU PLUS-1
.
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 reads a BIS report (RID), reformats the data found in each column into data items appropriately defined for storing in PCIOS files, and writes to the PCIOS file.

First, look at how the RDA is utilized below. The parameter block definition will begin at character position 1001 since it is included after the pseudo RDA definition called WORK-BASE. RDA position 1001 will leave ample room for the largest queue-alias/record to be read or written.

Since the BIS data is to be reformatted into data types more appropriate for writing to a PCIOS file, the DEFINE RA directive is used. By this means, an input data line can reside in the RDA alongside the reformatted output record.

```
DEF RDA WORK-BASE        (1000,1)  .  BASE WORK AREAS
.
. DTM Interface parameter block for I-QU
.   (Copied from SYS$LIB$*IQU-1.)
.
ADD DTM-PARM-BLK FROM SYS$LIB$*IQU-1
.
.  File definitions
.
DEF F FACTOR-BASE MAPPER PARAM-BLOCK .  BIS RID/file
DEF F FACTOR-FILE SEQ 51,200         .  SEQ output file
DEF RA FACTOR-FILE AFTER FACTOR-BASE .
```

The next part of the program shows the individual data item definitions for the BIS report and the PCIOS file. Notice that both the report and the file begin in RDA position 1. Since the FACTOR-FILE will reside in the RDA after the FACTOR-BASE (see DEFINE RA above), it will be necessary to qualify each data item name of the FACTOR-FILE when using the name on an I-QU PLUS-1 command.

The BIS columns that contain numeric data have a corresponding I-QU PLUS-1 data item definition with a data type of MAPNUM. This allows I-QU PLUS-1 to obtain BIS numeric display data and convert it TO non-display, numeric data types.

```
.
.  BIS    report field definitions...
.
DEF RDA FACTOR-BASE-REC    (1,80)  .  WHOLE LINE
DEF RDA PRODUCT-TYPE       (2,9)
DEF RDA SUB-KEY            (12,5)
DEF RDA PRODUC-COST        (18,6) MAPNUM
DEF RDA WHOLE-SALE$        (25,7) MAPNUM
DEF RDA RETAIL-$$$$        (33,8) MAPNUM
DEF RDA SALES-COMMISS      (42,7) MAPNUM
DEF RDA SPACE-REQ          (50,5) MAPNUM
DEF RDA DEMO-QUANTITY      (56,8) MAPNUM
DEF RDA DEMO-RESULTS       (65,15)
.
.  Output file definitions
.
DEF RDA OF-PRODUCT-TYPE    (1,9)
DEF RDA OF-SUB-KEY         (*,5)
```

```
DEF RDA OF-PRODUC-COST     (*,4) COMP
DEF RDA OF-WHOLE-SALE$     (*,4) COMP
DEF RDA OF-RETAIL-$$$$     (*,4) COMP
DEF RDA OF-SALES-COMMISS   (*,4) COMP .00
DEF RDA OF-SPACE-REQ       (*,2) COMP
DEF RDA OF-DEMO-QUANTITY   (*,4) COMP
DEF RDA OF-DEMO-RESULTS    (*,15)
```

The final part of the program initializes the parameter block field required to interface with DTM, READs each BIS report line, reformats the data, and WRITEs the PCIOS file. The parameter block fields, PB-DEST-QUEUE through PB-RUN-NAME must be filled in by the I-QU PLUS-1 program before opening the queue-alias file. These parameter values are described in the Unisys BIS SCHDLR Interface Programming Reference Manual with the exception of the PB-START-LINE and PB-XFER-LINES fields which are I-QU PLUS-1 parameters.

```
.  Initialize DTM parameter block to access RID 1C in mode 0...
RDA PB-DEST-QUEUE = 'BIS'
RDA PB-USERID = 'TEACHER'
RDA PB-DEPT = 1
RDA PB-PASSWORD = 'QPASS'
RDA PB-RUN-NAME = 'IQU$DTM'
RDA PB-MODE = '0'
RDA PB-TYPE = 'C'
RDA PB-RID = '1'
RDA PB-START-LINE = 6              .  Start reading at first tab line.
RDA PB-XFER-LINES = 0              .  Read all lines.
.  Now open files and process
OPEN FACTOR-FILE OUTPUT SEQ
OPEN FACTOR-BASE INPUT SEQ
IF RDA PB-STATUS <> 0
  GO DTM-ERROR
ENDIF
```

After the OPEN (shown on the previous page), the PB-STATUS parameter was checked to determine if the OPEN of the BIS report was successful. If the OPEN failed, SCHDLR would return a non-zero status in PB-STATUS. This same field is checked after the READ below. However, in addition to checking the PB-STATUS returned from SCHDLR, it is necessary to determine if an error has been returned from the BIS DTERR run or SCHDLR. When an error is returned, the first eleven positions of the data line returned will contain the literal, ".DTERR*****".

```
DO
  READ FACTOR-BASE AT END BREAK
  IF RDA PB-STATUS <> 0
    GO DTM-ERROR
  ENDIF
  IF RDA FACTOR-BASE-REC = '.DTERR***'
    GO BIS-ERROR
  ENDIF
  RDA OF-PRODUCT-TYPE OF FACTOR-FILE  = RDA PRODUCT-TYPE
  RDA OF-SUB-KEY OF FACTOR-FILE       = RDA SUB-KEY
  RDA OF-PRODUC-COST OF FACTOR-FILE   = RDA PRODUC-COST
  RDA OF-WHOLE-SALE$ OF FACTOR-FILE   = RDA WHOLE-SALE$
  RDA OF-RETAIL-$$$$ OF FACTOR-FILE   = RDA RETAIL-$$$$
  RDA OF-SALES-COMMISS OF FACTOR-FILE = RDA SALES-COMMISS
  RDA OF-SPACE-REQ OF FACTOR-FILE     = RDA SPACE-REQ
  RDA OF-DEMO-QUANTITY OF FACTOR-FILE = RDA DEMO-QUANTITY
  RDA OF-DEMO-RESULTS OF FACTOR-FILE  = RDA DEMO-RESULTS
  WRITE FACTOR-FILE
  X = X + 1
ENDDO
CLOSE FACTOR-FILE
TRIMDISP 'WROTE'X' FACTOR BASE RECORDS.'
STOP EXIT
```

```
.  DTM error display routine
DTM-ERROR
  DISPLAY '<<FATAL DTM ERROR ENCOUNTERED>>'
  DISPLAY RDA PB-STATUS
  DISPLAY RDA PB-ERR-CODE
  DISPLAY RDA PB-ERR-MESSAGE
  STOP EXIT
BIS-ERROR
  DISPLAY '<<FATAK ERROR IN BIS DTM RUN>>'
  DISPLAY RDA FACTOR-BASE-REC.
  STOP EXIT
RUN
```

7.3.2 Example 2: Displaying a Report on a DEMAND Terminal

The following I-QU PLUS-1 program opens a BIS report, reads the requested lines and displays them at the terminal. Special handling is provided for BIS TAB codes since this program is intended to run at the DEMAND terminal; i.e., TAB codes must be replaced with spaces in order to prevent the TAB code from acting as a screen control character.

```
INIT
INPUT
LISTOFF
DEF RDA LINE-BASE          (101,0)          A9
DEF RDA LINE-TAB           (LINE-BASE,1)    A9
DEF RDA LINE-RECORD        (LINE-BASE,132)  A9
.
.  DTM INTERFACE PARAMETER BLOCK FOR I-QU
.
ADD DTM-PARM-BLK FROM SYS$LIB$*IQU-1
.
.  DEFINE RDA REFERENCE FOR A TAB CODE
DEF RDA TAB-NUM            (1,1)            UB9
DEF RDA TAB-ALPHA          (TAB-NUM,1)      A9
.
.
DEF A PV-MODE 12
DEF A PV-TYPE 1
DEF A PV-RID 4
DEF A TAB-CHAR 1                    .  Search character for SCAN command
.
.
  RDA TAB-NUM = 9                   .  Decimal 9 to UB9 is
  TAB-CHAR = RDA TAB-ALPHA          .  Alpha TAB Char.
.
.
  RDA PB-DEST-QUEUE = 'BIS'
  RDA PB-USERID = 'THETEACH'
  RDA PB-DEPT = 7
  RDA PB-PASSWORD = ''
  RDA PB-RUN-NAME = 'IQU$DTM'
  ACCEPT PV-MODE 'MODE: '
  RDA PB-MODE = PV-MODE
  ACCEPT PV-TYPE 'TYPE: '
  RDA PB-TYPE = PV-TYPE
  ACCEPT PV-RID 'RID: '
  RDA PB-RID = PV-RID
  ACCEPT X 'START LINE: '
  RDA PB-START-LINE = X
  ACCEPT Y 'LINE QUANTITY: '
  RDA PB-XFER-LINES = Y
.
.
DEF F BIS BIS PARAM-BLOCK
```

```
DEF RA BIS 26                          .  No I/O before word 26
.
.
  OPEN BIS INPUT SEQ
  D 'OPEN READ: ' +
  D RDA PB-STATUS
  IF RDA PB-STATUS <> 0
    D RDA PB-ERR-CODE
    D RDA PB-ERR-MESSAGE
    STOP 97
  ENDIF
.
.
  DO
    READ BIS 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
      BREAK
    ELSE
      IF RDA FACTOR-BASE-REC = '.DTERR***'
        DISPLAY '****FATAK ERROR IN BIS DTM RUN****'
        DISPLAY RDA FACTOR-BASE-REC.
        BREAK
      ENDIF
.
    IF REC$LEN = 0
      L$=1
    ELSE
      L$ = REC$LEN                     .  Set length for SCAN
    ENDIF                              .    and DISPLAY
.
.
    DO
      X = 0
      SCAN RDA LINE-BASE TAB-CHAR X    .  Find TAB
      IF X = 0
        BREAK
      ENDIF
      RDA LINE-TAB :X = $SPACES        .  Put Space
    ENDDO
    D RDA LINE-BASE
  ENDDO
.
.
  IF RDA PB-STATUS = 1
    CLOSE BIS
    D 'CLOSE INPUT: ' +
    D RDA PB-STATUS
    IF RDA PB-STATUS <> 0
      D RDA PB-STATUS
      D RDA PB-ERR-CODE
      D RDA PB-ERR-MESSAGE
    ENDIF
  ENDIF
.
LISTON
```

Below is an execution of the above program. The partial output is shown as 80 columns plus the SOE:

```
▶ADD DTM FROM PF
▶Initializing I-QU Work Areas
```

```
▶Switched To Input Mode.
▶ 1 1 3C LISTOFF
▶RUN
▶MODE:
▶0
▶TYPE:
▶C
▶RID:
▶1
▶START LINE:
▶1
▶LINE QUANTITY:
▶0
▶OPEN READ: 0
▶.DATE 12 JUL 85  09:38:08  RID      1C   12 JUL 85   LOU1
▶.@991231              CORPORATE FACTORS BASE
▶C000004
▶* PRODUCT . SUB .PRODUC. WHOLE . RETAIL . SALES .SPACE.  DEMO  .
▶*  TYPE   . KEY . COST . SALE$ .  $$$$  .COMMISS. REQ .QUANTITY. DEM
▶*=========.=====.======.=======.========.=======.=====.========.====
▶ BLACKBOX1     A  13500  16875     23625 2362.50   100        1
▶ BLACKBOX2     A  13600  17000     23800 2380.00   110        2
▶ BLACKBOX3     A  13700  17125     23975 2397.50   120        4
▶ BLACKBOX4     B  13800  17250     24150 2415.00   130       10
▶ BLACKBOX5     B  13900  17375     24325 2432.50   140       50
▶.
▶.
▶.
▶CLOSE INPUT:                 0
```

7.3.3 Example 3: Merging DMS 2200 Data into a Report

The objective of this I-QU PLUS-1 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 original report will be read by the I-QU PLUS-1 program where part names will be inserted into each line. The completed report will be returned to BIS as a new report.

The 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. In addition, the output report will be limited to 2000 lines per RID. This implies that multiple output reports will be generated if there are more than 2000 lines in the input report.

The first part of the program adds the DTM parameter block definitions and invokes the subschema.

```
.  DTM INTERFACE PARAMETER BLOCK FOR I-QU PLUS-1
.
ADD DTM-PARM-BLK FROM SYS$LIB$*IQU-1
.
INVOKE INVENTORYSUB IN MFGSCHEMA .  Using a default schema file
.
```

For a second parameter block, define a dummy file (PB1) to be used as an offset for a second dummy file (PB2). PB2 is the subject of the first DEF RA directive below and will be used to qualify all references to the second, parameter block item names. Both dummy files MUST be at least the length of the parameter block (168 characters for the current release level).

```
.
DEF F PB1 SEQ 168,1
DEF F PB2 SEQ 168,1
DEF RA PB2 AFTER PB1                .  2nd PARAM-BLOCK beyond 1st
.
DEF F BIS-RID1 BIS PARAM-BLOCK OF PB1
```

```
DEF F BIS-RID2 BIS PARAM-BLOCK OF PB2
.
DEF RA BIS-RID1 AFTER PB2            .  BIS data after PARAM-BLOCK
DEF RA BIS-RID2 OVERLAY BIS-RID1     .  BIS Input/Output
DEF RA PART-MASTER AFTER BIS-RID2    .  DMS 2200 Input beyond BIS
.  Input
.
.  Define 1st 2 columns of BIS RID
.
DEF RDA BIS-REC                    (1,132)
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                       . Variable to get BIS input RID
DEF A PBUFF-FLUSH 1                  . Variable to flush print buffer
.
  RDA (1,1000) = $SPACES
  D
  RDA PB-DEST-QUEUE OF PB1 = 'MAPPER' . Set PARAM-BLOCK for input
  RDA PB-USERID OF PB1 = 'BOB'
  RDA PB-DEPT OF PB1 = 1
  RDA PB-PASSWORD OF PB1 = 'Q45'
  RDA PB-RUN-NAME OF PB1 = 'IQU$DTM'
  RDA PB-MODE OF PB1 = '100'
  RDA PB-TYPE OF PB1 = 'E'
  ACCEPT PV-RID 'Enter Input RID:
  RDA PB-RID OF PB1 = PV-RID         . Input RID
  RDA PB-XFER-LINES OF PB1 = 2000    . Limit # lines due to MQT
.
  RDA PB-DEST-QUEUE OF PB2 = 'MAPPER' . Set PARAM-BLOCK for output
  RDA PB-USERID OF PB2 = 'LEW'
  RDA PB-DEPT OF PB2 = 14
  RDA PB-PASSWORD OF PB2 = 'ENT'
  RDA PB-RUN-NAME OF PB2 = 'IQU$DTM'
  RDA PB-MODE OF PB2 = '74'
  RDA PB-TYPE OF PB2 = 'I'
  RDA PB-RID OF PB2 = ' '            . Allocate next available RID
  RDA PB-XFER-LINES OF PB2 = 0000    . Not used on OUTPUT
  RDA PB-START-LINE OF PB2 = 0       .  "   "   "    "
.
  Y = 0                              . Counter for output lines
  OPEN BIS-RID2 OUTPUT SEQ
  D '*** OPEN OUTPUT RID FIRST TIME ***'
  D RDA PARAM-BLOCK OF PB2           . Display for program trace
  D 'OPEN ERROR: ' +
  DO PB2-TEST
  Z = 2
  IMPART
  OPEN PART-AREA
  DBDN PART-ANAME = 'PART-AREA'
.
```

The following part of the program uses a nested DO loop. Due to the MTQ limit, the outer DO will control how many lines (2000) will be read from the report during one OPEN. The inner DO reads each line of the report. For each line read, an attempt is made to retrieve a DMS 2200 record by using the part number found in the report line. The DMS record contains a part description field that will be used to update the report line. All updated lines will be written to a newly allocated report. However, the MQT limit will force us to allocate multiple reports since the input report is very large. Once the reports have been written, they may be concatenated with the BIS ADTO function.

```
DO                                 . DO for multiple OPENs
  RDA PB-START-LINE OF PB1 = Z
```

```
    OPEN BIS-RID1 INPUT SEQ
    TD '### OPEN INPUT RID: START AT LINE # ' Z
    D RDA PARAM-BLOCK OF PB1       . Display for program trace
    D 'OPEN ERROR: ' +
    DO PB1-TEST
.
    DO                             . READ limited to 2000 lines per OPEN
      READ BIS-RID1 AT END BREAK
      D 'READ STATUS:' +
      DO PB1-TEST
.
.   *** Use PARTNO from BIS to get PM-PART-NAME from DMS 2200
      RDA PM-PART-KEY = RDA PARTNO OF BIS-RID1
      FETCH5 PART-MASTER
      IF ERROR-NUM <> 0
        D '--> Part name not found for number ' +
        D RDA PARTNO +
        D ' / Line not written to BIS'
        D
      ELSE
        RDA PARTNAME OF BIS-RID2 = RDA PM-PART-NAME
.
        WRITE BIS-RID2             . Write only if PARTNAME found
        Y = Y + 1
        IF Y = 2000                . Close each output RID at 2000 lines
          CLOSE BIS-RID2
          D 'CLOSE ERROR: ' +
          DO PB2-TEST
          TD '*** CLOSE OUTPUT: ' Y ' LINES WRITTEN TO RID ' +
          TD RDA PB-RID OF PB2     . Get allocated RID #
          D RDA PARAM-BLOCK OF PB2
          RDA PB-RID OF PB2 = ' '  . Blank to allocate new RID
          OPEN BIS-RID2 OUTPUT SEQ . More lines to new RID
          D 'OPEN ERROR: ' +
          DO PB2-TEST
          Y = 0
        ELSE
          D 'WRITE STATUS:' +
          DO PB2-TEST
        ENDIF
      ENDIF
      X = X + 1
    ENDDO
    IF X = 0                       . If no lines returned during
      BREAK                        .  this OPEN (the last), BREAK
    ENDIF                          .  out of this DO and end program.
    X = 0
    IF RDA PB-STATUS OF PB1 = 1
      CLOSE BIS-RID1
      D 'CLOSE ERROR: ' +
      DO PB1-TEST
    ENDIF
    Z = Z + 2000                   . Set variable to get next 2000
                                   . lines from input RID
  ENDDO
  CLOSE BIS-RID1
  D 'CLOSE ERROR: ' +
  DO PB1-TEST
  CLOSE BIS-RID2
  D 'CLOSE ERROR: ' +
  DO PB2-TEST
  TD '*** CLOSE OUTPUT RID:'Y' LINES WRITTEN TO RID ' +
  TD RDA PB-RID OF PB2
```

```
  D RDA PARAM-BLOCK OF PB2
  TD 'X = ' X ' / Y = ' Y +
  TD' / Z = ' Z
  STOP EXIT

.
PB1-TEST PROC                       . Test for input errors
  IF RDA PB-STATUS OF PB1 <> 0
    D RDA PB-STATUS OF PB1
    D RDA PB-ERR-CODE OF PB1
    D RDA PB-ERR-MESSAGE OF PB1
  ELSE
    IF RDA BIS-REC OF BIS-RID1 = '.DTERR***'
      D '*** FATAL ERROR IN BIS DTM RUN: '
    ELSE
      PBUFF-FLUSH = $PBUFF          . Flush print buffer if no error BREAK
    ENDIF
  ENDIF
  D RDA BIS-REC OF BIS-RID1
  STOP EXIT                         . Stop if error!
  ENDPROC
.
PB2-TEST PROC                       . Test for output errors
  IF RDA PB-STATUS OF PB2 <> 0
    D RDA PB-STATUS OF PB2
    D RDA PB-ERR-CODE OF PB2
    D RDA PB-ERR-MESSAGE OF PB2
  ELSE
    IF RDA BIS-REC OF BIS-RID2 = '.DTERR***'
      D '*** FATAL ERROR IN BIS DTM RUN: '

    ELSE
      PBUFF-FLUSH = $PBUFF          . Flush print buffer if no error BREAK
    ENDIF
  ENDIF
  D RDA BIS-REC OF BIS-RID2
  STOP EXIT .  Stop if error!
  ENDPROC
.
SAVE GET-P-NAMES
```

The DISPLAYed output of the program might appear as follows:

```
RUN GET-P-NAMES

*** OPEN OUTPUT RID FIRST TIME ***
BIS      LEW         0014        74         I       00000000IQU$DTM

### OPEN INPUT RID: START AT LINE # 2
BIS      BOB    0001QLINK       100         E   000500022000IQU$DTM

*** CLOSE OUTPUT: 2000 LINES WRITTEN TO RID 90
BIS      LEW         0014        74         I   90  00000000IQU$DTM

### OPEN INPUT RID: START AT LINE # 2002
BIS      BOB    0001QLINK       100         E   000520022000IQU$DTM

--Part name not found for number 21001 / Line not written to BIS

*** CLOSE OUTPUT: 2000 LINES WRITTEN TO RID 91
BIS      LEW         0014        74         I    91 00000000IQU$DTM

### OPEN INPUT RID: START AT LINE # 4002
BIS      BOB    0001QLINK       100         E   000540022000IQU$DTM
```

```
### OPEN INPUT RID: START AT LINE # 6002
BIS      BOB    0001QLINK       100         E   000560022000IQU$DTM

*** CLOSE OUTPUT: 239 LINES WRITTEN TO RID 92
BIS      LEW         0014        74         I    92 00000000IQU$DTM

X = 0 / Y = 239 / Z = 6002
** Exiting I-QU **
```

The original report might look like this before processing by I-QU PLUS-1 (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 new report after I-QU PLUS-1 processing:

```
*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^
```

7.4 Tables in I-QU PLUS-1

The I-QU PLUS-1 Processor does not allow definition of array variables; however, arrays or tables can be set up and manipulated in the RDA. The following examples should aid in developing table lookup logic for use in your application.

7.4.1 Example 1: 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
.
DEFINE RDA TAB-ENT (1001,12)         . Code and name
DEFINE RDA TAB-CODE (TAB-ENT,2) UN9  . Code
DEFINE RDA TAB-NAME (*,10)           . Name
DEFINE SUB TX TAB-ENT                . Subscript
DEFINE N TMAX                        . Table limit
DEFINE A RGN-NAME 12                 . Work area
DEFINE A TABENT 12                   . Accept variable
.
.  The following procedure retrieves the region table
.  from text following the I-QU PLUS-1 RUN directive
.
  DO
    ACCEPT TABENT AT END BREAK
    IF TX > 300                                . Above limit?
      DISPLAY 'TABLE EXCEEDS LIMIT'
      STOP EXIT
    ENDIF
    RDA TAB-ENT :TX = TABENT                   . Put in RDA table
    TX = TX + 1                                . Increment subscript
  ENDDO
  TMAX = TX                                    . Save limit of table
```

```
.
.  MAIN LINE PROCESSING.
...
...
.  This routine translates the numeric region code
.  to region name using the table coded above.
.
  RGN-NAME = '*UNKNOWN*'                        . Assume no find
  TX = 1
  DO WHILE TX <> TMAX
    IF RDA TAB-NAME :TX = RDA SLSM-REGION
      RGN-NAME = RDA TAB-NAME TX:
      BREAK
    ENDIF
    TX = TX + 1
  ENDDO
. . .
. . .
RUN
01SOUTHEAST         <— Data to load into region table
02EAST
03NORTHEAST
. . .
```

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 RDA subscript is checked while the table is being loaded to ensure that the RDA limit is not exceeded.

7.4.2 Example 2: Table of Accumulators

In this example, as records are read, a table of accumulators will be built to accumulate totals by region. The table will prevent 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, the table entries will be sorted and output as a report.

```
@IQU,I
INVOKE SUB-SALES IN MARKETING
DEFINE RDA SRT-TEC (1,6)
DEFINE RDA SRT-CODE (SRT-REC,2)
DEFINE RDA SRT-TOT (*,4) COMP .00
.
DEFINE RDA RGN-TABLE (1001,6)
DEFINE RDA RGN-CODE (RGN-TABLE,2)
DEFINE RDA RGN-TOT (*,4) COMP .00
DEFINE SUB RGN-TABLE RGNX                          . Subscript for table
.
  IMPART
  OPEN SALESMAN
  F4 F SALESMAN-REC SALESMAN A
  DO WHILE ERROR-NUM = 0
    RGNX = 1
    .
    .  This in-line DO accumulates sales totals by region
    .  in an RDA table.  If a region is not yet in the
    .  table, a new table entry is set up.  If the region
    .  is already in the table, sales are added to the total
    .  in the table entry.
    DO
      IF RDA RGN-CODE :RGNX = $LOVALS .  Empty entry??
        RDA RGN-CODE :RGNX = RDA SLSM-REGION        . Insert
        RDA RGN-TOT :RGNX = RDA SLSM-TOTAL
        BREAK
      ENDIF
      IF RDA RGN-CODE :RGNX = SLSM-REGION
```

```
          RDA RGN-TOT = RDA RGN-TOT :RGNX + RDA SLSM-TOT
          BREAK
        ENDIF
        RGNX = RGNX + 1                              . Inc to next entry
      ENDDO
      F4 N SALESMAN-REC SALESMAN A
    ENDDO .  *** End of DO WHILE ERROR-NUM = 0
    DO SORT-LIST
    DEPART
    STOP EXIT
.
.  This proc sorts the table entries by region code and
.  lists the totals accumulated by the TOTS procedure.
.
SORT-LIST PROCEDURE

    SORT 6,6 1,2,DISP,A .  Set sort by region code
    RGNX = 1
    DO WHILE RDA RGN-TABLE :RGNX <> $LOVALS

      RDA SRT-REC = RDA RGN-TABLE :RGNX              . Move to sort
      RELEASE 6                                      . Release entry
      RGNX = RGNX + 1                                . Inc to next
    ENDDO
    DO
      RETURN AT END BREAK
      DISPLAY 'REGION CODE: ' +
      DISPLAY RDA SRT-CODE +
      DISPLAY ' TOTAL SALES: ' +
      EDIT RDA SRT-TOT 'Z,ZZZ,ZZZ.99-'
    ENDDO
    ENDPROC
RUN
```

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 I-QU PLUS-1 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:

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

Chapter 8: Debugging I-QU PLUS-1 Programs

The I-QU PLUS-1 processor provides several debugging aids including a command trace that can be invoked on request, and a formatted object dump. This section will explain how these may be used.

8.1 Using I-QU PLUS-1 Command Trace

A helpful feature of I-QU PLUS-1 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. Using this information with the compile listing of the program, you can trace program logic. The following is a short example of INPUT mode session from a compile and run with the TRACE ON command:

Example of Program Command Trace

```
 1    1              TRACE  ON  .   Turn  on  command  trace.
 2    2              DATE
 3    3              TIME
 4    4              DISPLAY  'The  date  and  time  is  '  +
 5    5              DISPLAY  DATE  +
 6    6              DISPLAY ' - ' +
 7    7              DISPLAY TIME
 8    8              IF MONTH  =  12
 9    9                DISPLAY  'This  is  December!'
10    10             ELSE
11    11               DISPLAY  'It''s  not  December  yet.'
12    12             ENDIF
13    12             DISPLAY  'End  of  demo....bye'
14    13           RUN

<<< TRACE TURNED ON >>>
***TRACE PC = 0002
***TRACE PC = 0003
***TRACE PC = 0004
***TRACE PC = 0005
***TRACE PC = 0006
***TRACE PC = 0007
The date and time is 1993/12/03 - 21:32:24.205
***TRACE PC = 0008
***TRACE PC = 0009
This is December!
***TRACE PC = 0010
***TRACE PC = 0012
End of demo....bye
***TRACE PC = 0013
```

8.2 Reading an I-QU PLUS-1 Object Dump

If you have been using the I-QU PLUS-1 Processor for any length of time, you have probably discovered that when a runtime error is encountered in an I-QU PLUS-1 program, 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 I-QU PLUS-1 Processor. It is still used for this purpose; however, you will find this dump very helpful in debugging your I-QU PLUS-1 applications. The dump can also be produced by entering the OBJECT directive.

Refer to the figure containing an example I-QU PLUS-1 Object dump while going over the following description.

The first section of the dump contains a list of the Data Storage Index (DSI) containing 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 I-QU PLUS-1 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 alpha 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. If this is a decimal variable, the scale factor is located in the first quarter word (the implied number of decimal places) and remaining 7-quarter words contain the integer representation of the value. If this is a floating-point variable, these two words are represented like a COBOL COMP-2 item (double precision floating-point). A subscript variable will contain an octal 025 in the first quarter word, with the subscripted item's length in the next three bytes and the actual subscript value in the second word. The format of an alpha variable consists of a two word entry followed by enough words (in multiples of two) to accommodate the alpha string length. The two words preceding the alpha string data are made up of a one-quarter word special value indicator followed by 7-quarter words that contain the alpha string's character length.

If a DEFINE F is in effect when the dump is taken, the next section of the dump will be a listing of the DEFINE F files with their current usage and access modes, and the alternate record area word offset. No DEF F directives were in use when the example dump 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 I-QU PLUS-1 to terminate automatically. These codes must be handled by the I-QU PLUS-1 user when they occur.

If an INVOKE was executed, a list of all areas, records, sets and database datanames will be next on the dump listing. This list is obtained from the I-QU PLUS-1 primary data item index file, not from I-QU PLUS-1 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 will help 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.

Example of an I-QU PLUS-1 Object Dump

This example is an I-QU PLUS-1 object dump taken by entering the OBJECT directive while in INPUT mode. Note: Some spaces were eliminated from the dump during editing of this document in order to prevent the printed lines shown below from wrapping to multiple lines.

```
Initial Mode is INPUT
   1   1         ADD TSTOBJ FROM PFILE
   2   1     1C  INVOKE TESTERSUB IN TESTER FILE SCHFILE
** Invoke complete **
   3   1     2C  DEF RA R03-PART AFTER R01-CUST
   4   1     3C
   5   1     4C  DEF RDA MY-A9-FIELD (2,3)
   6   1     5C  DEF RDA MY-UN9-FIELD (2,3) UN9
   7   1     6C  DEF RDA MY-UB9-FIELD (2,3) UB9
```

```
   8   1     7C
   9   1     8C  DEF A MY-ALPHA-VAR '*** MY VARIABLE ***'
  10   1     9C  DEF N MY-NUMERIC-VAR 255
  11   1    10C  DEF SUB MY-SUB MY-A9-FIELD
  12   1    11C
  13   1    12C    IMPART
  14   2    13C    O A01-CUST
  15   3    14C    O A10-PART
  16   4    15C    F3 F A10-PART A
  17   5    16C    F3 F A01-CUST A
  18   6    17C
  19   6    18C  RUN
<Input Mode>
  20   7    19C  OBJECT

*** 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
C-O-T                            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
C-DBP                            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
IICODE                           INDEX-00051 WORD-00101 DEC-00032 TYPE-00001-V
$TAB                             INDEX-00054 WORD-00107 DEC-00032 TYPE-00001-V
RUNID                            INDEX-00056 WORD-00111 DEC-00032 TYPE-00001-V
MY-A9-FIELD                     ST CHR-00002 LEN-00003 TYPE-00016 RDA DEF
MY-UN9-FIELD                    ST CHR-00002 LEN-00003 TYPE-00018 RDA DEF
MY-UB9-FIELD                    ST CHR-00002 LEN-00003 TYPE-00000 RDA DEF
MY-ALPHA-VAR                     INDEX-00058 WORD-00115 DEC-00032 TYPE-00001-V
MY-NUMERIC-VAR                   INDEX-00062 WORD-00123 DEC-00000 TYPE-00000-V
MY-SUB                           INDEX-00063 WORD-00125 DEC-00021 TYPE-00000-V

*** Start of Data Storage Area ***
0001    000000000000 000000000000 000000000000 000001000001 ????????????????
0005    000000000000 000000000001 000000000000 000000000001 ????????????????
0009    000000000000 000000000000 000000000000 000000000000 ????????????????
0013    000000000000 000000000000 040000000000 000000000036 ???????? ???????
0017    040040040040 040040040040 040040040040 040040040040
0021    040040040040 040040040040 040040040040 040040040040
```

```
0025    040000000000 000000000036 122060061055 103125123124 ???????R01-CUST
0029    040040040040 040040040040 040040040040 040040040040
0033    040040040040 040040040040 040000000000 000000000036 ???????
0037    040040040040 040040040040 040040040040 040040040040
0041    040040040040 040040040040 040040040040 040040040040
0045    000000000000 000000252452 000000000000 000000000127 ??????????????W
0049    000000000000 000000000014 000000000000 000000000004 ????????????????
0053    040000000000 000000000010 070067057061 062057060064 ???????87/12/04
0057    040040040040 040040040040 000000000000 000000000000 ????????
0061    000000000000 000000000000 000000000000 000000000000 ????????????????
0065    040000000000 000000000014 101060061055 103125123124 ???????A01-CUST
0069    040040040040 040040040040 040000000000 000000000014 ???????
0073    040040040040 040040040040 040040040040 040040040040
0077    000000000000 000000000001 000000000000 000000000000 ????????????????
0081    040000000000 000000000014 040040040040 040040040040 ???????
0085    040040040040 040040040040 000000000000 000000000000 ????????
0089    000000000000 000000000000 000000000000 000003245444 ????????????????
0093    000000000000 000000000000 000000000000 000000000000 ????????????????
0097    000000000000 000000000000 000000000000 000000000000 ????????????????
0101    040000000000 000000000010 040040040040 040040040040 ???????
0105    040040040040 040040040040 040000000000 000000000001 ???????
0109    011040040040 040040040040 040000000000 000000000006 ? ???????
0113    114105127040 040040040040 040000000000 000000000023 LEW ???????
0117    052052052040 115131040126 101122111101 102114105040 *** MY VARIABLE
0121    052052052040 040040040040 000000000000 000000000377 *** ????????
0125    025000000003 000000000000 040040040040 ????????

*** INVOKE tables follow ***
AREA---->A01-CUST               SCODE=00001 SSCODE=00001
AREA---->A02-SLSM               SCODE=00002 SSCODE=00002
AREA---->A03-IPA                SCODE=00003 SSCODE=00003
AREA---->A04-IPAX               SCODE=00004 SSCODE=00004
AREA---->A05-REGION             SCODE=00005 SSCODE=00005
AREA---->A06-CATALOG            SCODE=00006 SSCODE=00006
AREA---->A07-CAT-IX             SCODE=00007 SSCODE=00007
AREA---->A08-COST               SCODE=00008 SSCODE=00008
AREA---->A09-COST-IX            SCODE=00009 SSCODE=00009
AREA---->A10-PART               SCODE=00010 SSCODE=00010
AREA---->A11-INV                SCODE=00011 SSCODE=00011
AREA---->A12-REQMTS             SCODE=00012 SSCODE=00012
AREA---->A13-OPS                SCODE=00013 SSCODE=00013
RECORD-->IPA$                    SCODE=04102 SSCODE=00014 RLEN=00002 ROFF=00000
RECORD-->R01-CUST                SCODE=00001 SSCODE=00001 RLEN=00023 ROFF=00000
RECORD-->R02-REGION              SCODE=00002 SSCODE=00002 RLEN=00009 ROFF=00000
RECORD-->R03-PART                SCODE=00003 SSCODE=00003 RLEN=00016 ROFF=00024
RECORD-->R05-CATALOG             SCODE=00005 SSCODE=00004 RLEN=00006 ROFF=00000
RECORD-->R06-COST                SCODE=00006 SSCODE=00005 RLEN=00006 ROFF=00000
RECORD-->R07-SALESMAN            SCODE=00007 SSCODE=00006 RLEN=00003 ROFF=00000
RECORD-->R10-CUST-DATE           SCODE=00010 SSCODE=00007 RLEN=00001 ROFF=00000
RECORD-->R20-INVENTORY           SCODE=00020 SSCODE=00008 RLEN=00009 ROFF=00000
RECORD-->R21-WAREHOUSE           SCODE=00021 SSCODE=00009 RLEN=00009 ROFF=00000
RECORD-->R22-REL                 SCODE=00022 SSCODE=00010 RLEN=00003 ROFF=00000
RECORD-->R30-REQMTS              SCODE=00030 SSCODE=00011 RLEN=00010 ROFF=00000
RECORD-->R32-OP-STEP             SCODE=00032 SSCODE=00012 RLEN=00009 ROFF=00000
RECORD-->R33-OP-MASTER           SCODE=00033 SSCODE=00013 RLEN=00009 ROFF=00000
SET---->S05-CUST-DATE           SCODE=00005 SSCODE=00001
SET---->S06-CUST-SLSM-IPA       SCODE=00006 SSCODE=00002
SET---->S07-REGION-CUST         SCODE=00007 SSCODE=00003
SET---->S08-REGION-WH           SCODE=00008 SSCODE=00004
SET---->S09-WH-REL              SCODE=00009 SSCODE=00005
SET---->S10-INV-REL             SCODE=00010 SSCODE=00006
SET---->S11-PART-INV            SCODE=00011 SSCODE=00007
SET---->S12-PART-CAT            SCODE=00012 SSCODE=00008
```

```
SET———>S13-COST-PART          SCODE=00013 SSCODE=00009
SET———>S14-PART-REQMTS        SCODE=00014 SSCODE=00010
SET———>S15-PART-OP            SCODE=00015 SSCODE=00011
SET———>S16-OP                 SCODE=00016 SSCODE=00012
SET———>S17-OP-REQMST          SCODE=00017 SSCODE=00013
DBDN———->A01-AN                TYPE=1, LEN=00003, SCODE=00001, SSCODE=00001
DBDN———->A02-AN                TYPE=1, LEN=00003, SCODE=00003, SSCODE=00003
DBDN———->A10-AN                TYPE=1, LEN=00003, SCODE=00004, SSCODE=00004
DBDN———->A11-AN                TYPE=1, LEN=00003, SCODE=00005, SSCODE=00005
DBDN———->A12-AN                TYPE=1, LEN=00003, SCODE=00006, SSCODE=00006
DBDN———->A13-AN                TYPE=1, LEN=00003, SCODE=00007, SSCODE=00007
DBDN———->R02-AKEY              TYPE=2, LEN=00001, SCODE=00002, SSCODE=00002
*** Non-fatal ERROR-NUM table ***
ERROR-NUM = 6
ERROR-NUM = 7
ERROR-NUM = 13
**** Dump of RDA follows:
0001    060060060063 062060061060 067066103165 163164157155 0003201076Custom
0005    145162040116 125115102105 122040060060 060063062040 er NUMBER 00032
0009    040040040040 040040040040 040040040040 040040040040
0013    040040040040 040040040040 040040040040 040040040040
0017    040040040040 040040040040 040040040040 040040040040
0021    040040040040 040040040040 040040000000 060060060060           ??0000
0025    060060060060 060060061061 120101122124 040104105123 00000011PART DES
0029    103122111120 124111117116 040116125115 040060060060 CRIPTION NUM 000
0033    060060060060 061061060060 060060060060 061060000000 000011000000010??
0037    000012060060 060060060060 060063040040             ??00000003
**** Remainder of RDA is unused ****
**** End of object dump ****
  21   7    20C  EXIT
** YOU HAVE NOT DEPARTED - DEPARTING WITH NO ROLLBACK.  **
Error-status = 000000 Error-num = 0000 Current page/rec = 000001/00001
** Exiting I-QU PLUS-1 **
```

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. . . . . .

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

Attn: Technical Documentation Section