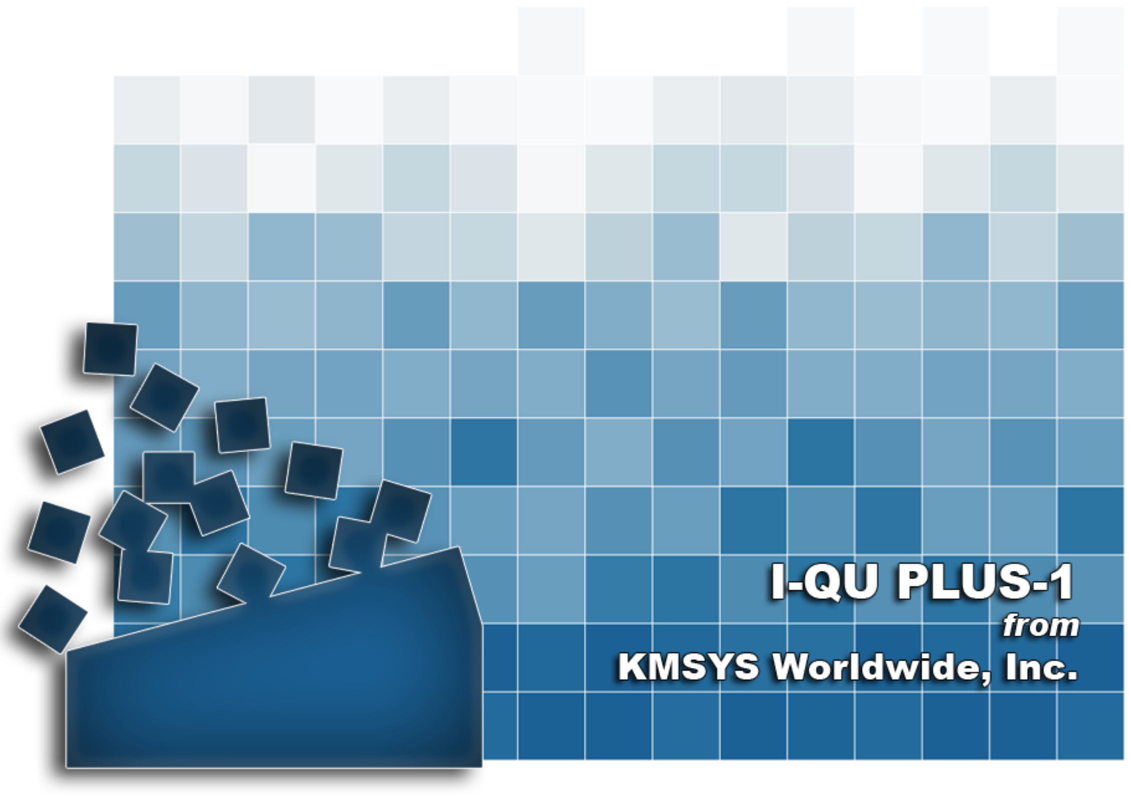


Programmer Reference

10 August 2011

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

©Copyright 1985-2008 by KMSYS Worldwide, Inc. All rights reserved.

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

eQuate, Host Gateway Server, I-QU PLUS-1, I-QU ReorgComposer, InfoQuest, InfoQuest Client, Q-LINK, QPlex, QPlexView, T27 eXpress IT, T27 eXpress Net, T27 eXpress Plus, T27 eXpress Pro, UTS eXpress IT, UTS eXpress Net, UTS eXpress Plus, UTS eXpress Pro and WinQ are trademarks or registered trademarks of KMSYS Worldwide, Inc. Microsoft, Windows, Visual Basic and Visual C++ are trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries. Delphi is a trademark of Borland International. Sperry, Unisys, UTS, UNISCOPE and BIS are trademarks of Unisys Corporation. Enable is a trademark of Cypress Software, Inc. All other trademarks and registered trademarks are the property of their respective owners.

RESTRICTED RIGHTS LEGEND

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

Table of Contents

Tables

Preface

This manual contains technical information regarding the use of the I-QU PLUS-1 processor. This manual is directed to programming and database administrative personnel. The manual contains information necessary to use the I-QU PLUS-1 processor interactively or to write I-QU PLUS-1 programs.

For those sites that are migrating from earlier release levels of I-QU PLUS-1, please read the I-QU PLUS-1 README.TXT file on the Documentation CD from KMSYS Worldwide prior to executing existing I-QU PLUS-1 programs. Some of the differences described in that document may effect the I-QU PLUS-1 operating environment at your site.

Any I-QU PLUS-1 programs SAVEd in object form with earlier released levels of I-QU PLUS-1 must be recompiled to execute under this released level. KMSYS Worldwide, Inc., provides an SSG routine called "COMPILE-RUNS" which will automatically recompile all I-QU PLUS-1 programs. This routine can be found in the second I-QU PLUS-1 product file, SYS$LIB$*IQU-1 (default mode install). For other modes of installation, refer to the COMUS SRL after COMUS product registration.

With DMS-1100, level 9R1 or higher, the term, "DMR", is no longer used. Instead, Unisys documentation now refers to a logical data manager (LDM) for DMS running under Universal Data System (UDS) Control. I-QU PLUS-1 can interface with any number of LDMs through UDS Control. However, this documentation will continue to use the term, "DMR", in a generic fashion since I-QU PLUS-1 can interface with non-UDS DMS applications (e.g., 8R3) as well as DMS applications running under UDS Control.

Chapter 1: Introduction

I-QU PLUS-1 is a high level, multi-mode database manipulation processor. I-QU PLUS-1 has been designed to give the programmer or analyst the ability to interactively retrieve, display, add, change and delete data in a multi-file system environment using simple procedural commands. This data can be accessed in a DMS 2200 database, standard PCIOS files, TIP/FCSS files, RDMS 2200 database tables and BIS reports via the DTM interface. The user may also write I-QU PLUS-1 command language programs for performing data manipulation or for producing lists and summaries without having to use compilers or collectors.

I-QU PLUS-1 is especially useful in systems development because of its ability to display, load, modify or dump data. Program test results can be easily verified after testing COBOL/DML or standard COBOL programs, without the need to write additional COBOL programs. Data can be written to PCIOS files from a test database, and used to refresh the database when needed, using I-QU PLUS-1 command language programs. I-QU PLUS-1 can be used in the production of ad hoc reports using its flexible output formatting commands.

I-QU PLUS-1 has an optional function that provides the more technical user with a powerful database reorganization utility package. An important feature of I-QU PLUS-1, when the optional I-QU PLUS-1 extensions are included, is its set of powerful DMS 2200 database pointer manipulation commands. These commands are designed to be used in conjunction with a group of special database reorganization utilities to speed up and simplify large scale database reorganizations.

1.1 Description of Chapters

Chapter 1 contains an introduction to the I-QU PLUS-1 processor, a summary of each chapter of the Programmer Reference and the rules for its use.

Chapter 2 contains general information regarding the basic structure of the I-QU PLUS-1 processor, its operation modes, its file systems' interfaces.

Chapter 3 describes how the I-QU PLUS-1 Processor is organized internally. It also contains the rules by which data items may be referenced in an I-QU PLUS-1 program or session.

Chapter 4 lists the I-QU PLUS-1 Processor call options and format.

Chapter 5 describes the types of user-defined variables that can be used within an I-QU PLUS-1 program or session.

Chapter 6 lists the system defined (reserved) variables that can be referenced within an I-QU PLUS-1 program or session.

Chapter 7 describes how to utilize the Record Delivery Area (RDA).

Chapter 8 contains all control directives other than DEFINE and INVOKE which are covered in chapters that are more appropriate to their use.

Chapter 9 contains all of the I-QU PLUS-1 procedural commands except those used for file I/O that are discussed in succeeding chapters.

Chapter 10 includes the directives and commands used for defining and accessing flat files, PCIOS and SFS 2200 files.

Chapter 11 describes the internal sort interface that may be used within an I-QU PLUS-1 batch program.

Chapter 12 contains all the directives and commands necessary to interface with DMS 2200 either through the standard multi-thread interface provided by UDS Control or the single-thread interface that may be collected with I-QU PLUS-1.

Chapter 13 describes the EXEC/FCSS/TIPDMS direct I/O (if configured) that may be used with any 2200 file structure.

Chapter 14 contains the syntax requirements to use the RDMS 2200 interface.

Chapter 15 describes the directives and commands required to use the BIS (MAPPER) DTM interface.

Chapter 16 lists all the command and keyword abbreviations that may be used through the I-QU PLUS-1 environments.

Chapter 17 illustrates the use of the special file format, DBDUMP, available only through I-QU PLUS-1.

Chapter 18 describes the I-QU PLUS-1 utility, QINDEX, used to build and maintain a data item index file, its processor call format, options and parameters.

1.2 Additional Documentation

The following manuals are available with the release of I-QU PLUS-1:

- I-QU PLUS-1 Applications Development User Guide
- I-QU PLUS-1 Installation Guide
- I-QU PLUS-1 Database Reorganization User Guide
- I-QU PLUS-1 Database Reorganization Utility Reference

The two User Guides contain many examples illustrating I-QU PLUS-1 being used for applications development, prototyping, program testing, restructuring and database reorganization.

1.3 Syntax Notation

The following conventions are used throughout this manual in the description of I-QU PLUS-1 commands:

- Changes to this document since its last publication are marked with a change bar (an elongated vertical bar) as shown to the right of this paragraph.
- Important notes and warnings are encased in a box as shown around this bullet.
- All words in UPPERCASE letters (not italicized) are reserved keywords and must be entered exactly as shown or in their abbreviated forms.
- Many keywords may be abbreviated. Please refer to Chapter 16, "Command and Keyword Abbreviations," for a full list of permissible abbreviations. In addition, abbreviations will be highlighted in BLUE throughout the syntax shown in this manual. For example, DEFINE may be abbreviated as "DEF".
- All italicized words (mostly in lowercase letters) are to be substituted by a user supplied name or value.

- Ellipsis (…) implies allowable, but omitted, repetitions in the published syntax. Please note that the ellipsis is ***<u>not</u>*** allowed in the command or directive when parsed. In the following example, multiple set names may be specified:

  set-name-1 [… *set-name-n*]
- A vertical bar (|) represents an "or" or "and/or" operator.
- Selections appearing within brackets, "[ ]", are lists of optional items of which one may be selected. In the following example, neither A nor B is required, but either one or the other may be selected:

  [A | B]
- An underlined word in optional brackets represents the default value when not entered. For example, in the syntax, "[<u>RECORDS</u> | CHARACTERS]," RECORDS is the default and would be assumed if omitted.
- Selections appearing within braces, "{ }", are lists of items of which one and only one must be selected. In the following example, one of either C, D or E must be selected:

  {C | D | E}
- Selections appearing within double vertical bars, "||", are lists of items of which one or more must be selected. The items between vertical bars are referred to as permutations and may be selected in any order. In the following example, one or more of F, G, and/or H must be selected:

  ||F | G | H||

  To use the SUPPRESS clause as an example:

```
SUPPRESS {ALL | ;
          || AREA | ;
             RECORD | ;
             {SET | ;
              set-name-1 [… set-name-n] } || }
```

  Either of the following would be correct:

```
. . .  SUPPRESS AREA RECORD
. . .  SUPPRESS RECORD AREA
```

The following example uses several of the above command notation rules:

DO procedure [{UNTIL | WHILE} *conditional-expression*]

In this example, the word DO is the command keyword and must be entered exactly as shown. The procedure is user-defined and must always be present. The UNTIL/WHILE clause is optional as indicated by the block formed by square brackets. If the UNTIL/WHILE clause is used, either the keyword WHILE or UNTIL (not both) must be entered as indicated by the block formed by braces, and a valid *conditional-expression* must follow. The following examples are possible forms of this command syntax:

```
DO TOTAL-ROUTINE .  (This is the minimal form)
DO TOTAL-ROUTINE UNTIL CNT NOT = 1
DO TOTAL-ROUTINE WHILE CNT < LIST-RESULT
```

Chapter 2: General Information

2.1 Basic Structure

The I-QU PLUS-1 processor is logically divided into two major components: the COMMAND EDITOR and the COMMAND EXECUTOR. The COMMAND EDITOR checks each command entered for correct syntax and attempts to translate it into an internally encoded object command. If an error is detected during this process, an error diagnostic is displayed and the command is rejected. If the command passes all edits, it is either passed to 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 the processing of control directives, which will be discussed later.

The COMMAND EXECUTOR interprets the encoded object command passed from the COMMAND EDITOR and actually performs the operation. The COMMAND EXECUTOR has been designed to do the least amount of interpretation possible, therefore achieving maximum throughput during program execution.

2.2 Modes of Operation

I-QU PLUS-1 commands may be entered in two modes: CONVERSATIONAL or INPUT. In CONVERSATIONAL mode, each command is edited and executed immediately. Command results are then displayed to the user. If a DML command is executed, the DMS 2200 command 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 and DO/ENDDO pairs. If I-QU PLUS-1 is unable to resolve any of these, or an error was detected during input and the error flag has not been CLEARed, I-QU PLUS-1 will not attempt to execute the object program. Otherwise, the command executor will begin executing the object program. All directives, including data definition directives, will be processed immediately in either CONVERSATIONAL or INPUT mode.

2.3 Commands vs. Directives

There are two types of control statements in I-QU PLUS-1: commands and directives.

Directives are used to establish and control the program environment of the I-QU PLUS-1 program. These are always executed during the “Command Editor” phase of the I-QU PLUS-1 program; i.e., immediately upon being received by the I-QU PLUS-1 processor regardless of the mode. They are used to:

- Define files, records, buffers and fields;
- Set the I-QU operating mode (conversational vs. input);
- Invoke subschemas and data item index files;
- Initialize data storage areas;
- Suppress or allow the listing of commands;
- Dump I-QU PLUS-1 internal areas;
- Run a program stored in the object program area or from an object library;
- Save a program in the object program area to an object library;
- Compile a program to resolve program label references.

Directives must always be entered beginning in column position one.

Commands are used to perform data manipulation, program flow control, input/output, etc. They are executed during the “Command Executor” phase of the I-QU PLUS-1 program.

When in INPUT mode, commands must begin after column one. When in CONVersational mode, commands must begin in column one. Directives (regardless of the mode) and program labels must begin in column one.

2.4 Command Line Format

Command formats must follow certain rules. In general, commands may be entered in free form. Restrictions are as follows:

- All directives must be entered starting in column 1.
- All procedural commands must be entered starting in column 1 when in CONVERSATIONAL mode, and after column 1 in INPUT mode.
- In INPUT mode, any word starting in column 1 will be regarded as either a control directive (if it matches one of the valid control directives) or a program or procedure label.
- A program or procedure label may be up to 30 characters in length. However, only the first 16 characters will be used by I-QU PLUS-1, and therefore must be unique.
- Only one command may be entered per line.
- A decimal numeric literal may not exceed 18 digits, and if signed, the sign must immediately precede the first digit. Example: 123, -5 or +1000.00.
- Floating-point numeric literals must be entered in standard floating-point notation with a decimal point. Example: 1.E-10.
- Alphanumeric literals may not exceed 150 characters. In addition, they must be enclosed within single or double quotes. If an alphanumeric literal definition begins with a double quote, the single quote (apostrophe) character may be used in the literal, and vice versa. Additionally a string of two single ('') or double ("") quotes within a literal will generate one single or double quote.

  Alphanumeric literal examples:

  ```
      'THIS IS AN ALPHANUMERIC LITERAL'
      "This is also 'ONE' literal."
  ```

```
'AND "THIS IS A LITERAL" TOO!'
'This is a single quote ('') within parentheses'
```

- All command components must be entered between columns 1 through 80. A semicolon following any word within a command line will cause I-QU PLUS-1 to skip to the next line to search for the next word of the command; therefore, line continuation is possible using the semicolon as follows:

```
IF    RDA CUSTOMER-NAME ;
OF CUSTOMER = 'APEX'
```

- To continue an alphanumeric literal, end the first part with a single (or double) quote followed by a semicolon. Note: A space must not precede the semicolon. Continue the literal anywhere on the following line enclosed in quotes. For example:

```
'...PQRSTUV';
'WXYZ'
```

- A period, followed by a space, is used in a command line to separate the command from commentary text. I-QU PLUS-1 will ignore any text following a period that is not embedded within a literal. Examples of comments are:

```
DO HEADER-RTN  .  This a Comment on a Command Line
.  *** This entire line is a Comment ***
```

2.5 File Systems Accessed by I-QU PLUS-1

All DMS 2200 database record placement strategies are supported by I-QU PLUS-1. There is an I-QU PLUS-1 Data Manipulation Language (DML) command corresponding to each COBOL/DML command. However, I-QU PLUS-1 syntax may differ from COBOL/DML, and I-QU PLUS-1 commands can be entered in an abbreviated format.

The I-QU PLUS-1 Processor can be used to read, write, and update standard Processor Common Input/Output System (PCIOS) files. PCIOS files may be accessed in combination with DMS 2200 database areas and other file systems, or separately.

I-QU PLUS-1 also supports access to other file types through its Direct I/O (DIO) feature. This feature provides the ability to access TIP/FCSS, TIP/DMS, or any sector-formatted EXEC mass-storage file or any tape file.

RDMS 2200 relational databases can be accessed through two I-QU PLUS-1 commands that provide RDMS 2200 interface information. This information is comprised of an RDML (or SQL) formatted command, RDML error status variables and application program variables (table column names, table column values, etc.). All RDML formatted commands are supported and may be used in conjunction with I-QU PLUS-1 commands that interface with other file systems.

The I-QU PLUS-1 processor can be used to read and write BIS reports via an interface with Data Transfer Module (DTM) of BIS. I-QU PLUS-1 allows “queue-aliases” to be defined in order to access more than one BIS report simultaneously through the same BIS queue. The DTM interface may be used in conjunction with any other I-QU PLUS-1 file system interface.

Chapter 3: Internal Structure

I-QU PLUS-1 internal storage is organized into three areas of interest to the user:

- Record Delivery Area (RDA);
- Variable Data Storage Area;
- Object Program Area.

This chapter will describe how these areas are used.

3.1 Record Delivery Area (RDA)

The RDA is used for all input and output for DMS2200 database records, PCIOS file records and and all other file interfaces. The size of this area will vary depending upon how I-QU PLUS-1 was generated. It is important to remember that all read, write, fetch, modify, etc. operations assume that the data record begin in the first position of this area. If multiple records are to be resident simultaneously, then data from the RDA can be moved to variable data storage prior to a subsequent I/O. Another way to access multiple records simultaneously is to define alternate record areas using the “DEFINE RA” directive. This directive allows the user to define alternate areas of the RDA to be used for I/O for specific records or files. See the DEFINE RA directive for more information.

Since the RDA will be generated larger than any record that may be read, the upper ranges may be used for additional workspace, and the allocation of tables, etc. When issuing the WRITE command, information may be built in an unused area of the RDA and appended to the record when it is written.

The RDA may be referenced in two ways: directly by character position, or by the data item name. The following sections describe RDA referencing in detail.

3.1.1 Direct RDA Reference

When using the direct RDA reference, the starting byte (or character) position, field length, data type, and subscript or index offset are specified.

The direct RDA reference format is:

RDA (*starting-byte-position*,*length*) [*data-type*] [:*index* | :*subscript*]

The *starting-byte-position* is the offset (relative to 1) of the referenced data from the beginning of the RDA or of a record within the RDA.

The *length* is in bytes and is the actual length of the referenced data. For computational data items, this is the actual byte length of the data, not the COBOL picture length. For example, if the COBOL picture is PIC 9(10) COMP, the actual byte length is 4 (assuming the data is ASCII). Table 3.2 may be used to determine the byte length of various computational field definitions.

The *index* specifies a numeric literal, or a decimal integer numeric variable, immediately following a colon (:). Refer to page 3-6 for detailed information regarding RDA indexing.

The *subscript* specifies the name of a defined subscript variable to be used in addressing the data item. Refer to page 3-6 for detailed information on RDA subscripting.

Data within the RDA may be referenced in ASCII (quarter word, 9-bit bytes) or FIELDATA (sixth word, 6-bit bytes) form. Data representation is indicated by the *data-type* field as shown in the following table:

Data Type	**Description**
DISP	ASCII alphanumeric display (default).
A9	ASCII alphanumeric display.
SN9	Signed ASCII numeric display.
UN9	Unsigned ASCII numeric display.
COMP	Signed ASCII aligned binary.
SB9	Signed ASCII aligned binary.
UB9	Unsigned ASCII aligned binary.
DISP-1	Fieldata alphanumeric display.
A6	Fieldata alphanumeric display.
SN6	Signed fieldata numeric display.
UN6	Unsigned fieldata numeric display.
COMP-4	Signed fieldata aligned binary.
SB6	Signed fieldata aligned binary.
UB6	Unsigned fieldata aligned binary.
COMP-1	Single-precision floating point.
FP1	Single-precision floating point.
COMP-2	Double-precision floating point.
FP2	Double-precision floating point.
DISP-2 or A18	Kanji. Note: For alignment purposes within the RDA, you must specify two bytes for each double-byte character required; therefore, if you have three Kanji double-byte characters, you would specify RDA (n,6) DISP-2.
MAPNUM	BIS numeric data format.

Table 3-1: Data Types

Please note that exact binary (i.e., PIC 1(3), etc.) is not currently supported in I-QU PLUS-1; however, exact binary is supported in QINDEX for alignment purposes only (see Chapter 18).

If using the ASCII data code, the *starting-byte-position* and *length* must be given as ASCII. If using the FIELDATA data code, locations must be given in terms of FIELDATA lengths.

The number of bytes required for signed aligned binary fields can be determined by utilizing the following table:

ASCII (COMP)		**FIELDATA (COMP-4)**	
Positions in Picture Clause	**Record Area Bytes**	**Positions in Picture Clause**	**Record Area Bytes**
1-2	1	1	1
3-5	2	2-3	2
6-7	3	4-5	3
8-10	4	6	4
11-13	5	7-8	5
14-15	6	9-10	6
16-18	7	11-12	7
		13-14	8
		15	9
		16-17	10
		18	11

Table 3-2: Computational Field Byte Lengths

COMP-1 and COMP-2 are always 4 bytes and 8 bytes respectively for ASCII alignment, and 6 bytes and 12 bytes for FIELDATA alignment (one and two words).

MAPNUM items must be set as if they are alpha items; however, they can be used to set numeric items and must be tested as numeric items (see the IF and SET commands).

Here are some examples of RDA direct references with the following data in the RDA:

```
1...5....10...15...20...25...30...35...
ABCDEFGHIJKLMNOPQRSTUVWXYZ123456789

1. RDA (2,3)
2. RDA (27,2) COMP
3. RDA (5,4) UB9
4. RDA (1,5) :X
5. RDA (7,6) COMP-4
6. RDA (2,9) MAPNUM
```

Explanations:

Example 1 would refer to the 3-character field starting in position 2 of the RDA, the value of which is “BCD” assuming the above data is ASCII.

Example 2 would refer to an ASCII computational field of two bytes starting in position 27.

Example 3 would refer to the second full word in the RDA as unsigned binary.

Example 4 would refer to positions 1 through 5 offset by the value of X. IfX=5,the value would be “FGHIJ” assuming the above data is ASCII (see the subsection “RDA Indexing and Subscripting” below).

Example 5 would refer to a full word FIELDATA computational field starting in byte position 7 (word 2 of the RDA).

Example 6 refers to an RDA field in BIS numeric data format. BIS numeric data is always in edited form, which may include a leading sign, spaces and decimal point. An example of a MAPNUM data item would be ‘ -123.58’.

3.1.2 Item Name RDA Reference

The second method of referencing the RDA uses the I-QU PLUS-1 data item index file. A primary data item index is created automatically during INVOKE processing for DMS database items (see "Subschema Invocation (INVOKE)" in the "DMS2200 Interface" chapter). A secondary data item index may also be specified via the INDEX directive for non-DMS data items and redefined DMS database items. The QINDEX Processor is used to create data item index files from COBOL definitions. Refer to Chapter 18, "QINDEX Reference," for an explanation on how to create alternate I-QU PLUS-1 data item index files. To use this method of referencing the RDA, it is necessary to specify the data item name and its index offset or subscript, if any.

The format for a named RDA reference is:

RDA item-*name* [{IN | OF} {*record-name* | *file-name* | *queue-alias* | SORT}] ;
[:*index* | :*subscript*]

The *item-name* must be defined in one of the data item index files currently in use (built by the INVOKE and INDEX directives) or by a previously issued DEFINE RDA directive.

The IN/OF option allows the *item-name* to be qualified to the proper database *record-name*, PCIOS *file-name*, BIS DTM *queue-alias* or SORT area.

3.1.3 RDA Indexing and Subscripting

An RDA index specifies a byte offset to be used when addressing the data. The index may be entered as a numeric literal, or a numeric variable, immediately following a colon (:).

Subscripting differs from indexing in that a subscript refers to an occurrence of an item within a table of items, while an index refers to a number of bytes from the start of the item. Subscripting in I-QU PLUS-1 is similar to that used in COBOL, except that a defined subscript variable must only be used to reference the OCCURS items for which it was defined. RDA subscripting may be used on multidimensional RDA references, whereas RDA indexes may not.

The index or subscript options are used in the same way as with a direct reference in specifying a byte offset from the beginning of the field (indexing), or a specific occurrence (subscripting).

Given the following portion of a record definition:

```
05  RATE-TABLE.
    10  RATE OCCURS 10 TIMES PIC 9(5).
05  HIGHEST-RATE             PIC 9(5).
```

An example of indexed references follows:

To refer to the first RATE, simply enter "RATE".

To refer to the second RATE, enter "RATE :5". (RATE offset by five bytes.)

To refer to any RATE, use "RATE :RATE-INDEX", where RATE-INDEX is a numeric variable used to index to the desired occurrence of RATE.

To use subscripted reference, a subscript variable must be defined using the DEFINE SUB directive. The defined subscript variable may then be used in much the same manner as a COBOL subscript.

The following program example illustrates the necessary I-QU PLUS-1 code to examine all 10 occurrences of RATE in the above record definition using RDA indexing:

```
DEF N RATE-CTR
DEF N RATE-INDEX
.....
  SET RATE-CTR = 0
  SET RATE-INDEX = 0
  DO RATE-LOOP UNTIL RATE-CTR = 10
.....

RATE-LOOP PROCEDURE
  DISPLAY RDA RATE :RATE-INDEX
  SET RATE-CTR = RATE-CTR + 1
  SET RATE-INDEX = RATE-INDEX + 5
  ENDPROC
```

Note that the numeric variable RATE-INDEX is incremented by the byte length of the field being examined, and not by one (1) as in standard COBOL subscripting or indexing.

The following is the same example using RDA subscripting:

```
DEF SUB RATE-SUA RATE
.....
  SET RATE-SUB = 1
  DO RATE-LOOP WHILE RATE-SUB NOT = 10
.....
RATE-LOOP PROCEDURE
  DISPLAY RDA RATE :RATE-SUB
  SET RATE-SUB = RATE-SUB + 1
  ENDPROC
```

In this case, a subscript variable, RATE-SUB, was defined for use in referencing the table. RATE-SUB is incremented the same way as a COBOL subscript. RATE-SUB may only be used to reference the table for which it was defined.

3.1.4 Variable RDA Referencing

A direct RDA reference allows the use of a variable *start-byte-position* and/or *length*. To use this feature, two reserved numeric variables, S$ and L$, have been provided. The contents of these variables will be used in a direct RDA reference whenever a zero *start-byte-position* and/or *length* are specified. The value of S$ will be substituted for a *start-byte-position* of zero; the value of L$ will be substituted for a length of zero.

Caution: If the values in either S$ or L$ cause an invalid RDA reference, runtime errors will result.

Examples:

```
1. RDA (0,0)
2. RDA (20,0) DISP-1
```

In the case of Example 1, both the *start-byte-position* and *length* will be obtained from the reserved variable S$ and L$.

Example 2 would refer to a FIELDATA field starting in position 20 with the length of the value in the reserved variable L$.

3.2 Variable Data Storage Area (DSA)

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

3.3 Object Program Area

The Object Program Area is where the internally encoded object program is stored for execution by the command executor. The size of this area is specified at generation time.

Chapter 4: Processor Call Format

I-QU PLUS-1 is called as a processor using the following format:

@IQU,[*options*][*password*]

The processor name will vary depending on a configuration parameter or installation mode used when I-QU PLUS-1 was installed. The default processor name for the default installation mode is shown above. Default names for installation modes IQUA through IQUK correspond to the mode chosen; e.g., a mode IQUA install would produce a default processor name of IQUA.

Valid options are:

A	Causes I-QU PLUS-1 to abort if a RUN directive is attempted and errors have been detected during command editing or program resolution. This option should be used if I-QU PLUS-1 is part of a multi-step runstream and it is not desirable to continue in the event of an abnormal step termination.
B	Causes the execution of I-QU PLUS-1 to be treated as a batch mode execution. All I-QU PLUS-1 output will use the 132-character print width.
C	Automatically puts I-QU PLUS-1 into CONVERSATIONAL mode, even if generated for initial mode of INPUT.
D	Causes the execution of I-QU PLUS-1 to be treated as a demand mode execution. This option is the converse of the B-Option. When used, it will cause I-QU PLUS-1 not to produce an object print and then abort when a fatal runtime error is encountered.
E	Causes input to be echoed when in CONVERSATIONAL mode.
I	Automatically puts I-QU PLUS-1 into INPUT mode, even if generated for initial mode of CONVERSATIONAL.
K	Causes the object program table area to be dumped when an OBJECT directive or DUMP ALL command is executed. The object program area is not normally dumped, as it is of very little use to the I-QU PLUS-1 programmer.
O	When interfacing with DMS 2200, execute in DMS test mode.
T	For KMSYS Worldwide debugging only. Use only if directed to by KMSYS Worldwide personnel.
U	Inhibits listing of command input when in INPUT mode. This is useful when running a tested I-QU PLUS-1 program and commands do not need to be listed.
V	Disables the ability to do II-keyins.
X	For KMSYS Worldwide debugging only. Use only if directed to by KMSYS Worldwide personnel.
Y	When interfacing with DMS 2200, execute in training mode.

The password entry is only required when the password feature of I-QU PLUS-1 security is enabled. It is used to control DMS 2200 subschema access and is set up during the I-QU PLUS-1 generation process (see the I-QU PLUS-1 Installation Guide).

Chapter 5: Defining Variables

I-QU PLUS-1 does not allow implicit variable definition. All variables must be defined using the DEFINE directive prior to being referenced. The DEFINE directive may be entered at any point, but must appear before the variable is referenced. There are only four types of data variables: signed decimal numeric, double precision floating point numeric, alphanumeric strings and double-byte Kanji.

5.1 Alphanumeric String Variables (DEFINE A)

Alphanumeric variables may be defined as any length; however, they may not exceed the generated size of the variable data storage area. They may be specified with an initial literal value from 1 to 150 characters in length. When alphanumeric variables are defined with a specified length and no initial value, their initial contents will be undetermined. If an initial literal is specified with no length, the size of the variable will be the size of the literal.

Format:

DEFINE A *variable-name* { || *length-of-variable* | '*string-literal*' || }

DEFINE may be abbreviated DEF.

If both *string-literal* and *length-of-variable* are specified, and *length-of-variable* is greater than the length of *string-literal*, the *string-literal* will be left justified and the variable will be padded with spaces to the right. If *length-of-variable* is less than *string-literal*, truncation will occur.

Examples:

```
1. DEFINE A STARS '***********'
2. DEFINE A HOLD-REC 80
3. DEFINE A COMPANY-VARIABLE 80 'KMSYS Worldwide, Inc.'
```

Explanations:

STARS in example 1 will contain 11 asterisks and is equivalent to:

```
DEFINE A STARS '***********' 11
```

In example 2, HOLD-REC will have a length of 80, but its initial contents will be unknown.

In example 3, COMPANY-variable will initially be set to “KMSYS Worldwide, Inc.” but will have a length of 80.

5.2 Decimal Numeric Variables (DEFINE N)

All decimal numeric variables are allocated as 18-digit signed numbers. They may be defined with an initial value. If a value is not specified, the variable is initialized to zero.

Format:

DEFINE N *variable-name* [*decimal-value-and-precision*]

DEFINE may be abbreviated DEF.

The *decimal-value-and-precision* determines both the initial value of the variable and the number of implied decimal positions (scale factor) it will have.

Examples:

```
1. DEFINE N COUNT 123
2. DEFINE N MINUS-FIVE -5
3. DEFINE N HOLD
4. DEFINE N TOT-AMT .00
```

Explanations:

In example 1, COUNT will have an initial value of 123.

In example 2, MINUS-FIVE will be assigned an initial value of -5.

In example 3, HOLD will have an initial value of zero.

In example 4, TOT-AMT will have an initial value of zero, and two implied decimal positions.

5.3 Floating Point Numeric Variables (DEFINE FP)

All floating-point variables are stored in double precision format for maximum accuracy. When floating-point variables are moved to or from RDA single precision fields or decimal numeric items, translation will be made according to ASCII COBOL rules.

Format:

DEFINE FP variable-name [value]

DEFINE may be abbreviated DEF.

The *value* of a floating-point variable may be specified as in either decimal form, or standard floating-point notation.

Examples:

```
1. DEFINE FP VARIANCE 1.2400E+2
2. DEFINE FP RESULT
```

Additional information on floating-point usage may be found on the EDIT command.

5.4 Kanji Variables (DEFINE K)

Kanji variables may be defined as any length, but may not exceed the generated size of the variable data storage area. They may be specified with an initial literal value from 1 to 75 double-byte characters in length. When kanji variables are defined with a specified length and no initial value, their initial contents will be undetermined. If an initial literal is specified with no length, the size of the variable will be the size of the literal.

Format:

DEFINE K *variable-name* { || *length-of-variable-in-double-bytes* | '*string-literal*' || }

DEFINE may be abbreviated DEF.

If both *string-literal* and *length-of-variable-in-double-bytes* are specified, and *length-of-variable-in-double-bytes* is greater than the length of *string-literal*, the *string-literal* will be left justified and the variable will be padded with spaces to the right. If *length-of-variable-in-double-bytes* is less than *string-literal*, truncation will occur.

Chapter 6: Reserved Words

I-QU PLUS-1 maintains a set of reserved words that can be used on many I-QU PLUS-1 commands. These reserved words fall into two categories: reserved variables and special names.

6.1 Reserved Variables

I-QU PLUS-1 automatically reserves a set of variables upon initialization. These variables occupy the same storage area as user variables, and may be accessed and modified in the same manner. The following is a list of reserved variables and an explanation of how each are used:

Reserved Name	**Description**
$TAB	$TAB is a one-character alpha variable initially set to octal 011 (decimal 9), which represents a horizontal tab character. This may be used for formatting a BIS output line instead of the automatic tab insertion feature.
C-AKEY	C-AKEY is a numeric variable with an initial value of 0. It will automatically be set to the AREA-KEY of the DMCA after the execution of a DML command.
C-AREA-NAME	C-AREA-NAME is a twelve-character alpha variable with an initial value of spaces. It is automatically set to the current AREA-NAME from the DMCA after all DML commands.
C-DBK	C-DBK is a numeric variable with an initial value of 0. It is set to the current of run unit database key when the SET CURRENT DBK command is executed.
C-DBP	C-DBP is a numeric variable with an initial value of 0. It is set to the current of run unit database pointer when the SET CURRENT DBP command is executed. NOTE: SET CURRENT DBP is part of the optional pointer manipulation command set and will not have any effect if pointer manipulation is not generated.
C-O-R	C-O-R is a thirty-character alpha variable with an initial value of spaces. It will be set automatically after the execution of DML commands and will contain the current RECORD-NAME from the DMCA.

Reserved Name	**Description**
C-O-T	C-O-T is a thirty-character alpha variable with an initial value of spaces. It is automatically set after a READ of a DBDUMP file. It will contain the record name of the record read.
C-PAGE	C-PAGE is a numeric variable with an initial value of 0. It will automatically be set to the current PAGE-NUM of the AREA-KEY after each DML command.
C-REC	C-REC is a numeric variable with an initial value of 0. It will automatically be set to the current RECORD-NUM of the AREA-KEY after each DML command.
DATE	DATE is an alpha variable with a length of 10 characters. It is initially set to the current Gregorian date in edited form (example: July 25, 1994 = “1994/07/25”). This variable is altered by the DATE command.
DATE-NUM	DATE-NUM is a numeric variable with an initial value of the current Gregorian date in the form YYYYMMDD. This variable is altered by the DATE command.
DAY	DAY is a numeric variable with an initial value of the current day of the month. This variable is altered by the DATE command.
ERROR-NUM	ERROR-NUM is a numeric variable with an initial value of 21. It will automatically be set to the value of ERROR-NUM from the DMCA after the execution of all DML commands.
G-AKEY	G-AKEY is a numeric variable with an initial value of 0. It is not set by I-QU PLUS-1, but may be used by the I-QU PLUS-1 user as desired.
G-AREA-NAME	G-AREA-NAME is a twelve-character alpha variable with an initial value of spaces. It is not used by I-QU PLUS-1, but may be used by the I-QU PLUS-1 user as desired.
G-PAGE	G-PAGE is a numeric variable with an initial value of 0. It is not set by I-QU PLUS-1, but may be used by the I-QU PLUS-1 user as desired.
G-REC	G-REC is a numeric variable with an initial value of 0. It is not set by I-QU PLUS-1, but may be used by the I-QU PLUS-1 user.
G-RECORD-NAME	G-RECORD-NAME is a thirty-character alpha variable with an initial value of spaces. It is not used by I-QU PLUS-1, but may be used by the I-QU PLUS-1 user. G-RECORD-NAME may be initialized with the change file name when executing I-QU PLUS-1 in DMS test mode. In this instance, an @USE statement is required to get the change file qualifier (see Section 9.8), and the O-option is required on the I-QU PLUS-1 processor call (see Chapter 4).
IICODE	IICODE is an eight-character alpha variable initially set to spaces. Upon receipt of an immediate interrupt (II), the value entered with the II-key-in will be moved here. EXEC II-key-ins are limited to 6 characters. It may be used within a program to take certain actions based on unsolicited console key-ins.

Reserved Name	**Description**
IMPART-DEPART	IMPART-DEPART is a numeric variable with an initial value of 0. It is set to 1 after the execution of the DML IMPART, and to 2 after the execution of the DEPART command.
J-DAY	J-DAY is a numeric variable with an initial value of the current Julian date in the form YYYYDDD. This variable is altered by the DATE command.
L$	L$ is a numeric variable with an initial value of 0. It is used to replace the *length* value in a direct RDA reference if a *length* of zero is specified.
MONTH	MONTH is a numeric variable with an initial value of the current month. This variable is altered by the DATE command.
RB-CODE	RB-CODE is a numeric variable with an initial value of zero. It will be set to the rollback error code when a DMS rollback error occurs.
REC$LEN	REC$LEN is a numeric variable with an initial value of 0. It is automatically set to the record length (in words) after each sequential file read. In the case of a DBDUMP file, only the length of the data portion of the record is represented. The CALSIM command also uses this variable to return the resulting chain number from the CALC simulation routine.
RUNID	RUNID is a 6-character alpha variable initially set to the original RUN-ID under which the I-QU PLUS-1 Processor is running. RUNID is not the generated run-id created by the operating system when two runs having the same run-id are started.
S$	S$ is a numeric variable with an initial value of 0. It is used to replace the start-byte value in a direct RDA reference if a start-byte of zero is specified.
TIME	TIME is a twelve-character alpha variable with an initial value of spaces. It is set to the time of day in display format by execution of the TIME command. The format is HH:MM:SS.DDD.
TIME-MSPM	TIME-MSPM is a numeric variable with an initial value of 0. It is set to the time of day in milliseconds past midnight (MSPM) by execution of the TIME command.
YEAR	YEAR is a numeric variable with an initial value of the current year (example: 1994). This variable is altered by the DATE command.
X,Y and Z	X, Y and Z are numeric integer variables with an initial value of zero with no implied decimal positions. They are set up for the convenience of the user and may be used for any purpose.

6.2 Special Names

I-QU PLUS-1 automatically reserves a set of special names that can be used on several I-QU PLUS-1 commands. The following is a list of these special names and how they are used:

Reserved Name	Description
$ALPHA	This special name can only be used in a conditional-expression on the IF or DO commands. The characters that meet the condition must be within the range of 'A' through 'Z', 'a' through 'z' (ASCII only), or space. An I-QU PLUS-1 configuration parameter can change the interpretation of this special name.
$HIVALS	$HIVALS correspond to ASCII COBOL HIGH-VALUES and may be used in setting alpha variables or RDA items (see the SET command). $HIVALS may also be used in a conditional-expression on the IF or DO commands. They are interpreted as all octal 177s (or 377s) for ASCII or all octal 77s for FIELDATA. An I-QU PLUS-1 configuration parameter can change the interpretation of this special name.
$LOVALS	$LOVALS correspond to ASCII COBOL LOW-VALUES and may be used in setting alpha variables or RDA items (see the SET command). $LOVALS may also be used in a conditional-expression on the IF or DO commands. They are interpreted as all binary zeroes.
$NUM	This special name can only be used in a conditional-expression on the IF or DO commands. The characters that meet the condition must be within the range of 0 through 9.
$PBUFF	$PBUFF refers to the current contents of the print buffer which is filled by using the I-QU PLUS-1 concatenation symbol on the DISPLAY, EDIT, TRIMDISP and TRIMEDIT commands. $PBUFF may be referenced on the SET and RDMS commands. It will be cleared by a DISPLAY, EDIT, TRIMDISP or TRIMEDIT without a concatenation symbol or when referenced by a SET or RDMS command.
$SPACES	$SPACES may be used to set an alpha variable or RDA item to all spaces (see the SET command). $SPACES may also be used in a conditional-expression on the IF command.

Chapter 7: Record Delivery Area (RDA)

This chapter covers the defining or partitioning of the input/output area of I-QU PLUS-1. This area is called the Record Delivery Area (RDA). The starting position of files, records, buffers, etc. can be defined through the use of the DEFINE RA directive. Data items can be defined by using the DEFINE RDA directive and subscripts established (DEFINE SUB directive) for repeating data item definitions (OCCURS clause in COBOL).

The DEFINE F (Define File) directive is not covered in this chapter since its use and format is dependent upon a particular file system; PCIOS, DTM, etc. Refer to each file system's chapter in this manual for the DEFINE F directive required.

7.1 Alternate Record Area Definition (DEFINE RA)

All input and output of either DMS 2200 records, PCIOS file records or BIS DTM queue-alias areas default to position one of the RDA. This default is not always desirable. When working with a database record whose location mode is via set and set occurrence selection is location mode of owner, or when data must be extracted from several different record types that must be current at the same time, having several records resident in the RDA is a requirement. The DEFINE RA directive allows alternate record areas to be used in I/O operations for DMS 2200 records, PCIOS files, DTM queue-alias areas and SORT records.

Format:

```
        DEFINE RA {filename | record-name  queue-alias | SORT} ;
           {absolute-word-number | {OVERLAY | AFTER} ;
           {file-name | record-name | queue-alias | SORT } }
```

DEFINE may be abbreviated DEF.

For proper displacement of items within a named file/record, the DEFINE RA directive should be coded prior to referencing those items in the program.

If alternate records delivery areas are to be defined for DMS records, they should be specified after INVOKE but prior to IMPART.

Filename applies to PCIOS files. The file must be currently defined with a DEFINE F directive (see Chapter 10, "PCIOS and SFS 2200 File Interface" in this manual). The *filename* may be an entry defined in a permanent data item index file by the QINDEX processor. By defining the file with the QINDEX processor, the I-QU PLUS-1 processor will be able to address file data items by name, and automatically determine the RDA location of the record.

For information on creating a permanent data item index file, see Chapter 18, "QINDEX Reference".

Record-name applies to DMS 2200 records. The record must be currently invoked (see "DMS 2200 Interface" in this manual). In addition, a DEFINE RA which references a DMS 2200 record must be coded prior to the IMPART for proper record offsetting to occur.

Queue-alias applies to BIS DTM reports. The *queue-alias* must be currently defined with a DEFINE F directive (see Chapter 15, "BIS DTM Interface," in this manual).

SORT specifies the area to be used for all sort RELEASEs and RETURNs.

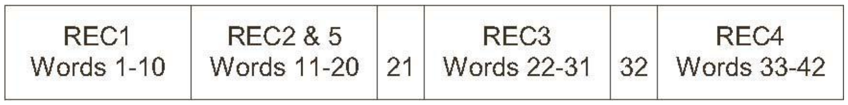
Absolute-word-number is the beginning word address within the RDA that will be used for I/O. Word addresses may range from 1 to the generated maximum RDA length. Word number is used here instead of character address to ensure that the record area is established on a full word boundary.

OVERLAY allows one record area to overlay another. The OVERLAY clause is used instead of an absolute word number. If the record to be overlaid is located in an alternate record area, its DEFINE RA must precede the definition in which the OVERLAY is specified.

AFTER is used when one record area is to follow another. The AFTER clause is used instead of an absolute word number. If the record to be followed is located in an alternate record area, its DEFINE RA must precede the definition in which the AFTER is specified.

When using absolute or direct RDA references, e.g., RDA (21,4), care should be taken when used in relationship to the DEFINE RA … AFTER directive. When the AFTER clause references a record/file named on a previous DEFINE RA … AFTER directive, the location of the record/file that is being defined will start one word (four ASCII bytes) beyond the length of the referenced record. Take, for example, five records that are ten words each. The DEFINE RA directives might be coded as follows:

```
DEF RA REC2 AFTER REC1
DEF RA REC3 AFTER REC2
DEF RA REC4 AFTER REC3
DEF RA REC5 AFTER REC1
```



REC1 Words 1-10	REC2 & 5 Words 11-20	21	REC3 Words 22-31	32	REC4 Words 33-42

Words 21 and 32 are not used. REC2 and REC5 do not reference a record named on a previous DEF RA directive; therefore, they begin immediately after REC1. REC 3 and REC 5 reference a record named on a previous DEF RA; therefore, they begin one word to the right of the referenced record. This situation was detected in an earlier release of I-QU PLUS-1 and cannot be changed for reasons of upward compatibility; i.e., changing I-QU PLUS-1 to eliminate this extra word, could adversely affect existing I-QU PLUS-1 programs that contain direct references.

An alternate record area for any single file, record or SORT may be defined only once in an I-QU PLUS-1 program.

Examples:

```
1. DEFINE RA R0001-PART-MASTER 100
2. DEFINE RA SORT AFTER R0001-PART-MASTER
3. DEFINE RA OUTFILE OVERLAY SORT
```

Explanations:

In example 1, DMS 2200 will use the area beginning at word 100 of the record delivery area for all input and output for the R0001-PART-MASTER record.

Example 2 causes all SORT Release/Return commands to use the area beginning at the next word following the R0001-PART-MASTER record in the RDA.

In example 3, all I/O for the PCIOS file OUTFILE will be done from the same area used by the SORT.

7.2 Record Delivery Area Field Definition (DEFINE RDA)

Areas within the RDA may be defined for reference by name without an external data item index, or items already defined in a data item index may be redefined by the DEFINE RDA directive. This is useful when there is a need to redefine fields, or when referencing files for which no data item index file has been generated. DEFINE RDA allows the user to assign a name to a direct, or relative, RDA reference, thus simplifying future references to the data.

Format:

> DEFINE RDA *item-name* (*start-byte-position*, *length*) ;
> [*data-type*] [*decimal-precision*]

Where:

> *start-byte-position* = {*RDA-absolute-start-position* | * | *item-name* | **item-name*}

DEFINE may be abbreviated DEF.

The *start-byte-position* may be specified in several ways. The *start-byte-position* may be given as an integer offset from the start of the RDA. An asterisk (*) in the *start-byte-position* indicates that this item begins immediately following the last RDA field defined. If an asterisk is used on the first DEFINE RDA in the program, position one (1) will be assumed. If the *start-byte-position* is given as a defined RDA *item-name*, either defined with a previous DEFINE RDA or in the current primary or secondary data item index, the item being defined will start at the same position of the named item. If an RDA *item-name* is used, preceded by an asterisk (*), the item being defined will start immediately following the named item. An RDA *item-name* used in a DEFINE RDA may be qualified by record or file name.

The *length* is the actual length of the referenced data. For computational data items, this is the actual byte length of the data, not the COBOL picture length. For example, if the COBOL picture is PIC 9(10) COMP, the actual byte length is 4 (assuming the data is ASCII). Table 3.2 may be used to determine the byte length of various computational field definitions.

If the definition specifies zero in either position of the RDA reference, the value of S$ and/or L$ will be used when *item-name* is referenced. In addition, *item-name* can be referenced with an index or subscript when used in a command.

The decimal-precision is a literal showing how many decimal positions are implied (the scale factor). The decimal-precision specification does not assign a value to the RDA field.

The data-type must be one of the allowed I-QU PLUS-1 RDA data types.

Examples:

```
1. DEFINE RDA PART-KEY (1,10)
2. DEFINE RDA PART-KEY-FIRST-4 (PART-KEY,4)
3. DEFINE RDA PART-KEY-REST (*,6)
4. DEFINE RDA PRICE-CODE (11,4) COMP
5. DEFINE RDA PRICE (*,4) COMP .00
6. DEFINE RDA PART-TABLE (0,3)
7. DEFINE RDA TITLE-FIRST-CHAR (*EMP-NAME OF PAYMAST,1)
```

Explanations:

> In example 1, PART-KEY defines a display field in the first 10 characters of the RDA.
>
> In example 2, PART-KEY-FIRST-4 redefines the first 4 positions of PART-KEY defined in example 1.
>
> PART-KEY-REST, in example 3, redefines the last 6 positions of PART-KEY. The asterisk indicates that PART-KEY-REST immediately follows PART-KEY-FIRST-4 in the RDA since it was the last RDA field defined.

In example 4, PRICE-CODE is a computational data item immediately following the PART-KEY in the RDA.

In example 5, PRICE is a computational field with 2 implied decimal positions. The equivalent COBOL picture would be PIC S9(8)V99 COMP. PRICE will immediately follow PRICE-CODE in the RDA.

In example 6, PART-TABLE defines a 3-character field with variable start position to be determined at run time based on the content of the reserved variable S$.

Example 7 demonstrates the redefinition of a portion of a data item defined in a DMS 2200 record. TITLE-FIRST-CHAR starts immediately following the item named EMP-NAME in record PAYMAST (see qualified RDA item referencing).

7.3 Subscript Variable Definition (DEFINE SUB)

Subscript variables may be defined for use in referencing RDA data item arrays. A subscript variable may only be used to reference items within a single array; therefore, each array to be referenced would require a different subscript variable. A subscript variable contains the length of the subscripted item as well as the desired occurrence number of the item to be referenced. The item length portion may not be accessed by the user. Only the occurrence number can be referenced.

Format:

DEFINE SUB *subscript-name RDA-reference* [:*subscript-name*]

DEFINE may be abbreviated DEF.

The *RDA-reference* either may be a direct RDA reference or may refer to a data item name from the currently invoked subschema, secondary data item index file or a user defined RDA reference.

The optional :*subscript-name* on the RDA-reference is used in setting up subscripts for multi-dimensional arrays. It names the subscript variable used to reference the RDA-reference name within an occurring group. When the optional :*subscript-name* is used, the RDA-reference may not be direct.

Examples:

Given the following COBOL two-dimensional table definition:

```
05 BUDGET-TABLE.
    10 YEAR-ENTS OCCURS 5.
        15 MONTH-ENTS OCCURS 12.
            20 BUDGET-AMT PIC 9(7)V99.
```

Two subscript variables would be defined as follows:

```
1. DEFINE SUB YSUB YEAR-ENTS
2. DEFINE SUB MSUB MONTH-ENTS :YSUB
```

Explanations:

In example 1, YSUB is set up to reference each of the 5 YEAR-ENTS entries.

In the second DEFINE SUB, MSUB is set up to reference MONTH-ENTS within YEAR-ENTS.

To reference the BUDGET-AMT for the 2nd month of the 3rd year, the following may be used:

```
YSUB = 3
MSUB = 2
TOT-BUDGET = TOT-BUDGET + RDA BUDGET-AMT :YSUB :MSUB
```

Chapter 8: Control Directives

The following directives are used to perform non-procedural control functions.

8.1 ADD

The ADD directive is used to obtain program input from a symbolic element in an external EXEC file. This will cause I-QU PLUS-1 to begin reading the specified element, which may contain DIRECTIVES and/or PROCEDURAL commands. When end-of-file is reached on the ADDed element, I-QU PLUS-1 will then resume reading the original input stream. Any number of ADD directives may be placed anywhere in the I-QU PLUS-1 program. ADD directives may be nested up to a depth of ten (10) levels. The formats for the ADD directive are:

Format 1:

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

Format 2:

ADD * FROM [*qualifier* *]*filename*

ADD may be abbreviated AD.

If the FROM clause is not specified, the default I-QU PLUS-1 program source code library file, I$QU*I$QULIB., will be used (the qualifier, I$QU, is a runtime configuration parameter and may differ from site to site).

If Format 2 is used, the *filename* is required and must be an SDF data file. Format 2 is used when adding a file instead of an element.

Examples:

```
ADD CUSTMAST/DEFS              .  Use default lib.
ADD MISC-DEFS FROM QLNK*PROCS
ADD * FROM DATA*FILE.
```

8.2 CLEAR

The CLEAR directive is used to set the internal error-flag off. The error-flag is turned on anytime an error is detected.

8.3 COMPILE

The COMPILE directive causes I-QU PLUS-1 to resolve all program and procedure labels, all IF/ENDIF pairs, and all PROCEDURE/ENDPROC pairs to complete the internal program and prepare it for execution or saving. If any errors are detected during this process, further processing of the program will not be attempted. The COMPILE is implied automatically if it has not been called before a RUN or SAVE directive. There are no parameters associated with the COMPILE.

8.4 CONV

The CONV directive is used to switch I-QU PLUS-1 into CONVersational mode. In this mode, commands entered are edited by the command editor, and passed to the command executor for immediate execution. The data storage area is not affected by switching to CONVersational mode, thus allowing access to the results of program execution. However, when switching from INPUT to CONVersational mode, the current object program is lost.

8.5 EXIT

The EXIT directive is used to terminate I-QU PLUS-1. If the user has not done a DML DEPART, I-QU PLUS-1 will automatically execute a DEPART without ROLLBACK prior to termination. PCIOS files that are open will be closed. The EXIT routine will also @FREE any files automatically assigned by I-QU PLUS-1. If a default alternate print file was assigned, it will be automatically @SYMed.

8.6 INDEX

The INDEX directive is used to assign a secondary data item index file. The INDEX directive must specify the name of an I-QU PLUS-1 data item index file created by the QINDEX processor, or by the I-QU PLUS-1 Processor. Only one secondary data item file may be assigned at any given time.

Format:

INDEX [*qualifier**]*filename*

INDEX may be abbreviated IND.

8.7 INIT

The INIT directive is used to reinitialize all data storage areas. All reserved variables will be reset to initial values and all user variables will be de-allocated.

8.8 INPUT

The INPUT directive is used to switch I-QU PLUS-1 to INPUT mode. In this mode, commands are entered, edited by I-QU PLUS-1's command editor, and stored for execution. When INPUT mode is entered, I-QU PLUS-1's data storage area is left unchanged. The program counter (PC) will be set to 1, causing any previous object program to be overlaid.

8.9 LISTOFF

The LISTOFF directive will suppress the listing of commands as they are edited by I-QU PLUS-1.

The issuing of the LISTOFF directive in INPUT mode does not effect the LISTON/LISTOFF condition of CONVersational mode and visa versa.

8.10 LISTON

The LISTON directive will cause resumption of the listing of commands as they are edited.

The issuing of the LISTON directive in INPUT mode does not effect the LISTON/LISTOFF condition of CONVersational mode and visa versa.

8.11 LOAD

The LOAD directive is used to load a previously compiled program into I-QU PLUS-1's internal areas. Once loaded, the program may be executed using the RUN directive, or saved to another object program file. See the RUN and SAVE directives for more information.

Format:

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

LOAD may be abbreviated L.

If the FROM clause is omitted, I-QU PLUS-1 will attempt to load the object program from the default file I$QU*I$QUOBJ (the qualifier, I$QU, is a runtime configuration parameter and may differ from site to site).

8.12 OBJECT

The OBJECT directive will produce a formatted dump of all internal areas. The dump will not destroy the current contents of the data storage areas.

8.13 PRINTER

The PRINTER directive is used to change the default device name to which alternate print output will be @SYMed when the program is complete. The device is selected by the user by entering a number between 0 and 4. Each number represents a device name specified during the I-QU PLUS-1 dynamic configuration process (see the I-QU PLUS-1 Installation and Operation Guide).

Format:

PRINTER *site-printer-number*

PRINTER may be abbreviated PR.

This directive applies only to alternate default print files.

8.14 RUN

The RUN directive is used to execute a I-QU PLUS-1 program that has been entered and compiled, or to load and execute an object version of a program that has been compiled previously and saved in object form (see SAVE directive).

Format:

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

RUN may be abbreviated R.

If both program-name and the FROM clause are omitted, I-QU PLUS-1 will begin executing the currently compiled program. If the program has not been previously compiled, the RUN directive will cause an implicit compilation prior to execution.

If the FROM clause is omitted, I-QU PLUS-1 will attempt to load the object program from the default object library, I$QU*I$QUOBJ (the qualifier, I$QU, is a runtime configuration parameter and may differ from site to site).

Examples:

```
RUN
```

The above would require that a program had just been entered, and would force a COMPILE if one had not already been done.

```
RUN CUST-QUERY
```

The above will load and run the object form of the program CUST-QUERY from the default object file I$QU*I$QUOBJ.

```
RUN SALES FROM SALES*QLINK-OBJ
```

The above will load and run the object program SALES from the user-cataloged file SALES*QLINK-OBJ.

8.15 SAVE

The SAVE directive is used to save the currently compiled I-QU PLUS-1 program, in object form, as an OMNIBUS element. The object form includes all internal tables (including D$WORK and S$WORK) required to execute the program. The SAVE directive may immediately follow a COMPILE directive. An implicit COMPILE of the current program will be performed if required.

Format:

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

SAVE may be abbreviated SA.

If the INTO clause is omitted, I-QU PLUS-1 will attempt to save the object program into the default object library, I$QU*I$QUOBJ (the qualifier, I$QU, is a runtime configuration parameter and may differ from site to site).

The SAVE directive may not be preceded by a RUN directive.

Examples:

```
SAVE CUST-QUERY
```

The above will save the currently compiled object program as CUST-QUERY in the default file I$QU*I$QUOBJ.

```
SAVE SALES INTO SALES*QLINK-OBJ
```

The above will save the object of the currently compiled program as SALES in the user-cataloged file SALES*QLINK-OBJ.

8.16 ?

This directive, entered as a question mark (?), displays the following information to the conversational user:

- Current schema file, schema and subschema;
- The maximum object program size;
- The maximum number of variable indexes and how many are currently available;
- The size of the variable data storage area and how much is currently in use.

8.17 Object Program Considerations

Using object forms of programs will provide substantial performance improvements because the INVOKE processing, command interpretation, and compilation are eliminated. An object I-QU PLUS-1 program is sensitive to the same types of schema and subschema changes that cause COBOL/DML programs to be recompiled; therefore, it may be necessary to recompile and save I-QU PLUS-1 programs after certain schema and/or subschema changes are implemented. In addition, certain changes in the configuration of the I-QU PLUS-1 processor may make an object program incompatible, requiring recompilation and saving.

The default object file I$QU*I$QUOBJ must be catalogued by the user site prior to use of I-QU PLUS-1. The qualifier of the default object file may be changed when configuring the runtime portion of I-QU PLUS-1.

Typically, while a program is in the development phase, full source would be processed by I-QU PLUS-1, including an INVOKE, COMPILE and RUN. Once a program is tested and ready for production use, it would be compiled and saved in object form (see SAVE directive). To execute the program for production, only the RUN directive would be required as follows:

```
RUN program-name              .  <---Load and execute object.
data-images                   .  <---Optional input data.
```

Chapter 9: General Procedural Commands

Procedural commands can be divided into seven categories:

- General Procedural Commands;
- COBOL File (PCIOS/SFS 2200) Handling Commands;
- Sort Interface Commands;
- Direct I/O Commands;
- DMS 2200 Data Manipulation Language Commands;
- RDMS 2200 Relational Database Commands;
- BIS DTM Interface Commands.

This chapter only describes the general procedural commands. The other categories will be covered in chapters that follow.

The general procedural commands give the I-QU PLUS-1 user the ability to display and change data in both the RDA and the variable data storage area.

9.1 ACCEPT

The ACCEPT command is used to set variables to values entered by the I-QU PLUS-1 user. When executed, ACCEPT will display a prompt string defined by the programmer (if specified), then read from the demand (terminal) or batch input stream or the system console. I-QU PLUS-1 will set the specified variable to the value entered. Input is limited to 256 characters. Numeric input may contain a leading sign with no intervening spaces. The optional *prompt-string* literal may not exceed 50 characters. It should only be used when the CONSOLE option is specified.

Format:

ACCEPT *variable* [‘*prompt-string*’] [[FROM] CONSOLE] ;
[AT END {BREAK | *program-label*}]

ACCEPT may be abbreviated A; while CONSOLE, CONS or CONSOL.

The AT END clause is optional. BREAK may only be used instead of a label when the ACCEPT is contained within a defined procedure, or an in-line DO block.

Examples:

```
1. ACCEPT BIS-DATA AT END END-RUN
2. ACCEPT X 'Enter Product Id:' CONSOLE
```

Example 1 shows how a variable may be accepted from the terminal or runstream. If an AT END condition occurs, control will be returned at the program label END-RUN.

In example 2, the ACCEPT will prompt and obtain input from the system console to put into variable X.

9.2 BITMERGE

This command is used to combine a specified number of bits from three numeric variables to form a result in a single numeric variable. For this operation, numeric variables are viewed as unsigned single-word entities. The same variable may be repeated in any position within the command syntax. There must be three counts specified. The first bit count may be zero, but the sum of the three must add up to 36. Bits will be extracted from the right-most positions of the source variables and deposited into the result variable starting from the left.

Format:

> BITMERGE *result-variable* ;
> FROM *variable-name-1*,*variable-name-2*,*variable-name-3* ;
> (*bit-count*-1,*bit-count-2*,*bit-count*-3)

BITMERGE may be abbreviated BM or BITM.

The *result-variable*, *variable-name-1*, *variable-name-2* and *variable-name-3* must be defined numeric variables. The *bit-count-1*, *bit-count-2* and *bit-count-3* must be given as unsigned integer literals.

Examples:

1. The following code builds a database key with 10 bits for the area, 17 for the page and 9 for the record number:

```
SET X = 121                          .  AREA CODE
SETY=50                              .  PAGE NUMBER
SETZ=1                               .  RECORD NUMBER
BITMERGE X FROM X,Y,Z (10,17,9)      .  MAKE DBK IN
```

2. The example below builds an area key in the pre-defined numeric variable G-AKEY. The page number is put in the first half of the word, and the record number in the second half. Y is assumed to contain the page number and Z the record number.

```
BITMERGE G-AKEY FROM Y,Y,Z (0,18,18) .  AREA KEY
```

> The first reference to Y in the area key example above is a dummy reference needed because all variables are required in the syntax.

9.3 BITSPLIT

The BITSPLIT command is used to disperse bits from one numeric variable into three numeric variables. Each variable used is viewed as an unsigned single-word entity. The same variable may be repeated in any position. There must be three-bit counts specified. The first specified *bit-count* may be zero, but the sum of the three must equal 36. Bits will be extracted from the source variable beginning from the right and deposited into the low-order positions of the result variables.

Format:

> BITSPLIT *source-variable* ;
> INTO *variable-name-1*,*variable-name-2*,*variable-name-3* ;
> (*bit-count*-1,*bit-count*-2,*bit-count*-3)

BITSPLIT may be abbreviated BS or BITS.

The *source-variable*, *variable-name-1*, *variable-name-2* and *variable-name-3* must be defined numeric variables. The *bit-count-1*, *bit-count-2* and *bit-count-3* must be given as unsigned integer literals.

Examples:

1. Split a database key into its three components (area code, page number and record number):

```
BITSPLIT C-DBK INTO X,Y,Z (10,17,9)
.  SPLIT DBK
```

2. Isolate the high-order bit of a word:

```
SET X = RDA (21,4) UB9
.  GET WORD FROM THE RDA
BITSPLIT X INTO X,Y,Z (0,1,35)
.  ISOLATE BIT IN Y
IFY=1
DISPLAY 'BIT SET'
ELSE
DISPLAY 'BIT NOT SET'
ENDIF
```

9.4 BREAK

The BREAK command is used only in conjunction with a DO command. It is used to force an immediate termination of an in-line DO or defined procedure, regardless of condition. For an in-line DO, control passes to the statement following the DO/ENDDO pair. If the BREAK is in a defined procedure, control passes to the command following the DO from which the procedure was entered.

Format:

BREAK

BREAK may be abbreviated B.

Examples:

1. To BREAK out of a defined procedure:

```
    DO COUNT WHILE Z < 900
...
COUNT PROCEDURE
    ACCEPT X 'Enter a value:'
    IF X = 0
        DISPLAY 'Zero entered, end of run'
        BREAK
. Exit DO loop immediately
    ENDIF
    Z = Z + X
    ENDPROC
```

3. To BREAK out of a DO/ENDDO in-line loop:

```
...
.  *** Search for first use type "A" or "G"
FETCH3 FIRST PART-USAGE SET
DO WHILE ERROR-NUM <> 7
    IF RDA PART-USE-TYPE = 'A'
    OR RDA PART-USE-TYPE = 'G'
        DISPLAY '*** FOUND IT ****'
        BREAK
    ENDIF
    FETCH3 NEXT PART-USAGE SET
ENDDO
...
```

9.5 CASE

The CASE command may be used to control case sensitivity for the DO, IF and SCAN commands. The default is CASE ON (for compatibility with older levels of I-QU PLUS-1) which implies that any test will be case sensitive. The CASE command is only significant when checking ASCII alpha display data.

Format:

```
CASE {OFF | ON}
```

Examples:

```
1.  ACCEPT REPLY 'Summary? Enter (Y)es/(N)o:' CONS
    CASE OFF
    IF REPLY = 'y' .  "Y" or "y" will test true.
        DO SUMMARY
    ENDIF
    CASE ON

2.  DEF F CMD-FILE SEQ 80,0
    DEF RDA COMMAND (1,7)
    :
    CASE OFF
    DO WHILE COMMAND <> 'process'
        READ CMD-FILE
        :
    ENDDO

3.  FETCH3 FIRST CUSTOMER-REC CUSTMST AREA
    CASE OFF
    DO WHILE ERROR-NUM = 0
        SET X = 0     .  Initialize length/result variable.
        SCAN RDA CUSTOMER-NAME 'JONES' X .  SCAN for string
        IF X > -1 .Found?
            DISPLAY RDA CUSTOMER-NAME    .  Display name
            SET FIND-CNT = FIND-CNT + 1  .  Count it
        ENDIF
        FETCH3 NEXT CUSTOMER-REC CUSTMST AREA
    ENDDO
```

9.6 CLEARSCREEN

The CLEARSCREEN command is designed to be used in an I-QU PLUS-1 program that would be run from a demand terminal. It outputs the ASCII control sequence ESC, "e", ESC, "M", which is a cursor to home and erase display on most terminals that support UNISCOPE (UTS) protocol.

Format:

```
        CLEARSCREEN
```

CLEARSCREEN may be abbreviated CLS or CLEAR.

9.7 CONNECT

The CONNECT command allows an I-QU PLUS-1 program to connect to the TIP on-line system. The command is only required when handling recoverable TIP/FCSS or TIPDMS files with the DIO command. The command may not be used while IMPARTed to DMS. This command may not be used to change the application association (application number).

Format:

```
    CONNECT [ || {TO [APPLICATION] | APPLICATION} application-number | ;
       AT level-number | ;
       OPTION recovery-options || ]
```

CONNECT may be abbreviated CONN, with APPL for APPLICATION and OPT for OPTION.

The *application-number* must be from 1 to 9. If not specified, application 0 is implied.

The *level-number* is the connection level which defines which TIP facilities are available to this program. Valid values are 1 through 3.

The *recovery-options* are specified as an integer number and the specified bits must be as discussed for the CONNECT TIP primitive. If not specified, "no recovery" is implied and the *application-number* is ignored by the EXEC.

No attempt is made to detect or prevent redundant CONNECT commands. I-QU PLUS-1 will issue requests to MCB/EXEC as submitted.

For more information, refer to the OS 2200 Transaction Processing Programming Reference Manual and the OS 2200 Transaction Processing Administration and Operations Reference Manual.

9.8 CSF

The CSF command allows the I-QU PLUS-1 user to submit any valid CSF image to the operating system. The status returned from the operating system will be placed in the numeric variable specified. I-QU PLUS-1 will not attempt to edit the image being submitted, so care must be taken to ensure that the format is acceptable to the operating system.

Format:

CSF *status-variable* {'*CSF-image-literal*' | *CSF-image-variable*}

CSF may be abbreviated CS.

Example:

```
CSF X '@ASG,A MY*FILE.,T,REEL01'   .  Assign a file
IF X < 0                           .  Test status
    FACERR X                       .  Display reasons
    STOP EXIT
ENDIF
...
CSF X '@START RUNFILE.LIST-DB '    .  Start a run
IF X NOT = 0
    DISPLAY 'Start failed, status follows:'
    DUMP X
ENDIF
```

The following control statements may be issued:

@ADD	@CAT	@LOG	@RSTRT	@USE
@ASG	@CKPT	@MODE	@START	
@BRKPT	@FREE	@QUAL	@SYM	

9.9 DATE

The DATE command is used to reset the related DATE reserved variables to the current date. The DATE command may also be used to translate a user-furnished date from Julian to Gregorian or from Gregorian to Julian. The DWTIME option will convert a value in DWTIME$ format to Gregorian and Julian formats and set a time variable.

Format:

DATE [{*date-variable* | RDA *RDA-reference*} TO ;
 {JULIAN | GREGORIAN | DWTIME}]

DATE may be abbreviated DA.

This command will set the values of the following predefined variables:

Variable	**Data Type**	**Format**
DATE	Alphanumeric	YYYY/MM/DD
DATE-NUM	Numeric	YYYYMMDD
YEAR	Numeric	YYYY
MONTH	Numeric	MM
DAY	Numeric	DD
J-DAY	Numeric	YYYYDDD
TIME (DWTIME option only)	Alphanumeric	HH:MM:SS.DDD

If the DATE command is used with no parameters, all date variables will be set to the current date.

The *date-variable* or *RDA-reference* must be defined as numeric and contain a positive numeric integer in the form YYMMDD or YYYYMMDD if converting to Julian, or YYDDD or YYYYDDD if converting to Gregorian. If the year is only two digits (YY), the century is derived from today's date. For example, if today were 2000-01-01 and the date variable contained 940729, DATE would reveal 2094/07/29.

When using the DWTIME option, the *date-variable* or *RDA-reference* must be defined as ASCII alphanumeric display for 8 bytes (characters); however, they must contain an ASCII aligned binary value, which represents nanoseconds since midnight, December 31, 1899. This value is said to be in DWTIME$ format and is used internally in such places as the ASCII System Log, the Universal Data System, LINC 2200 applications, etc. In addition to setting the six reserved variables associated with a date (see above), the DWTIME option also sets the reserved variable TIME.

If the date cannot be converted, the result variables will be set to zero.

Examples:

1. The DATE command used alone to set the current date into all date variables:

```
DATE
```

2. Use the DATE command to determine if a given year is a leap year. Assume the variable ENTRY-YEAR is 2000.

```
DEF A ENTRY-YEAR 4
DEF RDA LEAP-A (4001,6)
DEF RDA LEAP-N (4001,6) SN9
...
```

```
ACCEPT ENTRY-YEAR 'Enter year: '
TD ENTRY-YEAR '060' +
SET RDA LEAP-A = $PBUFF
DATE RDA LEAP-N TO GREGORIAN
IF DAY = 29

TD YEAR ' is a leap year!'
...
```

The answer would be:

```
2000 is a leap year!
```

3. Convert the date in the variable ST-DATE to Julian. Assume ST-DATE contains 940719. J-DAY contains the Julian result. The value of ST-DATE will be set up in all other date variables:

```
DEF N ST-DATE
...
DATE ST-DATE TO JULIAN
D 'START DATE ' DATE +
ED J-DAY ' IS JULIAN 99/999.'
```

The result would be as follows:

```
START DATE 1994/07/19 IS JULIAN 1994/200.
```

4. Convert the DWTIME$ formatted RDA reference, DWTIME-OF-START:

```
DEF RDA DWTIME-OF-START (*,8)
...
DATE DWTIME-OF-START TO DWTIME
D 'The program started on ' DATE +
D ' at ' TIME
```

The above example will yield results similar to the following:

```
The program started on 1994/09/25 at 16:17:41.041
```

9.10 DATESET

The DATESET command is used to convert a variable or RDA reference to a format suitable for displayed output or arithmetic operations.

Format:

DATESET {*variable* | RDA *RDA-reference*} ;
    [FORMAT {$DATE*n* | '*format-literal* ' | *format-variable*}] = ;
  {*variable* | RDA *RDA-reference*} ;
    [FORMAT {$DATE*n* | '*format-literal* ' | *format-variable*}]

The format field on either the sending or receiving side of the equal sign can be any of three choices, or omitted entirely if the default format is desired:

1. $DATE*n* provides a set of predefined date formats that should handle the majority of date formatting tasks. The "*n*" can be 0-8 for standard BIS date formats, or 20-23 for KMSYS Worldwide defined formats. These formats are defined in the table below.
2. The *format-literal* can be used to replicate the $DATE*n* formats or create other user-defined formats. The descriptors necessary to create these strings are defined in the date descriptor table on the following page.
3. The *format-variable* must be defined as an alphanumeric variable. The use of the *format-variable* allows the programmer to define format strings that may be set at execution time, which implies that the sending or receiving date string format is not necessarily known at compile time.
4. The default format (the FORMAT clause is omitted) is an internal Julian date based on 1/1/0001. Where 1 = 1/1/0001 , 2=1/2/0001 , 59 = 2/28/0001, etc. This format should be used when performing date calculations.

When converting dates from a 2-digit year to a 4-digit year, I-QU PLUS-1 assumes the missing century portion of the year to be the current century. Likewise, if a 4-digit year is input and the century portion is "00", the current century will be assumed. It must be noted that because of these assumptions, 4-digit years in the range of "0001" through "0099" will not convert properly to an internal Julian date.

The following $DATEn formats are predefined:

$DATEn	**Format**	**Type**	**Example**
$DATE0	YMMDD (Y1M2D2)	N	20327
$DATE1	YYMMDD (Y2M2D2)	N	920327
$DATE2	DD MMM YY (D2BM3BY2)	AN	27 MAR 92
$DATE3	YDDD (Y1D3)	N	2087
$DATE4	YYDDD (Y2D3)	N	92087
$DATE5	DDMMYY (D2M2Y2)	N	270392
$DATE6	MM/DD/YY (M2/D2/Y2)	AN	03/27/92
$DATE7	MMMMMMMMM DD,YYYY (M9BD2,Y4)	AN	MARCH 27,1992
$DATE8	MMDDYY (M2D2Y2)	N	032792
$DATE20	YYYYMMDD (Y4M2D2)	N	19920327
$DATE21	YYYYDDD (Y4D3)	N	1992087

$DATEn	**Format**	**Type**	**Example**
$DATE22	DDMMYYYY (D2M2Y4)	N	27031992
$DATE23	MMDDYYYY (M2D2Y4)	N	03271992

The predefined formats have a data type of either numeric (N) or alphanumeric (AN). A numeric format may be used on a numeric or alphanumeric variable or RDA reference; however, an alphanumeric format may only be used on an alphanumeric variable or RDA reference. Consider the following alpha variable, numeric RDA reference and DATESET commands:

```
    DEFINE A HDRDATE 17
    DEFINE RDA PODATE (10,6) UN9
       DATESET HDRDATE FORMAT $DATE8 = RDA PODATE FORMAT $DATE1 . Correct
       DATESET HDRDATE FORMAT $DATE8 = RDA PODATE FORMAT $DATE6 . Incorrect
```

The first DATESET command is correct. Both $DATE8 and $DATE1 are numeric and can reference either a numeric field (PODATE) or an alpha field (HDRDATE). The second DATESET command will cause an error stating that the alphanumeric format ($DATE6) is not compatible with the numerically defined field (PODATE).

The following date descriptors are allowed in the *format-literal* or *format-variable*:

Descriptor	**Type**	**Explanation**
-	A	Dash: Allows a dash in the input field, or inserts a dash in the output field.
,	A	Comma: Allows a comma in the input field, or inserts a comma in the output field.
.	A	Period: Allows a period in the input field, or inserts a period in the output field. The period is used in European date formats which are normally formatted as D2.M2.Y2 (e.g., "31.01.94").
/	A	Slash: Allows a slash in the input field, or inserts a slash in the output field.
B	A	Blank: Allows a blank in the input field, or inserts a blank in the output field. An actual blank or space character can also be used as the descriptor (e.g., "M9 D2, Y4" will produce the same result as "M9BD2,BY4").
D2	N	Day of the Month: Valid values are in the range of 01-31; however, the value must not exceed the maximum number of days allowed for the specified month.
D3	N	Day of the Year: Valid values are in the range of 001-366; however, 366 is only valid for a leap year.
I	N	Internal Julian Date: This format is the default when the FORMAT clause is omitted. This descriptor may not be used in combination with any other date descriptor. In this format, all dates are relative to the first day of year one.
M2	N	Month of the Year: Valid values are in the range of 01-12.
M3	A	Month of the Year Abbreviation: This format will alway be 3 alphabetic characters (e.g., "JAN", "FEB", etc.).

Descriptor	Type	Explanation
M9	A	Alphabetic Month of the Year: The longest month name is 9 characters. Shorter month names will be space filled to the right.
W1	N	Day of the Week: Valid values are in the range of 1-7 where 1 is Sunday, 2 is Monday, etc.
W3	A	Day of the Week Abbreviation: This format will alway be 3 alphabetic characters (e.g., "SUN", "MON", etc.).
W9	A	Alphabetic Day of the Week: The longest weekday name is 9 characters. Shorter names will be space filled to the right.
Yn	N	Year: n is the number of positions (1-4) required to represent the year. Valid values are in the range of 0001-2100. If n is less than 4, the high order positions of the year are truncated (e.g., if Y2 is specified, 1994 would be formatted as "94").

Each date descriptor is classified as either a numeric (N) or alphabetic (A) data type. If all the descriptors used in a format string are "A", the format is strictly alphabetic and can not be used with numeric variables or RDA references. If the string contains both "A" and "N" types, the format is alphanumeric and must be used with alphanumeric variables or RDA references. If all descriptors are "N", the format is numeric and may be used with both numeric or alphanumeric variables and RDA references.

The maximum number of date descriptors allowed in one *format-literal or format-variable* is six (6).

During execution, the DATESET instruction takes the input variable or RDA reference on the sending side of the equal sign and converts it to an internal Julian date based on the supplied format string or variable. If there is NOT enough information in the input data to exactly determine a specific date the missing values are used from today's date.

Assume the DATESET command looked something like the following:

```
DATESET AA1 FORMAT $DATE7 = AA2 FORMAT 'D2'
```

In this case, DATESET will only have the day of the month to work with. The month and year will be taken from the system clock and used to complete the date. This date will then be used to format the AA1 variable on the receiving side of the expression according to the $DATE7 format string.

Given the variety of data types and different format strings that can be used together with this instruction, it makes sense that some error may occur that the DATESET instruction cannot handle. In this case, DATESET will set the global variable ERROR-NUM to an error number (following table). Upon successful completion of the DATESET command, the ERROR-NUM variable will be 0. ERROR-NUM should be checked after each execution of this instruction.

ERROR-NUM	**Description**
0	No error status, the instruction completed successfully.
1	The format string does not contain any characters. This should only happen when the format string is empty.
2	Invalid descriptor in format string.
3	Invalid length for descriptor. This will occur if any descriptor is given with an invalid number of characters. For example if a long Month is required but the specification is M8 this error will be returned.
4	Invalid number of descriptors in the format string. The number of date descriptors is limited to 6.
5	Not enough characters in format string. This will happen if for example a "W" is used without a field width specified.
6	Format string contains a descriptor that can only occur alone. This will only happen when the "I" descriptor occurs with any other descriptor.
11	Invalid descriptor type for a numeric string. The type of the variable does not match the type indicated in the format string.
12	The sending field format is numeric but contains an alphabetic month descriptor.
13	The sending field format is numeric but contains an alphabetic day of the week descriptor.
21	RDA variable of type MAPNUM on the receiving side of the equal sign is not large enough to hold the result of date formatting.
31	Year is out of range, less than 0001.
32	Month is out of range, must be 1-12.
33	Day of month is invalid for month specified; this error considers leap year for the number of days in February.

Example 1:

```
DEF A AA1 20
DEF N NN1
AA1 = 'MARCH 23,1992'
DATESET NN1 FORMAT $DATE1 = AA1 FORMAT $DATE7
                        .  INPUT IN M9BD2,Y4
TD AA1 ' CONVERTS TO ' NN1
```

This example will produce the following result:

```
MARCH 23,1992 CONVERTS TO 920323
```

Example 2:

```
      DEF A CURR-DATE 18
      DATE
      DATESET CURR-DATE FORMAT 'M9BD2,BY4' = ;
          DATE-NUM FORMAT $DATE20
      TD "Today's date is " CURR-DATE '.'
```

This example will produce the following result:

```
      Today's date is AUGUST 12, 1994.
```

Example 3:

```
      DEF RDA MAPDATA (1,132)
      DEF RDA MAPDATE (14,8) MAPNUM  .  MM/DD/YY format
      DEF A MAPVAR 132
      ACCEPT MAPVAR AT END BREAK
      RDA MAPDATA = MAPVAR
      DATESET X = RDA MAPDATE FORMAT $DATE6
              .  Convert to internal Julian day for increment
      X = X + 120                     . Add 120 days to the date
      DATESET RDA MAPDATE FORMAT $DATE6 = X
                                       .  Convert to MM/DD/YY
      DISPLAY RDA MAPDATE
```

If the input was “02/15/92”, this example will produce the following result:

```
      06/14/92
```

9.11 DECIMAL

The DECIMAL command may be used to control decimal point sensitivity dynamically. Decimal sensitivity is initially ON by default, but may be turned off at any point in the program using this command. This command exists primarily for compatibility with older versions of the processor in which decimal point sensitivity did not exist.

Format:

```
        DECIMAL {ON | OFF}
```

DECIMAL may be abbreviated DEC.

9.12 DISCONNECT

The DISCONNECT command allows an I-QU PLUS-1 program that has issued a previous CONNECT command to detach itself from the TIP on-line system.

Format:

```
    DISCONNECT
```

DISCONNECT may be abbreviated DIS.

No attempt is made to detect or prevent redundant DISCONNECT commands. I-QU PLUS-1 will issue requests to MCB/EXEC as submitted.

9.13 DISPLAY

The DISPLAY command is used to display the contents of user or reserved variables or the contents of the RDA. DISPLAY may also be used to display alphanumeric literals.

Multiple display commands using the "+" symbol may be entered to build, but temporarily suppress, output. When "+" is encountered, data is moved into the print buffer, but no output is produced. When a DISPLAY without the "+" is executed, the data is moved to the print buffer and the contents of the print buffer are printed. (See the SET command to determine how to capture the contents of the print buffer, $PBUFF).

An output column may be specified by entering an integer or numeric variable following the command keyword DISPLAY. If the numeric variable is specified, it must be preceded by the "at" symbol (@). If no output column is specified, data is moved into the next available position of the print buffer. If a column is specified, data is moved to the specified position of the print buffer and the next available position will become the position following the data.

If automatic TAB character insertion is on (see TABS command), a tab character will be inserted in the first output position, followed by the data to be displayed.

Format:

> DISPLAY [[*column-literal* | @*numeric-variable*] {RDA *RDA-reference* | ;
> {*variable-1* | '*alpha-literal-1*'} … [*variable-4* | '*alpha-literal-4*'] } [+]]

DISPLAY may be abbreviated D.

A DISPLAY with no other options will generate a blank line.

Only one *RDA-reference* may be used within a single DISPLAY. However, from one to four alpha strings (*alpha-literal-n*) and variables (*variable-n*), or a combination of each, may be displayed using a single DISPLAY command.

Decimal numeric variables will be edited as 18-digit integers, zero suppressed with leading minus sign (COBOL edit picture '-----------------9'). Floating-point numeric variables will be displayed in general floating-point format. No other formatting will take place.

When using an *RDA-reference* in a DISPLAY, specified fields within the RDA may be printed using either direct RDA reference or data item name (if the data item index is available).

Examples:

Example 1: Display two literals on the same line:

```
DISPLAY '**** LIST OF PRODUCTS IN THE CURRENT ' +
DISPLAY 'DATABASE *** '
.  Use the + to concatenate the output
.  of two DISPLAY.
```

or

```
DISPLAY '**** LIST OF PRODUCTS IN THE CURRENT ';
    'DATABASE *** '
.  Use the ; immediately after the second string
.  delimiter (') to continue the literal.
```

or

```
DISPLAY '**** LIST OF PRODUCTS IN THE CURRENT ' ;
    'DATABASE *** '
.  Use the ; preceded by at least one space
.  to continue the command.
```

Regardless of the method used, the output will appear as follows:

```
**** LIST OF PRODUCTS IN THE CURRENT DATABASE ****
```

Example 2: Tab to column 5 and display three totals.

```
DISPLAY 5 '*** Totals for X, Y and Z are:'XYZ
```

Example 3: Display two lines containing one total on each.

```
DISPLAY 'VALUE OF X = '  X
DISPLAY 'VALUE OF Y = '  Y
```

Example 4: Display the same information as in Example 3, but on one line.

```
DISPLAY 'VALUE OF X = '  X  ' AND VALUE OF Y = '  Y
```

Example 5: Display characters 1 through 8 of the RDA.

```
DISPLAY RDA (1,8)
```

Example 6: Display a field in the RDA beginning in output position 5.

```
DISPLAY 5 RDA PRODUCT-ID
```

Example 7: Display an RDA field with the output beginning in the position designated by the numeric variable, X.

```
DISPLAY @X RDA PRODUCT-ID
```

9.14 DO

The DO command can be used in two ways. First, the DO command may be used like the COBOL PERFORM statement. Used in this manner the DO must reference a defined procedure. A procedure must begin with a PROCEDURE label command and end with an ENDPROC command. If the optional WHILE/UNTIL clause is omitted, the procedure will be executed until the ENDPROC command is encountered, or a BREAK command is executed. If the WHILE/UNTIL clause is used, the procedure will be repeated per the condition specified (see below).

The second way in which the DO may be used is to control the execution of a group of commands in-line. In this manner, commands following the DO through a matching ENDDO command are executed repeatedly. This is called an in-line DO block.

Format-1, DO a defined procedure:

DO *procedure* [{UNTIL | WHILE} *conditional-expression*]

Format-2, DO in-line code:

DO [{UNTIL | WHILE} *conditional-expression*]
 commands(s)
ENDDO

Where *conditional-expression* may take the following form:

{*numeric-literal* | *variable* | RDA *RDA-reference*} ;
 { [NOT] {= | < | >} | <> | <= | >=} ;
 {*numeric-literal* | *variable* | *special-name* '*alpha-literal* ' | RDA *RDA-reference*}

Conditional expressions may be in any form acceptable to the IF command including the AND/OR extensions. Please see the IF command for a complete discussion of conditional expressions.

ENDDO may be abbreviated ENDD.

Format-1 Rules:

DOing a procedure differs from an in-line DO in that it may be entered from several different places throughout the user's program. The procedure is formally defined by the PROCEDURE and ENDPROC commands. The format for defining a procedure is:

Procedure-name PROCEDURE
 Command(s)
[*label*] ENDPROC

PROCEDURE may be abbreviated PROC, with ENDP for ENDPROC.

If the procedure is entered with a DO UNTIL command, the condition will be tested each time the ENDPROC is executed. If the condition is met (TRUE), control will go to the command following the DO. Otherwise, the PROCEDURE will be reentered from the beginning. Using a DO UNTIL, the procedure will always be executed at least once.

If the procedure is entered with a DO WHILE command, the condition will be tested at the beginning (at the PROCEDURE command). If the condition is not true, control returns to the command following the DO, otherwise the procedure is reentered.

A procedure may be exited immediately by executing a BREAK command. The BREAK simply causes control to be passed back to the next command following the DO regardless of conditions.

A PROCEDURE / ENDPROC pair may not occur within a defined procedure (may not be nested). A PROCEDURE may call (DO) other procedures, including itself.

Format-2 Rules:

The DO must be terminated by a matching ENDDO. If no WHILE/UNTIL clause is used, the DO block will be repeated until a BREAK command is executed; otherwise, the same WHILE/UNTIL rules apply as in format-1. In-line DO blocks may be nested up to 50 levels.

Examples:

DOing defined PROCEDURES (Format-1):

```
MAIN-CONTROL
 X = 0
 DO PROC-1                          .  See NOTE-1
 X = 0
 DO PROC-1 UNTIL X = 100            .  See NOTE-2
   . . .
 DO PROC-1 WHILE X < 900            .  See NOTE-3
   . . .
PROC-1 PROCEDURE                    .  Procedure label
 X = X + 1
 IF X > 500
   BREAK                            .  Unconditional exit
 ENDIF
 DISPLAY X
 ENDPROC
```

NOTE-1: PROC-1 will be performed 1 time.

NOTE-2: PROC-1 will be performed 100 times and the values of X displayed will be in the range from 1 through 100.

NOTE-3: PROC-1 will be performed 400 times and the values of X displayed will range from 101 through 500. The WHILE condition will never be met because of the BREAK command within the IF.

DOing in-line commands (Format-2):

```
. . .
 FETCH4 NEXT PART-MAST PARTS AREA
 FETCH3 FIRST PART-USAGE SET
 DO WHILE ERROR-NUM = 0             .  See Note-4
   DISPLAY "Usage codes for step number:" +
   DISPLAY PU-STEP-NO
   SUB1 = 1
   DO WHILE SUB1 <= PU-NO-ENTS      .  Nested DO
   AND PU-USE-CODE :SUB1 <> $ALPHA
     DISPLAY PU-USE-CODE :SUB1
     SUB1 = SUB1 + 1
   ENDDO
   FETCH3 NEXT PART-USAGE SET
 ENDDO
. . .
```

NOTE-4: If the FETCH3 FIRST results in an ERROR-NUM = 13 (no members in the set), the entire DO block will be skipped.

9.15 DUMP

The DUMP command is used to produce an octal, and ASCII, dump of the contents of the RDA, a specified variable or database data name (DBDN), or to take a full object dump programmatically (same dump as produced by the OBJECT directive). When DUMPing the RDA, the user may specify any range within the RDA either by specific word boundaries, or by DMS 2200 record, BIS DTM record (queue-alias) or PCIOS file name. If DUMP is given with no parameters, the entire RDA will be dumped. An octal DUMP of a variable will include the internal control portions as well as the data. A DBDN DUMP will include only the current contents of the specified database data name. If the DUMP ALL is used, a full object dump will be taken and program execution will resume.

Format:

DUMP { [*start-word*] *number-of-words* | *filename* | *record-name* | *queue-alias* | ;
variable | DBDN *database-data-name* | ALL }

DUMP may be abbreviated DU.

Examples:

```
DUMP 20              .  Dump words 1 through 20.
DUMP 21,4            .  Dump words 21 through 60.
DUMP PART-MASTER     .  Dump PART-MASTER record area.
DUMP ALL             .  Take a full object dump and
                     .    continue processing
DUMP DBDN MS-ANAME   .  Dump a database data name
DUMP BIS-ROW         .  Dump queue-alias retrieved/sent
                     .    via  DTM
```

9.16 EDIT

The EDIT command is used to display either numeric variables or numeric data in the RDA using an edit mask.

Using the EDIT command, one numeric variable or RDA data item may be edited and moved to the print buffer. If the item contains implied decimal positions, it will be scaled to match the decimal position in the edit mask.

Format:

> EDIT ['*column-literal* ' | @*numeric-variable*] ;
> {*numeric-variable* | RDA *RDA-reference*} ;
> [*] {'*edit-mask-literal* ' | *edit-mask-variable*} [+]

EDIT may be abbreviated ED.

Edit masks are furnished as a literal or alpha variable containing mask characters. The edit mask is similar to that used in COBOL. The following edit mask characters may be used for editing decimal numbers:

1. Insertion characters:

   Any character may be used as an insertion character; however, there are two special insertion characters, which are:

   "B" blank;

   "H" hyphen (-).
2. Zero suppression control characters are:

   "Z" suppress leading zeros;

   "9" display leading zeros.
3. Sign control characters are:

   "+" print plus or minus sign depending on the value of the number;

   "–" print minus sign only if the number is negative.
4. Floating characters are:

   "$" print dollar sign to the left of the most significant digit when the dollar sign is the first character of the mask and followed by the character, "Z" ;

   "–" for negative numbers, print minus sign to the left of the most significant digit when the minus sign is the first character of the mask and is followed by the character, "Z".

Decimal number edit mask usage examples:

The number -1234, EDITed with a mask of '9999+' would print as:

```
'1234-'.
```

The same number EDITed with a mask of 'ZZ,ZZZ' would print as:

```
' 1,234'.
```

The number, 5678.909, EDITed with a mask of 'ZZZ.99+' would print as:

```
'678.91+'
```

Note: Number was scaled to mask and rounded.

The number, -1234, EDITed with a mask of '$ZZZ,ZZZ' would print as:

```
'$1,234'
```

The same number, EDITed with a mask of '$999,999' would print as:

```
'$001,234'
```

The same number , EDITed with a mask of '-ZZZ,ZZZ' would print as:

```
'-1,234'
```

The same number, EDITed with a mask of '-999,999' would print as:

```
'-001,234'
```

Output may be temporarily suppressed by using the "+" symbol, as with the DISPLAY command. Column number specifications and automatic TAB character insertions also function as with the DISPLAY command.

An option may be used to indicate when a value overflows an edit mask. If an asterisk (*) is placed in front of the edit mask literal or variable, and an overflow condition occurs, the entire output (allocated by the edit mask) will be filled with asterisks. Without the asterisk, if the numeric value being edited is greater than the significant output positions allowed by the edit mask, the remaining high-order digits are truncated.

Examples:

```
X = 1234567
.
EDIT X 'ZZ,ZZ9.99' .  No overflow indication
EDIT X * 'ZZ,ZZ9.99' .  Overflow indication
```

The above would produce the output "34,567.00" (high-order truncated) from the first EDIT, and "*********" (indicating overflow has occurred) from the second EDIT.

The use of the overflow indication option does not affect the rounding or truncation of low-order digits. The numeric value is always scaled to match the decimal precision of the edit mask, and either rounded or truncated as necessary depending upon whether or not rounding is turned on.

There are three possible methods to edit floating-point numbers: fixed decimal format, generalized format and scientific format. In each case, the total output width and maximum precision will be specified. The floating-point edit mask formats are as follows:

Fixed decimal format:

F*w*.*p*

Generalized format:

G*w*.*p*

Scientific format:

S*w*.*p*

Where "*w*" is an integer in the range of 1 to 32 that specifies the output field width, and "*p*" is an integer in the range of 0 to "*w*" that specifies the maximum precision to be displayed. In generalized format, "*p*" indicates the total number of significant digits.

In scientific format, the rules used to display images are as follows:

1. If *w* – *p* <= 5 and the value being displayed is positive or if *w* – *p* <= 6 and the value is negative, "*p*" significant digits are displayed. They will always be left justified. The exponent is displayed left justified and space filled up to *w* – *p* – 2 digits if positive or *w* – *p* – 3 digits if negative.
2. If *w* – *p* > 5 were the value is positive or *w* – *p* > 6 and the value is negative, "*p*" significant digits are displayed right justified and space filled with the sign preceding the value if negative. The exponent is displayed right justified and zero filled.

Examples:

Example 1: Assume X contains the value -12345.

```
EDIT X 'ZZ,ZZZ.99+' +
DISPLAY'/'+
EDIT X '-999999'
```

The output of the above commands would look like this:

```
Columns:1...5....10...15...20
        12,345.00- / -012345
```

Example 2:

```
DEF A PROD-MASK '999B9999'        .  Product-id
                                  .    EDIT MASK ...
EDIT 3 RDA PRODUCT-ID PROD-MASK   .  Display EDITed
                                  .    product-id.
```

If product-id is 1012046, output from the above EDIT statement will print starting in column 3 as follows:

```
Columns:1...5....10...15...20
          101 2046
```

Example 3:

```
EDIT RDA (6,4) '99/99'
```

If positions 6 through 9 of the RDA contain 0922, output from the above EDIT statement will print starting in column 1 as follows:

```
Columns:1...5....10...15...20
        09/22
```

Example 4: Edit the value of Y into the column designated by X:

```
EDIT @X Y '-ZZZ,ZZZ'
```

If X = 5 and Y = -2424, output from the command would appear as follows:

```
Columns:1...5....10...15...20
              -2,424
```

Example 5: Floating-point numeric editing. Given the following:

```
DEFINE FP FLOAT-NUM 1.2400E+2
    EDIT FLOAT-NUM 'F9.5'  .  Edit fixed decimal format
    EDIT FLOAT-NUM 'S9.5'  .  Edit scientific format
    EDIT FLOAT-NUM 'G9.5'  .  Edit generalized format
```

The edited output would look like this:

```
Columns:1...5...10...15...20
        124.00000
        1.2400E+2
        124.00
```

9.17 FACERR

This command is used to decode and display the facility status word after execution of the I-QU PLUS-1 CSF command. The specified variable must be numeric. FACERR will print the diagnostic message associated with each bit set in the facility status word. The display is directed to the current output (see the PCONTROL command).

Format:

FACERR *CSF-status-variabl*e

FACERR may be abbreviated FA or FE.

Example:

```
CSF X '@ASG,A MY*FILE.'     .  Assign file
IF X < 0                    .  Fac reject?
    FACERR  X               .  Display reason
    STOP 99                 .  Quit
ENDIF
```

9.18 GO

The GO command is used to transfer control of command execution to another point within the I-QU PLUS-1 program.

Format:

GO [TO] *program-label*

Example:

```
.  A SIMPLE LOOP...
START
    SET X = X + 1              .  START IS A PROGRAM LABEL
    IF X = 10
        DISPLAY 'DONE'
ELSE
    GO START                   .  GO BACK TO START
ENDIF
```

9.19 IF

The IF command is used to control conditional execution of I-QU PLUS-1 commands. The IF must be used in conjunction with an ENDIF command. If a false condition results, control is passed to the command following the corresponding ELSE or ENDIF. IF/ENDIF pairs may be nested to 10 levels.

Format:

```
IF conditional-expression
   [ {AND | OR} ] conditional-expression
      command(s)
[ELSE]
      command(s)
ENDIF
```

Where *conditional-expression* may be in the following forms:

```
{numeric-literal | variable | RDA RDA-reference} ;
   { [NOT] {= | < | >} | <> | >= | <=} ;
   {numeric-literal | variable | special-name | 'alpha-literal ' | RDA RDA-reference}
```

ELSE may be abbreviated EL, with ENDI for ENDIF.

When comparing two variables, both must be the same data type. When comparing two alphanumeric strings (variables or RDA), the length of the shortest argument will stop the comparison. For example: "ABCDEF" = "ABC" is true; "ABCDEF" = "ABCx" is false.

Alphanumeric string comparisons that use an "is less than" (<) or "is greater than" (>) operator will use one of two collating sequences depending upon the character set of the two operands. If both operands are ASCII, the ASCII collating sequence will be used. If either or both operands are FIELDATA, the FIELDATA collating sequence will be used.

When the IF command is used to test an RDA reference that has a data type of MAPNUM, the reference must be tested as a numeric item (e.g., IF RDA QTY-ON-HAND = 0).

The IF command may also be used to test for HIGH-VALUES, LOW-VALUES, ALPHABETIC, or NUMERIC values by using one of the following special names instead of an alphanumeric literal:

$HIVALS	All ASCII 177 (or 377) or Fieldata 77.
$LOVALS	All binary zeroes.
$ALPHA	All characters within ranges "A" through "Z", "a" through "z" (ASCII only) and space.
$NUM	All characters within range 0 through 9.
$SPACES	All spaces.

These special names may be used only with an equal or not equal test.

The IF command may be extended by using AND and OR commands immediately following the IF command. AND/OR logic operates as follows: all conditions connected by ORs are evaluated separately; all conditions connected by ANDs are evaluated together.

Consider the following compound condition:

```
IF A = 1
AND B = 2
OR C = 3
```

If A is 1, B is 3, and C is 3, the condition will be considered TRUE. This is because "C = 3" was true and ORed with "A = 1 and B = 2" that was false. This I-QU PLUS-1 condition is interpreted the same as "IF (A = 1 AND B = 2) OR C = 3" would be by COBOL.

The ELSE may also be used in conjunction with the IF. The only restrictions on the use of AND, OR and ELSE is that they must be entered on separate command lines.

Examples:

```
FETCH3 NEXT PART-USAGE SET .  Fetch next of set
IF ERROR-NUM = 7 .  End of set ...
OR ERROR-NUM = 13 .  OR set empty
    IF ERROR-NUM = 7 .Isit7?
        GO END-SET .  ERROR-NUM is 7
    ELSE
        GO ERROR-ROUTINE .  Must be 13
    ENDIF
ENDIF
GO PROCESS-REC .  Continue processing
...
```

In the following, C-O-R is a 30-character alpha-string variable. If it contains any string that begins with "PRODU", the IF will be evaluated as true.

```
DEF RDA NEW-RATE (*,10) MAPNUM .00
DEF N MAX-RATE 25.00
...
    IF C-O-R = 'PRODU'
        GO ....
    ENDIF

    IF RDA NEW-RATE > MAX-RATE
    AND RDA PRODUCT-TYPE = 'A'
        DO RATE-TOO-HIGH
    ENDIF
```

9.20 PCONTROL

The PCONTROL command is used to direct the disposition of I-QU PLUS-1 print output. I-QU PLUS-1 output may be directed to PRINT$ (the terminal in demand mode), the system console, or an alternate print file assigned by I-QU PLUS-1. In addition, print may be directed to both an alternate print file and PRINT$. PCONTROL functions are defined as follows:

BRKPT	Redirect all print to alternate print file.
CONSOLE	Redirect all print to the system console.
CLOSE	Close (@BRKPT and @FREE) the current print file.
ECHO	Redirect the output to both the reply and the alternate file.
EJECT	Page advance immediately (redirected output).
LOCAL	Direct output to PRINT$ (initial mode).
OPTION	Send an optional print control image to the current print file.

When in BRKPT mode, batch mode (@START) or if the B-option was used on the I-QU PLUS-1 processor call, I-QU PLUS-1 will use the 132-character print width for all output; in all other cases, the print width will be 76 characters.

Format:

```
PCONTROL {BRKPT ['[qualifier*]filename' | variable] | ;
          CONSOLE | ;
          CLOSE | ;
          ECHO ['[qualifier*]filename' | variable] | ;
          EJECT | ;
          LOCAL | ;
          OPTION {print-control-image-variable | 'print-control-image-literal '} }
```

PCONSOLE may be abbreviated PC, with CONS or CONSOL for CONSOLE, EC for ECHO, E for EJECT, L for LOCAL and OPT for OPTION.

If the CONSOLE option is used, the print line should be limited to 50 characters to prevent truncation at the system console.

If the optional [*qualifier*]**filename* is used with BRKPT or ECHO, I-QU PLUS-1 will use the specified file. If the file does not exist, it will be created. Each time a new file is specified, the previous file will be @BRKPTed (closed) and @FREEed (released).

If no [*qualifier*]**filename* is specified with BRKPT or ECHO, I-QU PLUS-1 will attempt to create a file with a name in the range I$QUPRT0 through I$QUPRT9 (I-QU PLUS-1's default alternate print files). If all ten alternate print files already exist, an error will occur.

Upon exiting I-QU PLUS-1, any I$QUPRTn files created will be @BRKPTed and @SYMed. User specified files will be @BRKPTed and @FREEed.

The [*qualifier*]**filename* may be given as either an alpha literal or an alpha variable.

The PCONTROL command with the OPTION clause is used to call for special forms, change margins, or print headings. PCONTROL will output the print control function image furnished by the user in the *print-control-image-variable*, or the *print-control-image-literal*, to the current print file. Valid print control function formats will be found in the EXEC Programmers Reference Manual. In LOCAL mode, all OPTION images are ignored.

Below, some of the commonly used print control function formats are shown:

Special Forms Request:

S,*forms-name-text*

Headings:

H,*options*,*page-number-to-start*,*heading-text*

Where *options* are: N = Do not print heading; X = Suppress printing page number and date.

Line Skip:

L,*line-number*

Where *line-number* is an integer value one less than the desired line number. For example, to skip to line 15, use "L,14".

Margins:

M,*lines-per-page*,*at-top,at-bottom*,*lines-per-inch*

More than one print control function may be entered in a single image if separated by periods. For example, the following image in the print output will call for form 1-PART 9X11 to be mounted, and will cause each page to be headed with CUSTOMER LIST.

```
'S,1-PART 9X11.H,,1,CUSTOMER LIST'
```

Examples:

```
PCONTROL BRKPT             .  Breakpoint to default file
PCONTROL BRKPT 'MY*PRTF'   .  Breakpoint to a user file
```

The following will set 88 lines per page, with four lines at top and bottom margin, and print eight lines per inch.

```
PCONTROL OPTION 'M,88,4,4,8'
```

The following will call for special INVOICE forms to be mounted when the print file is printed.

```
PCONTROL OPTION 'S,INVOICE'
```

The following will cause all report pages to have the title "A/R SUMM" with the current date and page numbers starting at page 1.

```
PCONTROL OPTION 'H,,1,A/R SUMM'
```

9.21 ROUND

The ROUND command is used to toggle automatic rounding of arithmetic results on and off. Automatic rounding is initially ON. Rounding may occur on any command that produces a numeric result.

Format:

ROUND {ON | OFF}

ROUND may be abbreviated ROU.

Examples:

```
DEF N INT-NUM 0
DEF N DEC-NUM .000
...

    DEC-NUM = 12.567
    INT-NUM = DEC-NUM
    DISPLAY INT-NUM
    ROUND OFF
    INT-NUM = DEC-NUM
    DISPLAY INT-NUM
```

The preceding would produce the following:

```
                       13 <--- (Result ROUNDed)
                       12 <--- (Result not ROUNDed)
```

9.22 SCAN

This command is used to search for a specified string of characters within a field in the RDA. If the string is found, its offset from the start of the field being searched is returned, else a -1 is returned.

Format:

SCAN RDA *RDA-reference* {'*alpha-literal* ' | *variable*} *length-result-variable*

SCAN may be abbreviated SC.

The *length-result-variable* is used to specify the length of the string to be compared in the scan. If it is set to zero, the length of the search literal ('*alpha-literal* ') or *variable* will be assumed. The same variable (*length-result-variable*) will be used to return the result of the scan. If the string is found, the variable is set to the offset from the first character of the field at which the find was made. If the string was not found, a -1 is returned.

The *length-result-variable* may not exceed the length of the search literal or variable.

The search literal (*alpha-literal*) or *variable* may not exceed 256 characters.

The *length-result-variable* should be initialized each time a SCAN is to be executed.

Examples:

Search for all customer names containing the name "JONES".

```
...
FETCH3 NEXT CUSTOMER-REC CUSTMST AREA
IF ERROR-NUM = 7
    GO END-AREA
ENDIF
SET X = 0                          .  Initialize length/result variable.
SCAN RDA CUSTOMER-NAME 'JONES' X   .  SCAN for string
IF X > -1                          .  Found?
    DISPLAY RDA CUSTOMER-NAME      .  Display name
    SET FIND-CNT = FIND-CNT + 1    .  Count it
ENDIF
...
```

The following procedure will perform a table lookup to convert an alpha code to its numeric value. The following table is stored in the RDA:

```
1001     1010      1020      1030      1040      1050      1060
   |...|....|....|....|....|....|....|....|....|....|....|....|...
   A 01B 02C 03D 04E 05F 06G 07H 08I 09J 10K 11L 12M 13N 14O 15...

DEFINE A ALPHA-CODE 2
DEFINE N NUM-CODE
    ...
    ALPHA-CODE = RDA REGION-CODE
    DO CONVERT-CODE
    IF NUM-CODE = 0
        DISPLAY 'ERROR-INVALID ALPHA REGION CODE'
    ELSE
        DISPLAY NUM-CODE
    ENDIF
    ...

CONVERT-CODE PROCEDURE
```

```
X = 0
NUM-CODE = 0
SCAN RDA (1001,104) ALPHA-CODE X       .  Search for code
IF X NOT = -1
    SET NUM-CODE = RDA (1003,2) UN9:X  .  Get value
ENDIF ...
```

9.23 SET

This form of the SET command will transfer data to the specified data storage area variable or RDA reference. The SET command can also determine the:

- Current or ending print position of the print buffer ($PBUFF);
- Start position, length and data type of RDA references;
- Length of an item if leading and trailing spaces were suppressed;
- Number of decimal positions on numeric items;
- Switch settings (ON/OFF) of the ROUND, DECIMAL and CASE commands.

Literals specified will first be stored in the data storage area by the command editor. In all cases of this form of SET, the data type of the object variable or RDA reference must be the same as the literal, variable or RDA reference from which the value will be obtained.

When the SET command is used to set an RDA reference that has a data type of MAPNUM, the reference must be set as an alphabetic item (e.g., SET RDA QTY-ON-HAND = '0'). If a numeric variable or numeric RDA reference is set to the contents of an RDA reference with a data type of MAPNUM, I-QU PLUS-1 will convert the MAPNUM item to an internal format and then decimal align it (if required) for the receiving field. Any excess fractional digits will be rounded/truncated.

Format:

[SET] {*variable* | RDA *RDA-reference*} = ;
  {*variable* | ;
   special-name | ;
   RDA *RDA-reference* | ;
   '*alpha-literal* ' | ;
   (*integer-literal*,*integer-literal*) | ;
   expression | ;
   $VAL *value-operation*}

The command keyword "SET" is implied and may be omitted. SET may be abbreviated S.

Where *expression* may have the following format:

{*numeric-literal* | *numeric-variable* | RDA *RDA-reference*} ;
  {+ | – | * | /} ;
  {*numeric-literal* | *numeric-variable* | RDA *RDA-reference*}

Special-name may be one of four possible values: $HIVALS, $LOVALS, $SPACES and $PBUFF. $HIVALS and $LOVALS correspond to ASCII COBOL HIGH-VALUES and LOW-VALUES and may be used in setting alpha variable or RDA items. $SPACES may be used to set an alpha variable or RDA item to all spaces. $PBUFF refers to the current contents of the print buffer. It may be used to set an alpha variable or RDA item. Using this option will set the specified variable to the contents of the print buffer, and then space fill the print buffer as if it had been printed.

Where *value-operation* may have the following format:

```
    {CURRPCOL | ;
     LASTPCOL | ;
     ROUND | ;
     DECIMAL | ;
     CASE | ;
     {STARTPOS {variable | RDA RDA-reference} ;
       TRIMLEN {variable | RDA RDA-reference} | ;
       LENGTH {variable | RDA RDA-reference} | ;
       DECPOS {variable | RDA RDA-reference} | ;
       TYPE {variable | RDA RDA-reference} } }
```

The $VAL option is used to determine a value based on a specific function. The value returned will always be numeric; therefore, the receiving *RDA-reference* or *variable* (to the left of the =) must always be numeric. The other *RDA-reference* or *variable* (to the right of the =) may be either numeric or alphabetic within the limits described in the table below. The functions of the $VAL option are:

Function	**Purpose**
CURRPCOL	This function returns the current print position maintained by I-QU PLUS-1 in the print buffer ($PBUFF). The current print position is the next available print position for a subsequent DISPLAY, TRIMDISP, EDIT or TRIMEDIT command unless overridden by a column specification. This function is only meaningful when preceded by a DISPLAY, TRIMDISP, EDIT or TRIMEDIT command utilizing the plus (+) option.
LASTPCOL	This function returns the highest print position currently in the print buffer ($PBUFF). This function is useful to ensure that printed output will not exceed the physical limitations of the output device. This function is only meaningful when preceded by a DISPLAY, EDIT, TRIMDISP or TRIMEDIT command utilizing the plus (+) option.
ROUND	This function returns the value of the ROUND switch. A value of zero is returned if the ROUND switch is off, and a value of 1 is returned if the ROUND switch is on (see the ROUND command, page 9-37).
DECIMAL	This function returns the value of the DECIMAL switch. A value of zero is returned if the DECIMAL switch is off, and a value of 1 is returned if the DECIMAL switch is on (see the DECIMAL command, page 9-17).
CASE	This function returns the value of the CASE switch. A value of zero is returned if the CASE switch is off, and a value of 1 is returned if the CASE switch is on (see the CASE command, page 9-6).
STARTPOS	This function returns the start position of the named RDA-reference within the RDA. If a variable is specified, a start position of zero (0) is returned.
TRIMLEN	This function returns the size of the significant portion of the RDA-reference or variable. The returned value represents the size as if displayed by the TRIMDISP command with leading and trailing spaces deleted. This function is useful when determining the size of a single data item or an assembled print line for centering or right justification (see the examples below).

Function	**Purpose**
LENGTH	This function returns the actual length of an RDA-reference.If a variable is specified, a length of zero (0) is returned.
DECPOS	This function returns the number of decimal positions defined on a numeric RDA-reference or numeric variable. If an alphanumeric RDA-reference or alphanumeric variable is specified, a zero (0) result is returned.
TYPE	This function returns the type code of the RDA-reference or variable. The following values are defined:

ASCII		*FIELDATA*		*Floating-Point*		*BIS*		*Variables*	
DISP	16	DISP	48	FP1	5	MAPNUM	22	DEF A	201
A9	16	A6	48	FP2	6			DEF FP	206
SN9	19	SN6	51					DEF N	200
UN9	18	UN6	50						
COMP	1	COMP4	33						
SB9	1	SB6	33						
UB9	0	UB6	32						

Examples:

Given the following definitions:

```
DEFINE N OLD-DBP
DEFINE RDA DBP-RDA (1000,4) UB9
.  Database Pointers (DBP)    and Database Keys
.  (DBK) are always UB9
DEFINE N N1 .000
DEFINE N N2
DEFINE A HOLD-1 6
DEFINE A HOLD-2 6
.
    RDA DBP-RDA = OLD-DBP         .  Named RDA ref.
    RDA (1000,4) UB9 = OLD-DBP    .  Direct RDA ref.
.
    X = 0
    N1 = 0
    N1 = X + 1
    N2 = -5
    N1 = N2 + N1                  .  N1 should now contain -4.000
.
    HOLD-1 = 'ABCDEF'
    HOLD-2 = HOLD-1               .  Move 'ABCDEF' to HOLD-2 also
    RDA (1,6) = HOLD-1            .  Move HOLD-1 to RDA pos. 1-6
    HOLD-1 = RDA (3,8)            .  Move RDA pos. 3-8 to HOLD-1
                                  .  Pos. 9-10 will be truncated
    HOLD-2 = $LOVALS              .  Move binary zeros to HOLD-2
.
    RDA PRODUCT-ID = 1011234
    RDA PRODUCT-DESC = 'ABCD'
    RDA (5,4) COMP = -505
    RDA PRODUCT-ID = RDA LINE-PRD
```

```
          RDA (1,100) = $LOVALS         .  Clear 100 chrs of RDA
```

The following form of the SET packs two integer numbers into a numeric variable.

```
      G-AKEY = (1,5)                    .  Set PAGE-NUM to 1
                                        .    and RECORD-NUM to 5
```

The actual decimal value of G-AKEY after the above SET would be 262149 (octal 0000001000005).

The following illustrates use of the $PBUFF special value. First, values are formatted into the print buffer using EDIT, DISPLAY, TRIMEDIT and TRIMDISP commands. The '+' is necessary to prevent I-QU PLUS-1 from printing the string.

```
      DISPLAY 'The total is ' +
      TRIMEDIT N1 '-ZZ,ZZZ.99' +
                                 .  + prevents $PBUFF from being cleared
      HOLD-THING = $PBUFF
```

HOLD-THING will then contain "The total is -4.00" and the print buffer will be cleared.

Since an RDA item that is defined as having a data type of MAPNUM cannot be set as a numeric item, $PBUFF can be used as a staging area for numeric data. For example:

```
      DEF RDA NEW-RATE (*,10) MAPNUM .00
      DEF N RATE-INCREASE .00
      . . .

          RATE-INCREASE = RDA NEW-RATE
                                 .  Numeric variable set to MAPNUM
          RATE-INCREASE = RATE-INCREASE * 1.10       .  Add 10%
          TRIMEDIT RATE-INCREASE 'ZZZ,ZZZ.99' +
          RDA NEW-RATE = $PBUFF
```

To avoid exceeding the length of an output device (e.g., 100 characters) the following $VAL LASTCOL and TRIMLEN function may be used. This example creates a record containing trimmed amounts delimited by a single space (123.56 98.76):

```
      DEF F AMOUNTS-OUT SEQ 100,10
      DEF RDA OUT (1,100)
      DEF RDA AMOUNT (101,10) .00
      . . .

          X = 0
          DO WHILE X < 500           .  Table contains 50 AMOUNTs
              DO UNTIL Y > 100       .  Next TD would be > 100
                  TD RDA AMOUNT :X +
                  DISPLAY ' ' +
                  Y = $VAL LASTCOL   .  Get last pos. of $PBUFF
                  X = X + 10         .  Bump by 10 for next AMOUNT
                  Z = $VAL
                  TRIMLEN RDA AMOUNT :X
                                     .  Get trim length of it
                  Y = Y + Z
          ENDDO
          RDA OUT = $PBUFF           .  Clears $PBUFF
          WRITE AMOUNT-OUT
      ENDDO
```

The following example will center a staged header using variable substitution:

```
DEF A HEADER 132
. . .
    DISPLAY 'The following information is for ' +
    TRIMDISP RDA C01-CUSTOMER-NAME +
    DISPLAY ', ' +
    TRIMDISP RDA C01-CITY +
    DISPLAY ', ' +
    TRIMDISP RDA C01-STATE +
    X = $VAL LASTPCOL         .  Get last pos.  in $PBUFF
    X = 132 - X
    X = X / 2                 .  Get start pos. of center
    HEADER = $PBUFF
    TRIMDISP @X HEADER        .  HEADER begins at (@) X
. . .
```

9.24 SHIFT

The SHIFT command shifts lower-case characters to uppercase characters, or vice versa. FIELDATA RDA data items will not be affected by the SHIFT command. The shift only changes characters "A" though "Z" (to LOWER), and "a" through "z" (to UPPER).

Format:

SHIFT {*variable* | RDA *RDA-reference*} TO {UPPER | LOWER}

SHIFT may be abbreviated SHI.

Example:

```
DEF A ANSWER 1
.
. *** Accept until a valid answer is received ***
    DO WHILE ANSWER <> 'Y'
    AND ANSWER <> 'Y'
        ACCEPT ANSWER "Want to continue (Y or N)?" CONSOLE
        SHIFT ANSWER TO UPPER
    ENDDO
    ...
```

9.25 STOP

All I-QU PLUS-1 programs must be terminated by the STOP command. A STOP command is automatically generated following the last command executed as a result of the RUN directive.

Format:

STOP {*exit-code-literal* | *exit-code-variable* | EXIT}

If the EXIT option is not used, the I-QU PLUS-1 program will not terminate. Instead, I-QU PLUS-1 will revert to the mode that was in effect at the time the RUN directive was issued. If the *exit-code-literal* or *exit-code-variable* is used, I-QU PLUS-1 will display the code as the STOP is executed. If EXIT is used, the I-QU PLUS-1 session will be terminated.

9.26 SWGET

The SWGET command provides a means of setting a numeric variable's value to the state of a switch in the run unit's condition word. The state can be either zero (for OFF), or one (for ON). The condition word switches are numbered to correspond with ASCII COBOL (See the ASCII COBOL Programmer Reference Section on Switch-Status Condition). The state of condition word switches 1 through 24 may be obtained. Condition word switches may be set externally within the run via another program or @SETC or at run initiation time via the condition word parameter on the @START command or the console ST key-in.

Format:

SWGET *switch-number variable*

SWGET may be abbreviated SWG.

The *switch-number* must be an integer in the range 1 through 24. The *variable* must specify a decimal integer numeric variable.

Example:

```
SWGET 5 COND
IF COND = 1
    DO MONTHLY-PROCESS
ELSE
    DO DAILY-PROCESS
ENDIF
```

9.27 SWSET

The SWSET command allows the setting of a switch in the run's condition word. Only switches 13 through 24 may be altered. The new value may be used internally in I-QU PLUS-1 via the SWGET command or externally via ECL @TEST and @JUMP commands.

Format:

SWSET *switch-number* {ON | OFF}

SWSET may be abbreviated SWS.

The *switch-number* must be an integer in the range 13 through 24.

Example:

```
PCONTROL CONSOLE
IF TRANSACTION-COUNT = 0
    DISPLAY 'No transactions, transaction';
        ' last step will be skipped.'
    SWSET 13 ON
ELSE
    DISPLAY 'Transactions total = ' +
    TRIMDISP TRANSACTION-COUNT
    SWSET 13 OFF
ENDIF
```

9.28 TABS

The TABS command is used to turn automatic tab insertion on and off. If on, a tab character will be inserted in the first output position of every DISPLAY, EDIT, TRIMDISP or TRIMEDIT followed by the data. The initial setting of TABS is off.

Format:

TABS {ON | OFF}

TABS may be abbreviated TA.

Example:

```
IF ERROR-NUM NOT = 13
    TABS ON
    .  Output TABS in front of each field
    DISPLAY RDA KEY       +
    DISPLAY RDA DESC      +
    DISPLAY 27 RDA STATUS +
...
    TABS OFF
ELSE
    DISPLAY '.  NO FIND ON ' +     .  No TABS
    DISPLAY KEY
ENDIF
```

The tab line produced in the above example would look like this:

```
columns     1...5....10...15...20...25...
            ^2009^DOUBLE WIDGET      ^S
```

The caret symbol (^) is used in the above example to represent the transparent tab character.

9.29 TIME

The TIME command is used to set the current time of day into the reserved variables TIME-MSPM and TIME. The variable TIME-MSPM will be set to the current time of day in milliseconds past midnight (MSPM). TIME will be set to an edited display consisting of hours, minutes, seconds and thousandths of seconds (example: 13:23:45:028). If the user furnishes the time in milliseconds as a numeric input variable, this command will convert the variable to the edited time format and place the result in the variable TIME. This command may be used to time various I-QU PLUS-1 program operations, or to limit run execution.

Format:

TIME [*time-variable* | *time-literal*]

TIME may be abbreviated TI.

Example:

```
DEFINE N ST-TIME                .  Variable to save start time.
DEFINE N ELAPSED-TIME           .  Variable to calc elapsed time.
...
TIME                            .  Set time -beginning of run.
ST-TIME = TIME-MSPM             .  Save for calculation later.
...
...
TIME                            .  Set current time -end of run.
ELAPSED-TIME = TIME-MSPM -ST-TIME
                                .  Calc elapsed time
IF ELAPSED-TIME > 60000         .  Is time > 60 seconds?
    TIME ELAPSED-TIME           .  Convert to edited form DISPLAY TIME
ENDIF
```

9.30 TRACE

The TRACE command is used to turn the I-QU PLUS-1 TRACE routines on or off. When TRACE is turned on, I-QU PLUS-1 will display the program counter (PC) value for each command executed. This command is often used in debugging I-QU PLUS-1 program logic.

Format:

TRACE {ON | OFF}

TRACE may be abbreviated TRA.

Example:

```
INVOKE ACCTSUB IN ACCT FILE ACCT*SCHEMA
DEF N TOT-ACCUM
...

    TRACE ON                  .  Will list the program counter of each
                              .  I-QU PLUS-1 command executed.
    F4 F ACCTREC ACCTAREA A
    DO WHILE ERROR-NUM = 0
    ...
```

9.31 TRANSFER

The TRANSFER command is used to perform record level movement of data within the RDA. All record TRANSFERs are specified by actual word addresses, defined file names, invoked DMS 2200 record names or defined BIS DTM queue-alias names. The TRANSFER command is much more efficient than the SET command when moving records within the RDA.

Format:

> TRANSFER {*filename* | *record-name* | *queue-alias* | SORT | *from-word*} TO ;
> {*filename* | *record-name* | *queue-alias* | ;
> SORT [*number-of-words*] | *to-word* [*number-of-words*] }

TRANSFER may be abbreviated TRANS.

Restrictions:

> Specifying the same name or word address in both the "from" and "to" fields is illegal.
>
> Specific word addresses cannot be used in both the "from" and "to" fields unless the *number-of-words* is specified.
>
> *Number-of-words* only applies when both *from-word* and *to-word* addresses are specific word addresses, or when SORT is specified in conjunction with a *from-word* or *to-word* address.
>
> The *number-of-words* to move when *filename*, *record-name* and/or *queue-alias* names are used will be determined by the length of the shortest or only defined area; *number-of-words* does not apply in this case.

Examples:

```
DEFINE RA R0009-MST 100       .  Alternate record area.
DEFINE RA SORT 200
...

    IF RDA 0009-KEY OF R0009-MST = 'AB0001'
        TRANSFER R0009-MST TO SORT
                              .  Move to SORT
        RELEASE R0009-MST     .  Release len.  of R0009-MST
    ENDIF
...
    RETURN AT END END-OF-JOB
    TRANSFER SORT TO 1,50     .  To words 1-50 of the RDA
    WRITE COBFIL
...
```

9.32 TRIMDISP

The TRIMDISP command performs the same functions as the DISPLAY for variables and RDA items, except that any leading and trailing spaces are not moved to the print buffer (except for alpha literals). Also, the next available output position will only be advanced by the number of significant characters actually moved to the print buffer. This command allows strings to be concatenated with no intervening spaces.

Format:

TRIMDISP [*column-literal* | @*numeric-variable*] {RDA *RDA-reference* | ;
{*variable-1* | '*alpha-literal-1*'} … {*variable-4* | '*alpha-literal-4*'] } [+]

TRIMDISP may be abbreviated TD or TRIMD.

A TRIMDISP with no other options will generate a blank line.

Only one *RDA-reference* may be used within a single TRIMDISP. However, from one to four alpha string literals (*alpha-literal-n*) and variables (*variable-n*), or a combination of each, may be displayed using a single TRIMDISP command.

Trailing and leading spaces will not be suppressed on alpha string literals.

Output may be temporarily suppressed by using the "+" symbol, as with the DISPLAY command. Column number specifications and automatic TAB character insertions also function as with the DISPLAY command.

Examples:

```
TRIMDISP MONTH '/' DAY '/' +
TRIMDISP YEAR
```

The above would display a date in the following manner:

```
3/5/1994
```

Right justify variable output on a 132-character print line:

```
DEF A SUBTOTALHOLD 132
DEF N SUBTOTAL
    DISPLAY 'Subtotal is ' +
    TRIMEDIT SUBTOTAL '$ZZ,ZZZ,ZZZ.99' +
    X = $VAL LASTPCOL         .  Get the length of display
    X = 132 - X               .  Compute starting column
    SUBTOTHOLD = $PBUFF       .  Set hold variable
    TRIMDISP @X SUBTOTHOLD    .  Print at (@) column X
```

9.33 TRIMEDIT

The TRIMEDIT command performs the same functions as the EDIT for variables and RDA items, except that any leading and trailing spaces are not moved to the print buffer. Also, the next available output position will only be advanced by the number of significant characters actually output to the print buffer. This command allows strings to be concatenated into the print buffer with no intervening spaces.

Format:

TRIMEDIT [*column-literal* | @*numeric-variable*] ;
{*numeric-variable* | RDA *RDA-reference*} ;
[*] {'*edit-mask-literal* ' | *edit-mask-variable*} [+]

TRIMEDIT may be abbreviated TE, TED or TRIME.

Output may be temporarily suppressed by using the "+" symbol, as with the DISPLAY command. Column number specifications and automatic TAB character insertions also function as with the DISPLAY command.

Example:

```
DEFINE N AVERAGE .000
DEFINE N COUNT 0
DEFINE A NAME 14
    ...
    TRIMDISP '*** ' NAME ' had ' +
    TRIMEDIT COUNT 'ZZZ,ZZ9' +
    DISPLAY ' sales, which averaged ' +
    TRIMEDIT AVERAGE '-ZZZ,ZZZ.999' +
    DISPLAY ' each.'
```

If NAME contains "SAM", COUNT contains 21 and AVERAGE contains 14.230, the preceding sequence of commands would produce the following line:

```
*** SAM had 21 sales, which averaged 14.230 each.
```

If the TRIMEDIT and TRIMDISP commands were replaced by EDIT and DISPLAY commands, respectively, the line would look like this:

```
*** SAM            had      21 sales, which averaged       14.230 each.
```

9.34 WAIT

The WAIT command is used to cause a voluntary delay in program execution. The delay is specified as an integer number of seconds.

Format:

WAIT {*numeric-literal* | *variable* | RDA *RDA-reference*}

Example:

```
ASG-FILE
CSF Z '@ASG,A MASTER.'
IF Z < 0                      .  Can't get file

    X = X + 1
    IF X < 10                 .  Less than 10 tries
        WAIT 5                .  Wait 5 more seconds
        GO ASG-FILE           .  Go try again
    ELSE
        DISPLAY "CAN'T ASSIGN MASTER"
        STOP EXIT
    ENDIF
ENDIF
```

9.35 WILDCARD

The WILDCARD command is used to set a wild card character to be used in alpha string compare operations of the IF and SCAN commands. The WILDCARD command may also be used to inhibit the use of any wild card characters in comparisons (the default).

Format:

WILDCARD IS {'*alpha-literal* ' | *variable* | RDA *RDA-reference* | NONE}

WILDCARD may be abbreviated WI.

If a *variable* or *RDA-reference* is used, it must define an alphabetic item. If the supplied item is greater than one character in length, the leftmost character will be used. The wild card character will stay in effect until it is changed either to another character, or deactivated by specifying the keyword NONE in the command.

Examples:

```
WILDCARD IS '%'                 .  Designate wild card char.
...
IF RDA PART-NUM = 'A%%%1%%%'
    DO SELECT
ENDIF
WILDCARD IS NONE                .  Clear wild card char.
```

Any time the PART-NUM contains an "'A" in the first position and a "1" in the fifth position, the test will be true regardless of the contents of the rest of the field.

Chapter 10: PCIOS and SFS 2200 File Interface

The number of Processor Common Input/Output System (PCIOS) and Shared File System (SFS) 2200 files that can be processed during one session is dependent upon how many files were configured when I-QU PLUS-1 was installed. The default is 10. Each file must be defined using the DEFINE F directive before being referenced by the PCIOS/SFS file handling commands. All file formats are supported.

For all PCIOS/SFS file input/output commands, data will be read into, and written from, the RDA beginning at position one, or at the beginning of the area specified by a DEFINE RA directive for the file.

Special PCIOS status codes returned from I-QU PLUS-1 can be found at the end of this chapter.

PCIOS error codes returned by the Processor Interface Module (PIM) can be found in the Unisys publication, PCIOS Administration and Programming Reference Manual (7831 0588). SFS 2200 error codes returned by the Logical Data Manager (LDM) can be found in the Unisys publication, UDS SFS 2200 Administration and Support Reference Manual (7831 0786).

10.1 PCIOS/SFS File Usage and Access Modes

The following table shows the allowed access and usage modes for each of the file types:

File Type	**Usage Modes**				**Access Modes**
	INPUT	OUTPUT	UPDATE	EXTEND	
SEQ* (Sequential)	YES	YES	YES*	YES	SEQ
DIRECT (Also called Relative or DSDF)	YES	YES	YES	YES	DIRECT
INDEXED** (Also called MSAM)	YES	YES	YES	NO	SEQ, RANDOM, DYNAMIC
ISAM	YES	YES	YES	NO	SEQ, RANDOM, DYNAMIC
TAPE	YES	YES	NO	YES	SEQ

Table 10-1: File Usage/Access Modes

* SEQ files may be either disk or tape; however, tape files cannot be opened with a usage mode of UPDATE.

** INDEXED files may have a primary key, or a primary key with up to 19 alternate or secondary keys.

The file's type is specified in the DEFINE F directive, while the USAGE and ACCESS modes are specified on the file OPEN command.

10.2 PCIOS/SFS File Definition (DEFINE F)

Before a PCIOS/SFS file may be referenced, it must be defined. I-QU PLUS-1 supports standard file formats. Files may be opened, referenced and closed as necessary. All file input and output is done through the RDA.

If multiple records are to be read and compared, the user may set up alternate record areas using the DEFINE RA directive, or move necessary data items to a defined variable storage location.

There are several formats used for file definition depending on the file type. The following file types may be defined:

SEQ	PCIOS sequential SDF files.
DIRECT	PCIOS direct (or relative) files (DSDF).
INDEXED	PCIOS indexed-sequential files (MSAM)
TAPE	PCIOS ANSI tape files.
ISAM	COBOL indexed files assigned to "MASS-STORAGE"

Format-1 SEQ:

DEFINE F[F] *filename* SEQ *record-length,block-size* [RECORDS | CHARACTERS]

Format-2 DIRECT:

DEFINE F[F] *filename* DIRECT [SHARED *record-length* ;
maximum-records-in-file,*relative-key-variable*

Format-3 INDEXED:

DEFINE F[F] *filename* INDEXED [SHARED] ;
record-length,*block-size* [RECORDS | CHARACTERS] *record-key* ;
[*secondary-key-1* [DUPS]] [*secondary-key-19* [DUPS]]

Format-4 TAPE:

DEFINE F[F] *filename* TAPE *record-length*,*block-size* [RECORDS | CHARACTERS] ;
[*ANSI-format* [UNIVAC | IBM]]

Format-5 ISAM:

DEFINE F[F] *filename* ISAM [RECORDS | CHARACTERS] *actual-key-variable*

DEFINE may be abbreviated DEF; RECS or RECORD for RECORDS; and CHARS for CHARACTERS.

If DEFINE F is used, the *record-length* is based on ASCII characters. If DEFINE FF is used, the *record-length* is in FIELDATA characters.

SEQ files may be externally assigned to either tape or disk. If assigned to tape, they will be in SDF format.

DIRECT, INDEXED (MSAM) and ISAM files must be assigned to disk.

TAPE files must be assigned to tape.

For DIRECT and INDEXED files, the optional SHARE parameter allows these file types to be accessed by SFS 2200 under the control of the Universal Data System (UDS). SFS 2200 files are "shared" files that provide a site the same recovery, locking and queuing features used for other file systems under UDS Control. SFS 2200 files must have File Description Tables (FDTs) created by the UDS Unisys Repository Manager (UREP). See "Defining and Maintaining Storage Areas" in the Repository for ClearPath OS 2200 Administration Guide (7830 8087), or contact the appropriate data processing support personnel at your site for assistance. To access SFS 2200 files, the I-QU PLUS-1 program must first perform an explicit (single) BEGIN THREAD prior to the file OPEN. Use the RDMS command to perform the BEGIN THREAD (see Section 14.1, “RDMS Command”). Also, use the RDMS command to perform the END THREAD at the end of the program.

The *record-length* parameter specifies the number of characters per record. The *block-size* can be specified as the number of records or characters per block. If the optional RECORDS/CHARACTERS keyword is not specified, I-QU PLUS-1 will default to records per block. If block-size is zero (0), I-QU PLUS-1 will determine the block size by reading the physical record header. The zero specification is only valid for files that are opened as INPUT, UPDATE or EXTEND.

The *relative-key-variable* and *maximum-records-in-file* parameters apply only to DIRECT files. The variable named must be numeric. The *maximum-records-in-file* parameter is a numeric integer literal. It is used to initialize all records in a DIRECT file when it is opened for OUTPUT or EXTEND. It is ignored when the file is opened as INPUT or UPDATE.

For INDEXED files, the *record-key* and *secondary-key-n* are given as RDA references, either direct or by data item name. When defining an INDEXED file to be opened for INPUT, UPDATE or EXTEND, all keys must be specified in the same order and for the same positions as when the file was originally created. The DUPS clause indicates a secondary key that has duplicates allowed. A maximum of 19 secondary keys may be specified.

For files defined as tape, the following ANSI-formats are supported in I-QU PLUS-1:

ANSII Format	Meaning
U	Undefined
F	Fixed
V	Variable
FB	Fixed Blocked
VB	Variable Blocked (Default)

If UNIVAC is specified for an ANSI TAPE file, the record length will be internally adjusted to the next multiple of four (4) characters for even word counts. If IBM is specified, the character length will be as specified. The default is UNIVAC.

The *actual-key-variable* applies only to ISAM files (older version single-key indexed sequential) and must be defined as alphanumeric. The length of the *actual-key-variable* cannot exceed 1024 characters.

Most programming shops use MSAM as opposed to the old ISAM format. The programming techniques required to handle ISAM are the same as that required for MSAM; however, multiple key access is not supported for ISAM. If there is a question as to whether a file created in a COBOL program is MSAM or ISAM, look for the ASSIGN clause on the SELECT statement in the COBOL ENVIRONMENT DIVISION. If it is assigned to DISC, it is MSAM; if assigned to MASS-STORAGE, it is ISAM. The ISAM key cannot be specified as an RDA reference in the record. A user-defined variable must be specified.

Examples:

```
1. DEFINE F SAVE-FILE SEQ 100 50
2. DEFINE F DIR-FILE DIRECT 80 2500 REL-KEY
3. DEFINE F MSAM-FILE INDEXED 122 24 (1,4) (9,8) DUPS
4. DEFINE F ISAM-FILE INDEXED 50 20 PROD-NUM
5. DEFINE F OUTFIL TAPE 334 3584 CHARACTERS
6. DEFINE F IBMTAPE TAPE 90 22 FB IBM
7. DEFINE F DIR-SFS DIRECT SHARED 875 5000 REC-NO
8. DEFINE F MSAM-SFS INDEXED SHARED 122,0 (1,4) KEY2 DUPS
```

Explanation:

The file in Example 1 is a PCIOS sequential file with 100-character records, blocked 50 records.

Example 2 defines a PCIOS DIRECT (or relative) file. Records contain 80 characters. The file will be initialized to 2500 records if opened for OUTPUT. The variable, REL-KEY, must be set to the relative record number for read, write and delete operations.

Example 3 is an INDEXED file with a primary and one secondary key. The primary key is 4 characters starting in position 1, and the secondary key is 8 characters starting in position 9. Each record contains 122 characters, blocked 14.

Example 4 is an INDEXED file with only a primary key. The key is specified by its name, requiring a data item index file. The records are 50 characters, blocked 20.

Example 5 defines a PCIOS ANSI tape file with 334 character records, blocked 3,584 characters.

Example 6 is an ANSI tape file in IBM fixed blocked format.

Example 7 defines a shared relative I/O file under SFS 2200. The file will contain a maximum of 5000 records, which are each 875 characters in length. The relative key variable used to access the records is called REC-NO.

Example 8 sets up an MSAM shared file under SFS 2200. Each record contains 122 characters. The zero block size indicates that the file's blocking factor will be determined from the file header when the file is OPENed for INPUT or UPDATE. The primary key begins in the first position of the record and extends for four characters. The secondary key is named KEY2, and duplicates are allowed.

10.3 CDELETE

The CDELETE command may be used on any file type. It is used to delete the record just read. The file must be opened for update.

Format:

CDELETE *filename* [INVALID KEY *program*-label | STATUS *status-variable*]

CDELETE may be abbreviated CD.

The INVALID KEY clause only applies to DIRECT, INDEXED (MSAM) and ISAM files. The STATUS clause may be used in lieu of the INVALID KEY clause to receive the PCIOS/SFS error status from the PCIOS Processor Interface Module (PIM) or from SFS 2200. The *status-variable* must be defined as a numeric variable. The INVALID KEY and STATUS clauses are optional in conversational mode.

Example:

```
SET REL-KEY = 22                     .  Set RELATIVE key
READ REL-FILE INVALID KEY NO-FIND    .  Read DIRECT rec.
CDELETE REL-FILE INVALID KEY ERROR   .  Delete record.
```

10.4 CLOSE

The CLOSE command is used to close a PCIOS/SFS file. If the file is opened for OUTPUT, end of file processing will take place. Disk files will not automatically be @FREEed; tape files will automatically rewind unless the NO REWIND option is used. Files may be closed or re-opened with different usage and access modes.

Format:

    CLOSE *filename* [[WITH] NO REWIND]

CLOSE may be abbreviated CL.

10.5 OPEN

The OPEN command is used to ready a file for use in a program. It establishes the usage and access modes for the file. The file must be assigned to the run before the OPEN is executed.

Format:

```
    OPEN filename {INPUT | OUTPUT | UPDATE | EXTEND} ;
       {SEQ | DIRECT | RANDOM | DYNAMIC} [ [WITH] NO REWIND] ;
          [number-of-buffers-literal [BUFFERS] ]
```

OPEN may be abbreviated O, with I for INPUT, O for OUTPUT and U for UPDATE.

INPUT, OUTPUT, UPDATE and EXTEND are usage modes and have the following meanings:

INPUT	The file will only be read.
OUTPUT	The file will only be written. Any existing data will be destroyed.
UPDATE	The file may be read and written.
EXTEND	The file will only be written to. All data will be written after the last data block found when the file is opened.

All file types, except TAPE, may be opened for INPUT, OUTPUT or UPDATE. TAPE files may not be opened for UPDATE. SEQ, TAPE and DIRECT files may be opened for EXTEND.

SEQ, DIRECT, RANDOM and DYNAMIC access modes have the following meanings:

SEQ	The file will only be accessed sequentially.
DIRECT	The file will only be accessed directly by relative record position. File type must be DIRECT.
RANDOM	The file will only be accessed randomly by primary or secondary record key. File type must be INDEXED or ISAM.
DYNAMIC	The file may be accessed both sequentially and randomly. File type must be INDEXED or ISAM.

SEQ files may only be accessed sequentially. Files defined as DIRECT may only be opened for DIRECT access. If a DIRECT file is to be accessed sequentially, it may be defined as a SEQ file with a block factor of one. INDEXED files may be accessed either SEQ, RANDOM or DYNAMIC.

The WITH NO REWIND clause can be used with files DEFINEd as TAPE or SEQuential files assigned to tape for the purpose of stacking files (i.e., multiple file volumes).

The *number-of-buffers-literal* can only be used for PCIOS INDEXED (MSAM) files. It provides a means of increasing the number of I/O buffers used by I-QU PLUS-1. If the number of buffers is not specified, I-QU PLUS-1 will calculate the number of buffers based on the following formula:

Number of buffers = 4 + the total number of keys

MSAM performance may be improved by allocating additional buffers at the expense of additional memory requirements. If buffer allocation exceeds memory available for allocation, a runtime error occurs. This runtime error may occur while executing an I-QU PLUS-1 program with many open files, large file blocks, or an excessive number of buffers allocated. If this error occurs, buffer allocation should be reduced, or unused files closed. If less than five buffers are specified, I-QU PLUS-1 will allocate five buffers; if more than 24 buffers are specified, I-QU PLUS-1 will allocate 24 buffers.

Examples:

1. Open to read a file sequentially:
   ```
   OPEN TX-FILE INPUT SEQ
   ```
2. Open to create a new file sequentially:
   ```
   OPEN NEW-MAST OUTPUT SEQ
   ```
3. Open to create a new MSAM file in random key order:
   ```
   OPEN BOND-MAST OUTPUT RANDOM
   ```
4. Open to read an MSAM file randomly or sequentially:
   ```
   OPEN DEPARTMENT INPUT DYNAMIC
   ```
5. Open MSAM file to update existing records accessed by key:
   ```
   OPEN EMP-MAST UPDATE RANDOM
   ```
6. Open to create new file by relative record number:
   ```
   OPEN TAG-FILE OUTPUT DIRECT
   ```
7. Open to append records to a file:
   ```
   OPEN LOG-DUMP EXTEND SEQ
   ```

10.6 READ

The READ command is used to read a standard file. The file must have been previously opened. The AT END label must be specified when not in conversational mode.

Format:

READ *filename* [AT END {*program-label* | BREAK} | ;
INVALID KEY *program-label* | ;
STATUS *status-variable*] ;
[USING KEY *key-number-literal*]

The AT END clause only applies to sequential reads. The INVALID KEY clause only applies to DIRECT, INDEXED and ISAM files. The STATUS clause may be used in lieu of the AT END or INVALID KEY clauses to receive the PCIOS/SFS error status from the PCIOS Processor Interface Module (PIM) or from SFS 2200. The *status-variable* must be defined as a numeric variable. The AT END, INVALID KEY and STATUS clauses are optional in conversational mode.

BREAK may only be used instead of a label when the READ is contained within a defined procedure or an in-line DO block.

The USING KEY clause applies to MSAM files when accessing by an alternate record key. The *key-number-literal* specifies which alternate key to use from the file's definition (example: 1 = the first secondary key defined after the primary key; 2 = the second, etc.).

Following the READ of a SEQ file, the reserved numeric variable REC$LEN will contain the actual length of the record read in words. This variable may be useful in processing variable length data records.

If the file is TAPE, REC$LEN will contain the record length in characters.

Examples:

SEQuential File:

```
READ SEQ-IN-FILE AT END END-OF-RUN
```

DIRECT File:

```
SET DIR-KEY = 20                  .  Relative key = 20.
READ DIR-FILE INVALID KEY NOFIND  .  Read DIRECT
GO FOUND
```

INDEXED File (MSAM) - Primary key:

```
SET RDA PART-NUMBER = '12-994 AB' .  Set Key in RDA
READ PARTMSTR INVALID KEY NO-FIND .  Read Record
```

INDEXED File (MSAM) - Secondary key:

File defined (with primary key plus two secondary keys) as follows:

```
DEFINE F XMAST INDEXED 120,1 (1,5) (6,8) DUPS (14,3) DUPS
...
    RDA (14,3) = SEARCH-VALUE
    READ XMAST INVALID KEY ERROR-1 ;
        USING KEY 2               .  Key (14,3)
```

10.7 READNEXT

The READNEXT command applies only to INDEXED (MSAM) or ISAM files opened as INPUT or UPDATE with an access mode of SEQUENTIAL or DYNAMIC. READNEXT is used to read the next logical record in the file. This command must follow a READ, START or a READNEXT.

Format:

READNEXT *filename* [AT END {*program-label* | BREAK} | STATUS *status-variable*]

READNEXT may be abbreviated READN.

The STATUS clause may be used in lieu of the AT END clauses to receive the PCIOS/SFS error status from the PCIOS Processor Interface Module (PIM) or SFS 2200. The *status-variable* must be defined as a numeric variable. The AT END and STATUS clauses are optional in conversational mode.

BREAK may only be used when the READNEXT is contained within a defined procedure or an in-line DO block.

See the START command for usage examples.

10.8 REWRITE

The REWRITE command may be used with any file type. The command is used to write the last record read, back to the same file location from which it was read. The file must be opened for update, and the number of words written must be the same as the original record length.

Format:

REWRITE *filename* [*length-literal* | *length-variable*] ;
 [INVALID KEY *program-label* | STATUS *status-variable*]

REWRITE may be abbreviated REW.

The *length-literal* or *length-variable* entry is optional. If omitted, the defined record length for the file will be written.

The INVALID KEY clause only applies to DIRECT, INDEXED (MSAM) and ISAM files. The STATUS clause may be used in lieu of the INVALID KEY clause to receive the PCIOS/SFS error status from the PCIOS Processor Interface Module (PIM) or from SFS 2200. The *length*-variable and *status-variable* must be defined as numeric variables. The INVALID KEY and STATUS clauses are optional in conversational mode.

Example:

```
READ WORK-FILE AT END END-RUN
RDA (1,5) = 'ABCDE'            .  Change the record.
REWRITE WORK-FILE              .  Rewrite it.
```

10.9 START

The START command applies only to INDEXED (MSAM) and ISAM files opened for INPUT or UPDATE with an access mode of DYNAMIC. START is used to position to a point in the file. The user may position to a specific key or a point just following a specified key. The START command does not read a record into the RDA. To read the record into the RDA, use the READNEXT command.

Format:

> START *filename* [INVALID KEY {*program-label* | BREAK} | ;
> STATUS *status-variable*] ;
> [USING KEY *key-number-literal*] [EQ | GT | EQGT]

START may be abbreviated ST.

The INVALID KEY clause only applies to INDEXED (MSAM) and ISAM files. BREAK may only be used instead of a label when the START is contained within a defined procedure or an in-line DO block. The STATUS clause may be used in lieu of the INVALID KEY clause to receive the PCIOS/SFS error status from the PCIOS Processor Interface Module (PIM) or from SFS 2200. The *status-variable* must be defined as a numeric variable. The INVALID KEY and STATUS clauses are optional in conversational mode.

BREAK may only be used instead of a label when the START is contained within a defined procedure or an in-line DO block.

The USING KEY clause is used to specify positioning on a secondary key, and therefore, only applies to INDEXED (MSAM) files. The *key-number-literal* corresponds to the position of the secondary key in the DEFINE for the file.

EQ, GT or EQGT are used to determine where to position in the file in relation to the key supplied in the RDA. If not present, EQ is assumed. EQ indicates that a record with a matching key must be found; otherwise, an invalid key condition will result. GT indicates that positioning to the next key following the given record key is to be used. If no higher key exists, an invalid key condition results. EQGT indicates to position to an equal key or to the next greater key in the file. If no equal condition or higher key exists, an invalid key error results.

Example:

The following directive defines an MSAM file with one secondary key:

```
DEFINE F MSTR-FILE INDEXED 100 1 MSTR-NUM MSTR-SCNDRY-KEY
...
    .  *** Position based on a secondary key.
    .  *** Position to record greater than the letter 'A'.
    RDA MSTR-SCNDRY-KEY = 'A'
    START MSTR-FILE INVALID KEY ERR2 USING KEY 1 GT
    .  *** Read the record.
    READNEXT MSTR-FILE AT END END-RUN
...
    .  Position based on primary key.
    RDA MSTR-NUM = 100000
    START MSTR-FILE INVALID KEY ERR1 EQ
    READNEXT MSTR-FILE AT END CLOSE-OUT  .  Read the record.
```

10.10 WRITE

The WRITE command is used to write a specified number of words or characters (depending on file type) from the RDA to a file. The file may be any standard format.

For standard PCIOS files, the length of the record is specified in characters. If the number of characters to write is omitted, I-QU PLUS-1 will assume the maximum record length specified in the file definition.

Format:

> WRITE *filename* [*length-literal* | *length-variable*] ;
>   [INVALID KEY *program-label* | STATUS *status-variable*]

WRITE may be abbreviated W.

The *length-literal* or *length-variable* entry is optional. If omitted, the defined record length for the file (as specified on the DEFINE F directive) will be written.

The INVALID KEY clause only applies to DIRECT, INDEXED (MSAM) and ISAM files. The STATUS clause may be used in lieu of the INVALID KEY clause to receive the PCIOS/SFS error status from the PCIOS Processor Interface Module (PIM) or SFS 2200. The *length-variable* and *status-variable* must be defined as numeric variables. The INVALID KEY and STATUS clauses are optional in conversational mode.

Examples:

```
DIR-KEY = DIR-KEY + 1              .  Set to write next record.
WRITE DIR-FILE INVALID KEY ERR-1   .  Write the record.

.  *** Write a variable length INDEXED record...
R-LEN = SUB1 * 45                  .  Calc length of occurs items.
R-LEN = R-LEN + 32                 .  Add on fixed part of rec.
WRITE PAYMAST R-LEN INVALID KEY KEY-ERROR
```

10.11 Special PCIOS Status Returned

The following status codes can be returned from I-QU PLUS-1:

Status	**Meaning**
000000	Normal status returned.
900000	User has reached end-of-file.
900001	File is not assigned. Assign the file with a CSF command.
900002	Not enough memory to allocate file buffers. Use a smaller block size or open fewer files at one time.
900003	Invalid key returned on READ, START, REWRITE or WRITE command.
900004	User specified a zero (0) block size and block size information not found in file header. The file is probably not a PCIOS file.

Chapter 11: SORT Interface

This group of commands gives the user the ability to SORT records within an I-QU PLUS-1 program. Records are passed to the SORT and returned from the SORT using the RDA. SORT commands may not be used in conversational mode. The SORT interface routine will require SORT work files to be assigned (XA, XB, through XZ are possible). The size of these files, and the number of SORT work files, must be determined by the user depending on the number and size of records to be SORTed and the core SORT buffer size. (Refer to the SORT manual). I-QU PLUS-1 will dynamically assign the XA, XB and XC files as follows:

```
@ASG,T XA.,///5000
@ASG,T XB.,///5000
@ASG,T XC.,///5000
```

11.1 RELEASE

This command RELEASEs a specified number of characters from the RDA to the SORT. The number of characters RELEASEd may be specified as an integer value or as a database record name. If a record name and an integer value are specified, the length of the named DMS 2200 record plus the integer value will be RELEASEd. The RELEASE may be given as RELEASEF, in which case the *integer-characters* parameter will be assumed to be FIELDATA.

Format:

RELEASE[F] {*record-name* [*integer-characters*] | *integer-characters*}

RELEASE may be abbreviated REL.

Examples:

```
.  *** Release using the length of the PRODUCT-RECORD
.  *** plus 20 additional characters.
RELEASE PRODUCT-RECORD 20
...

.  *** Release a record of 35 FIELDATA characters in
.  *** length.
RELEASEF 35
```

11.2 RETURN

This command is used to RETURN records from the SORT subroutine to the RDA. The first execution of the RETURN will cause the termination of the SORT release phase. The record will be placed in the RDA exactly as released. The AT END clause is required.

Format:

RETURN AT END {*program-label* | BREAK}

RETURN may be abbreviated RET, with B for BREAK.

BREAK may only be used instead of a label when the RETURN is contained within a defined procedure or an in-line DO block.

Example:

```
SORT-OUT PROCEDURE
    RETURN AT END BREAK        .  Return a record
    DISPLAY RDA PART-NAME +    .  list data from
    DISPLAY '-' +              .  the record area
    DISPLAY RDA PART-NUM
    ...
    ENDPROC
```

11.3 SORT

The SORT command is used to initialize the SORT interface. It is used to specify the length of records to be SORTed, and to define the SORT KEYS. This command must be used each time a new SORT is to be done. A maximum of ten SORT KEYS may be specified. The SORT command may be given as either SORT or SORTF. If SORT is used, the maximum/minimum record lengths are assumed ASCII characters, while SORTF indicates FIELDATA characters. The representation code specified affects the calculation used by I-QU PLUS-1 in determining the record size in words.

Format:

> SORT[F] *maximum-record-length minimum-record-length* ;
> *key-definition-1* [… *key-definition-10*] [(*core-buffer-size*)]

SORT may be abbreviated SOR.

The *minimum-record-length* field indicates the shortest record being SORTed, when SORTing variable length records. If records are fixed length, *maximum-record-length* and *minimum-record-length* must be equal. All SORT KEYS must fall within the shortest record length.

The format of *key-definition-n* is:

> *start-character*,*key-byte-length*,*data-type*,*sequence-specifier*

Allowed *data-types* are the same as those used in direct RDA reference. *Start-char* is relative to the first character of the SORT area. The SORT area may be defined by the DEFINE RA SORT directive. If the DEFINE RA SORT directive is not used, the SORT area begins in position one (1) of the RDA. See Section 3.1.1 for all possible data types.

Allowed *sequence-specifiers* are:

> A for ascending,
>
> D for descending.

The *core-buffer-size* parameter is optional. It is used to change the default SORT buffer from 22.5K to the size required, depending on the SORT volume. Currently the maximum sort buffer size allowed is 40,000. See the Unisys SORT manual for SORT file and buffer sizing.

Examples:

The following initializes the SORT interface to SORT 80-character records on a 9(10) COMP field starting in position 1, and an X(20) field starting in position 5.

```
SORT 80 80 1,4,UB9,A 5,20,DISP,D
```

This command sets up a SORT of variable length FIELDATA records on positions 2 through 7 alphanumeric ascending. A 40,000-word sort buffer will be allocated.

```
SORTF 100 20 2,6,DISP-1,A (40000)
```

Chapter 12: DMS 2200 Interface

The following commands are the I-QU PLUS-1 equivalent of standard ASCII COBOL/DML statements. I-QU PLUS-1 rules for record selection and database currency are the same as in standard DML. For a complete explanation of the function of DML commands, refer to the DMS 2200 ASCII COBOL/DML Manual.

12.1 Subschema Invocation (INVOKE)

As with any COBOL/DML program, before database access can begin, a DMS 2200 subschema must be invoked. Unlike COBOL/DML, I-QU PLUS-1 is able to invoke a subschema dynamically at runtime by using the INVOKE directive. The INVOKE directive must be issued prior to the DML IMPART command. INVOKE will access the object schema and subschema to initialize I-QU PLUS-1's S$WORK and D$WORK. Upon INVOKE, the I-QU PLUS-1 Primary Data Item Index file will be assigned, initialized, and built automatically to be used with DML command editing and decoding, and in RDA item name reference decoding. The INVOKE directive may be used to switch from one subschema to another so long as the current subschema is not IMPARTed.

Format:

> INVOKE *subschema* [{OF | IN} *schema*] ;
>   [{FILE *qualifier*filename* | TIP [FILE] *file-code*} [FOR *DMR-name*] ;
>   [|| APPLICATION *application-number* | ;
>   KEY IS *INVOKE-key* | ;
>   POINTER AREA LOAD | ;
>   {NX | INDEX} ||]

INVOKE may be abbreviated INV, with APPL for APPLICATION.

The *subschema* specification may reference an ACOB-generated or UCOB-generated subschema.

The OF/IN *schema* parameter is optional only when a default schema is configured for a particular DMR or LDM (see the note, below).

The FILE/TIP clause is optional only when a default schema file is configured for a particular DMR.

Please note: With DMS 2200, level 9R1 or higher, the term, "DMR", is no longer used. Instead, Unisys documentation now refers to a logical data manager (LDM) for DMS running under Universal Data System (UDS) Control. I-QU PLUS-1 can interface with any number of LDMs through UDS Control. This documentation, however, will continue to use the term, "DMR", in a generic fashion since I-QU PLUS-1 can interface with non-UDS DMS applications (e.g., 8R3) as well as DMS applications running under UDS Control.

The FOR *DMR-name* parameter is also optional. The *DMR-name* refers to the multi-thread invoke name specified for a particular "DMR" during dynamic configuration of I-QU PLUS-1 through COMUS. If it is not specified on the INVOKE directive, the first multi-thread DMR configured will be used. The security feature allows the configuration of additional DMR default values.

The optional APPLICATION *application-number* parameter allows I-QU PLUS-1 to explicitly connect to an application group upon INVOKE. This parameter may be used when invoking from a TIP schema file.

The EXEC will not allow a program to connect to two application groups within the same program execution. You must exit I-QU PLUS-1 after utilizing an application group before attempting to access a different application group. If you fail to exit before attempting access to a different application group, DMS will return a rollback error 94 and display a console message indicating that a STEP CONTROL error 067 was encountered.

The *INVOKE-key* literal, on the KEY IS parameter, is the LOCK FOR INVOKE literal specified in the subschema. It is only required when coded in the subschema.

The POINTER AREA LOAD parameter is only required when initially loading pointer array areas.

The optional NX/INDEX parameter controls the building of the primary data item index file on INVOKE. If the NX option is used, the I-QU PLUS-1 primary data index file will be built without data items. The NX option may be used if a permanent data item index file (previously built by the QINDEX processor) is to be used instead of the dynamically built primary index. The INDEX option will cause the primary data item index file to be built with data items. If this parameter is not specified, the default, as dynamically configured through COMUS, will be used.

For information on creating a permanent data item index file, see Chapter 18, "QINDEX Reference".

Note: If alternate records delivery areas are to be defined (DEFINE RA) for DMS records, they should be specified after INVOKE but prior to IMPART.

Examples:

```
1. INVOKE ACCTS OF FINANCE TIP FILE 77 NX
   INDEX DMS*DB-INDEX

2. INVOKE ACCTS OF FINANCE FILE TST*SCHFILE FOR STHREAD

3. INVOKE TRIV OF FINANCE FILE DMS*SCHFILE KEY IS 'FTRV'
```

Explanations:

In the first example, a TIP object schema file will be used. In addition, no data item entries will be created in the primary data item index (NX option). A secondary data item index will be required to reference data items by name in this case (INDEX directive). In addition, the DMR name is not given, implying that the INVOKE will default to the first multi-thread DMR configured.

In the second example, the object subschema and schema will be obtained from the EXEC schema file, "TST*SCHFILE", and is being used with the "STHREAD" DMR.

In the last example, the "TRIV" subschema requires an INVOKE key, "FTRV", for processing.

12.2 INVOKE Considerations

The INVOKE accesses the object schema and subschema specified by the user. Using the object schema and subschema tables, the INVOKE performs two major tasks: initialization of D$WORK and S$WORK, and creation of the Primary Data Item Index File. In a COBOL program, D$WORK is created by the ASCII Data Manipulation Language Preprocessor (ADMLP), while S$WORK is created when the Subschema Data Definition Language (SDDL) is compiled. Both D$WORK and S$WORK are collected into the object program by the MAP processor. I-QU PLUS-1 creates both of these dynamically when the INVOKE directive is processed.

The Primary Data Item Index File is used by I-QU PLUS-1 only during the editing of commands. It contains definitions of areas, records, sets, database data names and data items from the invoked subschema. The data item index file allows the processor to translate names entered by the user to codes or RDA references to be used during actual execution of commands.

The D$WORK and S$WORK areas are in all DMS 2200 programs, including I-QU PLUS-1. They are used in communications between the program and the data management routine (DMR). D$WORK contains lists of WORKING and/or COMMON STORAGE addresses of each record and database data name in the user program. The

DMR uses these addresses to find records and database data names in your program. S$WORK is where the DMR maintains information on run unit, area, record and set currency. S$WORK is also used as workspace for many DMR operations.

S$WORK also contains the name of the schema and subschema, and the file from which the DMR must obtain its copy of the schema and subschema absolutes. This fact is a very important and often misunderstood point. When I-QU PLUS-1 processes the INVOKE, the subschema file name is copied to S$WORK directly from the subschema absolute, not from the file named on the INVOKE directive. Upon IMPARTing, the DMR uses the file name found in S$WORK to find the subschema. The DMR then uses the schema file named in the subschema tables to find the object schema, which is why I-QU PLUS-1 cannot simply move the file name on the INVOKE directive to S$WORK. The DMR would still look for the schema in the schema file named in the subschema tables. It is very important to insure that the subschema and schema INVOKEd are the same as what the DMR will use. In a production application environment, it is best to INVOKE from the production on-line schema file. On the other hand, when using I-QU PLUS-1 for database reorganization, it is desirable to use an alternate schema file. For reorganization purposes only, the SCHUTL processor has been furnished to allow the subschema and schema object to be modified to force the DMR to obtain the object from a specified alternate file (see the “I-QU PLUS-1 Database Reorganization Utility Reference”).

12.3 DMS 2200 CALC Routine Definition (DEFINE C)

CALC routine definition is only required when the CALSIM command is to be used. The purpose of this definition directive is to describe all parameters needed by I-QU PLUS-1 to call DMSCALC or RANDENTIAL (independent of DMS 2200) to produce a resultant page number when given a record key. By redefining the CALC key and using CALSIM, record distribution may be tested in order to determine optimum, record placement strategies. CALSIM is also often used to determine the page to which a CALC record will be placed on initial load, and thus can facilitate a sort of the records by page number before initial load.

Format:

> DEFINE C *CALC-id* {D | R | U1 | U2 | U3} [ASCII | FDATA] ;
> *allocated-pages*,*number-of-chains* ;
> [*upper-range*,*lower-range*,*ovfl-every-pages*,*spcd-ovfl-pages*,*at-end-ovfl-pages*] ;
> (*key-def-1* [… *key-def-n*])

DEFINE may be abbreviated DEF.

The *CALC-id* may be any name desired by the user.

"D" indicates that the Unisys supplied calc routine, DMSCALC, is to be called; "R", that RANDENTIAL is to be called. "U1", "U2" and "U3" apply to user installed CALC routines (see the I-QU PLUS-1 Installation Guide).

ASCII or FDATA indicates that all key definitions will be either ASCII or FIELDATA characters. If omitted, the assumed mode is ASCII.

The *allocated-pages* parameter specifies the total number of pages allocated to the area.

The *number-of-chains* parameter specifies the number of CALC chains to be used.

The *upper-range* and *lower-range* parameters specify the page range bounds in which the record may be stored.

The *ovfl-every-pages* parameter specifies the interval at which interspersed overflow pages are allocated.

The *spcd-ovfl-pages* parameter specifies the number of overflow pages in each interval.

The at-*end-ovfl-pages* parameter specifies the number of overflow pages allocated at the end of the area.

See the DMS 2200 Schema Definition Manual for a more complete explanation of these parameters.

The format of *key-def-n* is:

> *cccswwww*

The *key-def* must be exactly eight (8) digits. The first three (*ccc*) digits specify the total number of characters in the key field. The fourth digit (*s*) specifies the position within the first word of the key field in which the first character of the key will be located. This will be a value of from 1 to 4 for ASCII keys, or 1 to 6 for FIELDATA keys. The last four digits (*wwww*) specify the starting word of the key within the record, where the first word is 0000. The starting word is relative to the beginning of the RDA. For example, if a six-character key, starting in character position 3 of the record were being defined, the *key-def* would look like this: 00630000.

The definition allows a group level CALC key containing up to five elementary key items. A group item is acceptable for RANDENTIAL, but is not normally allowed by DMSCALC. If using the standard DMSCALC, the first *key-def* will be used. When using RANDENTIAL, up to 10 *key-defs* may be specified, but the total number of significant characters in all the keys put together must be 13 or less if any of the keys contain alphabetic characters, or 17 or less if all the keys are pure numeric.

If any of the parameters from *upper-range* through *at-end-ovfl-pages* are present, they must all be present. Unused parameters within this group must be entered as zero (0).

Examples:

RANDENTIAL

```
DEFINE C REC-25 R 1200,1 (00630000,00210002)
```

In this example, a 1,200-page area with one CALC chain will be used by RANDENTIAL when the CALSIM command is executed. The record has a group level key made up of two fields. The first field is six characters beginning in character position three. The second field is two characters beginning in character position nine. The data looks like this:

```
                          1111111
Character position -----1234567890123456
Word position ----------0000111122223333
Data elements ----------..xxxxxxyy......
```

DMSCALC

```
DEFINE C REC-25 D 2000,2,0,0,0,0,0 (00810001)
```

In this example, a 2,000-page area with two CALC chains will be used by DMSCALC when the CALSIM command is executed for this definition. The record has a single elementary level key of eight (8) characters beginning in position five (5). The data looks like this:

```
                          1111111
Character position -----1234567890123456
Word position ----------0000111122223333
Data elements ----------....kkkkkkkk....
```

12.4 DBDUMP File

I-QU PLUS-1 provides a file format that may be used for database record unloading and reloading.

12.4.1 DBDUMP File Definition (DEFINE F)

The DBDUMP file format is fixed, and is called DBDUMP. The DBDUMP format may also be read or written by COBOL programs. The COBOL record definition of the DBDUMP file may be found in Chapter 17.

Format DBDUMP:

DEFINE F[F] *filename* DBDUMP [*length*,*block-size* {RECORDS | CHARACTERS}]

DEFINE may be abbreviated DEF, with RECS or RECORD for RECORDS and CHARS for CHARACTERS.

DBDUMP files may be externally assigned to either tape or disk. If assigned to tape, they will be in SDF format.

The *length* parameter specifies the number of characters per record. The *block-size* can be specified as the number of records or characters per block. If the optional RECORDS/CHARACTERS keyword is not specified, I-QU PLUS-1 will default to records per block. If *block-size* is zero (0), I-QU PLUS-1 will determine the block size by reading the physical record header. The zero specification is only valid when opening DBDUMP files for INPUT. If the *length*,*block-size* parameters are omitted, the default record size is the same as the RDA size with a blocking factor of two records per block. With a default RDA size of 4,000 words, the first DBDUMP file opened will require 12,000 words of memory (8k for buffers and 4k for the record transfer area) and each subsequent DBDUMP file opened will require 8,000 additional words of memory (the record transfer area is shared).

The required size of the record area (see Chapter 17 "DBDUMP File Description") is the sum of the largest data record to be written plus the number of control words to be written plus 12 words for the record header. In other words, if the largest DMS record is 100 words and 10 words are going to be written for control words, the required record size is 122 words or 488 ASCII characters (the record size is specified in characters as it is for all DEF F directives). Specifying the record size and a blocking factor of two records per block would require only 366 words of storage for the first DBDUMP format file opened versus the 12,000 words required in the default case.

Additionally, the DEFINE FF format may be used if you wish to specify your record size in FIELDATA characters rather than ASCII characters. Note that the data format is NOT changed — only the conversion from characters to words for the record size is affected. Thus the required space in the record for the overhead of the header on the DBDUMP records is 12*6 (72) FIELDATA characters or 12*4 (48) ASCII characters.

12.4.2 CLOSE (DBDUMP)

The CLOSE command is used to close a DBDUMP file. If the file is opened for OUTPUT, end-of-file processing will take place. Disk files will not automatically be @FREEed; tape files will automatically rewind. Files may be closed and re-opened with a different usage mode.

Format:

CLOSE *filenam*e

CLOSE may be abbreviated CL.

12.4.3 OPEN (DBDUMP)

The OPEN command is used to ready a DBDUMP file for use in a program. It establishes the usage and access modes for the file. The file must be assigned to the run before the OPEN is executed.

Format:

OPEN *filename* {INPUT | OUTPUT} SEQ

OPEN may be abbreviated O, with I for INPUT and O for OUTPUT.

INPUT and OUTPUT are usage modes and have the following meanings:

INPUT	The file will only be read.
OUTPUT	The file will only be written. Any existing data will be destroyed.

DBDUMP files may only be accessed sequentially.

Note: If more than one DBDUMP needs to be OPEN concurrently, the file with the largest number of characters per record (see the DEF F directive) should be opened first. This requirement is due to an internal addressing constraint.

12.4.4 READ (DBDUMP)

This form of the READ command is used to read an I-QU PLUS-1 formatted DBDUMP file. The file must have been previously opened (OPEN filename INPUT SEQ). The AT END label must be specified when not in conversational mode.

If the CONTROL clause was specified on the WRITE, the control words will be put into the RDA in the same position from which they were written. After each read of the DBDUMP file, the reserved variable C-O-T will be set to the record name contained in the DBDUMP record header. The C-O-T variable may then be tested to determine the record type just read.

Format:

READ *filename* [AT END { *program-label* | BREAK}]

The AT END clause must be present when not in conversational mode.

BREAK may only be used instead of a label when the READ is contained within a defined procedure or an in-line DO block.

Following the read of a DBDUMP file, the reserved numeric variable REC$LEN will contain the actual length of the record read in words. This length may be useful when processing variable length data records.

Example:

```
LOOP
    READ DBFILE AT END END-RUN .  Read DBDUMP file
    IF C-O-T = 'PRODUCT-RECORD' .  PRODUCT record?
        STORE PRODUCT-RECORD .  Yes, store record
    ENDIF
```

12.4.5 WRITE (DBDUMP)

The WRITE command is used to write a specified number of words from the RDA to an I-QU PLUS-1 formatted DBDUMP file. The file must have been previously opened (OPEN filename OUTPUT SEQ).

The user may specify a string of control words within the RDA to be inserted in front of the data being written. This option may be used to move sort keys to a common position in the dump file record when multiple record types are being written. Each time a DBDUMP file record is written, the current value of the reserved variable C-O-R will be placed in the record's header. This value will be available in the reserved variable C-O-T when the file is read back into I-QU PLUS-1 as a DBDUMP file. The number of data words to be written to a DBDUMP file may be specified as an integer, or by naming an invoked database record. If a record name is used with an integer value, the length of the record will be the length of the database record plus the integer value. This allows the user to append additional information to the database record being written.

Format:

> WRITE *filename* {*record-name* [*integer-words*] | *integer-words*} ;
> [CONTROL (*start-word*,*number-of-words*)]

WRITE may be abbreviated W.

The CONTROL clause is only valid if the file is defined as a DBDUMP file.

The *record-name* parameter is used to indicate that the length of the write is to be taken from the named DMS 2200 database record, not the area from which the data is to be written. If *record-name* is used with *integer-words*, the two lengths will be added together.

Example:

```
DEFINE F DUMPTAPE DBDUMP
...
WRITE DUMPTAPE REQDTL 5 CONTROL (251,9)
```

In the above example, the record written will be the length of the database record REQDTL plus five words. Nine (9) words of control information is being retrieved from word 251 (RDA ASCII character position 1001) and placed in the record prior to the data. When this record is read back in by I-QU PLUS-1, this control information will be placed back in the same place: word 251 for 9 words.

12.5 DML Commands

Most commands available in standard COBOL/DML are also available in I-QU PLUS-1. The ON ERROR and AT END clauses are not included in I-QU PLUS-1 DML syntax. The I-QU PLUS-1 procedural IF command is used to determine these conditions.

Unless otherwise stated, area-name, record-name and set-name may be given as a specific area, record or set name, or as alphanumeric variables, which contain a valid area, record or set name (known as generalized names in DMS 2200 DML terminology). When an alphabetic variable is used in place of an actual name, the variable length must be specified as follows:

area-name	12 characters
record-name	30 characters
set-name	30 characters

I-QU PLUS-1 may be dynamically configured in COMUS to allow abbreviation of AREA, RECORD and SET names. If this option is taken, the user may enter either the first “N” characters of the name, or the entire name, in all cases except where the name is used as a literal enclosed in quotes. “N” characters will vary depending on the number of characters required to identify the AREA, RECORD, or SET. For example, if all record names begin with the letter “R” and the three-digit record code (example: R0002-PARTS-MASTER-CONTROL), I-QU PLUS-1 may be configured to recognize the record name when only the first five characters have been entered (example: R0002).

12.6 ACQUIRE

The ACQUIRE command is used to retrieve a list of database keys for a specified set whose mode is pointer array or indexed pointer array. The ACQUIRE command functions in the same manner as the standard COBOL/DML ACQUIRE command. The only difference is the I-QU PLUS-1 acquire list is specified as a location within the RDA rather than a COBOL WORKING-STORAGE location. The acquire list in the RDA is in the same format as in a COBOL/DML program. The first word of the specified RDA location contains the number of DBKs retrieved followed by the list of DBKs.

Format:

ACQUIRE *number-of-keys* [BEGINNING *database-key*] ;
TO RDA *RDA-reference* FROM *set-name* [USING DEFINED KEYS]

ACQUIRE may be abbreviated ACQ.

The *RDA reference* must be a multiple of four (4) because each database key will occupy one word. Each DBK may be referenced using the RDA reference "(n,4) UB9" (n = start position).

Example:

The following is an example of the ACQUIRE command used to obtain two lists of database keys which will then be matched for retrieval of records that participate in both pointer array sets.

```
DEF N DBKVAR                      .  Number of DBKs
DEF RDA DBKLIST1 (1001,400)       .  A 100-word ACQUIRE list
DEF RDA DBK1 (1001,4) UB9         .  A DBK in ACQUIRE list 1
DEF RDA DBKLIST2 (1501,400)       .  Another ACQUIRE list
DEF RDA DBK2 (1501,4) UB9         .  A DBK in ACQUIRE list 2
DEF SUB X1 DBK1                   .  Subscript for list 1
DEF SUB X2 DBK2                   .  Subscript for list 2
    ...
    .  *** Owner of sets S0002-PART-USE and S0003-PART-MOD
    .      have been made made current of set.  Now
    .      ACQUIRE lists of DBKs for both sets.
    RDA DBKLIST1 = $LOVALS        .  Clear list of binary zeros
    RDA DBKLIST2 = $LOVALS
    DBKVAR = 50                   .  Set to retrieve 50 keys
    ACQUIRE DBKVAR TO RDA DBKLIST1 FROM S0002-PART-USE
    ACQUIRE DBKVAR TO RDA DBKLIST2 FROM S0003-PART-MOD
    DO DBK-MATCH
...
    .  ***  This proc matches DBKs from both ACQUIRE lists.
    .       When a DBK is found in both lists, the
    .       FETCH-IT procedure is executed.
DBK-MATCH PROCEDURE
    IF DBK1 = 0                   .  If either list is empty
    OR DBK2 = 0                   .  or no rec meets criteria
        GO DM-EXIT                .  EXIT this routine
    ENDIF
    X1 = 1                        .  1stDBKinlist1
DM005
    X2 = 1                        .  1st DBK in list 2
    Z = RDA DBK1 :X1              .  Isolate 1 DBK from list 1
```

```
DM010
    IF RDA DBK2 :X2 = Z           .  Is it in list 2?
        DO FETCH-IT               .  In both lists, FETCH it.
    ENDIF
    X2 = X2 + 1                   .  Increment list 2 subscript
    IF RDA DBK2 :X2 = $LOVALS     .  End-of-list-2?
        X1 = X1 + 1               .  Increment list 1 subscript
        IF RDA DBK1 :X1 = $LOVALS .  End-of-list-1?
            GO DMEXIT             .  Yes, done......
        ENDIF
        GO DM005                  .  Repeat match
    ENDIF
    GO DM010                      .  Match next entry
DMEXIT
    ENDPROC
    ...
FETCH-IT PROCEDURE
    FETCH1 Z                      .  FETCH member using DBK
    DISPLAY RDA ......
    ...
ENDPROC
```

12.7 CALSIM

The CALSIM command is used to interface with the DMS 2200 CALC routines: DMSCALC or RANDENTIAL. CALSIM is used in conjunction with the DEFINE C (CALC key definition) directive to obtain a resultant page number. The page number obtained may be used to determine record distribution throughout an area. The CALC routine will use the data currently in the RDA relative to position one and the parameters in the specified CALC definition. The resultant page number will be set into the named numeric variable. The resultant chain number will be placed in the predefined variable REC$LEN.

Format:

CALSIM *CALC-id result-variable* [*area-name-variable*]

CALSIM may be abbreviated CA.

The *CALC-id* is the name used on a previously defined, DEFINE C directive.

The optional *area-name-variable* specification only applies when locally installed user CALC routines are being simulated (requires local installation). It is used to receive the resulting area name output by some user CALC routines.

Example:

```
DEFINE C CALC-X D 12000 1 (01210000)   .  CALC key def.
DEFINE N PAGE-RESULT                   .  Resulting pg no.
    ...
    .  CALC on current RDA data and place result in
    .  PAGE-RESULT.

CALSIM CALC-X PAGE-RESULT
```

12.8 CLOSE

The CLOSE command releases database areas and associated quick-look files. The command may be used to close a single area or all areas currently open.

Format:

```
CLOSE {ALL | area-name}
```

CLOSE may be abbreviated CL.

12.9 DELETE

The DELETE command is used to delete a record from the database. If ONLY is specified, the record will not be deleted if it is the owner of any VIA-SET records. If ALL is specified, the record and all members of any sets of which it is the owner will be deleted. The ONLY and ALL options have the same meaning as in standard COBOL/DML.

Format:

DELETE *record-name* [ALL | ONLY]

DELETE may be abbreviated DEL.

Examples:

```
FETCH4 FIRST PRODUCT-REC PRODUCTS AREA
DELETE PRODUCT-REC ONLY       .  Delete, if no members
...
FETCH3 NEXT PRODUCT-REC PRODUCTS AREA
DELETE PRODUCT-REC ALL        .  Delete it & all members.
```

12.10 DEPART

The DEPART is used to disconnect I-QU PLUS-1 from the DMR. The ROLLBACK option may be used to negate any updates that were done during the session.

Format:

```
DEPART [ROLLBACK]
```

DEPART may be abbreviated DEP.

12.11 DISPLAY Database Error

This form of DISPLAY will print a fixed format DML error status. When executed, it will not disturb the current contents of the print buffer.

Format:

```
        DISPLAY DBERROR
```

DISPLAY may be abbreviated D, with ERROR for DBERROR.

12.12 FETCH/FIND

I-QU PLUS-1 supports all formats of the FETCH/FIND command. In the following descriptions, the word FETCH will be used in all formats. The same formats will apply to the FIND command.

12.12.1 FETCH/FIND Format-1

The FETCH Format 1 is used to obtain any record directly instead of by its specific location mode. A Format 1 FETCH can be executed in one of two ways:

1. A numeric variable containing a database key can be set;
2. An alpha variable containing an area-name and a numeric variable containing an area-key can be set.

The record-name is optional on the Format 1 FETCH. If used, it must be given as an actual record name, not a variable.

Format:

{FETCH1 | FIND1} [*record-name*] ;
{*database-key-variable* | *area-name-variable*,*area-key-variable*}
[*SUPPRESS-clause*]

FETCH1 and FIND1 may be abbreviated F1 and FN1, respectively.

The format of the suppress clause can be found in Section 12.25 at the end of this chapter.

Examples:

```
DEFINE N HOLDDBK               .  To hold database key.
    HOLDDBK = RDA DBK-OF-LINE-ITEM .  Get DBK
    FETCH1 HOLDDBK             .  Fetch the line item.
    ...
    G-AREA-NAME = 'PRODUCTS'   .  Set area-name.
    G-AKEY = (1,1)             .  Set area-key to page 1
                               .    and record 1.
FIND1 PRODUCT G-AREA-NAME G-AKEY
```

12.12.2 FETCH/FIND Format-2

The FETCH Format 2 is used to make a record current of run unit. The record must be current of that record type (i.e., previously accessed).

Format:

```
{FETCH2 | FIND2} [record-name] ;
   {database-key-area | area-name-variable,area-key-variable} ;
      [suppress-clause]
```

FETCH2 and FIND2 may be abbreviated F2 and FN2, respectively.

The format of the suppress clause can be found in Section 12.25 at the end of this chapter.

12.12.3 FETCH/FIND Format-3

The FETCH Format 3 is used to retrieve unspecified record types within sets or areas using set or area currency.

Format:

{FETCH3 | FIND3} {CURRENT | NEXT | PRIOR | FIRST | LAST | OWNER} ;
  {*set*-name SET | *area-name* AREA} ;
    [*SUPPRESS-clause*]

FETCH3 and FIND3 may be abbreviated F3 and FN3, respectively. CURRENT may be abbreviated CURR, with N for NEXT, P for PRIOR, F for FIRST, L for LAST, O for OWNER, S for SET and A for AREA.

The format of the suppress clause can be found in Section 12.25 at the end of this chapter.

Examples:

```
FETCH3 LAST PRODUCTS AREA

FIND3 CURRENT PROD-USAGE SET
```

12.12.4 FETCH/FIND Format-4

The FETCH Format 4 is similar to the FETCH Format 3, except that a specific record type is to be selected.

Format:

{FETCH4 | FIND4} {NEXT | PRIOR | FIRST | LAST} *record-name* ;
  {*set-name* SET | *area-name* AREA} ;
    [*SUPPRESS-clause*]

FETCH4 and FIND4 may be abbreviated F4 and FN4, respectively. NEXT may be abbreviated N, with P for PRIOR, F for FIRST, L for LAST, S for SET and A for AREA.

The format of the suppress clause can be found in Section 12.25 at the end of this chapter.

Examples:

```
FETCH4 FIRST PRODUCT-REC PRODUCTS AREA

FIND4 LAST ASSEMBLY-REC PARTS-USAGE SET
```

12.12.5 FETCH/FIND Format-5

The FETCH Format 5 is used to retrieve a record using its primary location mode. Before a Format 5 FETCH is executed, it may be necessary to set database data names specified in the schema using the SET DBDN command. If a record key is to be initialized, it may be set into the RDA using the SET RDA command.

Format:

> {FETCH5 | FIND5} [NEXT [DUPLICATE] | PRIOR | FIRST | LAST] *record-name* ;
> [*SUPPRESS-clause*]

FETCH5 and FIND5 may be abbreviated F5 and FN5, respectively. NEXT may be abbreviated N, with DUP for DUPLICATE, P for PRIOR, F for FIRST and L for LAST.

The format of the suppress clause can be found in Section 12.25 at the end of this chapter.

Examples:

```
DBDN PROD-AREA = 'PRODUCTS'
RDA PRODUCT-ID = 101100            .  Set record’s key.
FETCH5 PRODUCT-REC                 .  Fetch the record.
FIND5 NEXT DUPLICATE PRODUCT-REC   .  Find dup., if any.
...
DBDN DIR-AREA = 'DIRECT-AREA'      .  Set area name
DBDN DIR-KEY = 5,1                 .  Set area key
FETCH5 DIRECT-REC                  .  Fetch direct rec.
```

12.12.6 FETCH/FIND Format-6

The FETCH Format 6 is used to retrieve a record via a specific set. If the optional USING clause is included, the record will be selected based on the values of the data items specified.

Format:

{FETCH6 | FIND6} *record-name set-name* ;
  [USING *data-item-1* [… ,*data-item-n*]] ;
    [*SUPPRESS-clause*]

FETCH6 and FIND6 may be abbreviated F6 and FN6, respectively.

Where *data-items* may be one or more data field names defined within the object record. A *data-item* must be defined in the schema and may not be qualified.

The format of the suppress clause can be found in Section 12.25 at the end of this chapter.

Example:

```
RDA PROD-TYPE = 'T'
RDA PROD-CAT = '2A'
FETCH6 PROD-DET PROD-DET-SET USING PROD-TYPE PROD-CAT
```

12.12.7 FETCH/FIND Format-7

The FETCH Format 7 is generally used following positioning within a set to find a record with matching values on the specified data items. The *record-name* is optional. If used, it must be current of run unit and cannot be given as a variable.

Format:

{FETCH7 | FIND7} [*record-name*] *set-name* USING *data-item-1* [… ,*data-item-n*] ;
[*SUPPRESS-clause*]

FETCH7 and FIND7 may be abbreviated F7 and FN7, respectively.

Where *data-items* may be one or more data field names defined within the object record. A *data-item* must be defined in the schema and may not be qualified.

The format of the suppress clause can be found Section 12.25 at the end of this chapter.

Example:

```
FETCH7 PROD-DET PROD-DET-SET USING PROD-TYPE PROD-CAT
```

12.13 FREE

The FREE command is used to release all locks caused by the KEEP command, or any updates that have taken place. The FREE also releases all QUICK-LOOKS, making all updates permanent.

Format:

FREE

FREE may be abbreviated FR.

12.14 IF (DML)

The DML IF command is used to determine whether a record participates in a set, or whether a set is empty. When in input mode, the IF command must be accompanied by an ENDIF command. For a complete description of IF/ENDIF pairs, refer to the IF description in the “General Procedure Commands” chapter of this manual.

In conversational mode, I-QU PLUS-1 will display the resulting ‘TRUE’ or ‘FALSE’ condition upon execution.

Format:

IF { [NOT] {OWNER | MEMBER} *set-name* [SET] | ;
set-name [NOT] EMPTY}

SET may be abbreviated S.

Restrictions: The DML IF cannot be extended by AND and OR operators as with the procedural IF; however, the DML IF may be used in conjunction with the ELSE and may be nested within other DML IF or procedural IF commands.

Examples:

```
FETCH4 NEXT PRODUCT-REC PRODUCTS AREA
IF NOT OWNER PRODUCT-ON-ORDER     .  Any members?
...
ENDIF
```

The following is an example of a DML IF nested within a procedural IF:

```
IF ERROR-NUM = 0
    IF PRODUCT-ON-ORDER EMPTY     .  Is the set empty?
        GO END-RUN
    ELSE
        DO PRINT-RECS             .  Set not empty.
    ENDIF
ENDIF
```

12.15 IMPART

The IMPART command is used to connect I-QU PLUS-1 to the DMR. An IMPART cannot be done unless a successful INVOKE has been completed. If a ROLLBACK label is specified, control will be returned at the label upon receipt of a rollback error. The rollback error code will be placed in the reserved variable RBCODE. If no ROLLBACK label is specified, I-QU PLUS-1 will perform a controlled abort upon receipt of a rollback error. Control is always returned to the user in conversational mode.

Format:

IMPART [ROLLBACK *program-label*]

IMPART may be abbreviated IMP.

Note: If alternate records delivery areas are to be defined (DEFINE RA) for DMS records, they should be specified after INVOKE but prior to IMPART.

Example:

```
    IMPART ROLLBACK RB-ERR-PROCESS
    ...
RB-ERR-PROCESS
    DISPLAY DBERROR
    DISPLAY "Processing abandoned"
    STOP EXIT
```

12.16 INSERT

The INSERT command is used to include the current record as a manual member of a set. The record must be defined as a participant in the set, and the owner record must be current of set type. The user may specify ALL sets, in which case all owner records must be current.

Format:

INSERT *record-name* {*set-name* | ALL}

INSERT may be abbreviated INS.

Example:

```
FIND5 ELT-MSTR                .  Find manual owner
FIND2 STEP-DTL                .  Make current
INSERT STEP-DTL ELT-USAGE     .  INSERT into set.
```

12.17 KEEP

The KEEP command is used to extend a record currency lock. Currency locking is done to keep other runs from updating a particular record occurrence while the user is retrieving other records. The object of the KEEP must be current of run unit.

Format:

KEEP *recor*d-*nam*e

KEEP may be abbreviated K.

12.18 MODIFY

The MODIFY command is used to replace the contents of a record in the database with the contents of the RDA. The object record must be current of run unit. The optional data-item-name(s) may be used to change the record's set participation (qualified modify). The owner identified by the data-item-name must be current of the set where participation is to change.

Format:

MODIFY *record-name* [*data-item-name-1* [… *data-item-name-n*]]

MODIFY may be abbreviated M.

The optional *data-item-name(s)* must be defined in the schema and may not be qualified. A *data-item-name* must be a field name in a defined owner record of the record being modified. The *data-item-name* identifies the set in which set participation is to change.

Examples:

```
FETCH5 PRODUCT-REC              .  Fetch the product record.
RDA PRODUCT-DESC = 'XYZ'        .  Change a field
MODIFY PRODUCT-REC              .  Modify the database.
```

12.19 OPEN

The OPEN command is used to specify which database areas are to be accessed and in which usage mode. The default usage mode is retrieval.

Format:

OPEN {*area-name* | ALL} [{RETRIEVAL | UPDATE} [EXCLUSIVE] | LOAD]

OPEN may be abbreviated O, with RET for RETRIEVAL, U for UPDATE, EX for EXCLUSIVE and L for LOAD

The ALL option may not be specified with a usage mode of LOAD.

Examples:

```
OPEN CUSTOMERS UPDATE EXCLUSIVE
OPEN PRODUCTS LOAD
OPEN ALL                      .  Opens all areas for retrieval
```

12.20 REMOVE

The REMOVE command is used to remove a record from a manual set relationship. The record must be current of run unit. The ALL option may be used, if the record is to be removed from all manual sets in which it currently participates.

Format:

REMOVE *record-name* {*set-name* | ALL}

REMOVE may be abbreviated REM.

Examples:

```
FETCH3 NEXT SERIAL SET          .  Get manual member.
REMOVE SERIAL-REC SERIAL        .  Remove record.
```

12.21 SET CURRENT

Format 1 and 2 of the SET CURRENT command is equivalent to a MOVE CURRENCY STATUS command in ASCII COBOL/DML. The third format allows a user change file to be specified when executing in the DMS test mode (O-option on the I-QU PLUS-1 processor call).

Format-1:

[SET] CURRENT {DBK | AKEY | ANAME} ;
[= {RU | RECORD *record-name* | AREA *area-name* | SET *set-name*}]

Format-2:

[SET] CURRENT {AKEY | ANAME} = *database-key-variable*

Format-3:

[SET] CURRENT CHANGEFILE = {RDA *RDA-recerence* | *alpha-variable*}

The command keyword "SET" is implied and may be omitted. SET may be abbreviated S. CURRENT may be abbreviated CURR, with A for AREA and S for SET.

When the DBK option is used, the resulting database key is set in the reserved variable, C-DBK. When using the AKEY option, the resulting area key is set in the reserved variable C-AKEY. When the ANAME option is used, the resulting area name is set into the reserved variable C-AREA-NAME.

Format 3 only applies when interfacing with DMS 10R1 or a higher release level. The SET CURRENT CHANGEFILE command is not applicable to the single-thread interface. The *RDA-reference* or *alpha-variable* must contain a valid UREP 2200 storage area name (see UDS DMS 2200 Administration and Support Guide, 7830 7568, for more on change file considerations). The storage area name must be comprised of 12 or fewer ASCII characters. The SET CURRENT CHANGEFILE command must be issued prior to IMPART. If the change file is an EXEC file (not TIP), it must be assigned prior to IMPART. If test mode is specified and no SET CURRENT CHANGEFILE command is issued prior to IMPART, the system change file will be used.

Examples:

```
1.   FETCH4 PRODUCT-REC PRODUCTS AREA
     CURRENT DBK                          .  Save currency
     ...
     FIND1 PRODUCT-REC C-DBK
     CURRENT ANAME = RECORD PROD-REC      .  C-AREA-NAME = area
                                          .    name of the current
                                          .    PROD-REC.


2. DEF A CFILE 12 'DBUFILE'               .  DBUFILE is storage-area
                                          .    in UREP 2200
     CSF X '@ASG,A MY*DBUFILE.'           .  EXEC area must be
     IF X < 0                             .    pre-assigned.
       FACERR X
       STOP EXIT
     ENDIF
     CSF X '@USE DBUFILE.,MY*DBUFILE.'
     IF X < 0
       FACERR X
       STOP EXIT
     ENDIF
```

```
      SET CURRENT CHANGEFILE = CFILE
      IMPART
```

3.
```
DEF RDA CHGFILE (*,12)
 . . .
 SET RDA CHGFILE = 'MY-CHANGE'       .  MY-CHANGE is TIP
 CURR CHANGEFILE = RDA CHGFILE        .  storage area.
 IMPART
```

12.22 SET DBDN (Database Data Name)

This form of the SET command takes a value from the data storage area and places it in a specified database data name for use by a DML command. The database data name may be an area key, database key, area name, etc. Data types must be compatible. The referenced database data name must be defined in the currently invoked subschema.

Format:

[SET] DBDN *database-data-name* = {*integer-literal* [,*integer-literal*] | ;
variable-name [,*variable-name*] | ;
'*string-literal* ' } ;
[(*byte-length*) {DISP | DISP-1 | COMP | COMP-4}]

The command keyword "SET" is implied and may be omitted. SET may be abbreviated S.

The *database-data-name* must be defined in the currently invoked subschema.

The *byte-length* is the length of the database data name in bytes. The data type (DISP, DISP-1, COMP and COMP-4) are used to describe the form of the database data name. The *byte-length* and data type parameters must only be used when the database data name item being set does not have a usage mode of AREA-KEY, DATABASE-KEY, AREA-NAME, RECORD-NAME, or SET-NAME. These database data names have a form specifically defined within the object schema — no others, such as alias keys, do.

The *byte-length* for COMP and COMP-4 fields should be obtained from the conversion chart shown in Section 3.1.1.

Examples:

```
DBDN PRODAREA = 'PRODUCTS'           .  Set an area-name for calc.
DBDN LOCAKEY = 501,1                 .  Set an area-key to page
                                     .    501, record 1.
DBDN LOCAKEY = HOLD-AREA-KEY         .  Set an area-key from a
                                     .    variable.
DBDN CONTROL-KEY = HOLD-PG, HOLD-REC
                                     .  Set an area-key from two
                                     .    variables.
DBDN ACCESS-CTL = '$XXX123' (12) DISP
                                     .  Set an access control
                                     .    key.
DBDN PART-ALIAS = '123-0931-12' (30) DISP-1
                                     .  Set a FIELDATA alias key.
```

12.23 SET NON-FATAL (DML Errors)

This form of the SET command is used to eliminate the need to check for fatal error conditions resulting from the execution of DML commands. When a DML general error condition is encountered during execution, the non-fatal list will be checked. If the error number is not included in the list, I-QU PLUS-1 will execute its DML error termination routine and abort. Initially, the non-fatal list includes general error numbers 6, 7 and 13. Error numbers may be added and removed from the list at any time using this command. Up to 50 general errors may be specified. Rollback errors may not be specified as non-fatal. Check your DML Reference Manual for the meanings of these error codes.

Format:

[SET] NON-FATAL *error-num* {ON | OFF}

The command keyword "SET" is implied and may be omitted. SET may be abbreviated S.

Examples:

```
NON-FATAL 5 ON                .  Make error-number 0005 non-fatal.
NON-FATAL 13 OFF              .  Make error-number 0013 fatal.
```

12.24 STORE

The STORE command is used to store the contents of the RDA to a specific DMS 2200 record type. The data in the RDA may be created by using the SET RDA command, or it may be the result of a previous FETCH.

Format:

STORE *record-name* [*SUPPRESS-clause*]

STORE may be abbreviated STR.

The format of the suppress clause can be found in Section 12.25 at the end of this chapter.

Examples:

```
RDA PRODUCT-ID = 1011234
RDA PRODUCT-DESC = 'ABCDE'
DBDN PROD-AREA = 'PRODUCTS'    .  Set the area name DBDN.
STORE PRODUCT-REC              .  Store the record.
```

12.25 SUPPRESS Clause

The SUPPRESS CLAUSE may be used as the last parameter of any FETCH/FIND or STORE command. It is used to suppress updating specified levels of currency when executing DML commands.

Format:

```
SUPPRESS {ALL | ;
          || AREA | ;
             RECORD | ;
             {SET | ;
              set-name-1 [… set-name-n] } || }
```

SUPPRESS may be abbreviated SUP, with A for AREA and S for SET.

The AREA, RECORD and SET options may be used in combination. If the ALL option is used, no other option may be used. If the SET option is used, then *set-names* may not be listed.

Examples:

Suppress update of area and set currency.

```
FETCH3 NEXT PROD-DETAIL-SET SET SUPPRESS AREA SET
```

Suppress update of set currency for the two named sets. All other currency indicators will be updated.

```
FETCH5 PROD-MASTER SUPPRESS PROD-DET-SET PRSPL-SET
```

Chapter 13: Direct I/O Access

The I-QU PLUS-1 Processor can access sector formatted mass-storage and tape files using direct I/O.

13.1 DIO (DEFINE F)

For DIO, there is no I/O buffering used; data is transferred directly to and from the RDA. Mass-storage files may be EXEC, TIP/FCSS or TIP/DMS files. Tape files may be any format.

Format:

DEFINE F *filename* DIO *maximum-I/O-length-in-words*

DEFINE may be abbreviated DEF.

To define a TIP/FCSS file, specify the filename as follows:

TIP#*nnn*

To define a TIP/DMS file, specify the filename as follows:

TIPDMS#*nnn*

Where *nnn* is the FCSS or DMS file code, i.e., TIPDMS#21 for DMS area code 21, or TIP#200 for FCSS file 200. If an FCSS or DMS file code of 0 (zero) is specified, the value in the reserved variable S$ will be used as the file code when the file is accessed in a DIO command.

The *maximum-I/O-length-in-words* must not exceed the length of the RDA. This value will be used if the record is offset using a DEFINE RA directive.

Examples:

1. `DEFINE F WORK-FILE DIO 1000`
2. `DEFINE F TIPDMS#151 DIO 448`
3. `DEFINE F TIP#66 DIO 896`

Explanations:

Example 1 defines an EXEC file for DIRECT I/O. A maximum of 1000 words may be read from, or written to, the file in a single operation.

Example 2 defines a TIPDMS file (possibly a DMS 2200 database area) for DIRECT I/O.

Example 3 is a definition for a TIP/FCSS file.

13.2 DIO

The DIO command is used to perform direct I/O functions on a file defined for direct I/O. Data will be transferred directly between the RDA and the file using the IOW$ (DM$IOW in the case of TIP/DMS files) executive request, or the TIP FCSS file control primitives. Files accessed via the DIO command are not opened or closed as with PCIOS files. In the case of FCSS I/O, the three status words associated with the FCSS request will precede the data portion of the record in the RDA, automatically created for the user by the file DEFine.

Format:

DIO *filename function status-variable count* {*sector-address* | *record-number*}

To access a TIP/FCSS file, specify the filename as follows:

TIP#*nnn*

To access a TIP/DMS file, specify the filename as follows:

TIPDMS#*nnn*

Where *nnn* is the FCSS or DMS file code, i.e., TIPDMS#21 for DMS area code 21, or TIP#200 for FCSS file 200. If an FCSS or DMS file code of 0 (zero) is specified, the value in the reserved variable S$ will be used as the file code when the file is accessed in a DIO command.

For EXEC and TIP/DMS files, the available *functions* are:

WRITE	Mass-storage or tape;
WRITEEOF	Tape only;
READ	Mass-storage or tape;
READBACK	Tape only (read backwards);
RDLOCK	Mass-storage read with lock;
UNLOCK	Unlock mass-storage locked by RDLOCK;
REWIND	Tape only;
MOVE+	Tape only (move forward nn files);
MOVE-	Tape only (move backward nn files);
SPACE+	Tape only (space forward nn blocks);
SPACE-	Tape only (backspace nn blocks).

EXEC files must be @ASGed to the run before being accessed (see CSF command).

For TIP/FCSS files, the available *functions* are:

RD	Read;
RL	Read and Lock;
LK	Lock;
WR	Write;
WW	Write without lock;
WL	Write and keep lock;
UN	Unlock;
FL	File lock;
FR	File write lock;
AS	Assign file;
RV	Reserve file;
RE	Release file;
CG	Change file;
LF	List file.

In all cases, FCSS will execute synchronous I/O (FCDONE option).

The *status-variable* must be a defined numeric variable. It will contain the status at the completion of the command.

Count is used to specify the number of words to read or write; or the number of files or blocks in a MOVE or SPACE function. *Count* must be 0 for REWIND and WRITEEOF functions. The *count* may be entered as either a positive numeric integer literal or a numeric variable.

Sector-address applies to EXEC and TIP/DMS files, and is required on mass-storage READ and WRITE functions. It may be entered as either a positive numeric integer literal or a numeric variable.

Record-number applies only to FCSS files, and is required on READ and WRITE functions. It may be entered as either a positive numeric integer literal or a numeric variable.

Upon return from the DIO command, the I/O status will be put into the *status-variable*. For EXEC and TIP/DMS Read and Write functions, the final word count will be put into the reserved variable REC$LEN. These values will be displayed to the user when executing in conversational mode.

Examples:

EXEC Direct I/O:

```
.  Define tape and mass-storage files for Direct I/O.
DEFINE F TAPEIN DIO 2000
DEFINE F DISKFILE DIO 2000
...
    DIO TAPEIN MOVE+ X 1            .  Move forward past
                                    .    one file.
    IF X NOT = 0                    .  Check I/O status.
        DISPLAY 'I/O ERROR'
    ENDIF
    DIO TAPEIN READ X 2000          .  Read 2000 words.
...
    DIO DISKFILE READ X 1000,64     .  Read 1000 words at
                                    .    sector 64.
```

TIP/FCSS Direct I/O:

```
.  Define a TIP/FCSS file.
DEFINE F TIP#121 DIO 31
...
    DIO TIP#121 RL TIPSTAT 28,1    .  Read record number 1
    IF TIPSTAT < 0                 .  FCSS error?
        GO TIP-ERROR
    ENDIF
    DISPLAY RDA (13,4) UB9         .  Display 1st word of
                                   .    record (positions
                                   .    1 thru 12 are the
                                   .    3 status words).
    SET RDA (13,4) UB9 = 0         .  Set 1st word of
                                   .    record to zero.
    DIO TIP#121 WW TIPSTAT 28,1    .  Write record back.
    IF TIPSTAT < 0                 .  Check for FCSS error
                                   .    again ...
```

Chapter 14: RDMS 2200 Interface

The I-QU PLUS-1 commands, RDMS and RDMS+, provide the interface to RDMS 2200 databases. The commands are used in a similar fashion to the following COBOL commands:

```
ENTER MASM 'ACOB$DMR' ...
ENTER MASM 'RSA$PARAM' ...
```

The RDMS+ command is conceptually similar to ‘RSA$PARAM’, but must immediately follow the preceding RDMS (or RDMS+) command (not even a label is allowed in between). Also, the RDMS call must always pass the first three parameters. As many additional parameters as desired can be passed on the RDMS call subject to the following limitations: a maximum of 110 fields can appear in any I-QU PLUS-1 command including a maximum of 70 alphabetic words or names (“RDA RDA-reference” counts as two), 30 numeric literals and 10 quoted string literals. If more fields are required, the RDMS+ command can be used.

Refer to the Enterprise Relational Database Server for ClearPath OS 2200 SQL Programming Reference Manual, 7830 8160, for a complete discussion of available RDML commands.

14.1 The RDMS Command

The RDMS command provides the necessary RDMS 2200 interface information to access RDMS 2200 databases. This information is comprised of an RDML or SQL formatted command, RDML error status variables and application program variables. These variables may be RDA references or reserved/user variables. In certain cases, references to $PBUFF may be used as well as numeric and non-numeric literals.

Format:

RDMS *RDML/SQL-command error-status auxiliary-information* ;
[*program-variable-1* [,,, *program-variable-n*]]

Where *RDML/SQL-command* can be any, valid RDML command stated in one of the following ways:

{'*command-literal* ' | $PBUFF | RDA *RDA-reference* | *variable*}

Note: Refer to the Relational Database Server for ClearPath OS 2200 SQL Programming Reference Manual (7830 8160), for a complete discussion of available RDML commands.

Where *error-status* may be one of the following ASCII alphabetic references, 4 characters in length:

{RDA *RDA-reference* | *alpha-variable*}

Where *auxiliary-information* may be one of the following binary numeric references:

{RDA *RDA-reference* | *numeric-variable*}

If the "RDA *RDA-reference*" is used, the item must be defined as UB9, SB9 or COMP.

Where *program-variable-1* through *program-variable-n* can be stated in one of the following ways:

{RDA *RDA-reference* | *variable* | *numeric-literal* | '*alpha-literal* ' | $PBUFF}

Any non-result parameter on the RDMS call can be a quoted string literal. I-QU PLUS-1 allows a maximum of 10 string literals on a single command. For purposes of this check, RDMS and RDMS+ are separate commands; i.e., there may be up to 10 string literals on each occurrence of the RDMS and RDMS+ commands (see The RDMS+ in the next section).

Examples:

```
RDMS 'BEGIN THREAD FOR APPLICATION UDSSRC UPDATE ;' ;
    RDMS-STAT RDMS-AUX
RDMS 'END THREAD ;' RDMS-STAT RDMS-AUX
```

Notice the use of the semicolons in the above example. The semicolon within the quoted string is a part of the RDML syntax and denotes the end of the command. The second semicolon (outside the literal) is standard I-QU PLUS-1 notation. In this example, it is preceded by a space and it tells I-QU PLUS-1 that the command is continued on the next line.

The semicolon can also be used to continue lengthy alphabetic literals as shown in the following example. Here, the semicolon is placed immediately after (no space) the second single quote on the first line. The semicolon used in this manner tells I-QU PLUS-1 that the literal is continued on the next line. The second and third lines use command continuation as explained above.

```
RDMS 'GET DESCRIPTION INTO ';
    '$P1, $P2, $P3, $P4, $P5, $P6, $P7, $P8, $P9;' ;
    RDMS-STAT RDMS-AUX ;
    QUAL TAB VERS COL TYPE LEN DEC NULL KEY
```

The RDMS command allows the use of the $PBUFF special name for any non-result parameter. Thus, you can build long commands (or data items) for RDMS by:

```
      DISPLAY ' (long command or data, part 1) ' +
      DISPLAY ' (long command or data, part 2) ' +
      DISPLAY ' (long command or data, part 3) ' +
      ...
      RDMS $PBUFF status-var aux-info-var ...
```

As with other uses of $PBUFF, the print buffer is cleared once the call to RDMS is performed. $PBUFF could appear multiple times in the command but that only makes sense if the SAME data needs to be supplied multiple times since the data passed would be the same for each occurrence.

Further examples:

```
      D 'DECLARE LST CURSOR SELECT * FROM ' +
      TD TABLE +
      D ' WITH DESCRIPTION ;' +
      RDMS $PBUFF RDMS-STAT RDMS-AUX

      D 'GET DESCRIPTION INTO ' +
      D '$P1, $P2, $P3, $P4, $P5, $P6, $P7, $P8, $P9;' +

      RDMS $PBUFF RDMS-STAT RDMS-AUX ;
          QUAL TAB VERS COL TYPE LEN DEC NULL KEY
```

14.2 The RDMS+ Command

A maximum of 110 fields can appear in any I-QU PLUS-1 command including a maximum of 70 alphabetic words or names ("RDA RDA-reference" counts as two), 30 numeric literals and 10 quoted string literals. If an RDMS command requires more parameters than can be parsed on a single command (or if you just don't like long, continued commands ' ;') the RDMS+ command can be used to extend the number of parameters supplied for the RDMS call. The RDMS+ command must immediately follow the preceding RDMS (or RDMS+) command (not even a label is allowed in between).

The first three parameters for the RDMS call must be supplied on the RDMS call itself. The fourth and subsequent parameters can be supplied via the RDMS+ command. The RDMS+ command may not be used in conversational mode (the previous RDMS command would have already been completely processed and executed before you could enter the RDMS+ command).

Format:

RDMS+ *program-variable-1* [… *program-variable-n*]

Where *program-variable-1* through *program-variable-n* can be stated in one of the following ways:

{RDA *RDA-reference* | *variable* | *numeric-literal* | '*alpha-literal* ' | $PBUFF}

Examples:

```
D 'FETCH NEXT CUST_CURSOR INTO $P1,' +
D '$P2,$P3,$P4,$P5,$P6,$P7,$P8,$P9,$P10,$P11' +
RDMS $PBUFF STAT-VAR AUX-INFO
RDMS+ RDA CUST_KEY RDA CUST_LAST_NAME
RDMS+ RDA CUST_FIRST_NAME RDA CUST_MI ADDR1 ADDR2
RDMS+ RDA ADDR3 RDA ADDR4 RDA CITY RDA STATE
RDMS+ RDA ZIP
```

14.3 Automated RDA Definitions

Several RDML commands afford the ability to reference many program variables (column names). As shown in the previous example, these program variables may be RDA references. With this release of I-QU PLUS-1, an I-QU PLUS-1 program is provided to facilitate and automate the definition of these RDA references. The seventh file (Q6) of the release tape contains a source element called RDMS-PICS. After I-QU PLUS-1 installation, this element can normally be found in SYS$LIB$*IQU-1.

RDMS-PICS retrieves the column names from an RDMS table and creates COBOL 02-level definitions suitable for input to the QINDEX processor. The QINDEX utility can then be used to create a data item index file that can be referenced by the I-QU PLUS-1 INDEX directive. In addition to the COBOL 02-level definitions generated, RDMS-PICS generates a QINDEX FILE directive and a COBOL 01-level definition for each RDMS table chosen. The RDMS table name is used for the file name on the FILE directive and the 01-level data name. RDMS column names become the data names on the 02-level statements.

The QINDEX FILE directive generated allows an I-QU PLUS-1 program to use a pseudo DEFINE F directive for each table and thus, take advantage of alternate record area processing (DEFINE RA) as with PCIOS files. For example, assume an I-QU PLUS-1 program included the following definition of a PCIOS file, PAYMAST, and the pseudo definition for an RDMS 2200 table, PAY-RATE-TAB:

```
DEFINE F PAYMAST SEQ 120,0          .  PCIOS file
DEFINE F PAY-RATE-TAB SEQ 42,0      .  Pseudo define for
                                    .    RDMS Table
DEFINE RA PAY-RATE-TAB AFTER PAYMAST
```

The column values of PAY-RATE-TAB will occupy a different part of the RDA than the data item values of the file PAYMAST. The QINDEX FILE directive generated also makes it possible to reference column names without having to qualify a reference with the pseudo DEFINE F file name. However, since file names are restricted to 12 characters (table names can have as many as 30 characters), shorten the FILE directive file name to 12 or less characters prior to executing QINDEX. Decreasing the file name to 12 or fewer characters is only necessary if you plan to use the pseudo DEFINE F.

The file cataloged and created by RDMS-PICS is QKMS*RDMSFILES.

The following is the source listing of RDMS-PICS:

```
.  Create data item QINDEX input from RDMS 2200 column names
INPUT
INIT
LISTOFF
.   RDMS VARS
DEF A APPLICATION 30                .   Solicited from user to BEGIN
DEF A QUALIFIER 30                  .   THREAD and USE appripriate
DEF A VERSION 30                    .   QUALIFIER.TABLE:VERSION
DEF A TABLE 30                      .   (QUALIFIER  =  schema).
DEF A RDMS-STAT4''                  .  Error number from RDMS
DEF N RDMS-AUX 0                    .  Column if syntax error
DEF A ERROR-MSG 132                 .  Long message
DEF A PREVTAB 30                    .  Hold table for FILE and 01 gen
.  RDMS $P1 -$P9                    .  Values returned from GET DESC
DEF A QUAL 30                       .  Not used
DEF A TAB 30                        .  Used to gen FILE and 01 name
DEF A VERS 30                       .  Not used
DEF A COL 30                    .  Used to gen data name
DEF A TYPE 14                   .  Not used
DEF N LEN                       .  Used to gen PIC clause
DEF N DEC                       .  Used to gen V dec positions
DEF N NULL                      .  Not used
DEF N KEY                       .  Not used
.
.  Solicit user input -QUIT on any query will terminate program
  ACCEPT APPLICATION 'ENTER APPLICATION (<UDSSRC>/application_name): '
  IF APPLICATION = $SPACES
      APPLICATION = 'UDSSRC'      .  UDS default application name
  ELSE
      SHIFT APPLICATION TO UPPER
      IF APPLICATION = 'QUIT'
          GO QUITIT
      ENDIF
  ENDIF
  DO UNTIL QUALIFIER <> $SPACES
      ACCEPT QUALIFIER 'ENTER TABLE QUALIFIER (qualifier_name/';
                                                'schema_name): '
      IF QUALIFIER = $SPACES
          D '*** QUALIFER MUST BE NON-BLANK ***'
      ELSE
          SHIFT QUALIFIER TO UPPER
          IF QUALIFIER = 'QUIT'       .  No default schema/qualifier
              GO QUITIT
          ENDIF
      ENDIF
  ENDDO
  ACCEPT VERSION 'ENTER TABLE VERSION (<PRODUCTION>/version_name): '
  IF VERSION = $SPACES
      VERSION = 'PRODUCTION'      .  DDS default version name
  ELSE
      SHIFT VERSION TO UPPER
      IF VERSION = 'QUIT'
          GO QUITIT
      ENDIF
  ENDIF
.
.  BEGIN THREAD in RETRIEVE mode and set DEFAULTs
.
  TD 'BEGIN THREAD FOR APPLICATION ' APPLICATION ' RETRIEVE ;' +
  RDMS $PBUFF RDMS-STAT RDMS-AUX
  DO ERRCHK
  TD 'USE DEFAULT QUALIFIER ' QUALIFIER ' ;' +
  RDMS $PBUFF RDMS-STAT RDMS-AUX
  DO ERRCHK
  TD 'USE DEFAULT VERSION ' VERSION ' ;' +
  RDMS $PBUFF RDMS-STAT RDMS-AUX
  DO ERRCHK
  DO                              .  Do until AT END or BREAK

      ACCEPT TABLE 'ENTER TABLE (Table_name/QUIT): ' AT END BREAK
      SHIFT TABLE TO UPPER
```

```
            IF TABLE = $SPACES
            OR TABLE = 'QUIT'
                BREAK
            ENDIF
            PC BRKPT 'QKMS*RDMSFILES'   .   Will be input to QINDEX
            DO TABLE
            PC LOCAL                    .   Reset temporarily for next ACC
        ENDDO                           .     - will append to BRKPT file
        GO FINALE

      TABLE PROCEDURE
        TD 'DECLARE LST CURSOR SELECT * FROM ' TABLE ' WITH DESCRIPTION ;' +
        RDMS $PBUFF RDMS-STAT RDMS-AUX
        DO ERRCHK
        DO
            RDMS 'GET DESCRIPTION INTO ';
                 '$P1, $P2, $P3, $P4, $P5, $P6, $P7, $P8, $P9 ;' ;
                RDMS-STAT RDMS-AUX ;
                QUAL TAB VERS COL TYPE LEN DEC NULL KEY
            IF RDMS-STAT = '6001'       .  End of column names in table
                BREAK
            ENDIF
            DO WHILE TAB <> PREVTAB     .  Gen FILE and 01 name 1 time
                TD 'FILE ' TAB          .  per table
                TD 8 '01  ' TAB '.'
                PREVTAB = TAB           .  Hold current table name
            ENDDO
            SHIFT TYPE TO UPPER
            IF TYPE = 'NCHAR'           .  16-bit char set not supported
                GO NEXT-COLUMN
            ENDIF
            D 12 '02  '+
            D 16 COL +                  .  Data name = column name
            IF TYPE = 'REAL'
            OR TYPE = 'FLOAT'
            OR TYPE = 'DOUBLE'
                D 47 ' COMP-2.'
            ELSE
                D 47 'PIC '+
                IF TYPE = 'DECIMAL'
                OR TYPE = 'NUMERIC'
                OR TYPE = 'DATE'
                OR TYPE = 'TIME'
                OR TYPE = 'TIMESTAMP'
                    D 52 'S9(' +        .  All decimal fields are signed
                    IF DEC = 0
                        TD LEN ') COMP.'
                    ELSE
                        LEN = LEN - DEC
                        TD LEN ')V9(' DEC ') COMP.'
                    ENDIF
                else
                if type = 'INTEGER'
                or type = 'SMALLINT'
                    d 52 'S9(10) COMP.'
                else
                    TD 52 'X(' LEN ').'   .  CHARACTER/NCHARACTER
                endif
            ENDIF
        ENDIF
NEXT-COLUMN
  ENDDO
  RDMS 'DROP CURSOR LST;' RDMS-STAT RDMS-AUX
  DO ERRCHK
  ENDPROC

FINALE
  RDMS 'COMMIT WORK ;' RDMS-STAT RDMS-AUX
  DO ERRCHK
  RDMS 'END THREAD ;' RDMS-STAT RDMS-AUX
  DO ERRCHK

QUITIT
  STOP EXIT

ERRCHK PROCEDURE
  IF RDMS-STAT <> '0000'
      PC LOCAL
      D 'RDMS ERROR STATUS:' +
      D RDMS-STAT +
      D ' RDMS AUX INFO:' +
      D RDMS-AUX
      DO UNTIL ERROR-MSG = $SPACES
          RDMS 'GETERROR INTO $P1 ;' ;
              RDMS-STAT RDMS-AUX ERROR-MSG
          D '<ERRMSG>' +
          D ERROR-MSG
      ENDDO
      STOP EXIT
    ENDIF
  ENDPROC
```

The program can be run as a batch or interactive job utilizing the IQU processor as the following example illustrates:

```
►@IQU,I
►IQU IQU11R6 (Release 11R6) (961114 1146:10) ...
►Copyright 1986-1999 by KMSYS Worldwide, Inc. – All Rights ...
►Initial Mode is INPUT
►ADD RDMS-PICS FROM SYS$LIB$*IQU-1
►ENTER APPLICATION (<UDSSRC>/application_name): ►
►ENTER TABLE QUALIFIER (qualifier_name/schema_name): ►DEMOSCHEMA
►ENTER TABLE VERSION (<PRODUCTION>/version_name): ►
►ENTER TABLE (Table_name/QUIT): ►CUST_KEY_TAB
►ENTER TABLE (Table_name/QUIT): ►CUST_ADDR_TAB
►ENTER TABLE (Table_name/QUIT): ►QUIT
►@CAT,P CUSTINDEX.
►@QINDEX,I CUSTINDEX.
►@ADD QKMS*RDMSFILES.
```

The interactive session above uses the RDMS 2200 default application name (UDSSRC) and the DDS 2200 default version (PRODUCTION) by simply transmitting a blank line.

The output from RDMS-PICS for the above example would appear as follows:

```
      FILE  CUST_KEY_TAB
             01  CUST_KEY_TAB.
                 02  CUST_KEY                      PIC X(9).
                 02  NAME                          PIC X(33).
                 02  CHG_TIME                      PIC S9(9).
      FILE  CUST_ADDR_TAB
             01  CUST_ADDR_TAB.
                 02  STATE                         PIC X(2).
                 02  CITY                          PIC X(18).
                 02  ACCOUNT                       PIC X(9).
                 02  CUSTNAME                      PIC X(33).
                 02  ADDR1                         PIC X(30).
                 02  ADDR2                         PIC X(30).
                 02  ADDR3                         PIC X(30).
                 02  ZIP                           PIC X(9).
                 02  AREACODE                      PIC X(3).
                 02  EXCHANGE                      PIC X(3).
                 02  TELNUM                        PIC X(4).
```

The RDA references available by using the INDEX directive in an I-QU PLUS-1 program would be:

```
      CUST_KEY_TAB                   (00001,00051)
      CUST_KEY                       (00001,00009)
      NAME                           (00010,00033)
      CHG_TIME                       (00043,00009) SN9
      CUST_ADDR_TAB                  (00001,00171)
      STATE                          (00001,00002)
      CITY                           (00003,00018)
      ACCOUNT                        (00021,00009)
      CUSTNAME                       (00030,00033)
      ADDR1                          (00063,00030)
      ADDR2                          (00093,00030)
      ADDR3                          (00123,00030)
      ZIP                            (00153,00009)
      AREACODE                       (00162,00003)
      EXCHANGE                       (00165,00003)
      TELNUM                         (00168,00004)
```

Chapter 15: BIS DTM Interface

I-QU PLUS-1 programs can interface with BIS's Data Transfer Module (DTM). Two I-QU PLUS-1 directives and four commands are provided to handle requests through BIS queues: the DEFINE F and DEFINE RA directives; and the OPEN, CLOSE, READ and WRITE commands.

For a complete example using the BIS DTM interface, see the I-QU PLUS-1 Application Development User Guide.

15.1 Queue-alias Definition (DEFINE F)

Before BIS data can be referenced, a queue-alias must be defined with the DEFINE F directive. A queue-alias can be looked at as an internal file name to be associated with a particular BIS RID. Lines of data can be read from or written to this RID via a BIS queue name (configured with a BIS system through the DTM configuration report). Since it is possible to be accessing more than one RID simultaneously through the same BIS queue, I-QU PLUS-1 requires that a separate queue-alias be defined for each of these opened reports.

Queue-alias files may be opened, referenced and closed as necessary. All Queue-alias file and database input and output is done through the RDA. If multiple records are to be read and compared, the user may set up alternate record areas using the DEFINE RA directive, or move necessary data items to a defined variable storage location.

Format:

DEFINE F *queue-alias* MAPPER [*maximum-transfer-size* ;
parameter-block [{IN | OF} *filename*]

DEFINE may be abbreviated DEF.

The *maximum-transfer-size* may be between 8 and 256 characters. The default is 132 characters.

Note that the DEFINE FF format (available for PCIOS files) is NOT supported since the DTM interface only supports ASCII (FCS) transfers.

The *queue-alias* is used on the OPEN, READ, WRITE and CLOSE commands to refer to the specific BIS OPEN-*queue-alias/parameter-block* combination. The *queue-alias* is a logical name and does not convey the actual destination queue name. The use of the *queue-alias* allows the I-QU PLUS-1 program to have several *queue-aliases* open concurrently to the same BIS queue name. The actual destination queue name is passed as a field in the *parameter-block*.

The *parameter-block* contains fields necessary to affect data transfer with the DTM. It must be defined prior to any *queue-alias* DEFINE F directive. The format of the *parameter-block* is 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 p--b-err-message        (*,80)           A9
.                        168 character positions total
.
```

This data structure is defined in the element DTM-PARM-BLK in the file, SYS$LIB$*IQUx-1, and can be copied directly into an I-QU PLUS-1 program using the ADD directive. Consult the person responsible for installing I-QU PLUS-1 in order to determine the actual file name of this file on your system.

The *parameter-block* may be qualified by a filename previously defined on a DEFINE F directive.

The *parameter-block* specifications are established by KMSYS Worldwide software development and as such are subject to change from release level to release level. For this reason, KMSYS Worldwide highly recommends that the definition for the *parameter-block* be copied from the file, SYS$LIB$*IQUx-1, rather than hard coding it in an I-QU PLUS-1 program.

The 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, under similar but different COBOL names, are described in the Unisys Business Information Server for 2200 SCHDLR Interface Programming Reference Manual, 7832 1122, with the exception of the PB-START-LINE and PB-XFER-LINES fields, which are described below.

The PB-FILLERx fields are for structure alignment and ***must*** be present. The PB-STATUS, PB-ERR-CODE and PB-ERR-MESSAGE fields are returned following any DTM related I-QU PLUS-1 command and must be tested by the I-QU PLUS-1 program to determine the success of the operation. This is different from the I-QU PLUS-1 PCIOS implementation where any error other than end of file or invalid key is fatal to the program.

KMSYS Worldwide supplies a BIS run under the default name of IQU$DTM in the file, SYS$LIB$*IQUx. It is in non-BIS format without headers and should be retrieved into BIS and registered by the BIS Coordinator. This run can be restricted by BIS mode and/or user-id at the discretion of the BIS Coordinator.

15.2 Alternate Queue-alias Areas (DEFINE RA)

The DEFINE RA directive for DTM performs similarly to its PCIOS implementation. Note that it defines an alternate record source/delivery area, which MUST be at least 256 characters (132 for BIS level 34 and earlier) in length for input operations regardless of the actual width of the BIS report being retrieved. This directive does not affect the location of the param-block referenced on the DEFINE F directive.

Format:

DEFINE RA {*filename* | *record-name* | *queue-alias* | SORT} ;
{*absolute-word-number* | ;
{OVERLAY | AFTER} {*filename* | *record-name* | *queue-alias* | SORT}

DEFINE may be abbreviated DEF.

15.3 CLOSE

The CLOSE command is similar to its PCIOS equivalents in that it releases any pending buffers, requests SCHDLR/BIS to complete the request and returns the completion status to the I-QU PLUS-1 program. This command will be performed implicitly if an INIT directive is entered or if I-QU PLUS-1 is about to terminate normally.

Format:

CLOSE *queue-alia*s

CLOSE may be abbreviated CL.

15.4 OPEN

Currently BIS reports can be retrieved (INPUT) or created/replaced (OUTPUT). Other access modes are not currently supported. A partial report can be retrieved by setting the PB-START-LINE and PB-XFER-LINES fields in the parameter block BEFORE opening the queue-alias file. These parameters are not used when opening for OUTPUT. Other access modes (particularly APPEND) may be supported in future I-QU PLUS-1 releases. Note that the DTM/MTQ interface in BIS limits the size of messages, which may be passed to 113,000 words (approximately 5100 80-character or 3000 132-character ASCII [FCS] lines). Thus, reports written to BIS that exceed this limitation must be written into multiple RIDs and combined within BIS. Larger reports can be conveniently retrieved from BIS into an I-QU PLUS-1 program by opening the queue-alias multiple times using different line range specifications. The default BIS run supports referencing reports either by name or by mode number. If the RID number is supplied as zero on an OPEN for OUTPUT, the next available RID number will be used and the report actually created (by the AR [Add Report] function will be returned in the parameter block following the CLOSE command.

Format:

OPEN *queue-alias* {INPUT | OUTPUT} SEQ

INPUT may also be abbreviated INP.

15.5 READ

The READ command performs similarly to the PCIOS equivalent for sequential input files. The data is transferred to the record delivery area (its location is either position one of the RDA or as was modified by the DEFINE RA directive). The REC$LEN reserved I-QU PLUS-1 variable is set with the actual number of significant characters in the record (from zero to 256). Note that 256 characters (132 for BIS level 34 and earlier) is always delivered to the record delivery area, even if the significant data is shorter. Short records are padded with ASCII spaces.

Format:

READ *queue-alias* [AT END {*program-label* | BREAK}]

15.6 WRITE

This command performs similarly to the PCIOS equivalent for sequential output files. The data is transferred from the record delivery area (its location is either position one of the RDA or as was modified by the DEFINE RA directive). The length transferred is either as specified on the WRITE command or is 256 characters (132 for BIS level 34 and earlier) if omitted.

Format:

WRITE *queue-alia*s [*length*]

WRITE may be abbreviated W.

Chapter 16: Command and Keyword Abbreviations

The following is a list showing the minimum number of characters required for keywords and commands. Also shown are accepted abbreviations and mnemonics:

ACCEPT	A
ACQUIRE	ACQ
ADD	AD
APPLICATION	APPL
AREA	A
BITMERGE	BITM or BM
BITSPLIT	BITS or BS
BREAK	B
BRKPT	B
CALSIM	CA
CDELETE	CD
CHARACTERS	CHARS
CLEAR	CLE
CLEARSCREEN	CLS or CLEARS
CLOSE	CL
COMPILE	C
CONNECT	CONN
CONSOLE	CONS or CONSOL
CONV	CON
CSF	CS
CURRENT	CURR
DATE	DA
DBERROR	ERROR
DECIMAL	DEC
DEFINE	DEF
DELETE	DEL
DEPART	DEP
DISCONNECT	DIS
DISPLAY	D
DUMP	DU
DUPLICATE	DUP
ECHO	EC
EDIT	ED
EJECT	E
ELSE	EL
ENDDO	ENDD
ENDIF	ENDI
ENDPROC	ENDP
EXCLUSIVE	EX
EXIT	EXI
FACERR	FA or FE
FETCH1	F1
FETCH2	F2
FETCH3	F3
FETCH4	F4
FETCH5	F5
FETCH6	F6
FETCH7	F7
FIND1	FN1
FIND2	FN2
FIND3	FN3
FIND4	FN4
FIND5	FN5
FIND6	FN6
FIND7	FN7
FIRST	F
FREE	FR
GETPTR	GETP
GREGORIAN	G
IMPART	IMP
INDEX	IND
INIT	INI
INPUT	INP or I
INSERT	INS
INSPTR	INSP
INVOKE	INV
JULIAN	J
KEEP	K
LAST	L
LISTOFF	LISTOF
LOAD	L
LOCAL	L
MODIFY	M
NEXT	N
NULPTR	NULP
OBJECT	OB
OPEN	O
OPTION	OPT
OUTPUT	O
OWNER	O
PCONTROL	PC
PRINTER	PR
PRIOR	P
PROCEDURE	PRO
PUTPTR	PUTP
READNEXT	READN
RECORDS	RECS or RECORD
RELEASE	REL
REMOVE	REM
REMPTR	REMP
RETRIEVAL	RET
RETURN	RETU
REWRITE	REW
ROUND	ROU
RSTPTR	RSTP
RUN	R
SAVE	SA
SCAN	SC
SET	S
SETPTR	SETP
SHIFT	SHI
SORT	SOR
START	ST
STORE	STOR or STR
SUPPRESS	SUP
SWGET	SWG
SWSET	SWS
TABS	TA
TIME	TI
TRACE	TRA
TRANSFER	TRANS
TRIMDISP	TD or TRIMD
TRIMEDIT	TE, TED or TRIME
UPDATE	U
WILDCARD	WI
WRITE	W
XREF	XR
XTRPTR	XTRP

Table 16-1: Keyword Abbreviations

Chapter 17: DBDUMP File Description

The following is a COBOL description of I-QU PLUS-1's DBDUMP formatted file. This file format may be input to any other COBOL program, or created by a COBOL program, as long as it is specified as follows:

The SELECT statement:

SELECT DBFILE ASSIGN TO DISC *

* File may be externally assigned to tape.

The FD:

```
FD  DBFILE
LABEL RECORDS ARE STANDARD
BLOCK CONTAINS 2 RECORDS.
01  DUMP-RECORD.
    05  FIXED.
        10 RECORD-NAME PIC X(32).
        10 NUMBER-CTL-WORDS PIC 9(10) COMP.
        10 CONTROL-WORDS-POS PIC 9(10) COMP.
        10 NUMBER-DATA-WORDS PIC 9(10) COMP.
        10 TOTAL-WORDS PIC 9(10) COMP.
    05 VARIABLE-LENGTH-DATA.
        10 DATA-WORDS OCCURS 1 TO max-rec-length
                DEPENDING ON TOTAL-WORDS.
            15 DATA-WORD PIC X(4).
```

Field Description:

The *max-rec-length* may not exceed the length of the RDA generated for the version of I-QU PLUS-1 that will read the file.

RECORD-NAME will be moved to the reserved variable C-O-T after each record is read by I-QU PLUS-1. I-QU PLUS-1 program logic may then be based on the contents of this field.

NUMBER-CTL-WORDS must contain the number of words in the control segment of the record. If the control segment is present, it will start in the first occurrence of DATA-WORD. If no control segment is present, this field must be zero.

CONTROL-WORDS-POS must contain the word position with I-QU PLUS-1's RDA into which the control segment will be placed. If NUMBER-CTL-WORDS is zero, this item must also be zero.

NUMBER-DATA-WORDS must contain the total number of data words contained in the record. Data words will immediately follow the control segment, if one is present, or will begin in the first occurrence of DATA-WORD.

TOTAL-WORDS must contain the total number of control and data words in the record. This field must equal the sum of NUMBER-CTL-WORDS plus NUMBER-DATA-WORDS.

Chapter 18: QINDEX Reference

QINDEX is a processor used to create a data item index file of data item definitions. QINDEX is executed in demand mode and is installed on your system when I-QU PLUS-1 is installed. This chapter describes the purpose and rules for using the QINDEX processor.

18.1 Introduction

The data item index file is used by the I-QU PLUS-1 processor to obtain the location and format of data items in the record delivery area (RDA) automatically. I-QU PLUS-1 automatically creates a primary data item index file on start-up. When an INVOKE directive is processed, information related to DMS 2200 areas, records, sets and database datanames, and information related to data items (fields within records) is added to the primary index. This information is used in editing and encoding DMS 2200 DML commands, and in the resolution of RDA names used within various commands and directives. Data item definitions in the primary data item index are obtained directly from the object schema and subschema.

Data redefinitions coded in the subschema are not available via the INVOKE process, because they are not included in the object subschema. They are only present in the S$PROC element created by the SDDL and copied into user programs by the ADMLP processor. To make redefinitions available to I-QU PLUS-1, a secondary data item index may be created using QINDEX.

When a data item name is used in an I-QU PLUS-1 command or directive, the processor will first search its internal data definition table for the item name. Then, if the user has not defined the item (DEFINE RDA), the search will continue to the primary data item index for the correct definition. If the name is not found in the primary index, the secondary index, if present, is searched. The search of either index file will also match the record qualifier, if one was used. If the item name appears in more than one record and no record qualification is used, the first matching definition will be used.

The search order used in resolving a data item name is as follows:

1. Current DEFINE RDA names (always overrides data item index);
2. The primary data item index;
3. The secondary data item index.

A major use of the QINDEX processor is to create a single definition for all data within an entire application. The definition may be composed from a DMS 2200 subschema and schema plus any number of non-DMS 2200 file definitions. Non-DMS 2200 files may include any file types supported by the I-QU PLUS-1 processor.

18.2 QINDEX, General Information

The QINDEX processor is designed to create compatible data item index files from object schemas and subschemas, and standard COBOL definitions for use in the I-QU PLUS-1 processor. The data item index file created by QINDEX may be invoked by the I-QU PLUS-1 processor by using the INDEX directive (see the I-QU PLUS-1 Programmer Reference).

18.2.1 Input to QINDEX

The input to QINDEX consists of several directives, COBOL data definitions and optionally, an object schema and subschema. COBOL data definitions from many elements may be processed at one time (the processor call may be followed by many @ADD statements). No limit exists to the number of items that can be included in the data item index file. COBOL data definitions are input in standard COBOL source image format. The following restrictions apply:

- QINDEX will assume that the COBOL source input has been processed through the COBOL compiler at some point, and therefore will perform no COBOL syntax checking.
- Each 01 level definition from the source input will begin a new record definition. The generated RDA position pointer will be reset to character position one. The first data item image input to the QINDEX Processor must be an 01 level.
- Input images may contain comments, COBOL PROC headings and END statements. These will be ignored by the QINDEX processor.
- QINDEX will ignore all 66, 77 and 88 level items. FILLER items will be used to determine data item positions, and will not be output to the data item index file. Any VALUE clauses encountered will also be ignored.
- Input must not contain mixed FIELDATA and ASCII definitions. Allocation is based on the processor option settings. QINDEX will not make a distinction between DISPLAY and DISPLAY-1 or COMP and COMP-4. QINDEX will not support or recognize FIELDATA binary alignment (for example, PIC H99999).
- QINDEX supports UCOB data types of BINARY and BINARY-1. Also, both ACOB and UCOB S$PROC elements may be input to QINDEX.
- Double-byte characters as specified by the DISPLAY-2 data type are supported for the Asian character sets.
- On items within an OCCURS clause, QINDEX will generate a definition that refers to the first item occurrence. References to subsequent occurrences may be made by using either the RDA Indexing or Subscripting features of the I-QU PLUS-1 Processor.
- QINDEX supports exact binary notation for both DMS 2200 records (INVOKE directive) and PCIOS files (FILE directive). QINDEX can properly align data items that are included in record/file definitions (COBOL 01 level) containing exact binary data items. For example:

  ```
  PIC 1(36)
  PIC 1(5)
  PIC 1(1)
  ```

  This feature is currently supported for field alignment purposes only. A future release of I-QU PLUS-1 will support access-to-bit definitions.

18.2.2 Output of QINDEX

The final output of QINDEX is an I-QU PLUS-1 compatible secondary data item index file. In addition to the data item index file, QINDEX may optionally produce a listing of the data item index file's contents. The listing will show each entry in alphabetical sequence, grouped by areas, records, sets, database datanames and data items.

18.3 Running QINDEX

QINDEX must be run as a processor (not an @XQT) using the following format:

@IQINDX,*options index-file-name*

Followed by DIRECTIVES and user-supplied COBOL data definition source images.

The processor call name given to QINDEX (the default name) is determined when QINDEX is installed. For the correct processor name at your site, consult with the person responsible for installing QINDEX at your site.

The *index-file-name* must be the name of an existing mass-storage file that will be used as the Data Item Index File.

Following the processor call line are the optional QINDEX directives and user-supplied COBOL source images that will be used in processing the Data Item Index File.

18.3.1 QINDEX Processor Options

Available execution options are:

B	Causes the execution of QINDEX to be treated as a batch mode execution.
D	Causes the execution of QINDEX to be treated as a demand mode execution. This option is the converse of the B-Option.
F	Option "F" indicates that the definitions being processed are FIELDATA. The default is ASCII.
I	The "I" option will cause the index file to be initialized. Any existing definitions will be lost. If this option is not present, the index file will be opened for update. The INVOKE directive may only be used when the I option is used.
L	The "L" option will result in the listing of all source input images as they are read, and will cause the entire data item index file contents to be listed upon completion of processing of the source input.
N	Option "N" is used to produce a list of the contents of the index file with no update and no directives or source input. If this option is used, all other options are ignored.
S	Option "S" will cause the listing of all source input images as they are read.
T	For KMSYS Worldwide debugging only. Use only if directed to by KMSYS Worldwide' personnel.
O	The "O" option will cause the processor to overwrite any duplicate definitions encountered. The definition of the last duplicated item name will be used. If this option is not used, a warning message will be displayed for each duplicated item name encountered, and the first definition will be used.
W	Option "W" causes QINDEX to produce a data item list, replacing the "RDA REF" portion of the listing with starting word, starting bit and bit length of each item. The starting word is relative to zero, that is, for the first word in a record/file, starting word = 00000. The starting bit begins at bit one of the word and proceeds from left to right. If the "W" option is not specified, the default RDA reference (starting character, number of characters) will be displayed.

18.3.2 QINDEX Directives

There are several QINDEX directives that control the creation or update of the Data Item Index File. The following describes the function of each.

The QINDEX directives are:

INVOKE
S$PROC
RECORD
FILE

18.3.2.1 INVOKE

The INVOKE directive for QINDEX works in a similar manner to the INVOKE in the I-QU PLUS-1 processor in that it accesses the object schema and subschema to build the Data Item Index File. The difference is that no initialization of D$WORK and S$WORK is involved.

The INVOKE will create a data item index file record for each area, record, set, database dataname and data item included in the specified subschema. The data item index file will not include redefinitions of data items in the subschema, because redefinitions are not included in the subschema or schema object. Redefinitions are only present as source images in the S$PROC element for the subschema which is copied into COBOL programs by the ADMLP Processor prior to program compilation (see the S$PROC directive on the following page).

The INVOKE may only be used when initializing ("I" option) a new Data Item Index file, and must be the first directive read.

Format:

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

The *filename* or *file-code* must contain both the subschema and schema object elements.
Example:

```
INVOKE SUB-REQMTS IN MFG-SCHEMA FILE DMS*SCHEMAFILE
```

18.3.2.2 S$PROC

The purpose of the S$PROC directive is to apply data item redefinitions to an initial Data Item Index file created by the INVOKE directive. The S$PROC element contains the COBOL source for the subschema as generated by the SDDL process, including any item redefines statements. INDEX will match the S$PROC source to the records built by the INVOKE to apply redefinitions to the appropriate records. The S$PROC directive is optional, and must immediately follow the INVOKE directive.

Format:

S$PROC [*filename*]

The *filename* is optional, and only needs to be specified if the S$PROC element for the subschema named on the INVOKE is in a different file, or if the INVOKE specified a TIP schema file. If omitted, QINDEX will used the same file name used in the INVOKE. S$PROC cannot be processed from a TIP schema file.

18.3.2.3 RECORD

The RECORD directive is used to add additional item definitions to an existing DMS 2200 record. In this manner, a record can be redefined in any way desired beyond any definitions that exist in either the schema or subschema. This redefinition may be required if an application uses a generic record definition in the schema with the record defined in many different formats within various application programs. Definitions added via the RECORD directive will be tied directly to the specified record. All RECORD directives must follow the INVOKE and S$PROC directives, if present.

Format:

RECORD *subschema-record-name*

Followed by COBOL description source

The *subschema-record-name* must currently exist in the data item index file. It is placed in the data item index file by the INVOKE when the file is initialized.

If the subschema specifies only selected items of the record via an ITEMS ARE clause, care must be taken to ensure that added definitions match the mapped form of the record. The mapped form is the form containing only those items included in the ITEMS ARE clause.

Examples:

```
RECORD PART-MSTR
        01  PART-MSTR.
       *** THE FOLLOWING DEFINES THE FIRST 80 POSITIONS
       *** OF THE PART MASTER RECORD FOR USE AS A
       *** HEADER (THIS IS NOT DEFINED IN THE SCHEMA).
            05  PART-MSTR-HDR      PIC X(80).
            05  FILLER             PIC X(102).
```

The above definitions will be applied to the PART-MSTR record defined in the currently invoked subschema.

18.3.2.4 FILE

The file directive allows the addition of non-DMS 2200 file records. These definitions are not created by the INVOKE, S$PROC or RECORD directives. When a file definition is included in a data item index file, the I-QU PLUS-1 processor will be able to address file data items by name, and automatically determine the RDA location of the record.

When a file is defined in QINDEX, it is assigned an internal file code beginning with 4097. Each subsequent file is assigned the next higher number. This range is used to distinguish non-DMS 2200 files from DMS 2200 records (the highest possible record code is 4095). These codes are used in the I-QU PLUS-1 processor to determine the record's RDA position automatically when items within records are referenced. In order for this feature to function correctly in I-QU PLUS-1, the INDEX directive must be processed before all file definitions (DEFINE Fs).

Format:

FILE *filename* [IBM]

Followed by COBOL description source

The *filename* must be the same name used when the file is defined in I-QU PLUS-1 with the DEFINE F directive.

Example:

```
FILE PART-HIST
        01  PART-HIST.
            05  PH-PART-NUMBER     PIC 9(5).
            05  PH-PART-NAME       PIC X(50).
            05  PH-PRICE           PIC 9(5)V999.
            05  PH-PRICE-DATE      PIC 9(6).
```

The file definition may contain REDEFINES and additional 01 levels as needed. Once a file has been defined in this manner, RDA item referencing and automatic record area offset operate the same as for DMS 2200 records.

QINDEX can be instructed to treat the COBOL PIC clause as if it had been defined for an IBM 360/370 environment: i.e., allocate/align each data item according to the rules required for the EBCDIC character set by adding the “IBM” option to the QINDEX FILE directive.

The allocation rules are defined below:

Data Type	**UNISYS 2200 Allocation**	**IBM 360/370 Allocation**
DISPLAY	9 bits per character (quarter word aligned)	9 bits per character (quarter word aligned)
COMP	1-2 digits, 9 bits 3-5 digits, 18 bits 6-7 digits, 27 bits 8-10 digits, 36 bits 11-13 digits, 45 bits 14-15 digits, 54 bits 16-18 digits, 63 bits (quarter word aligned)	1-4 digits, 18 bits 5-9 digits, 36 bits 10-18 digits, 72 bits (quarter word aligned)
COMP-1	36 bits (word aligned)	36 bits (quarter word aligned)
COMP-2	72 bits (word aligned)	72 bits (quarter word aligned)
COMP-3	9 bits per digit (quarter word aligned)	1 digit 9 bits 2-3 digits 18 bits 4-5 digits 27 bits 6-7 digits 36 bits 8-9 digits 45 bits 10-11 digits 54 bits 12-13 digits 63 bits 14-15 digits 72 bits 16-17 digits 81 bits 18 digits 90 bits (quarter word aligned)

Table 18-1: QINDEX Data Allocations

18.4 Building an Application Definition

The following example builds a new application Data Item Index file from the base subschema and schema, applies redefinitions from the S$PROC element, adds more redefinitions to database records and finally adds PCIOS file definitions. The Data Item Index File built may then be considered an entire application data definition to be used in I-QU PLUS-1 applications.

```
@ASG,UP MFG*MFGQINDEX.
@IQINDX,IL MFG*MFGQINDEX.
INVOKE REQMTS-SUB IN MFG-SCHEMA FILE DMS*SCHEMAFILE
S$PROC
RECORD PART-MSTR
        01  PART-MSTR.
       *** THE FOLLOWING DEFINES THE FIRST 80
       *** POSITIONS OF THE PART MASTER RECORD FOR
       *** USE AS A HEADER (NOT DEFINED IN SCHEMA).
            05  PART-MSTR-HDR      PIC X(80).
            05  FILLER             PIC X(102).
FILE DLY-TRANS
        01  DLY-TRANS-REC.
```

```
                  05  DT-ACCOUNT-NUM     PIC 9(10).
                  05  DT-ACCOUNT-ALP REDEFINES DT-ACCOUNT-NUM.
                      10  DT-DIVISION    PIC 9(3).
                      10  DT-REGION      PIC 9(3).
                      10  DT-SERIAL      PIC 9(5).
                  05  DT-CTGY-CODE       PIC X.
                  05  DT-TRANS-CODE      PIC X.
                  05  DT-AMOUNT          PIC S9(5)V99 COMP.
      @EOF
```

The proper sequence for using the sample data item index in an I-QU PLUS-1 application program is as follows:

```
      INVOKE REQMTS-SUB IN MFG-SCHEMA FOR MT NX
      INDEX MFG*MFGQINDEX.
      DEFINE F DLY-TRANS SEQ 16,200
      DEFINE RA DLY-TRANS AFTER PART-MSTR
      ...
```

The file name used in the DEFINE F directive must match the file name used when the file was defined in QINDEX in order for I-QU PLUS-1 to detect record area offsets automatically.

Once the application has been defined as shown here, any data item in any DMS 2200 record or non-DMS file may be referenced without qualification. The referencing of DMS 2200 records or non-DMS files without qualification is possible because each item definition has been linked to its DMS 2200 record or file name internally by QINDEX. When an item is referenced in I-QU PLUS-1, the command editor automatically looks up the item's record or file entry to determine if the record or file has been relocated within the RDA via a DEFINE RA directive. If an item name appears within more than one record or file within the application, it will be necessary to qualify its reference.

18.5 Example

The following example shows the executions of QINDEX to build the data item index file used in the examples shown throughout this guide. The file contains both DMS and PCIOS data item definitions. In the execution, a FILE directive is supplied so that the I-QU PLUS-1 DEFINE RA (record area) directive can be used in the generated DBM code for InfoQuest.

```
      @DELETE,C MKTG*CO-INDE.
      FURPUR 31R5 (990611 1158:59) 1999 Sep 23 Thu 1545:20
      END DELETE.
      @ASG,UPV MKTG*CO-INDEX.
      I:002333 ASG complete.
      @QINDXA,LI MKTG*CO-INDEX.
      QINDEX 6R5-0826 (Release 6R5) (990826 1425:41) 1999 Sep 23 Thu 1545:21
      (C) Copyright 1985-1997 by KMSYS Worldwide, Inc.  All Rights reserved.
      This program licensed for use by KMSYS Worldwide, INC.
      File:MKTG*????????CO-INDEX.???? already assigned
      ***Index will be INITIALIZED.
      (00001)INVOKE DEMOSUB IN DEMOSCH FILE uds$$src*schabs
      (00002)RECORD CUST-ADDR-REC
      (00003)      *** THE FOLLOWING REDEFINES TO CA-LOCKEY FIELD ***
      (00004)       01  CUST-ADDR-REC-RE-DEF.
      (00005)           05  CA-LOCKEY-REDEF.
      (00006)               10  CA-STATE           PIC XX.
      (00007)               10  CA-CITY            PIC X(18).
```

```
(00008)                10  CA-ACCOUNT        PIC 9(9).
(00009)FILE ORDERFILE
(00010)       01  ORDER-RECORD.
(00011)      ** Secondary key, duplicates allowed **
(00012)           05  OR-CUSTKEY.
(00013)               10  OR-CUSTDIV           PIC 9.
(00014)               10  OR-CUSTNUM           PIC X(5).
(00015)               10  OR-CUSTSHIPTO        PIC 999.
(00016)      ** Primary key **
(00017)           05  OR-ORDER-IDENT.
(00018)               10  OR-ORDER-LOC         PIC 99.
(00019)               10  OR-ORDER-KEY         PIC X(7).
(00020)               10  OR-SHIP-LOC          PIC 99.
(00021)           05  OR-ORDER-TYPE-CODE       PIC X.
(00022)           05  OR-PRODLINE-CODE         PIC X.
(00023)           05  OR-ENTRY-DATE.
(00024)               10  OR-ENTRY-MO          PIC XX.
(00025)               10  OR-ENTRY-DA          PIC XX.
(00026)               10  OR-ENTRY-YR          PIC XX.
(00027)           05  OR-BUYER.
(00028)               10  OR-BYPASS            PIC X.
(00029)               10  FILLER               PIC X(10).
(00030)           05  OR-PURCHASE-ORD          PIC X(8).
(00031)           05  OR-REQ-SHIPDATE          PIC X(6).
(00032)           05  OR-SHIP-VIA              PIC X(11).
(00033)           05  OR-CREDIT-HOLD           PIC X.
(00034)           05  OR-HOLD-CODE             PIC X.
(00035)           05  OR-INVOICE-CODE          PIC X.
(00036)           05  OR-BOL-PRT-CODE          PIC X.
(00037)           05  OR-PAY-METHOD-CODE       PIC XXX.
(00038)           05  OR-WORKORD-CODE          PIC X.
(00039)           05  OR-SPECIAL-TERMS         PIC X(20).
(00040)           05  OR-TERMS.
(00041)               10  OR-TERM-CODE         PIC X.
(00042)               10  OR-TERM-PER          PIC V9(4) COMP.
(00043)               10  OR-TERM-DATE-DAYS    PIC 9(6).
(00044)           05  OR-DELETE-FLAG           PIC X.
(00045)           05  OR-INPROCESS-HOLD        PIC X.
(00046)           05  FILLER                   PIC XX.
(00047)           05  OR-ACTUAL-SHIP-DATE.
(00048)               10  OR-SHIP-YR           PIC XX.
(00049)               10  OR-SHIP-MO           PIC XX.
(00050)               10  OR-SHIP-DA           PIC XX.
(00051)           05  OR-PIECES                PIC 9(5) COMP.
(00052)           05  FILLER                   PIC XX.
(00053)           05  OR-WEIGHT                PIC 9(7) COMP.
(00054)           05  OR-SHIP-FEE              PIC 9(5)V99 COMP.
(00055)           05  OR-TOT-CHARGES           PIC 9(6)V99 COMP.
(00056)           05  OR-TOT-CLC               PIC 9(6)V99 COMP.
(00057)           05  OR-DISCOUNT              PIC 9(5)V99 COMP.
(00058)           05  OR-CREDIT-REL-DATE.
(00059)               10  OR-CREL-YR           PIC XX.
(00060)               10  OR-CREL-MO           PIC XX.
(00061)               10  OR-CREL-DA           PIC XX.
(00062)           05 OR-WORKORD-PRT-DATE.
(00063)               10  OR-WKORD-PRT-YR      PIC XX.
(00064)               10  OR-WKORD-PRT-MO      PIC XX.
```

```
(00065)              10  OR-WKORD-PRT-DA      PIC XX.
(00066)          05  OR-STATE-TAX             PIC S9(4)V99 COMP.
(00067)          05  OR-CITY-TAX              PIC S9(4)V99 COMP.
(00068)          05  OR-COUNTY-TAX            PIC S9(4)V99 COMP.
(00069)          05  OR-CREDIT-USERID         PIC X(8).
(00070)          05  OR-NBR-PALLETS           PIC S9(2) COMP.
(00071)          05  OR-PALLET-CHG            PIC S9(3)V99 COMP.
(00072)          05  OR-TOT-PALLET-COST       PIC S9(5)V99 COMP.
(00073)          05  OR-AUTHDLR-CODE          PIC X.
(00074)          05  OR-LINE-COUNT            PIC 9(10) COMP.
(00075)          05  OR-ORDER-LINE-DATA OCCURS 1 TO 50
(CONT.)              DEPENDING ON OR-LINE-COUNT.
(00076)     ***  Order line item data ***
(00077)              10  OR-PRODUCT           PIC X(6).
(00078)              10  OR-TYPE-ORD-CODE     PIC X.
(00079)              10  OR-QUANTITY          PIC S9(5) COMP.
(00080)              10  OR-UNIT-PRICE        PIC 9(5)V99 COMP.
(00081)              10  OR-DESC              PIC X(25).
(00082)              10  OR-LINE-WEIGHT       PIC 9(5)V99 COMP.
(00083)              10  OR-PACKAGE           PIC X(8).
(00084)              10  OR-PRICE-CODE        PIC XX.
(00085)              10  OR-BOL-KEY           PIC 999 COMP.
(00086)              10  OR-TAX-CODE          PIC X.
(00087)              10  OR-REG-CODE          PIC X.
(00088)              10  OR-SHIP-QTY          PIC 9(5) COMP.
(00089)              10  OR-BILL-ONLY-CODE    PIC X.
(00090)              10  OR-PRICE-CHANGE      PIC X.
(00091)              10  OR-LAST-DATE         PIC X(6).
(00092)              10  OR-PRIORITY          PIC X.
(00093)              10  OR-EXCEPTION-SW      PIC X.
(00094)              10  OR-SUB-ITEM          PIC X(6).
(00095)FILE CUSTFILE
(00096)       01  CUSTORMER-MASTER-REC.
(00097)      ** Primary key **
(00098)          05  CM-ACCOUNT               PIC X(9).
(00099)      ** Secondary key, duplicates allowed **
(00100)          05  CM-LOCKEY.
(00101)              10  CM-STATE             PIC XX.
(00102)              10  CM-CITY              PIC X(18).
(00103)          05  CM-CUSTNAME              PIC X(33).
(00104)          05  CM-ADDR1                 PIC X(30).
(00105)          05  CM-ADDR2                 PIC X(30).
(00106)          05  CM-ADDR3                 PIC X(30).
(00107)          05  CM-ZIP                   PIC X(9).
(00108)          05  CM-TELEPHONE.
(00109)              10  CM-AREACODE          PIC XXX.
(00110)              10  CM-EXCHANGE          PIC XXX.
(00111)              10  CM-TELNUM            PIC XXXX.
.....End of QINDEX processing.
**************************************************
*** Date Item Index File List ***
**************************************************
( Subschema names and codes listed )

<<< Schema/Subschema >>>
DEMOSCH     DEMOSUB              SCHEMA/SUBSCHEMA

******* COUNT 0000000001
```

```
<<< Areas >>>
DEMO-ADDR                        AREA CODE: 00001
DEMO-CKEY                        AREA CODE: 00003
DEMO-INDX                        AREA CODE: 00002
DEMO-ORD                         AREA CODE: 00004

******* COUNT 0000000004 remove

<<< Records >>>
CUST-ADDR-REC                    RECORD CODE:00002
CUST-KEY-REC                     RECORD CODE:00001
ORDER-COMMENT-REC                RECORD CODE:00004
ORDER-HEADER-REC                 RECORD CODE:00003
ORDER-LINE-REC                   RECORD CODE:00005

******* COUNT 0000000005

<<< Sets >>>
CUST-KEY-ADDR-SET                SET CODE: 00001
CUST-ORD                         SET CODE: 00004
ORDH-CMT                         SET CODE: 00003
ORDH-LINE                        SET CODE: 00002

******* COUNT 0000000004

<<< Database Datanames >>>
AREA-DEMO-CKEY                   DBDN CODE: 00002
AREA-DEMO-ORD                    DBDN CODE: 00003
SH-MISC-ANAME                    DBDN CODE: 00001

*******  COUNT  0000000003

<<< Files >>>
CUSTFILE                         FILE CODE: 04098
ORDERFILE                        FILE CODE: 04097

*******  COUNT  0000000002

<<< Data Items >>>
IX|ITEM NAME                       |  RDA REF    |DATA TYPE|DEC POS|REC/FILE COD
--|--------------------------------|-------------|---------|-------|------------
18:CA-ACCOUNT                       (00021,00009) UN9                 00002
17:CA-ADDR1                         (00063,00030) A9                  00002
17:CA-ADDR2                         (00093,00030) A9                  00002
17:CA-ADDR3                         (00123,00030) A9                  00002
17:CA-AREACODE                      (00162,00003) A9                  00002
16:CA-CITY                          (00003,00018)                     00002
17:CA-CUSTNAME                      (00030,00033) A9                  00002
17:CA-EXCHANGE                      (00165,00003) A9                  00002
17:CA-LOCKEY                        (00001,00029) A9                  00002
16:CA-LOCKEY-REDEF                  (00001,00029)                     00002
16:CA-STATE                         (00001,00002)                     00002
17:CA-TELEPHONE                     (00162,00010) A9                  00002
17:CA-TELNUM                        (00168,00004) A9                  00002
17:CA-ZIP                           (00153,00009) A9                  00002
00:CK-CHG-TIME                      (00043,00004) UB9                 00001
17:CK-CUST-KEY                      (00001,00009) A9                  00001
17:CK-NAME                          (00010,00033) A9                  00001
16:CM-ACCOUNT                       (00001,00009)                     04098
16:CM-ADDR1                         (00063,00030)                     04098
```

```
16:CM-ADDR2                         (00093,00030)                     04098
16:CM-ADDR3                         (00123,00030)                     04098
16:CM-AREACODE                      (00162,00003)                     04098
16:CM-CITY                          (00012,00018)                     04098
16:CM-CUSTNAME                      (00030,00033)                     04098
16:CM-EXCHANGE                      (00165,00003)                     04098
16:CM-LOCKEY                        (00010,00020)                     04098
16:CM-STATE                         (00010,00002)                     04098
16:CM-TELEPHONE                     (00162,00010)                     04098
16:CM-TELNUM                        (00168,00004)                     04098
16:CM-ZIP                           (00153,00009)                     04098
17:CUST-ADDR-REC                    (00001,00180) A9                  00002
16:CUST-ADDR-REC-RE-DEF             (00001,00029)                     00002
17:CUST-KEY-REC                     (00001,00048) A9                  00001
16:CUSTORMER-MASTER-REC             (00001,00171)                     04098
17:OC-BOL-TYPE                      (00082,00001) A9                  00004
17:OC-COMMENT                       (00001,00080) A9                  00004
17:OC-TYPE                          (00081,00001) A9                  00004
17:OH-ACTUAL-SHIP-DATE              (00106,00006) A9                  00003
17:OH-AUTHDLR-CODE                  (00168,00001) A9                  00003
17:OH-BOL-PRT-CODE                  (00068,00001) A9                  00003
17:OH-BUYER                         (00029,00011) A9                  00003
17:OH-BYPASS                        (00029,00001) A9                  00003
04:OH-CITY-TAX                      (00148,00003) SB9         02      00003
04:OH-COUNTY-TAX                    (00151,00003) SB9         02      00003
17:OH-CREDIT-HOLD                   (00065,00001) A9                  00003
17:OH-CREDIT-REL-DATE               (00133,00006) A9                  00003
17:OH-CREDIT-USERID                 (00154,00008) A9                  00003
17:OH-CREL-DA                       (00137,00002) A9                  00003
17:OH-CREL-MO                       (00135,00002) A9                  00003
17:OH-CREL-YR                       (00133,00002) A9                  00003
18:OH-CUSTDIV                       (00001,00001) UN9                 00003
17:OH-CUSTKEY                       (00001,00009) A9                  00003
17:OH-CUSTNUM                       (00002,00005) A9                  00003
18:OH-CUSTSHIPTO                    (00007,00003) UN9                 00003
17:OH-DELETE-FLAG                   (00102,00001) A9                  00003
03:OH-DISCOUNT                      (00130,00003) UB9         02      00003
17:OH-ENTRY-DA                      (00025,00002  A9                  00003
17:OH-ENTRY-DATE                    (00023,00006) A9                  00003
17:OH-ENTRY-MO                      (00023,00002) A9                  00003
17:OH-ENTRY-YR                      (00027,00002) A9                  00003
17:OH-HOLD-CODE                     (00066,00001) A9                  00003
17:OH-INPROCESS-HOLD                (00103,00001) A9                  00003
17:OH-INVOICE-CODE                  (00067,00001) A9                  00003
01:OH-NBR-PALLETS                   (00162,00001) COMP                00003
17:OH-ORDER-IDENT                   (00010,00011) A9                  00003
17:OH-ORDER-KEY                     (00012,00007) A9                  00003
18:OH-ORDER-LOC                     (00010,00002) UN9                 00003
17:OH-ORDER-TYPE-CODE               (00021,00001) A9                  00003
04:OH-PALLET-CHG                    (00163,00002) SB9         02      00003
17:OH-PAY-METHOD-CODE               (00069,00003) A9                  00003
00:OH-PIECES                        (00112,00002) UB9                 00003
17:OH-PRODLINE-CODE                 (00022,00001) A9                  00003
17:OH-PURCHASE-ORD                  (00040,00008) A9                  00003
17:OH-REQ-SHIPDATE                  (00048,00006) A9                  00003
17:OH-SHIP-DA                       (00110,00002) A9                  00003
03:OH-SHIP-FEE                      (00119,00003) UB9         02      00003
```

```
18:OH-SHIP-LOC                     (00019,00002) UN9                  00003
17:OH-SHIP-MO                      (00108,00002) A9                   00003
17:OH-SHIP-VIA                     (00054,00011) A9                   00003
17:OH-SHIP-YR                      (00106,00002) A9                   00003
17:OH-SPECIAL-TERMS                (00073,00020) A9                   00003
04:OH-STATE-TAX                    (00145,00003) SB9         02       00003
17:OH-TERM-CODE                    (00093,00001) A9                   00003
18:OH-TERM-DATE-DAYS               (00096,00006) UN9                  00003
03:OH-TERM-PER                     (00094,00002) UB9         04       00003
17:OH-TERMS                        (00093,00009) A9                   00003
03:OH-TOT-CHARGES                  (00122,00004) UB9         02       00003
03:OH-TOT-CLC                      (00126,00004) UB9         02       00003
04:OH-TOT-PALLET-COST              (00165,00003) SB9         02       00003
00:OH-WEIGHT                       (00116,00003) UB9                  00003
17:OH-WKORD-PRT-DA                 (00143,00002) A9                   00003
17:OH-WKORD-PRT-MO                 (00141,00002) A9                   00003
17:OH-WKORD-PRT-YR                 (00139,00002) A9                   00003
17:OH-WORKORD-CODE                 (00072,00001) A9                   00003
17:OH-WORKORD-PRT-DATE             (00139,00006) A9                   00003
17:OL-BILL-ONLY-CODE               (00057,00001) A9                   00005
00:OL-BOL-KEY                      (00051,00002) UB9                  00005
17:OL-DESC                         (00013,00025) A9                   00005
17:OL-EXCEPTION-SW                 (00066,00001) A9                   00005
17:OL-LAST-DATE                    (00059,00006) A9                   00005
17:OL-PACKAGE                      (00041,00008) A9                   00005
17:OL-PRICE-CHANGE                 (00058,00001) A9                   00005
17:OL-PRICE-CODE                   (00049,00002) A9                   00005
17:OL-PRIORITY                     (00065,00001) A9                   00005
17:OL-PRODUCT                      (00001,00006) A9                   00005
01:OL-QUANTITY                     (00008,00002) COMP                 00005
17:OL-REG-CODE                     (00054,00001) A9                   00005
00:OL-SHIP-QTY                     (00055,00002) UB9                  00005
17:OL-SUB-ITEM                     (00067,00006) A9                   00005
17:OL-TAX-CODE                     (00053,00001) A9                   00005
17:OL-TYPE-ORD-CODE                (00007,00001) A9                   00005
03:OL-UNIT-PRICE                   (00010,00003) UB9         02       00005
03:OL-WEIGHT                       (00038,00003) UB9         02       00005
16:OR-ACTUAL-SHIP-DATE             (00106,00006)                      04097
16:OR-AUTHDLR-CODE                 (00168,00001)                      04097
16:OR-BILL-ONLY-CODE               (00229,00001)                      04097
00:OR-BOL-KEY                      (00223,00002) UB9                  04097
16:OR-BOL-PRT-CODE                 (00068,00001)                      04097
16:OR-BUYER                        (00029,00011)                      04097
16:OR-BYPASS                       (00029,00001)                      04097
01:OR-CITY-TAX                     (00148,00003) COMP        02       04097
01:OR-COUNTY-TAX                   (00151,00003) COMP        02       04097
16:OR-CREDIT-HOLD                  (00065,00001)                      04097
16:OR-CREDIT-REL-DATE              (00133,00006)                      04097
16:OR-CREDIT-USERID                (00154,00008)                      04097
16:OR-CREL-DA                      (00137,00002)                      04097
16:OR-CREL-MO                      (00135,00002)                      04097
16:OR-CREL-YR                      (00133,00002)                      04097
18:OR-CUSTDIV                      (00001,00001) UN9                  04097
16:OR-CUSTKEY                      (00001,00009)                      04097
16:OR-CUSTNUM                      (00002,00005)                      04097
18:OR-CUSTSHIPTO                   (00007,00003) UN9                  04097
16:OR-DELETE-FLAG                  (00102,00001)                      04097
```

```
16:OR-DESC                         (00185,00025)                      04097
00:OR-DISCOUNT                     (00130,00003) UB9         02       04097
16:OR-ENTRY-DA                     (00025,00002)                      04097
16:OR-ENTRY-DATE                   (00023,00006)                      04097
16:OR-ENTRY-MO                     (00023,00002)                      04097
16:OR-ENTRY-YR                     (00027,00002)                      04097
16:OR-EXCEPTION-SW                 (00238,00001)                      04097
16:OR-HOLD-CODE                    (00066,00001)                      04097
16:OR-INPROCESS-HOLD               (00103,00001)                      04097
16:OR-INVOICE-CODE                 (00067,00001)                      04097
16:OR-LAST-DATE                    (00231,00006)                      04097
00:OR-LINE-COUNT                   (00169,00004) UB9                  04097
00:OR-LINE-WEIGHT                  (00210,00003) UB9         02       04097
01:OR-NBR-PALLETS                  (00162,00001) COMP                 04097
16:OR-ORDER-IDENT                  (00010,00011)                      04097
16:OR-ORDER-KEY                    (00012,00007)                      04097
16:OR-ORDER-LINE-DATA              (00173,00072)                      04097
18:OR-ORDER-LOC                    (00010,00002) UN9                  04097
16:OR-ORDER-TYPE-CODE              (00021,00001)                      04097
16:OR-PACKAGE                      (00213,00008)                      04097
01:OR-PALLET-CHG                   (00163,00002) COMP        02       04097
16:OR-PAY-METHOD-CODE              (00069,00003)                      04097
00:OR-PIECES                       (00112,00002) UB9                  04097
16:OR-PRICE-CHANGE                 (00230,00001)                      04097
16:OR-PRICE-CODE                   (00221,00002)                      04097
16:OR-PRIORITY                     (00237,00001)                      04097
16:OR-PRODLINE-CODE                (00022,00001)                      04097
16:OR-PRODUCT                      (00173,00006)                      04097
16:OR-PURCHASE-ORD                 (00040,00008)                      04097
01:OR-QUANTITY                     (00180,00002) COMP                 04097
16:OR-REG-CODE                     (00226,00001)                      04097
16:OR-REQ-SHIPDATE                 (00048,00006)                      04097
16:OR-SHIP-DA                      (00110,00002)                      04097
00:OR-SHIP-FEE                     (00119,00003) UB9         02       04097
18:OR-SHIP-LOC                     (00019,00002) UN9                  04097
16:OR-SHIP-MO                      (00108,00002)                      04097
00:OR-SHIP-QTY                     (00227,00002) UB9                  04097
16:OR-SHIP-VIA                     (00054,00011)                      04097
16:OR-SHIP-YR                      (00106,00002)                      04097
16:OR-SPECIAL-TERMS                (00073,00020)                      04097
01:OR-STATE-TAX                    (00145,00003) COMP        02       04097
16:OR-SUB-ITEM                     (00239,00006)                      04097
16:OR-TAX-CODE                     (00225,00001)                      04097
16:OR-TERM-CODE                    (00093,00001)                      04097
18:OR-TERM-DATE-DAYS               (00096,00006) UN9                  04097
00:OR-TERM-PER                     (00094,00002) UB9         04       04097
16:OR-TERMS                        (00093,00009)                      04097
00:OR-TOT-CHARGES                  (00122,00004) UB9         02       04097
00:OR-TOT-CLC                      (00126,00004) UB9         02       04097
01:OR-TOT-PALLET-COST              (00165,00003) COMP        02       04097
16:OR-TYPE-ORD-CODE                (00179,00001)                      04097
00:OR-UNIT-PRICE                   (00182,00003) UB9         02       04097
00:OR-WEIGHT                       (00116,00003) UB9                  04097
16:OR-WKORD-PRT-DA                 (00143,00002)                      04097
16:OR-WKORD-PRT-MO                 (00141,00002)                      04097
16:OR-WKORD-PRT-YR                 (00139,00002)                      04097
16:OR-WORKORD-CODE                 (00072,00001)                      04097
```

```
16:OR-WORKORD-PRT-DATE           (00139,00006)                      04097
17:ORDER-COMMENT-REC             (00001,00084) A9                   00004
17:ORDER-HEADER-REC              (00001,00172) A9                   00003
17:ORDER-LINE-REC                (00001,00072) A9                   00005
16:ORDER-RECORD                  (00001,03772)                      04097

******* COUNT 0000000195
**** End of Data Item Index List ****
```

Index

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

Name ____

Company ____

Address ____

____

City ______________________________ State/Province _________

Country ______________________________ Zip/Mail Code __________

Document Name ______________________________ OS Level ______________

KMSYS Worldwide Product ______________________________ Level ________________

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

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

What information did you expect to find that was omitted?

____

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

____

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

____

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

____

Is the documentation intimidating? ❑ Yes ❑ No.

____

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

____

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

____

(Additional Comments)

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

KMSYS WORLDWIDE, INC

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

Attn: Technical Documentation Section