HP RPG/XL Programmer's Guide

HP 3000 MPE/iX Computer Systems

Edition 1

Manufacturing Part Number: 30318-90001
E1288

U.S.A. December 1988

Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for direct, indirect, special, incidental or consequential damages in connection with the furnishing or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Restricted Rights Legend

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. Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19 (c) (1,2).

Acknowledgments

UNIX is a registered trademark of The Open Group.

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

HP RPG/XL Programmer's Guide

Product 900 Series HP 3000 Computers

HP RPG/XL Programmer's Guide

Printed in U.S.A.
HP Part No. 30318-90001
Edition E1288
Printed Dec 1988

Notice

Printing History

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The dates on the title page change only when a new edition or a new update is published. No information is incorporated into a reprinting unless it appears as a prior update; the edition does not change when an update is incorporated.

The software code printed alongside the data indicates the version level of the software product at the time the manual or update was issued. Many product updates and fixes do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one to one correspondence between product updates and manual updates.

First Edition December 1988 30318A.00.00

Preface

The *HP RPG/XL Programmer's Guide* explains how to perform many of the common programming functions in RPG. It does not include an exhaustive discussion of these tasks, but covers the commonly-used ones and the ones that use features unique to Hewlett-Packard computers.

This manual is directed to experienced RPG programmers, who may or may not be familiar with Hewlett-Packard computers. The manual discusses the language features available with the MPE XL operating system.

This manual is organized as follows:

Chapter 1 Discusses the HP RPG logic cycle.

Chapter 2 Tells you how to enter an RPG program at the terminal.

Chapter 3 Explains how to use disc files in RPG programs.

Chapter 4 Explains how to use the terminal in RPG programs.

Chapter 5 Discusses tables, arrays, data structures and subroutines.

Chapter 6 Explains how to compile an RPG program and how to enter compiler options. It also explains the compiler listings.

Chapter 7 Explains how to execute and debug an RPG program.

Chapter 8 Tells how RPG programs can exchange information and use operating system facilities such as intrinsics.

Chapter 9 Gives tips on writing efficient RPG programs.

Appendix A Gives instructions on converting IBM RPG programs to the HP 3000. It also explains how to migrate RPG programs from the (HP) MPE V operating system to MPE XL.

Related Documentation

Refer to the following documents for further information on features available in the RPG programming language:

HP RPG/XL Reference Manual (30318-90003) - This manual includes a complete discussion of the language elements of RPG.

RPG Utilities Reference Manual (32104-90006) - This manual explains how to use these RPG utilities: XSORT, RISE, SIGEDITOR and RPGINIT.

Data Entry and Forms Management System VPLUS/3000 (32209-90001) - This manual includes a complete discussion about the screen management software product, VPLUS. You can use this product within RPG programs when using a terminal.

EDIT/3000 Reference Manual (03000-90012) - This manual explains how to use the text processor software product, EDITOR. *KSAM/3000 Reference Manual* (30000-90079) - This manual explains how to use KSAM disc files and how to access them.

KSAM/3000 Reference Manual (30000-90079) - This manual explains how to use KSAM disc files and how to access them.

TurboIMAGE/XL Database Management System (30391-90001) - This manual discusses the TurboIMAGE database software product.

MPE XL Intrinsics Reference Manual (32650-90028) - This manual discusses the operating system routines that can be used by external subroutines in

an RPG program.

Native Language Programmer's Guide (32650-90022) - This manual discusses how to create and use Native Language Support message files.

Message Catalogs Programmer's Guide (32650-90021) - This manual discusses how to create and use non-Native Language Support message files.

FCOPY Reference Manual (03000-90064) - This manual explains how to use the FCOPY file utility.

SORT-MERGE/XL Programmer's Guide (32650-90080) - This manual explains how to use the SORT/MERGE file utility.

Accessing Files Programmer's Guide (32650-90017) - This manual discusses the ways MPE XL files can be processed.

MPE XL General User's Reference Manual (32650-90002) - This manual discusses file, group and account structures.

MPE XL Commands Reference Manual (32650-90003) - This manual describes all of the MPE XL commands including FILE.

Example Conventions

Throughout this manual, examples of RPG program code are shown using figures similar to the one below. The first two lines are a ruler to help you quickly see the column positions for the code. The shaded numbers on the left are not sequence numbers. Rather, they are used as reference numbers for the comments which follow the figure. Lines are referenced only to highlight specific concepts. Additionally, some examples show lines containing dots only. Dots indicate that, to clarify examples, code has been omitted.

```
     1         2         3         4         5         6         7
    67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
1   C                   RLABL           IN20
2   C                   RLABL           PNAME
    C                     .
    C                     .
    C                     .
3   C                   EXIT SUB01
```

Figure 5-16. Using RLABL to Pass Information to an External Subroutine

Comments

1 This line makes indicator 20 available to an external subroutine.

Columns 43-48 specifies that indicator 20 is passed to the external subroutine. (Prefix indicator names by IN.)

2 This line makes the field, PNAME, available to an external subroutine.

3 This line executes the external subroutine, SUB01.

Chapter 1 How RPG Works

RPG programs can be thought of as having two different authors. You are the primary author. You enter the RPG program *specifications* that describe the input and output data and the calculations to perform on that data. For example, if you want to print a total line on a report, you must describe the format of the total line and define the calculations that will produce it. The RPG compiler is the second author of RPG programs. When you compile an RPG program, the compiler determines when, and the order in which your specifications are executed. Thus, it supplies the logic framework of the RPG program. This framework, the RPG *logic cycle*, reads and writes files and prints detail and total lines on reports. The RPG logic cycle notifies you of events that it controls by means of *indicators*. For example, when RPG reads the last record in an input file, it turns on the last-record (LR) indicator. You can then use this indicator in your specifications to direct processing. Conversely, you can use indicators to control certain RPG logic cycle events. For example, you can cause a program to end immediately by turning on the last-record indicator.

The next sections in this chapter go into more detail about specifications, the RPG logic cycle and indicators. Specifications are presented first to give you a clear idea of the types of events that they control. Next, an overview of the RPG-supplied logic is presented in the section, "The Basic RPG Logic Cycle." Indicators are discussed in the section, "RPG Indicators." You can see how indicators provide the communications link between the RPG logic cycle and your specifications. The last section, "More About the RPG Logic Cycle," completely describes each step of the logic cycle. The section discusses all of the specifications and indicators that are used during the cycle.

RPG Specifications

Specifications are the source code for RPG programs. There are different kinds of specifications, each used for a particular function. For example, you use Output Specifications to define and describe data to be printed on a report. The specifications are listed below in the order that you enter them into a program:

Specification	Description
Header (H)	Sets RPG compiler options, certain indicators and the collating sequence.
File Description (F)	Describes the files used in the program.
File Extension (E)	Defines arrays and tables, and gives additional information about certain types of files.
Line Counter (L)	Defines printer page lengths, overflow lines and channel numbers.
Input (I)	Defines input records and data fields used in the program.
Calculation (C)	Defines computations and other operations to be performed.
Output (O)	Defines output records and data fields used in the program.
Array/Table File Name (A)	Names a file which contains a compile-time array or table.

The Basic RPG Logic Cycle

Specifications that you enter in an RPG program are executed in the order determined by the RPG compiler. The RPG compiler fits your specifications into the standard logic framework, producing a complete program.

It is important to understand the RPG-supplied logic to effectively use many features of RPG. The RPG logic cycle has three phases which are repeated for each record that is processed. Figure 1-1 illustrates the three basic phases: record input (input time), total calculations/output (total time), and detail calculations/output (detail time). The word *detail* refers to operations performed on individual input records while *total* refers to the operations performed on the results of previous records.

See the section in this chapter titled "More About the RPG Logic Cycle" for a detailed breakdown of the three phases of the basic RPG logic cycle.

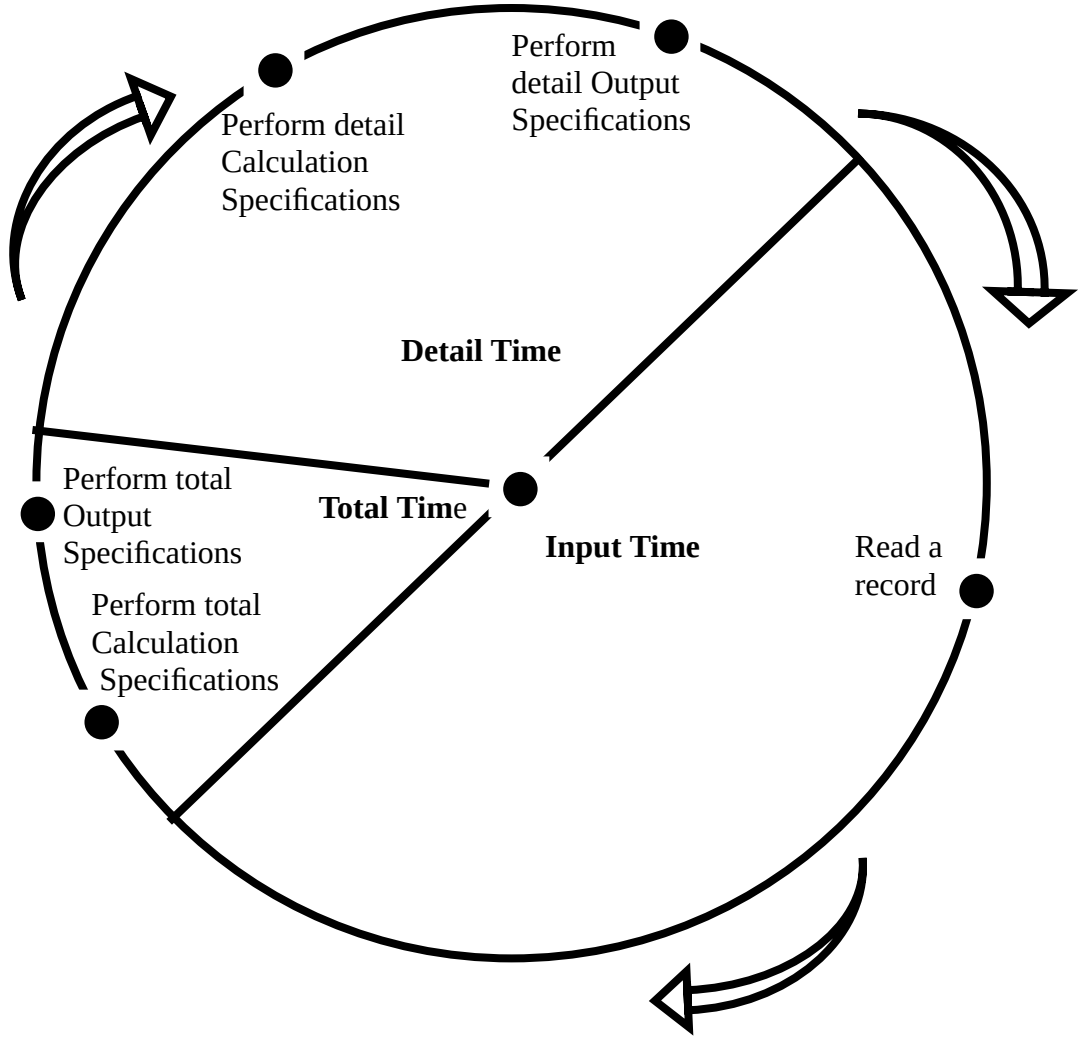


Figure 1-1. The Basic RPG Logic Cycle

RPG Indicators

RPG turns indicators on and off during the RPG logic cycle to indicate that certain processing events have occurred. You can use the settings of the indicators to select the specifications to perform in your program. For instance, control fields trigger total operations such as printing total lines on a report. To define a control field, you assign a control-level indicator (L1-L9) to it on the Input Specification. When this input field changes, RPG turns on the control-level indicator. Then, at the proper time in the logic cycle, RPG performs those specifications conditioned by the indicator you selected. Figure 1-2 shows how control-level indicators fit into the RPG logic cycle.

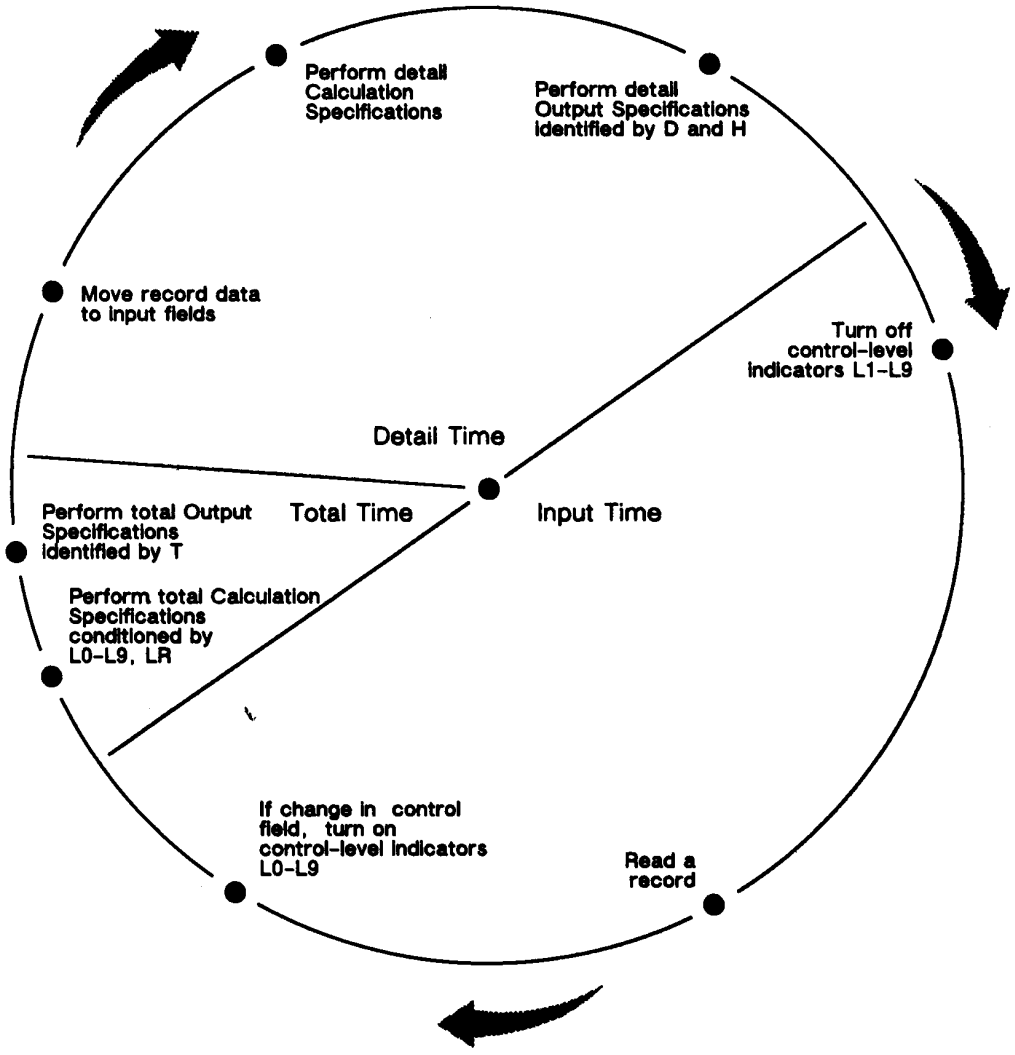


Figure 1-2. Control-Level Indicators and Total/Detail Processing

The control-level indicators, as well as the other RPG indicators, are described below. The last section in this chapter titled "More About the RPG Logic Cycle" explains when these indicators are turned on and off and how they are used in the logic cycle. The RPG indicators are:

Indicator	Description
Command Key (KA-KN, KP-KY)	Lets you control read and write operations on the terminal when using RSI terminal files.

Control-Level (L1-L9)	Lets you sense and perform total operations, such as printing part number totals on a report.
First-Page (1P)	Lets you sense when the first record is processed and lets you perform first record processing.
Function Key (F1-F9)	Lets you use function keys with VPLUS and with the Calculation Specification operations, SET, DSPLY and DSPLM.
General (01-99)	Lets you control operations on Input, Calculation and Output Specifications. Field indicators are a special kind of general indicator used with Input Specifications. They test input fields for plus, minus, zero or blanks. Field indicators are turned on after total-time processing and remain on until after the next total-time cycle.
Halt (H1-H9)	Stops an RPG program at the end of the current logic cycle. You can stop an RPG program based on record codes, field and result values. Once the program is stopped, the operator can resume it by entering an appropriate error response. To avoid halts, you can enter pre-responses to individual errors using the Header Specification.
Last-Record (LR)	Lets you sense when the last record is processed and lets you perform last record processing.
Overflow (OA-OG, OV)	Signals the logical end of the printed page. Overflow indicators are turned on when the overflow line is printed. Sensing overflow lets you perform output operations at the bottom of the current page or at the top of the next page.
Matching Record (MR)	Lets you select the processing order of input records when using more than one input file. This indicator is turned on after total-time processing in the logic cycle is performed.
User (U1-U8)	Allows communication between RPG programs. User indicators are read into the program when it starts and are passed to the next RPG program upon termination. For more information on user indicators see the "Communicating Switches" section of Chapter 8.

More About the RPG Logic Cycle

The following sections in this chapter give details about how the RPG logic cycle works and how indicators are used. Use this section when you have specific questions about how a particular RPG feature is implemented.

The Primary Steps in the RPG Logic Cycle

When an RPG program is first executed, certain initialization and housekeeping operations (described under Step 1 below) are performed. Then the main RPG cycle begins (Steps 2-8). Steps 2 through 8 are repeated for each set of input records. Figure 1-3 diagrams these steps:

Step	Description
1	**Pre-Cycle Initialization and Housekeeping** RPG performs certain preliminary functions once before the main logic cycle begins. During these operations, all data areas are

initialized and the 1P and L0 indicators are turned on. All files are prepared for processing and all preexecution-time tables and arrays are read into memory. First-page alignment is performed for printer files and all output controlled by the 1P indicator is written. A record is read from each primary and secondary input file. The program is now ready to begin the main logic cycle.

2 Heading and Detail-Time Output

This begins the main logic cycle. The RPG program writes all heading and detail records whose conditions are satisfied. (Heading records are identified by an H in the Type Field of the Output Specification; detail records are indicated by a D in the Type Field.) The program tests the status of all halt indicators (and halts if any are turned on), and turns off all record-identifying and control-level indicators that are on. All overflow indicators that were turned on before the last detail calculations (done in Step 8) are turned off. RPG also tests the LR indicator for end-of-program status and transfers to Step 6 if this indicator is on.

NOTE The term "detail-time" used throughout this manual collectively refers to detail-time calculations (occurring in Step 8) followed by detail output (occurring in this step).

---

3 Record Input

If there are no primary or secondary files, control goes to Step 6; otherwise, a record is read from the current file. If it is end-of-file, control skips to Step 4. If the record has an invalid record type or is in the wrong sequence and columns 56-71 of the Header Specification contains a pre-response to the error, the pre-response is performed. If there is an error and no pre-response is entered, the response comes from the operator (session mode) or the job file (job mode).

4 Record Selection

If the program has just one input file, the next record from that file is selected for processing. If the program has more than one input file, the record is selected as follows:

a. If a file is FORCEd (uses the FORCE Calculation Specification operation), the next record from that file is selected.

b. If any input records without matching fields have been read, the highest-priority record among them is chosen. Primary file records have greatest priority, followed by secondary file records in the order that they are entered in the File Description Specifications.

c. If all input records have matching fields and the input sequence is ascending, the lowest-sequenced record is chosen. If all input records have matching fields and the input sequence is descending, the highest-sequenced record is chosen. If the matching fields are the same, the record with the highest priority (see Step 4b.) is selected.

5 Control Break Check

If a control break occurs, the appropriate control-level indicator and all lower-level indicators are turned on.

6 Total-Time Operations

If this is the first record with control level fields, the program skips total-time operations. But, if this is not the

first record with control level fields, the program performs total-time calculations and output (including total-time overflow output). Total-time operations are all those calculations with L0-L9 or LR indicator entries in the Control Level Field (columns 7-8) and output operations of record type T. These operations are done after each control break occurs or after the last input record is read, but before the information on the input record selected in Step 4 is actually made available for processing. The program also sets all resulting indicators as specified. If there are no more records to process in any of the files, the LR indicator is turned on, output tables and arrays are written, all files are closed and the program ends.

7 Data Movement

The program moves the data from the record selected into the input fields for processing. If this is a matching record, it sets the matching-record (MR) indicator. If this is a look-ahead record, the program reads the look-ahead fields and moves them into the input fields. If input chaining is used, the chained file(s) are read, their records identified and their fields moved to the data area.

8 Detail-Time Calculations

The program performs detail-time calculations for the record specified, and returns to Step 2. Detail-time calculations are those calculations not conditioned by control-level indicators in the Control Level Field (columns 7-8.) Notice that these calculations and the detail-time output, described in Step 2, are both done after the information (from the record selected in Step 4) becomes available. They are usually based on information from this record. At this point, the program also turns on resulting indicators used with these calculations.

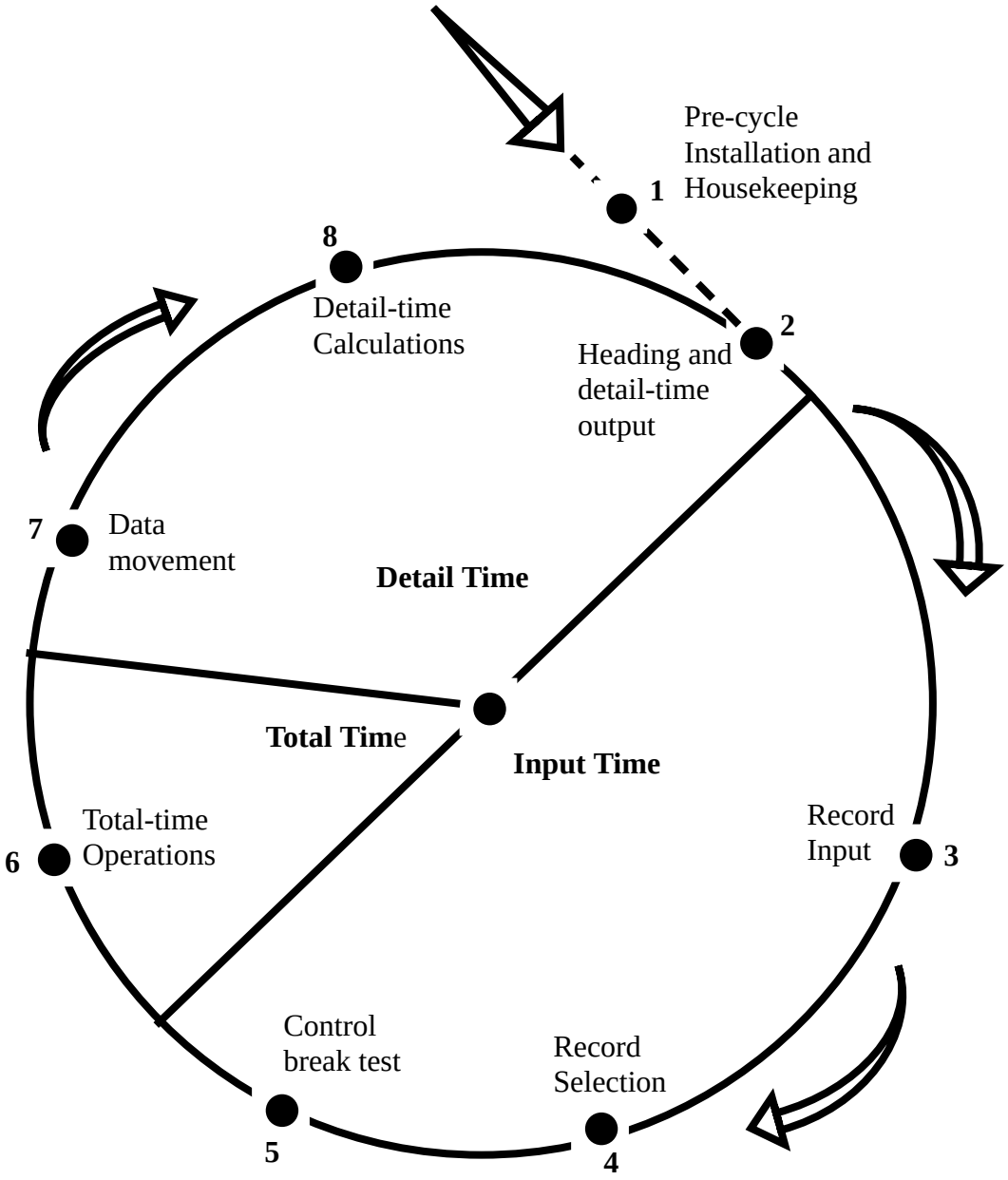


Figure 1-3. The Primary Steps in the RPG Logic Cycle

Substeps in the RPG Logic Cycle

This section describes in detail each step of the RPG logic cycle including when indicators are turned on and off. The substeps listed below expand upon the primary logic steps discussed in the previous section and they correspond to the steps shown on the HP RPG Logic Cycle diagram (Figure 1-4) at the end of this chapter.

Substep	Description
1	**Pre-Cycle Processing**

The operations in this step are performed once. They are not part of the main cycle.

1A All data areas are initialized. This includes compile-time tables and arrays and the user date (UDATE) fields obtained from the system. User indicators (U1-U8) are fetched from the specified source. The Local Data Area (LDA) is initialized with the contents of LDAFILE, if specified. The 1P and L0 indicators are turned on. All files, including TurboIMAGE databases and data sets, are opened unless the program has been suspended by the JCW RPGSUSP (see "End-of-Job Processing", Step E-2B). Preexecution-time tables and arrays are loaded into the data area.

1B If forms alignment is requested, the first output line for each print file conditioned by 1P is printed. The first-page heading records are then written to the specified files.

1C A record is read from each input (primary and secondary) file. The matching and control fields and the record indicator for each file are identified.

2 Heading and Detail-Time Output

This is the beginning of the main logic cycle.

2A RPG writes all heading (except those conditioned by the 1P indicator) and detail lines whose conditions are satisfied. It turns on the appropriate overflow indicator (if used), if the overflow line is reached, and it turns off any overflow indicators that were already processed. It performs overflow processing if Fetch Overflow is specified and the overflow indicator is assigned and turned on.

2B If overflow processing was performed by the previous pass through the cycle or by Fetch Overflow, turn off the overflow indicator. If the overflow indicator was turned on because Fetch Overflow was specified and the overflow line was reached during detail calculations in the previous cycle (or during detail output time in the current cycle), leave the overflow indicator on (you can use the overflow indicator to condition Calculation Specifications during the current cycle).

2C If a halt indicator (H0-H9) is on, control goes to "Run-Time Error Processing", Step E-1.

2D The first-page indicator (1P) and the control-level indicators (L1-L9) are turned off. The L0 indicator is turned on.

2E If this is the first time through the logic cycle, control skips to Step 4A.

2F All record-identifying indicators are turned off.

2G If the last-record (LR) indicator is on, control skips to Step 6B.

3 Record Input

3A If there are no primary or secondary files defined in the program, control skips to Step 6A. (If there are no primary or secondary files, the program loops indefinitely unless the program contains Calculation Specifications that turn LR on.)

3B If the file is not an update or combined file and it contains look-ahead fields, control skips to Step 3G. (A record from the file was read before detail Calculation Specifications were performed in the previous cycle.)

3C If there are spread records defined for an input file and there are more fields to be processed before reading the next record, control skips to Step 4A.

3D A record is read from the last file processed. If this file is retrieved using a Record Address File, the data in the RAF defines the record to be selected.

3E If the file just read is at end-of-file, control skips to Step

4A.

3F If the record just read contains an unidentified record type, or an incorrect record sequence (as specified in the Group Sequence Field of the Input Specification), control skips to "Run-Time Error Processing", Step E-1.

3G If the record just read is a blank spread record, control goes back to Step 3D.

4 Record Selection

4A The next record is selected for processing. If a FORCE Calculation Specification operation was executed during the detail-time calculations in the previous pass through the cycle, the record is selected from the FORCEd file. If no matching-record processing is used, the next record is selected according to the primary/secondary processing order. If matching-record processing is used, the next record is selected according to the matching-record processing order.

4B The program determines if all end-of-file conditions are met (all records in files having an E in the End-of-File Field of the File Description Specification are processed and all matching secondary records are processed). If so, control skips to Step 6B.

4C The record-identifying indicator (associated with the record selected for processing) is turned on.

5 Control Break Check

If a control break occurred for the selected input record, turn on the appropriate control-level indicator and all lower-level indicators. (A control break occurs when the value in the control-level fields of the record differs from the control fields of the previous record.)

6 Total-Time Operations

6A If this is the first time through the logic cycle or this is the first record with control fields, control skips to Step 6E; otherwise, control proceeds to Step 6C.

6B If there are no more records to process in any of the files, all control-level indicators (L1-L9 and LR) are turned on.

6C Total-time calculations are performed. If CHAIN or READ operations are used, the appropriate records are retrieved. The appropriate indicators resulting from total calculations are turned on; overflow conditions resulting from EXCPT output operations are set and Fetch Overflow processing is performed if required by EXCPT output operations.

6D Total-time lines are written. Overflow processing is performed if it was not done in Step 6C.

6E If the LR indicator is on, control skips to "End-of-Job Processing", Step E-2A.

7 Data Movement

7A Overflow lines are written for total-time then detail-time records that are conditioned by overflow indicators but that have not yet been processed by Fetch Overflow.

7B If matching fields are specified and the records match, the MR indicator is turned on; otherwise, it is turned off. When it is turned on, it remains on until the matching record is processed.

7C The data fields from the record selected in Step 4A are extracted and moved into the input data area. Field indicators, if specified, are turned on or off according to the data in the field.

7D If chaining field codes (C1-C9) are entered in the Chaining

Field of one or more Input Specifications, a record is retrieved from each specified file; the appropriate record-identifying indicators are turned on; the data fields are moved to the input data area and field indicators are set.

7E If look-ahead fields are specified, the look-ahead fields are extracted and moved to the look-ahead data area. For combined and update files, the look-ahead fields are extracted from the current record; for all other files, the look-ahead fields are extracted from the next record in the file currently being processed.

8 Detail-Time Calculations

Detail-time calculations are performed. If CHAIN or READ operations are used, the appropriate records are retrieved. The appropriate indicators resulting from detail calculations are turned on; overflow conditions resulting from EXCPT output operations are set and Fetch Overflow processing is performed if required by EXCPT output operations.

E-1 Run-Time Error Processing

E-1A If there is a pre-selected response for this error in the Header Specification, control skips to Step E-1C.

E-1B The response is obtained from the computer operator or from the job file.

E-1C If the response is 0 (CONTINUE), control proceeds to Step E-1D.
If the response is 1 (BYPASS), control skips to Step 2B.
If the response is 2 (REGULAR TERMINATION), control skips to Step 6B.
If the response is 3 (IMMEDIATE TERMINATION), control skips to Step E-2F.
If the response is 4 (REGULAR TERMINATE WITH DUMP), control skips to Step E-1G.
If the response is 5 (IMMEDIATE TERMINATE WITH DUMP), control skips to Step E-1F.

E-1D Each halt indicator (H0-H9) is tested. If an indicator is on, it is reset and control skips back to Step E-1A.

E-1E Processing is resumed at the point in the logic cycle where the error occurred.

E-1F An Error Dump is printed and the program skips to step E-2F.

E-1G An Error Dump is printed and control skips to Step 6B.

E-2 End-of-Job Processing

E-2A Output tables and arrays are written. All files are closed (except when the program is in suspend mode). The settings of the user indicators (U1-U8) update the Job Control Word and the Local Data Area is written back to the LDAFILE, if specified.

E-2B If the JCW RPGSUSP does not equal 1 (the program is not suspended), the program ends.

E-2C Processing is suspended. Files are left open for later access.

E-2D Control is transferred back to the father process.

E-2E If the father process reactivates the suspended program, control begins at Step 1.

E-2F Close files.

Chapter 2 Creating an RPG Program

You can create an RPG program using any editor or word processor that produces a standard ASCII file.

For example, you can use EDITOR. EDITOR comes with your HP system, produces a standard text file and is widely used for other applications. Another popular editor is TDP. Instead of EDITOR or TDP, you may want to use the RPG Interactive System Environment (RISE). RISE is specifically designed for entering RPG programs and lets you use the entire screen. It also provides interactive compiling and debugging features.

EDITOR and RISE are discussed briefly in the next two sections.

Using EDITOR to Create an RPG Program

EDITOR is a general-purpose line editor. You enter and modify lines in your RPG programs one at a time. EDITOR may be useful for entering simple and short RPG programs and for making modifications to those programs. For longer programs, use RISE (discussed in the next section).

For details on using EDITOR, see the *EDIT/3000 Reference Manual*.

Using RISE to Create an RPG Program

RISE is an RPG utility that is ideal for entering moderate to large RPG programs. Besides using it to enter programs, you can use it to interactively compile and debug them as well. Though RISE may take some extra time to learn, it is easier and faster to use than EDITOR.

RISE lets you enter and modify RPG programs in two different modes. In line mode (similar to EDITOR), you work with individual lines. In screen mode, you enter program specifications onto screen forms that resemble the specification sheets themselves. RISE is user-friendly. You can browse specifications, display errors together with their meanings and undo delete operations.

For more information on RISE, see the *RPG Utilities Reference Manual*. To become acquainted with the features of RISE, use the self-paced RISE tour. To start the tour, enter these two commands:

```
:RUN RISE.PUB.SYS
>GET RISETOUR.PUB.SYS
```

Chapter 3 Using Disc Files in an RPG Program

RPG contains language elements that let you process several kinds of disc files. You can use MPE files, Keyed Sequential Access Method (KSAM) files and TurboIMAGE databases. The following list summarizes these files and how you can use them in RPG programs:

File type:	Ways you can access the file:
MPE	Sequentially, randomly.
KSAM	Sequentially, sequentially within key limits, randomly and chronologically.
TurboIMAGE	Sequentially, sequentially within key limits, randomly.

A KSAM file consists of a data file and a key file. Data files contain the actual data records and are maintained in chronological sequence (records are placed in the file in the order that they are added). The key file lets you access the data file records in key sequence. When you create a KSAM file, you specify the fields that are the keys.

A TurboIMAGE database is a collection of files (data sets) whose data field relationships are maintained automatically. You can easily query data in TurboIMAGE databases and there are extensive file security provisions that you can use.

This chapter discusses KSAM files and TurboIMAGE databases and how you access and use them in RPG programs.

Indexed Disc Files (KSAM)

The following list summarizes the advantages and disadvantages of using KSAM files. For in-depth information on KSAM files, see the *KSAM/3000 Reference Manual*.

Advantages:

* Can access records sequentially, chronologically, and by partial or generic key value
* Efficient disc usage; the data portion of the file grows as records are added
* No special utilities are required for loading and unloading data
* Multiple keys
* Key fields can be updated
* Deleted records are retrievable
* Duplicate keys
* KSAM files can be processed by FCOPY, SORT, EDITOR; you can also use the operating system FILE command with them
* Fast updating when processing data sequentially
* RPG Specifications are easy to code
* Fixed or variable length data records

Disadvantages:

* No query facilities

* No automatic maintenance of data relationships
* Extra memory usage when several users access the same file simultaneously

Creating a KSAM Disc File

There are three ways to create a KSAM disc file:

Use this method:	When:
KSAMUTIL	You want to build an empty KSAM file or when you cannot use an RPG program to build it (RPG programs won't let you specify more than one key field, for instance). (KSAMUTIL is a KSAM utility that creates an empty KSAM file.)
FCOPY	You want to create an empty KSAM file or when you want to copy all or part of a KSAM or MPE file to an existing KSAM file. (FCOPY is a general-purpose system command that creates and copies files.)
RPG program	You want to create a new KSAM file and load it with data at the same time.

When you create a KSAM file, you specify the record keys (key fields) that it contains. The first key that you enter is called the *primary* key. The second and successive keys that you enter are called *secondary* keys. Secondary keys are independent of primary keys; they are not subordinate to them. Secondary keys can be thought of as alternate keys. Both primary and secondary keys are maintained in the same KSAM key file.

Creating KSAM files using KSAMUTIL, FCOPY and RPG programs are discussed in the next three sections.

Creating a KSAM File Using KSAMUTIL. The most flexible method of creating a KSAM file is to use KSAMUTIL. You define the file size and key fields using this utility. KSAMUTIL creates an empty file. It does not load the file with actual data. (Use FCOPY or an RPG program to load data into a file created by KSAMUTIL.)

The following steps tell you how to create a KSAM file using KSAMUTIL. For detailed information about KSAMUTIL commands, see the *KSAM/3000 Reference Manual*.

Follow these steps to create a KSAM file using KSAMUTIL:

1. Enter the following command at the operating system colon prompt (:),

   RUN KSAMUTIL.PUB.SYS

   You see the prompt, >. KSAMUTIL is prompting you to enter information about the KSAM file.

2. Enter a KSAMUTIL BUILD command to create the KSAM file.

   For example, the following command creates a file MASTFL having 256 characters per record. Its key file is MASTFLK. Its primary key is a 5-byte field (KEY=B) starting in position 1 of the record:

   >BUILD MASTFL;REC=-256,1,F,ASCII;KEYFILE=MASTFLK;KEY=B,1,5

3. End KSAMUTIL by typing,

   EXIT

Creating a KSAM File Using FCOPY. Use FCOPY when you want to create an empty KSAM file or when you want to copy all or part of a KSAM or MPE file to an existing KSAM file.

To use FCOPY, enter an FCOPY command at the operating system prompt. (For detailed information on the FCOPY command, see the *FCOPY Reference*

Manual.)

For example, the following command copies 50 records (records 0 through 49) from the KSAM file, TRANSFL, to the KSAM file, MASTFL. MASTFL is a new file and is created along with its key file, MASTFLK.

```
FCOPY FROM=TRANSFL;TO=(MASTFL,MASTFLK);SUBSET=0,49
```

If MASTFL and its key file already exist, the following command can be used to copy the first 50 records in TRANSFL to it.

```
FCOPY FROM=TRANSFL;TO=MASTFL;SUBSET=0,49
```

NOTE You can create an empty KSAM file by specifying SUBSET=0,0 in the FCOPY command. To copy all records to the new file (to duplicate it), omit the SUBSET=option.

Creating a KSAM File Within an RPG Program. You can use an RPG program to create and load data into a single-key KSAM file. You define the KSAM file as an output file and include KSAM entries in the File Description and File Description Continuation Specifications. If the KSAM file does not already exist, RPG creates it using these specifications.

Figure 3-1 shows part of the transaction file used by the program in Figure 3-2 to create a KSAM file. The transaction file, TRANSFL, contains customer charge information that is used to create a master KSAM transaction file, MASTFL. The first field of each record in the transaction file is the customer's identification number and it is the key field for MASTFL. For example, the first customer identification number is 00216. (Records do not have to be sorted by customer identification number before running the RPG program; KSAM automatically orders the records.) Following the customer number in each transaction record is the charge amount field. For example, 00342 ($3.42) is the charge amount for customer 00216.

```
0021600342
0365416514
0720001517
0021601802
0321004532
0045318455
8206603324
5122101088
0321006702
      .
      .
      .
```

Figure 3-1. Creating a KSAM File Within an RPG Program - A Sample Input File TRANSFL

The KSAM file in Figure 3-2 can contain up to 1023 records (this is the default). If your KSAM file is larger, enter a FILE command before running the program (for information on the FILE command, see the *MPE XL Commands Reference Manual*). For example, this FILE command specifies a MASTFL size of 10,000 records:

```
:FILE MASTFL;DISC=10000
```

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
1 FTRANSFL IPE F      80             DISC
2 FMASTFL  O   F     256  5AI     1 DISC
3 F                                          KKEYFL MASTFLK  DC

4 ITRANSFL NS  01
  I                                       1  80 REC

5 OMASTFL  D        01
  O                        REC       80
```

Figure 3-2. Creating a KSAM File Within an RPG Program Comments

Comments

1 This line defines the input customer transaction file, TRANSFL.

2 This line defines the master transaction file, MASTFL.

Column 15 is O to indicate that the file is an output file.

Columns 20-23 are blank to specify that there is one record per block.

Columns 24-27 contain 256 to specify the number of characters per record.

Columns 29-30 contain 5 to specify the number of characters in the key field.

Columns 35-38 contain 1 to specify the starting position of the key field.

Column 66 is blank (not A) to start writing data at the beginning of the output file.

3 This line gives more details about the KSAM file, MASTFL.

Column 53 is K to indicate that this line is a Continuation line.

Columns 54-59 contain the option name, KEYFL.

Columns 60-65 contain the name of the KSAM key file, MASTFLK.

Column 69 contains D to allow duplicate keys (optional).

Column 70 contains C to maintain the chronological order of duplicate keys (optional).

4 This line starts the description of the input record format for TRANSFL.

5 This line starts the description of the output record format for MASTFL.

Reading a KSAM Disc File

There are several ways to read records in a KSAM file. Three methods, sequential, random and chronological, are discussed in this chapter.

Reading a file sequentially means that RPG retrieves records automatically in key sequence. You can process the entire file automatically or you can process a portion of the file. To process part of the file, you give RPG the key value of the first record to access. Records are retrieved starting with that record and proceeding sequentially until the last record (having a certain key value or end-of-file) is processed.

Reading a file randomly means that you supply RPG with the key values for each record to be processed. RPG goes to those records directly, without passing through others first.

Reading a KSAM file chronologically means retrieving records in the sequence in which they were added to the file.

Reading a KSAM File Sequentially. This section explains how to read records in a KSAM file sequentially, starting with the first record in the file and ending with the last. You can access records in sequence by any key field. You can also read records in sequence by a non-key field. To do this, the KSAM file must first be sorted on that field.

The following lines list the KSAMUTIL commands that create the KSAM file used in Figure 3-3 and Figure 3-4. The KSAM file, MASTFL, has two keys. The primary key field (KEY=B,1,4) does not have duplicates. The secondary key field (KEY=B,25,15,,RDUP) can have duplicates but they are not maintained in chronological order.

```
:RUN KSAMUTIL.PUB.SYS
>BUILD MASTFL;REC=-256,4,F,ASCII;DISC=20000,20,4;KEYFILE=MASTFLK;&
>KEY=B,1,4;KEY=B,25,15,,RDUP
```

Reading a KSAM File Sequentially by Key

This section explains how to read an entire KSAM file sequentially by key. You can read a KSAM file in order by any field that was specified as a key when the file was created.

Figure 3-3 shows the File Description Specification that reads the KSAM file MASTFL by its primary key. (The previous section shows how MASTFL is created.) MASTFL is defined as a primary file. If the KSAM file is defined as a demand file, include a Calculation Specification containing the READE operation.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
```

```
1  FMASTFL  IPE F 256 256  4AI     1 DISC
```

Figure 3-3. Reading a KSAM File Sequentially by Key

Comments

1 This line defines the KSAM file, MASTFL.

Columns 29-30 contain 4 to specify the length of the key, DEPT.

Column 31 contains A to specify that the key field is alphanumeric.

Column 32 contains I to specify that this is a KSAM file.

Columns 35-38 contain 1 to specify the starting location of the key field.

Reading a KSAM File Sequentially by a Non-Key Field

This section explains how to read an entire KSAM file sequentially by a non-key field. You can use any field that was not defined as a key when the file was created. (This method of reading KSAM files also applies to MPE files.)

To read a KSAM file sequentially by a non-key field, you must first create a Record Address File (RAF). The RAF contains addresses of the data records in the KSAM file in sorted order. You can use either SORT/3000 or XSORT to create a RAF. XSORT is an RPG utility that is documented in the *RPG Utilities Reference Manual*. SORT/3000 is a general-purpose sort described in the *SORT-MERGE/XL Programmer's Guide*. Once you create a RAF, you can use it in an RPG program to retrieve the KSAM records in sorted order.

The following lines show how to enter XSORT parameters to order an employee file, MASTFL, by employee zip code. (When MASTFL was created, zip code was not specified as a key field.) The zip code field occupies positions 60 through 64 in the KSAM records. The H Specification of XSORT directs XSORT to create an Address Output (*ADDROUT*) file. ADDROUT files are a special type of RAF containing sorted record addresses.

```
:FILE XSORTIN=MASTFL
:FILE XSORTOUT=RAFFILE;SAVE
:RUN XSORT.PUB.SYS
     HSORTA     5A
     FNC  60  64
:EOD
```

To process the ADDROUT file created by XSORT above, enter File Description and File Extension Specifications as shown in Figure 3-4.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
1  FRAFFILE IRE F       4  4 T      EDISC
2  FMASTFL  IP  F     256R  I        DISC

3  E    RAFFILE MASTFL
```

Figure 3-4. Reading a KSAM File Sequentially by a Non-Key Field

Comments

1 This line defines the RAF, RAFFILE.

Column 16 contains an R to indicate that RAFFILE contains record addresses.

Column 17 is E to specify that reading continue to the end of the file.

Columns 24-27 specify the record length, 4. (XSORT always creates ADDROUT records that are four bytes long.)

Columns 29-30 contain the key field length, 4.

Column 32 is T to specify that this is an ADDROUT file.

Column 39 is E to indicate that there is a File Extension Specification for this file.

2 This line defines the KSAM file, MASTFL.

Column 28 is R to indicate that MASTFL is accessed randomly.

Column 31 is I to specify that records in MASTFL are accessed by their record addresses.

3 This line specifies that RAFFILE contains the record addresses for accessing records in MASTFL.

Reading a KSAM File Sequentially Within Key Limits. When you need to process a set of records in a KSAM file that have continuous key values, you can process that file sequentially within key limits. Reading within key limits may be used to process a range of part numbers (15000-15999) in an inventory file, for example.

To read sequentially within key limits, specify the key value for the first record then read the KSAM file sequentially until all records with that key have been processed.

The following lines list the KSAMUTIL commands that create the KSAM file used in Figure 3-5, through Figure 3-8. The KSAM file, MASTFL, has two keys. The primary key field (KEY=B,1,4) cannot have duplicates. The secondary key field (KEY=B,25,15,,RDUP) can have duplicates.

```
:RUN KSAMUTIL.PUB.SYS
>BUILD MASTFL;REC=-256,4,F,ASCII;DISC=20000,20,4;KEYFILE=MASTFLK;&
>KEY=B,1,4;KEY=B,25,15,,RDUP
```

Supplying Full Key Values (Method 1)

Figure 3-5 shows how to read all records in a KSAM file having a specific key value. The KSAM file, MASTFL, is read by its secondary key field, LNAME (positions 25-39). In this example, a user enters a specific last name from the terminal. The program reads all records in the file with this last name. When there are no more records for the name, the program prompts the user to enter another name.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
                                                                                
1    FMASTFL  IF  F 256 256L15AI    25 DISC

     IMASTFL  NS  01
     I                                          25  40 NAMEL

     C           "LST-NAME"DSPLY          LNAME  15        NAME PROMPT
2    C           LNAME     SETLLMASTFL
     C           RLOOP     TAG
3    C           LNAME     READEMASTFL                    20(20=NO MATCH)
     C*  .... (do conditioned output)
     C  N20      NAMEL     DSPLY
     C  N20                GOTO RLOOP
```

Figure 3-5. Reading a KSAM File Sequentially Within Key Limits - Supplying Full Key Values

Comments

1 This line defines the KSAM file, MASTFL.

Column 16 is D to indicate that MASTFL is a demand file.

Column 28 is L to specify that MASTFL will be processed within key limits.

Columns 29-30 contain 15 to specify the length of the secondary key, LNAME.

Column 31 contains A to specify that the secondary key is an alphanumeric key.

Column 32 contains I to specify that this is a KSAM file.

Columns 35-38 contain 25 to specify the starting location of the key field.

2 This line specifies the starting secondary key value for reading MASTFL.

Columns 28-32 are SETLL to position the file pointer to the first record having a key equal to the value placed in the LNAME field.

3 This line reads the next record in last name sequence in MASTFL (it is included in a loop that processes only those records whose last name is equal to LNAME.)

Columns 28-32 are READE to specify the Read Equal key operation.

Supplying Full Key Values (Method 2)

If you know the key values of records to be accessed in a KSAM file, you can place them into a RAF and let RPG use the RAF to access records in the KSAM file. Figure 3-6 shows ranges of department numbers (the primary key) to be accessed in a KSAM file. The ranges are: 0005-0100, 1200-1280, 2000-2475 and 5000-5999. (The department numbers are shown exactly as they are entered into the RAF.) You can create a RAF using any

standard text editor, for example EDITOR. Enter key ranges starting with position one. Enter the lower key value first followed immediately by the upper key value. (See the *HP RPG Reference Manual* for details on creating a RAF.)

```
|                                  |
|     00050100                     |
|     12001280                     |
|     20002475                     |
|     50005999                     |
|                                  |
|                                  |
```

Figure 3-6. Reading a KSAM File Sequentially Within Key Limits - RAF Entries

To process a KSAM file sequentially using a RAF, enter both a File Description and a File Extension Specification in your program similar to those shown in Figure 3-7. Once the program in Figure 3-7 finds a record, it displays the first 20 characters in it.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
                                                                                
1  FRAFFILE IRE F      80  4        EDISC
2  FMASTFL  IP  F     256L 4AI     1 DISC
   FDISPLAY O   F      80            $STDLST

3  E    RAFFILE MASTFL

   IMASTFL  NS  01
   I                                        1  20 FLD01

   ODISPLAY D        01
   O                        FLD01     20
```

Figure 3-7. Reading a KSAM File Sequentially Within Key Limits - Using a RAF

1 This line defines the RAF, RAFFILE.

Column 16 contains R to indicate that RAFFILE is a RAF.

Column 17 contains E to specify that the program will not end until all records in the RAF are processed.

Column 30 is 4 to specify the length of key values in RAFFILE.

2 This line defines the KSAM file, MASTFL.

Column 28 is L to indicate that MASTFL is processed within limits.

Column 32 contains I to indicate that MASTFL is a KSAM file.

3 This line specifies that RAFFILE contains the key values for records to be processed in MASTFL.

Supplying Partial Key Values

Figure 3-8 shows how to read all records in a KSAM file having a range of key values. All records in the KSAM file, MASTFL, are read that have last names starting with a specific character (for example "A"). A user at a terminal enters the letter and the program reads the first record in alphabetical sequence that starts with the letter. When there are no more records in the file, the user is prompted to enter another letter. The KSAM file, MASTFL, is created using the KSAMUTIL commands shown at the beginning of this section.

```
      1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

1  FMASTFL  ID  F 256 256L15AI    25 DISC

     IMASTFL  NS  01
     I                                              1   4 ACCTNO
     I                                              5   6 CODE
     I                                              7  23 FNAME
     I                                             24  24 MINIT
     I                                             25  25 CHAR1
     I                                             25  39 LNAME
     I                                                 .
     I                                                 .
     I                                                 .

     C            RNAME     TAG
     C            "LNAME-1" DSPLY          NCHAR   1
     C                        .
     C                        .
     C                        .
2    C            NCHAR     SETLLMASTFL
     C            RLOOP     TAG
3    C                      READ MASTFL                    20(20=EOF)
     C   20                 GOTO RNAME
     C            CHAR1     COMP NCHAR                     21
     C  N21                 GOTO RNAME
     C            LNAME     DSPLY
     C                        .
     C                        .
     C                        .
     C                      GOTO RLOOP
```

Figure 3-8. Reading a KSAM File Sequentially Within Key Limits - Supplying Partial Key Values

Comments

1 This line defines the KSAM file, MASTFL.

Column 16 is D to indicate that MASTFL is a demand file.

Column 28 is L to specify that MASTFL will be processed within key limits.

Columns 29-30 contain 15 to specify the length of the secondary key (last name).

Column 31 contains A to specify that the secondary key is an alphanumeric key.

Column 32 contains I to specify that this is a KSAM file.

2 This line specifies the starting secondary key value for reading MASTFL.

Columns 28-32 are SETLL to position the file pointer to the first record having a key equal to the value placed in the NCHAR field.

3 This line reads the next record in MASTFL. READ is included in a loop that processes only those records whose last name starts with a specific character.

Columns 28-32 are READ to specify the Read key operation.

Reading a KSAM File Randomly. This section explains how to randomly access records in a KSAM file by one of its key fields. There are two ways to do this. The first method lets you supply the key value dynamically in your RPG program. Use it when the key values are computed by the program or when the program is run interactively. The second method lets you enter the key values ahead of time into a RAF. RPG accesses records in the KSAM file that match key values in the RAF. Use the RAF method when you know what the key values are or when they can be generated automatically by other programs. Examples of both of these methods are included in the next two sections.

The following lines show how to use KSAMUTIL to create a KSAM file with one (primary) key. The KSAM file is called MASTFL and duplicate keys are not allowed:

```
:RUN KSAMUTIL.PUB.SYS
>BUILD MASTFL;REC=-256,4,F,ASCII;DISC=20000,20,4;KEYFILE=MASTFLK;&
>KEY=B,1,4
```

Specifying the Key Dynamically

If the records that you need to access in the KSAM file vary from one program run to the next, you probably need to supply the record keys dynamically. That is, you must retrieve or calculate them in the RPG program before reading the KSAM file.

To process a KSAM file randomly by key, enter both a File Description and a Calculation Specification similar to those shown in Figure 3-9. See the KSAMUTIL commands in the previous section, "Reading a KSAM File Randomly" for information on how this KSAM file is created.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
                                                                                
1    FMASTFL  IC  F 256 256R 4AI     1 DISC

     IMASTFL  NS  01
     I                                          1  70 RECRD

     C           "DEPT-NO" DSPLY          DEPT    4
2    C           DEPT      CHAINMASTFL                60
     C           RECRD     DSPLY
```

Figure 3-9. Reading a KSAM File Randomly by Key

Comments

1 This line defines the KSAM file, MASTFL.

Column 16 contains C to indicate that the KSAM file is accessed in chained fashion.

Column 28 is R to specify random processing.

Columns 29-30 contain 4 to specify the key length.

Column 31 contains A to specify that the key is alphanumeric.

Column 32 contains I to specify that this is a KSAM file.

2 This line reads MASTFL by primary key.

Columns 18-21 contain the name of the key field, DEPT.

Columns 28-32 contain the word CHAIN to read the file in a chained fashion during the calculation portion of the logic cycle.

Columns 54-55 contain the resulting indicator (60) that is turned on when no record can be found for the value in DEPT.

Specifying the Keys Using a RAF

If you need to process KSAM records randomly by key and you know (before running a program) what the key values are, use a RAF to hold them. RPG will then use the key values in the RAF to access records in the KSAM file. RAFs let you change key values without modifying and recompiling the RPG program.

You can create a RAF using any standard text editor, for example EDITOR. Enter key values starting with position one. You can enter more than one key on each line, if you wish, but there must be no intervening spaces. (See the *HP RPG Reference Manual* for details on creating a RAF.)

Figure 3-10 and Figure 3-11 show how to read a KSAM file randomly using a RAF. Figure 3-10 lists the key values in the RAF and Figure 3-11 gives the RPG program that processes the RAF. The keys in the RAF are four digits long. They are: 0006, 0010, 0028, 0012, 0013, 0016, 0020, 0036, 0040, 0026, 0011, 0029, 0030 and 0080.

```
|                                  |
|     00060010002800120013         |
|     00160020                     |
|     00360040                     |
|     0026001100290030             |
|     0080                         |
|                                  |
```

Figure 3-10. Reading a KSAM File Randomly by Primary Key - RAF Entries

The KSAMUTIL entries to create a file that can be accessed randomly is shown in the previous section "Reading a KSAM File Randomly". To process a KSAM file randomly using a RAF, enter both a File Description and a File Extension Specification in your program similar to those shown in Figure 3-11. Once the program in Figure 3-11 finds a record, it displays the first 20 characters in it.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
                                                                                
1  FRAFFILE IRE F      80  4         EDISC
2  FMASTFL  IP  F     256R 4AI     1 DISC
   FDISPLAY O   F      80            $STDLST

3  E    RAFFILE MASTFL

   IMASTFL  NS  01
   I                                        1  20 FLD01

   ODISPLAY D        01
   O                       FLD01     20
```

Figure 3-11. Reading a KSAM File Randomly by Primary Key - Using a RAF

Comments

1 This line defines the RAF, RAFFILE.

Column 16 contains R to indicate that RAFFILE is a RAF.

Column 17 contains E to specify that the program will not end until all records in the RAF are processed.

Column 30 is 4 to specify the length of key values in RAFFILE.

2 This line defines the KSAM file, MASTFL.

Column 28 is R to indicate that MASTFL is processed randomly.

Column 32 contains I to indicate that MASTFL is a KSAM file.

3 This line specifies that RAFFILE contains the key values for records to be processed in MASTFL.

Reading a KSAM File Randomly and Sequentially Using the Same Key. This section explains how to read a KSAM file both randomly and sequentially using the same key field. Use this method when you want to process the

file as a CHAINed file and as a demand file in the same program.

To read randomly and sequentially by the same key define the file as a full procedural file in the File Description Specification. Figure 3-12 show how to read a KSAM file randomly and sequentially using the key field, DEPT. A department number is entered by a user from the terminal. The program randomly reads the KSAM file, MASTFL, to access that department. Once the department record is read, the program reads subsequent records sequentially until no more records for that department are found.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
                                                                                
1  FMASTFL  IF  F 256 256R 4AI     1 DISC

   IMASTFL  NS  01
   I                                            1  70 RECRD

   C           "DEPT-CH" DSPLY          DEPT    4
2  C           DEPT      CHAINMASTFL                60
   C  N60      RECRD     DSPLY
   C           "DEPT-RD" DSPLY          DEPT    4
3  C           DEPT      SETLLMASTFL
4  C           RLOOP     TAG
5  C           DEPT      READEMASTFL                    20(20=NO MATCH)
   C  N20      RECRD     DSPLY
   C  N20                GOTO RLOOP
```

Figure 3-12. Reading a KSAM File Randomly and Sequentially - Using Full Procedural Files

Comments

1 This line defines the KSAM file, MASTFL.

Column 16 is F to indicate that the file is a full procedural file.

Column 32 is I to specify that this is a KSAM file.

2 This line reads MASTFL randomly by the department number field, DEPT.

Columns 28-32 are CHAIN to specify that chained random processing be performed for MASTFL. This key is DEPT.

3 This line starts the code that processes MASTFL sequentially by key.

Columns 28-32 are SETLL to set the beginning key value for MASTFL. SETLL sets the file pointer to the key value placed in DEPT.

4 This line starts the loop that reads the KSAM file sequentially by key.

5 This line reads MASTFL until there are no more records having a department equal to DEPT.

Reading a KSAM File Randomly and Sequentially Using Different Keys. This section explains how to access a file both randomly and sequentially using different key fields. There are two ways to do this. For both methods you describe the files in the program as if they were two separate entities. In the first method, enter two FILE commands (*file equations*) before executing the program. This method is easier to use and less likely to result in access conflicts than the second method. In the second method, enter File Description Continuation lines for each file, equating them to the same physical file. The second method treats the two files as one in the program and is more memory-efficient. The next two sections give examples of these two methods.

Using File Equations

To use this method of processing a file randomly and sequentially, enter File Description and Calculation Specifications similar to those shown in Figure 3-13. The file MASTERC is a KSAM file that is processed randomly and MASTERC is a KSAM file that is processed sequentially within key limits.

Before executing the program shown in Figure 3-13 two FILE commands see the *MPE XL Reference Manual* for information on the FILE command) must be entered.

```
:FILE MASTERC=MASTFL
:FILE MASTERD=MASTFL
```

The FILE commands equate both MASTERC and MASTERD to the KSAM file, MASTFL. MASTERC and MASTERD each have their own operating system file number and current record pointer and each is opened and processed independently of the other.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
1    FMASTERC IC  F 256 256R 4AI     1 DISC
2    FMASTERD ID  F 256 256L15AI    25 DISC

3    C           DEPT      CHAINMASTERC              60
     C                        .
     C                        .
     C                        .
4    C           LNAME     SETLLMASTERD
5    C           RLOOP     TAG
6    C           LNAME     READEMASTERD                   20(20=NO MATCH)
     C  N20
        .....(do conditioned output)
     C  N20                 GOTO RLOOP
```

Figure 3-13. Reading a KSAM File Randomly and Sequentially - Using File Equations

Comments

1 This line defines the KSAM file, MASTERC.

Column 16 is C to indicate that the file is processed in a CHAINed fashion.

Column 28 is R to indicate that MASTERC is processed randomly.

Column 32 is I to specify that this is a KSAM file.

2 This line defines the KSAM file, MASTERD.

Column 16 is D to indicate that MASTERD is a demand file.

Column 28 is L to specify that MASTERD will be processed within key limits.

Column 32 contains I to specify that this is a KSAM file.

3 This line reads MASTERC randomly by department number, DEPT.

Columns 28-32 are CHAIN to specify that chained (sequential) processing be performed for the file MASTERC. The key is DEPT.

4 This line starts the code that processes MASTERD sequentially by secondary key.

Columns 28-32 are SETLL to set the beginning secondary key value for MASTERD. SETLL sets the file pointer to the key value placed in LNAME.

5 This line starts the loop that reads the KSAM file sequentially by secondary key.

6 This line reads MASTERD until there are no more records having a last name equal to LNAME.

Using File Description Continuation Lines.

The second way to read a KSAM more than one way in a program is to enter two separate files in the program but include File Description Continuation lines for each of them. These lines equate the two files to the same physical KSAM file (you don't need FILE equations to do this). Figure 3-14 shows how this is done. The file, MASTERC, is processed randomly and the files, MASTERD is processed sequentially within key limits. The File Description Continuation lines contain DSNAME entries that equate the files to the KSAM file, MASTFL. When you use File Description Continuation lines in this way, both files are treated as one. They have the same file number and pointer and they share the same buffer. Therefore, when you use this method, make sure that the read operations restore the file pointer values properly.

```
-       1         2         3         4         5         6         7
    6789012345678901234567890123456789012345678901234567890123456789012345
    ===============================================================================
1   FMASTERC IC  F 256 256R 4AI     1 DISC
2   F                                                KDSNAMEMASTFL
3   FMASTERD ID  F 256 256L15AI    25 DISC
4   F                                                KDSNAMEMASTFL

    IMASTERC NS  01
    I                                          1  70 RECRD
    IMASTERD NS  02
    I                                          1  70 RECRD

    C            "DEPT-CH" DSPLY          DEPT    4
5   C            DEPT      CHAINMASTERC                60
    C  N60       RECRD     DSPLY
    C            "LST-NAME"DSPLY          LNAME   15
6   C            LNAME     SETLLMASTERD
7   C            RLOOP     TAG
8   C            LNAME     READEMASTERD                    20(20=NO MATCH)
    C  N20       RECRD     DSPLY
    C  N20                 GOTO RLOOP
```

Figure 3-14. Reading a KSAM File Randomly and Sequentially - Using File Description Continuation Lines

Comments

1 This line defines the KSAM file, MASTERC.

Column 16 is C to indicate that the file is processed in a CHAINed fashion.

Column 28 is R to indicate that MASTERC is processed randomly.

Column 32 is I to specify that This is a KSAM file.

2 This line equates the KSAM file, MASTERD to MASTFL.

Column 53 is K to indicate that this line is a Continuation line for MASTERC.

Columns 54-58 are DSNAMEMASTFL to specify that the file MASTFL is to be used for MASTERC.

3 This line defines the KSAM file, MASTERD.

Column 16 is D to indicate that MASTERD is a demand file.

Column 28 is L to specify that MASTERD will be processed within key limits.

Column 32 contains I to specify that this is a KSAM file.

4 This line equates the KSAM file, MASTERD, to MASTFL.

Column 53 is K to indicate that this line is a Continuation line for MASTERD.

Columns 54-58 are DSNAMEMASTFL to specify that the file MASTFL

is to be used for MASTERD.

5 This line reads MASTERC randomly by department number, DEPT.

Columns 28-32 are CHAIN to specify that chained (random) processing be performed for the file MASTERC. The key is DEPT.

6 This line starts the code that processes MASTERD sequentially by secondary key.

Columns 28-32 are SETLL to set the beginning secondary key value for MASTERD. SETLL set the file pointer to the key value placed in LNAME.

7 This line starts the loop that reads the KSAM file sequentially by secondary key.

8 This line reads MASTERD until there are no more records having a last name equal to LNAME.

Reading a KSAM File Chronologically. When you read a KSAM file in chronological order, you're reading the KSAM data file from beginning to end. Records are read from the data file in the order that they were originally added to the file. The KSAM file key file is not used.

Figure 3-15 gives an example of how records are stored in a KSAM data file. In this example, only the key values are shown and the key field occupies the first two bytes of the record. Notice that the records are not physically ordered by key value. Records that are inactive (deleted) have a value of FFFF (hexadecimal) in the first two bytes (regardless of where the key is located). When you read chronologically, inactive records are bypassed automatically.

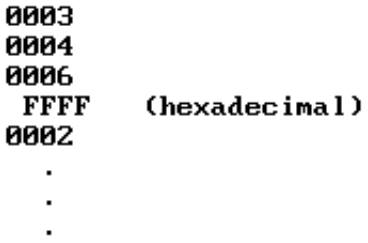


Figure 3-15. The Chronological Order of Records in a KSAM Data File

The following File Description Specification processes a KSAM file in chronological order.

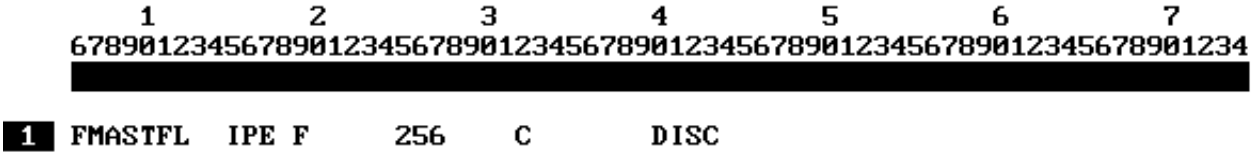


Figure 3-16. Reading a KSAM File in Chronological Order

Comments

1 This line defines the KSAM file, MASTFL.

Column 17 is E to specify that processing proceed to the end of the file.

Column 32 contains C to indicate the file that the file be processed in chronological order.

Updating a KSAM Disc File

This section shows how to randomly update a KSAM file using its primary key.

Figure 3-17 lists a program that updates a customer master file (MASTFL) for a large department store. The master file is updated by a daily sales transaction file (TRANSFL). Each sales transaction contains an identification code that is used to access the customer's record in the customer master file. Each transaction also contains the amount charged by the customer. This amount is used to update the customer's account balance.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-----------------------------------------------------------------------------------------

1    FTRANSFL IP  F      80             DISC
2    FMASTFL  UC  F     256R 4AI     1 DISC
3    FMSGFL   O   F     132            LP

4    ITRANSFL AA  01
     I                                          1   4 IDNO
     I                                         11  162CHARGE
5    IMASTFL  BB  02  05 CA
     I                                          1   4 ID
     I                                          6  20 NAME
     I                                         21  302CURBAL
     I                                         36  50 STREET
     I                                         51  60 CITY
     I                                         61  65 STATE
6    I        CC  03  05 CI
     I                                          1   4 IDNO

7    C           IDNO      CHAINMASTFL              60
8    C   02                ADD  CHARGE    CURBAL

9    OMASTFL  D        01 02
10   O                          CURBAL    30
11   OMSGFL   D  2     01 03
     O                          IDNO      10
12   O                                    34 "DELETED/OBSOLETE RECORD"
13   O        D  2     01 60
     O                          IDNO      10
14   O                                    25 "MISSING RECORD"
```

Figure 3-17. Updating a KSAM File Randomly by Primary Key

Comments

1 This line defines the sales transaction file, TRANSFL. This file is an MPE file.

2 This line defines the KSAM customer master file, MASTFL.

Columns 7-14 contain the name of the customer master file, MASTFL.

Column 15 is U to indicate that MASTFL will be updated.

Column 16 is C to indicate that MASTFL will be processed in a chained fashion.

Column 28 is R to specify that MASTFL will be processed randomly.

3 MSGFL is a message file that is used to report program and other errors. MSGFL is directed to the line printer.

4 This line defines the record type for all records in TRANSFL and is followed by the field definitions for that record.

5 This line defines the record type for active records in MASTFL and is followed by the field definitions for that record.

6 This line defines the record type for inactive records in MASTFL.

Column 27 specifies that inactive records have an I in position 5. Inactive records may exist, but they cannot be updated. (Inactive records reflect customers whose information either has been deleted or is obsolete. They remain in the file to facilitate error handling.)

7 This line reads MASTFL randomly by primary key.

Columns 18-21 define IDNO as the key to be used for reading MASTFL.

Columns 28-32 are CHAIN to specify a chained read operation for MASTFL.

Columns 54-55 turn on indicator 60 when a record matching IDNO is not found in MASTFL.

8 This line adds the contents of field CHARGE to the filed CURBAL (customer balance field in MASTFL) when an active record for IDNO is found.

9 This line defines the output record for MASTFL.

Columns 24-25 specify that indicator 01 must have been turned on (a transaction record must have been read) before records in MASTFL are updated.

Columns 27-28 specify that indicator 02 must be turned on (an active record is found in MASTFL) before records in MASTFL are updated.

10 This line defines the customer balance field, CURBAL. This field is the only field that is updated in MASTFL.

11 This line defines the first output record format for the message file, MSGFL.

Columns 27-28 contain 03 to print an error message when a

deleted record is encountered in MASTFL.

12 The message, DELETED/OBSOLETE RECORD, is printed when indicator 03 is on.

13 This line defines the second record format for the message file, MSGFL.

Columns 27-28 contain 60 to print an error message when a record is not found in MASTFL.

14 The message MISSING RECORD, is printed when indicator 60 is on.

NOTE The CHAIN operation in line 7 turns on indicator 60 when a record is not found for the key. The program handles the error by displaying "MISSING RECORD". If no indicator is used with the CHAIN operation, the halt indicator H0 is turned on and, if an error response was not entered with the Header Specification, the program halts and performs run-time error processing.

Adding Records to a KSAM Disc File

This section explains how to add Records to a KSAM file that already contains data. You can add records in any order. They do not have to be in sequence by a key field.

Figure 3-18 shows how to add records to a KSAM file. You define the file as a CHAINed file by entering C in Column 16 of the File Description Specification. The program in this example adds records using EXCPT output (you can add records at detail-time also). The CHAIN operation reads the file to see if a record already exists for the value in the key field, DEPT. If a record does not exist, it is added.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
                                                                                
    FINPUT   IP  F     256            DISC
 1  FMASTFL  UC  F     256R 4AI     1 DISC                          A

    IINPUT   NS  01
    I                                        1   4 DEPT
    I                                        1 256 RECRD
    IMASTFL  NS  01

 2  C           DEPT      CHAINMASTFL                80
 3  C   80                EXCPT          ADDREC

 4  OMASTFL EADD             ADDREC
    O                         RECRD    256
```

Figure 3-18. Adding Records to a KSAM File Randomly by Primary Key

Comments

1 This line defines the KSAM file, MASTFL.

Column 16 is U to indicate that the KSAM file will be updated.

Column 17 is C for CHAINed (random) access.

Column 66 is A to indicate that records will be added to the file.

2 This line reads MASTFL randomly.

Columns 18-27 contain DEPT to specify the primary key field for MASTFL.

Columns 28-32 contain CHAIN to read a record randomly.

Columns 54-55 contain 80 to turn on indicator 80 when a record is not found in MASTFL.

3 This line directs RPG to perform exception output when a department record is not found (indicator 80 is turned on).

Columns 10-11 contain 80 to specify that this line be executed when indicator 80 (line 2) is turned on.

Columns 28-32 contain EXCPT to direct RPG to write records while calculations are in progress.

Columns 43-48 are ADDREC, the EXCPT Name for the record to be added.

4 This line defines the MASTFL file output record for new (added) records.

Column 15 is E to identify this record as an exception output record.

Columns 16-18 are ADD to specify that records be added to the KSAM file.

Columns 32-37 are ADDREC to specify the EXCPT Name for the record to be added.

Deleting Records from a KSAM Disc File

This section tells you how to delete records from a KSAM file.

When you delete a record, you mark it as inactive. KSAM automatically enters a -1 (hexadecimal FFFF) in the first two bytes of the record. Deleted records remain in the file but are ignored when you read it.

When using KSAM files, it is a good idea to start the data fields of each record in position three. This enables you to recover inactive records in their entirety, if this becomes necessary. There are two ways to access records that have been marked as deleted. You can use an RPG program to read the file chronologically or you can use FCOPY. When using FCOPY, copy the KSAM file with the NOKSAM file option.

To physically remove inactive records from a KSAM file, use FCOPY (with the KSAM default option). FCOPY copies the file,except for inactive records to either an MPE file or another KSAM file. You can also use an RPG program to remove inactive records by copying the KSAM file to another file.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
                                                                                
     FINPUT   IP  F      80            DISC
1    FMASTFL  UC  F     256R 6AI    10 DISC

     IINPUT   NS  01
     I                                        1   6 EMPNO
     IMASTFL  NS  01

2    C           EMPNO     CHAINMASTFL              80
3    C  N80                 EXCPT          DELREC

4    OMASTFL  EDEL              DELREC
```

Figure 3-19. Deleting Records From a KSAM File

Comments

1 This line defines the KSAM file, MASTFL.

Column 15 is U to indicate that the file will be updated.

2 This line reads the KSAM file, MASTFL, in a chained fashion.

Columns 18-27 contain EMPNO, which is key field name.

Columns 28-32 contain CHAIN to specify a CHAINed read operation.

Columns 33-42 contain MASTFL to name the KSAM file.

3 This line performs exception output when the appropriate record in MASTFL is found.

4 This line defines the delete operation to be performed for exception output.

Columns 16-18 are DEL to delete the current output record from MASTFL.

Columns 32-37 contain DELREC to specify the EXCPT Name for the record to be deleted.

Providing Security for KSAM Disc Files

When a KSAM file is going to be accessed by several programs, you must decide on the type of security the file should have. If you want to allow the programs to access the file simultaneously or do you want only one program at a time to access the file? You specify the type of file security with the operating system FILE command (see the *MPE XL Commands Reference Manual* for details on the FILE command).

The following list summarizes the level of file security that you can use for KSAM files.

* **Exclusive**

  Exclusive access prevents others from accessing a file that is used

in your program. When you close the file, or when your program finishes, the file is available to others. For example, entering this FILE command before running a program gives the program exclusive access to the file, MASTER:

```
:FILE MASTER;EXC
```

* **Semi-exclusive**

Semi-Exclusive access lets others read a file but prevents them from updating it. When you close the file, or when your program finishes, the file is available to others. For example, entering this FILE command before running a program gives the program semi-exclusive access to the file, MASTER:

```
:FILE MASTER;SEMI
```

* **Shared**

Shared access lets others read and write records in the same file simultaneously. Different records can be accessed independently. Each user has separate buffers, record pointers and file control information. For example, entering this FILE command before running a program gives the program shared access to the file, MASTER:

```
:FILE MASTER;SHR
```

There are two ways to process shared files in an RPG program. You can have RPG keep track of the records that are being accessed and lock and unlock the file automatically. Alternatively, you can control the locking process, yourself. If you do this, you must be careful to lock the file before updating it and unlock it when you're finished. The next two sections discuss automatic and manual locking in detail.

NOTE All RPG programs processing the same file concurrently must either use manual or automatic locking or must not use locking at all.

Both automatic and manual locking enable the MPE Dynamic Locking Facility. RPG programs that use automatic or manual locking for a file cannot run concurrently with programs that do not have it enabled. For non-RPG programs, enable the MPE Dynamic Locking Facility by entering a FILE command similar to FILE MASTER;LOCK.

Automatically Locking and Unlocking Shared KSAM Files. Using LOCK in the File Description Continuation Specification enables RPG automatic looking. When you read or write a record, RPG locks and unlocks the file for you. Figure 3-20 shows how to use LOCK in a program that adds records to the KSAM disc file, MASTFL.

Each user, before running a program that accesses a KSAM file concurrently with another user, must enter a file equation similar to the following:

```
:FILE MASTFL;SHR
```

```
      1         2         3         4         5         6         7
  6789012345678901234567890123456789012345678901234567890123456789012345

  FINPUT   IP  F     256            DISC
1 FMASTFL  UC  F     256R 4AI     1 DISC                      A
2 F                                              KLOCK

  IINPUT   NS  01
  I                                         1   4 DEPT
  I                                         1 256 RECRD
  IMASTFL  NS  01

3 C           DEPT      CHAINMASTFL               80
4 C   80                EXCPT          ADDREC

5 OMASTFL  EADD              ADDREC
  O                          RECRD    256
```

Figure 3-20. Automatically Locking and Unlocking KSAM Files

<u>Comments</u>

1 This line defines KSAM file, MASTFL, as an Update CHAINed file.

2 This line enables the automatic lock facility for MASTFL.

Column 53 is K to indicate that this is a Continuation line.

Columns 54-59 are LOCK to enable RPG automatic locking.

3 The line reads MASTFL randomly by department number, DEPT.

4 This line directs RPG to perform exception output when a department record is not found (indicator 80 is turned on).

5 This line defines the MASTFL file output record for new (added) records.

Manually Locking and Unlocking Shared KSAM Files. Using NOLOCK in the File Description Continuation Specification lets you manually lock and unlock a KSAM file yourself.

Figure 3-21 shows how to enter the File Description and Calculation Specifications that lock and unlock a KSAM file. The purpose of the program is to add records to the file. Before they are added, the program checks to see if they already exist. The KSAM file is locked before the check is made and unlocked after the new record is added.

When manually locking and unlocking a KSAM file, you must enter a file equation similar to the following before running the program:

```
:FILE MASTFL;SHR
```

```
       1         2         3         4         5         6         7
   678901234567890123456789012345678901234567890123456789012345678901234

   FINPUT   IP  F     256            DISC
1  FMASTFL  UC  F     256R 4AI     1 DISC                      A
2  F                                              KNOLOCK

   IINPUT   NS  01
   I                                        1   4 DEPT
   I                                        1 256 RECRD
   IMASTFL  NS  01

3  C                     LOCK MASTFL                 1112
4  C   12      DEPT      CHAINMASTFL               80
5  C   12 80             EXCPT          ADDREC
6  C   12                UNLCKMASTFL                 1112

7  OMASTFL  EADD             ADDREC
   O                         RECRD    256
```

Figure 3-21. Manually Locking and Unlocking KSAM Files

Comments

1 This line defines the KSAM file,MASTFL, as an Update CHAINed file.

2 This line enables the manual locking facility for the file, MASTFL.

Column 53 is K to indicate that this is a Continuation line.

Columns 54-59 are NOLOCK to enable RPG manual locking.

3 This line locks the file, MASTFL, before a record is read from it.

Columns 28-32 contain LOCK. This locks MASTFL before the chain operation (next line).

Columns 33-42 contain the name of the KSAM file, MASTFL.

Columns 58-59 contain 12 to turn on indicator 12 when the lock operation is successful.

4 This line reads MASTFL in a chained fashion.

Columns 10-11 contain 12 to execute the CHAIN operation when resulting indicator 12 is turned on (the previous LOCK operation was successful).

5 This line performs exception output when a department record is not found (indicator 80 is turned on).

6 This line unlocks the file, MASTFL.

7 This line defines the MASTFL file output record for new (added) records.

Database Disc Files (TurboIMAGE)

The following list summarizes the advantages and disadvantages of using TurboIMAGE databases. For detailed information on TurboIMAGE, see the *TurboIMAGE/XL Database Management System* manual.

Advantages:

* Query (QUERY) facilities
* Security to the data field level
* Data integrity in a multiuser environment
* Automatic maintenance of data interrelationships
* Data-dictionary approach to files
* Efficient memory usage in a multiuser environment
* Flexible locking strategies
* Rapid access by key

Disadvantages:

* No chronological access
* No partial, generic or sequential-within-limits key access
* Inefficient disc usage when actual file size varies considerably (the maximum file size must be allocated regardless of the space that is actually used)
* Inability to update a key field
* No undo for data deletes
* No duplicate keys in master data sets
* Databases are not processed by most system software (for example, EDITOR, FCOPY and SORT)
* No one-step method for using standalone sorts

The remainder of this chapter tells you how to create and use TurboIMAGE databases within RPG programs. To simplify the examples, a single database (MARKET) is used throughout. MARKET keeps customer account information and is used online by a marketing department. MARKET has four data sets. D-ACCOUNTS is a detail data set having NAME-LAST and ACCOUNT-NO as keys. The automatic masters, A-LAST-NAME and A-ACCOUNT-NO keep track of these keys. M-SOURCE is a manual master data set that is used to validate source codes and to keep track of customer responses by source code. It has only a few fields, and is updated infrequently. Figure 3-22 shows the structure of the MARKET database.

[ERROR:] Click here to view figure.

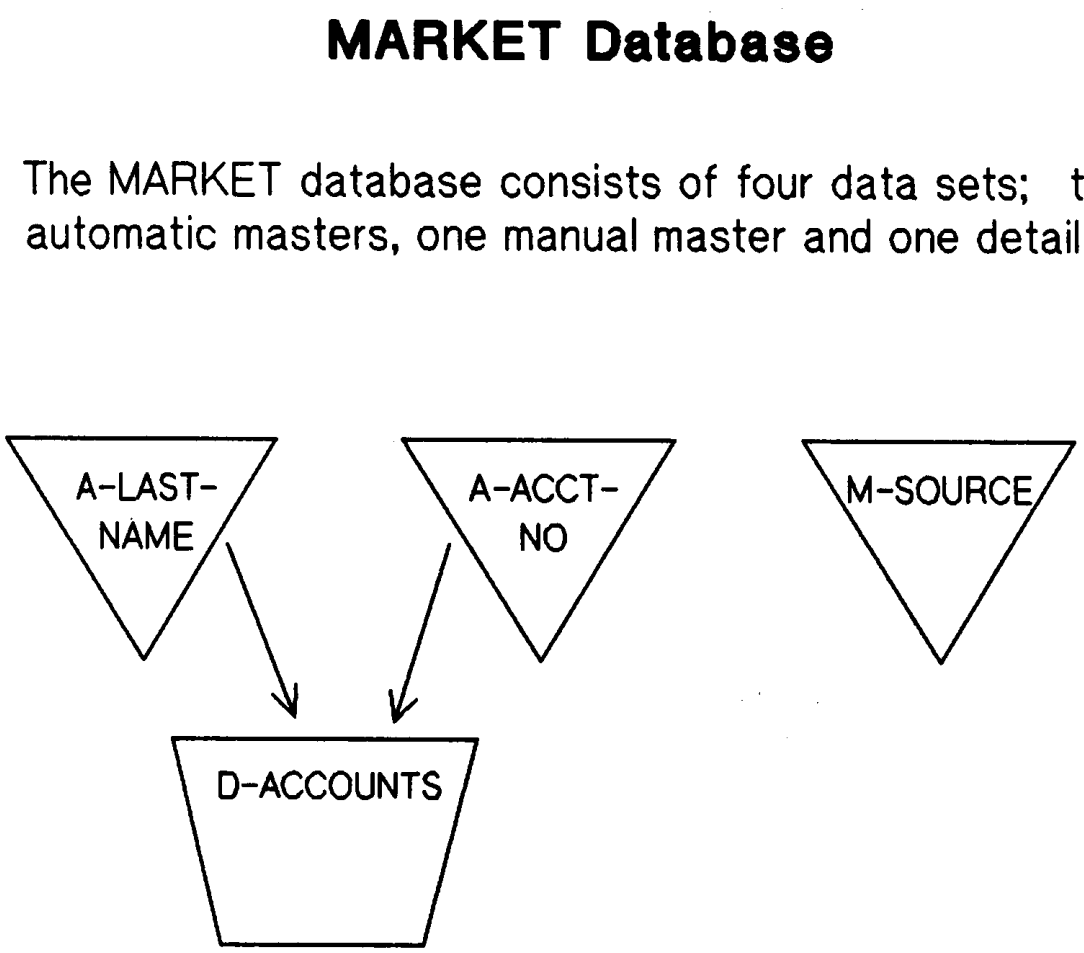


Figure 3-22. The MARKET Database

Creating a TurboIMAGE Database

There are four steps to creating and loading data into a TurboIMAGE database. First, you create a schema file that defines the database, its data sets and data items. Next, you build the database by creating a root file and then running DBUTIL. Finally, you write a program to load data into the database. The next sections explain each of these steps in detail.

Defining a TurboIMAGE Database Schema. The first step in creating a TurboIMAGE database is to create the schema for it.

It is assumed that you are familiar with TurboIMAGE, and that you know how to code the various entries to define the schema. You can use EDITOR or any standard text processor to create the schema file. For information on TurboIMAGE, see the *TurboIMAGE/XL Database Management System* manual.

Figure 3-23 shows the schema that defines the MARKET database shown in Figure 3-20. The data sets in MARKET are shaded for easy reference.

```
$CONTROL TABLE,BLOCKMAX=512,LIST
$TITLE "DIRECT- AND TELE-MARKETING DATABASE SCHEMA"

BEGIN

DATA BASE     MARKET;

<<            DATABASE FOR MARKETING THROUGH DIRECT AND PHONE SALES

SCHEMA NAME   MARKETSC.SOURCE.RPGCLASS

AUTHOR        JOE SMITH

DATE          APRIL 6,1988

CONVENTIONS

The ITEMS portion of the schema is organized in field order
alphabetically by name.  Data sets are listed in the order:
1) automatic masters, 2) manual masters, 3) detail data sets.

Dates used as sort fields are stored as YYYYMMDD alphanumeric (X8).

Naming of fields is done in COBOL fashion, using up to 15
characters per field name.  This convention allows clarity
in naming when accessing the database with QUERY.  The
RPG abbreviated name is listed as a comment with each field.
>>

PASSWORDS:  1 READER;
            2 WRITER;
$PAGE
```

Figure 3-23. The Schema For the Market Database

```
<<
QUERY NAME      SIZE/TYPE      DESCRIPTION                          RPG NAME
-------------------------------------------------------------------------
>>
ITEMS:

ACCOUNT-NO,       X6 ;    << Account number                        ACTNO
                          >>
ACTIVITY-DATE,    X8 ;    << Date of last account activity         ADATE
                          >>
ADDRESS1,         X28;    << Address lines one through three       ADDR1
                          >>
ADDRESS2,         X28;    <<                                       ADDR2
                          >>
ADDRESS3,         X28;    <<                                       ADDR3
                          >>
BEGIN-DATE,       J2 ;    << Date that the customer was first      BDATE
                             contacted or placed on account list
                             (YYYYMMDD format)
                          >>
BLANK02,          X2 ;    << Unused filler space
                          >>
BLANK38,          X38;    << Unused filler space
                          >>
MAIL-CODES,       X8 ;    << Mail code array, holds up to eight    MAILC
                             different mail codes
                          >>
NAME-FIRST,       X10;    << Customer first name                   FNAME
                          >>
NAME-LAST,        X16;    << Customer last name                    LNAME
                          >>
PHONE,            P12;    << Phone number - area,prefix,number     PHONE
                             (unpacked length = 11)
                          >>
PHONE-EXT,        J1 ;    << Phone extension number                PHEXT
                          >>
SOURCE-CODE,      X4 ;    << Source code indicating how the        SRCCD
                             account was first added to files
                          >>
SOURCE-DATE,      J2 ;    << Date of mailing, advertisement, etc   SRCDT
                          >>
SOURCE-DESC,      X30 ;   << Source code description (adver-       SRCDS
                             tisement, direct mail piece, etc)
                          >>
```

Figure 3-23. The Schema For the Market Database (Continued)

```
SOURCE-QTY,       J2 ;    << Current number of accounts started   SRCQY
                             from this source code
                          >>
SOURCE-TYPE,      X2 ;    << Source type code                      SRCTP
                          >>
TYPE-CODE,        X2 ;    << Type code for further classification TYPEC
                             of customer account
                          >>
ZIP-CODE,         X10;    << ZIP+4 code, left justified, zero     ZIPCD
                             filled on right
                          >>
$PAGE
```

Figure 3-23. The Schema For the Market Database (Continued)

```
SETS:

NAME:    A-LAST-NAME,          AUTOMATIC (1/2);
ENTRY:   NAME-LAST(1)             << Key Field >>
         ;
CAPACITY:    97;

NAME:    A-ACCOUNT-NO,         AUTOMATIC (1/2);
ENTRY:   ACCOUNT-NO(1)            << Key Field >>
         ;
CAPACITY:    97;

NAME:    M-SOURCE,             MASTER(1/2);
ENTRY:
          SOURCE-CODE(0)           <<    1    4     Key Field    >>
         ,SOURCE-DATE              <<    5    8B >>
         ,SOURCE-DESC              <<    9   38  >>
         ,SOURCE-QTY               <<   39   42B >>
         ,SOURCE-TYPE              <<   43   44  >>
         ;
CAPACITY:     29;

<<  D-ACCOUNTS is a detail data set using NAME-LAST and
    ACCOUNT-NO as keys, which are maintained in the
    automatic masters, A-LAST-NAME and A-ACCOUNT-NO.     >>

NAME:    D-ACCOUNTS,           DETAIL(1/2);
ENTRY:
          BLANK02                  <<    1    2  >>
         ,ACCOUNT-NO(A-ACCOUNT-NO) <<    3    8     Key Field    >>
         ,NAME-FIRST               <<    9   18  >>
         ,NAME-LAST(!A-LAST-NAME)  <<   19   34     Primary Path >>
         ,ADDRESS1                 <<   35   62  >>
         ,ADDRESS2                 <<   63   90  >>
         ,ADDRESS3                 <<   91  118  >>
         ,ZIP-CODE                 <<  119  128  >>
         ,PHONE                    <<  129  134P >>
         ,PHONE-EXT                <<  135  136B >>
         ,MAIL-CODES               <<  137  144  >>
         ,BEGIN-DATE               <<  145  148B >>
         ,SOURCE-CODE              <<  149  152  >>
         ,TYPE-CODE                <<  153  154  >>
         ,ACTIVITY-DATE            <<  155  162  >>
         ,BLANK38                  <<  163  200  >>
         ;
CAPACITY:    100;

END.
```

Figure 3-23. The Schema For the Market Database (Continued)

Creating a TurboIMAGE Root File. Once you define a TurboIMAGE schema, you then create a TurboIMAGE root file.

For example, to create the root file for our sample MARKET database, enter these commands at the operating system colon prompt:

```
:FILE DBTEXT=MARKET.PUB
:FILE DBSLIST=$STDLIST
:RUN DBSCHEMA.PUB.SYS;PARM=3
```

If there are any schema or execution errors, they are displayed next. When the root file is successfully created, you see the message END OF PROGRAM.

NOTE Instead of equating DBSLIST to $STDLIST in the file equation, you can assign it to other devices. For example, $NULL or DEV=LP.

Running DBUTIL. After creating a TurboIMAGE root file, you create an empty database.

For example, to create an empty database for our sample MARKET database, enter these commands:

```
:RUN DBUTIL.PUB.SYS
>>CREATE MARKET
>>EXIT
```

When an empty database is created successfully, you see the message data base MARKET has been created.

CAUTION Enter the CREATE command carefully. DBUTIL commands perform many functions, one of which purges databases.

Loading Data into a TurboIMAGE Database. You can use an RPG program to load data into a TurboIMAGE database. You do this by entering TurboIMAGE-specific fields in the File Description and File Description Continuation Specifications.

The program in Figure 3-24 shows how to load the D-ACCOUNTS data set (see our sample MARKET database in Figure 3-23) with data from an MPE file, ACCTSIN. Records in ACCTSIN are sorted by account number (ACTNO).

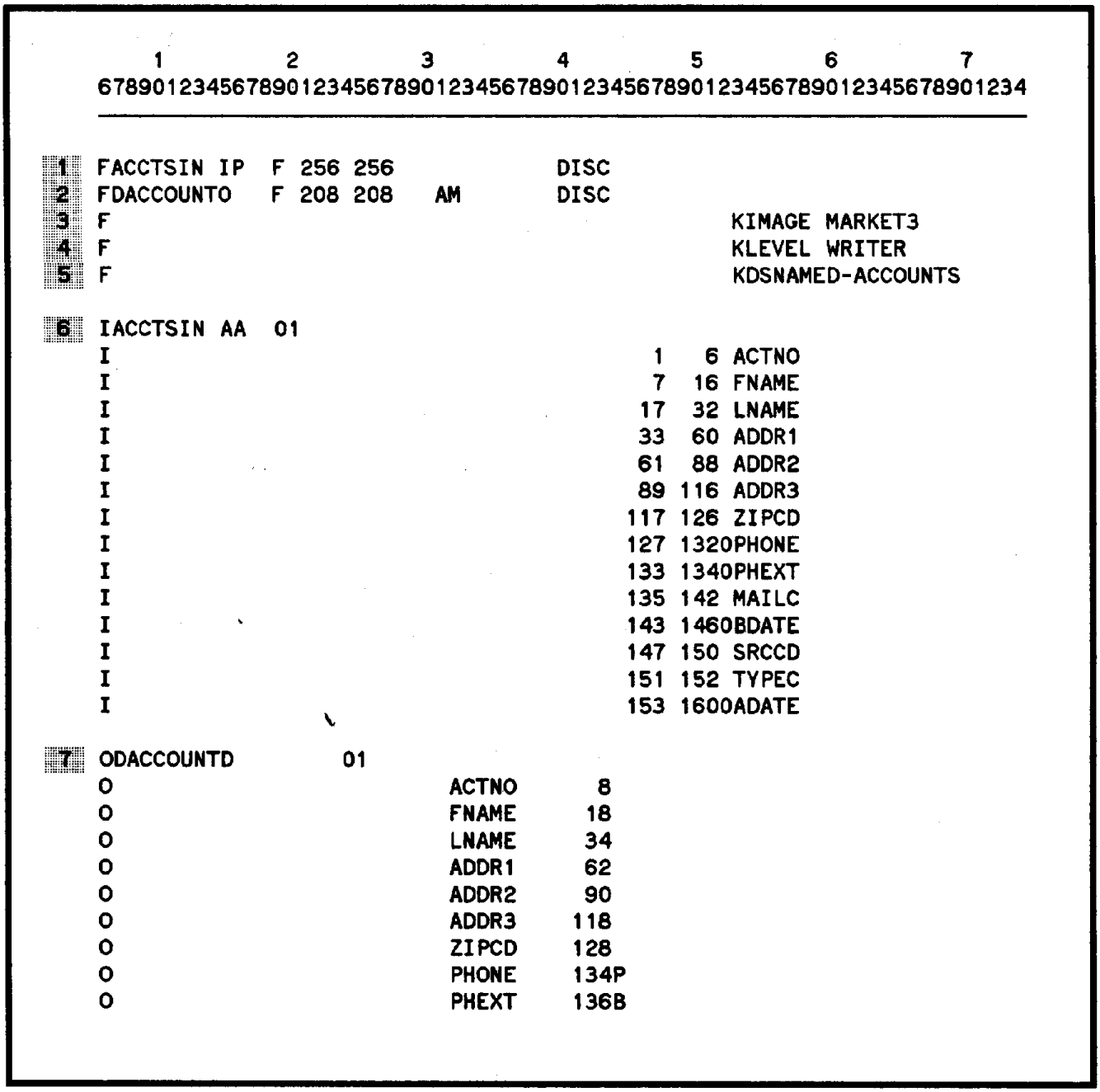


Figure 3-24. Loading a TurboIMAGE Database

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
```

```
0                       MAILC    144
0                       BDATE    148B
0                       SRCCD    152
0                       TYPEC    154
0                       ADATE    162
```

Figure 3-24. Loading a TurboIMAGE Database (Continued)

Comments

1 This line defines the accounts file, ACCTSIN.

2 This line defines the TurboIMAGE data set file, DACCOUNT.

Column 31 is A to indicate that DACCOUNT has an alphanumeric key.

Column 32 contains M to indicate that DACCOUNT is a TurboIMAGE file.

3 This line specifies that DACCOUNT is part of the MARKET database.

Column 53 is K to indicate that this line is a Continuation line.

Columns 54-65 identify MARKET as the TurboIMAGE database that is used.

Column 66 is 3 to specify Open Mode - Exclusive Access.

Column 67 is blank to specify Input/Output Mode - Write, Output File.

4 This line establishes write access to the database.

Column 53 is K to indicate that this is a Continuation line.

Columns 54-65 contain LEVEL WRITER to specify the write-access password for the database.

5 This line identifies the key field for the data set.

Column 53 is K to indicate that this is a Continuation line.

Columns 54-65 contain DSNAMED-ACCOUNTS to identify the data set, D-ACCOUNTS to be accessed (this line is not needed when line 2 contains the actual data set name).

6 This line begins the description of the ACCTSIN input record.

7 This line begins the description of the DACCOUNT output record.

Reading a TurboIMAGE Data Set

You can access data in a TurboIMAGE data set three different ways and each of these is discussed in the sections which follow:

* Sequentially (not in key order)

* Sequentially by full key values

* Randomly by key value

Reading a TurboIMAGE Data Set Sequentially. Since TurboIMAGE does not maintain the sequential order of keys in its data sets, you cannot read the entire data set from beginning to end by a specific key. However, you can retrieve all of the records in their stored unsorted sequence. This may be useful, for example, for extracting data set records for a sort.

Figure 3-25 shows how to retrieve all of the records in the TurboIMAGE data set, D-ACCOUNTS (see the schema for this data set in Figure 3-23), and how to display the first four fields in them.

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
                                                                                
1   FDACCOUNTIP  F 200 200   AM       DISC
2   F                                                KIMAGE MARKET62
3   F                                                KLEVEL READER
4   F                                                KDSNAMED-ACCOUNTS
    FDISPLAY O   F      80            $STDLST

5   IDACCOUNTNS  02
    I                                      3   8 ACTNO
    I                                      9  18 FNAME
    I                                     19  34 LNAME
    I                                     35  62 ADDR1

6   ODISPLAY D        02
    O                       ACTNO      6
    O                       FNAME     16
    O                       LNAME     32
    O                       ADDR1     60
```

Figure 3-25. Reading a TurboIMAGE Data Set Sequentially

Comments

1 This line defines the TurboIMAGE data set file, DACCOUNT.

2 This line defines the database containing D-ACCOUNTS and the security provisions for accessing the database.

Column 53 is K to indicate that this is a Continuation line.

Columns 54-65 identify MARKET as the TurboIMAGE database that is used.

Column 66 is 6 to indicate Open Mode 6 - Shared Read Access.

Column 67 is 2 to specify Input/Output Mode 2 - Serial Read.

3 This line establishes the password for the database.

Columns 54-65 contain LEVEL READER to specify the password, READER. This password establishes a user class identification

that permits read access to the database.

4 This line defines the data set to be used.

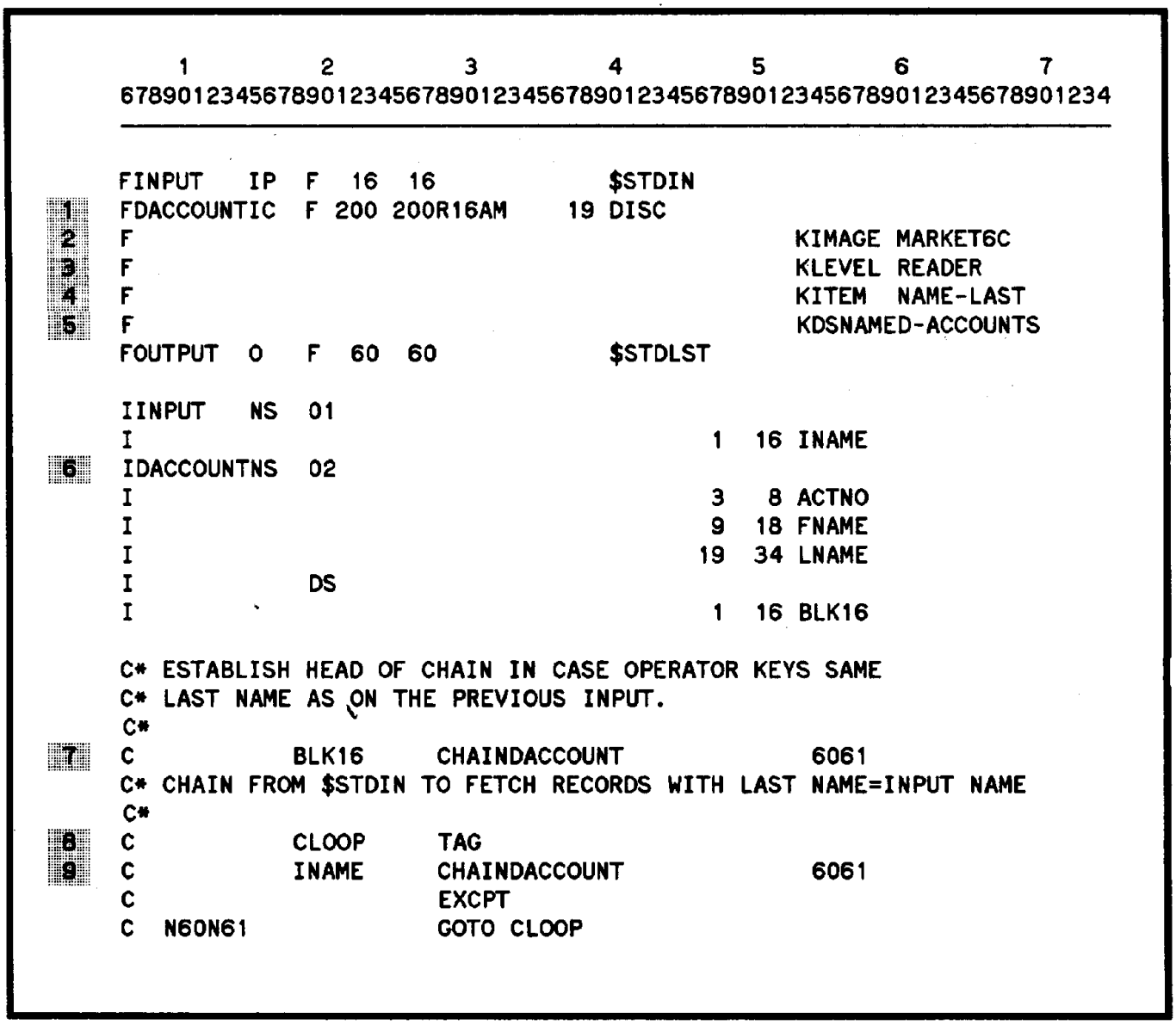
Columns 54-65 contain DSNAMED-ACCOUNTS to identify the data set, D-ACCOUNTS.

5 This line begins the input record description of the DACCOUNT file.

6 This line begins the output record description of the DISPLAY file.

Reading a TurboIMAGE Data Set Sequentially By Key. This section explains how to read all records in a detail data set having a specific key value. This method is useful when a data set contains records with duplicate keys. Unlike KSAM files, you cannot read a TurboIMAGE data set sequentially by a range of key values (reading sequentially within key limits).

The program in Figure 3-26 finds and displays all records in the data set, D-ACCOUNTS, having the same last name. The last name field (NAME-LAST) is defined as a key in the data set (see the schema for the data set in Figure 3-23). A user enters a last name from a terminal. When the last name does not exist, one of two warning messages (NO SUCH NAME or END OF CHAIN) is displayed.



```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

     FINPUT   IP  F  16  16            $STDIN
1    FDACCOUNTIC  F 200 200R16AM    19 DISC
2    F                                        KIMAGE MARKET6C
3    F                                        KLEVEL READER
4    F                                        KITEM  NAME-LAST
5    F                                        KDSNAMED-ACCOUNTS
     FOUTPUT  O   F  60  60            $STDLST

     IINPUT   NS  01
     I                                      1  16 INAME
6    IDACCOUNTNS  02
     I                                      3   8 ACTNO
     I                                      9  18 FNAME
     I                                     19  34 LNAME
     I            DS
     I                                      1  16 BLK16

     C* ESTABLISH HEAD OF CHAIN IN CASE OPERATOR KEYS SAME
     C* LAST NAME AS ON THE PREVIOUS INPUT.
     C*
7    C           BLK16     CHAINDACCOUNT             6061
     C* CHAIN FROM $STDIN TO FETCH RECORDS WITH LAST NAME=INPUT NAME
     C*
8    C           CLOOP     TAG
9    C           INAME     CHAINDACCOUNT             6061
     C                     EXCPT
     C  N60N61             GOTO CLOOP
```

Figure 3-26. Reading a TurboIMAGE Data Set Sequentially by Key

```
      1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

10 OOUTPUT  E         01
   O                  N60N61    FNAME     10
   O                  N60N61    LNAME     27
   O                  N60N61    ACTNO     34
   O                   60                 15 "NO SUCH NAME"
   O                   61                 60 "END OF CHAIN"
```

Figure 3-26. Reading a TurboIMAGE Data Set Sequentially by Key (Continued)

Comments

1 This line defines the TurboIMAGE data set file, DACCOUNT.

Columns 28-38 indicate that DACCOUNT is a TurboIMAGE file to be processed randomly by the NAME-LAST key field (positions 19-34).

2 This line specifies that DACCOUNT is part of the MARKET database.

Columns 54-65 identify MARKET as the TurboIMAGE database that is used.

Column 66 is 6 to indicate Open Mode 6 - Shared Read Access.

Column 67 is C to indicate Input/Output Mode C - Chained Sequential Read. In this mode, records are read sequentially along the data set chain.

3 This line defines the password for the database.

Columns 54-65 contain LEVEL READER to specify the password, READER. This password establishes a user class identification that permits read access to the database.

4 This line identifies the key field for the data set.

Columns 54-65 contain ITEM NAME-LAST to identify the key field, NAME-LAST.

5 This line identifies the data set to be used.

Columns 54-65 contain DSNAMED-ACCOUNTS to identify the data set, D-ACCOUNTS.

6 This line begins the input record description for the DACCOUNT file.

7 This line establishes the head of the chain in the D-ACCOUNTS data set.

Columns 18-27 contain BLK16 to reset the file pointer to the head of the chain.

Columns 28-32 contain CHAIN to specify a "dummy" chained read operation. This operation simply establishes a new head of chain.

Columns 33-42 specify the name of the TurboIMAGE data set file, DACCOUNT.

8 This line is the beginning of the loop that reads records in the D-ACCOUNTS data set.

9 This line reads records randomly from the D-ACCOUNTS data set.

Columns 18-27 contain INAME to specify the key field.

Columns 28-32 contain CHAIN to specify a chained read operation.

Columns 33-42 specify the name of the TurboIMAGE data set file, DACCOUNT.

Columns 54-55 contain 60 to turn on indicator 60 when a record is not found for the key value in INAME.

Columns 56-57 contain 61 to turn on indicator 61 when there are no more records in the data set chain.

10 This line starts the description of the output record to be displayed on the terminal.

Reading a TurboIMAGE Data Set Randomly. This section explains how to retrieve master data set records using random key values.

The following example takes an account number (ACTNO) entered by a user and verifies it against a list of valid account numbers in the A-ACCOUNT-NO data set. (The schema for this data set is shown in the section "Defining a TurboIMAGE Database Schema.") When an account does not exist, a message (NO HIT ON CHAIN) is displayed.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
FINPUT   IP  F  16  16           $STDIN
1 FAACCOUNTIC  F   6   6R 6AM     1 DISC
2 F                                          KIMAGE MARKET67
3 F                                          KLEVEL READER
4 F                                          KITEM  ACCOUNT-NO
5 F                                          KDSNAMEA-ACCOUNT-NO
FOUTPUT  O   F  60  60           $STDLST

IINPUT   NS  01
I                                      1   6 ACTNO
6 IAACCOUNTNS  02
C*
C* CHAIN FROM $STDIN INPUT TO SEE IF ACCOUNT EXISTS
C*
7 C           ACTNO     CHAINAACCOUNT             60H0

8 OOUTPUT  D        01
O                N60               15 "ACCOUNT EXISTS"
O                 60               15 "NO HIT ON CHAIN"
```

Figure 3-27. Reading a TurboIMAGE Data Set Randomly

Comments

1 This line defines, AACCOUNT, which is a TurboIMAGE automatic master data set.

Columns 28-38 indicate that AACCOUNT is a TurboIMAGE file to be processed randomly by the ACCOUNT-NO key field (positions 1-6).

2 This line specifies that AACCOUNT is part of the MARKET database.

Columns 54-65 identify MARKET as the TurboIMAGE database that is used.

Column 66 is 6 to indicate Open Mode 6 - Shared Read Access.

Column 67 is 7 to specify Input/Output Mode 7 - Calculated Read. This mode applies to master data sets only.

3 This line defines the password for the database.

Columns 54-65 contain LEVEL READER to specify the password READER. This password establishes a user class identification that permits read access to the database.

4 This line identifies the key field for the data set.

Columns 54-65 contain ITEM ACCOUNT-NO to specify that the key for the data set is ACCOUNT-NO.

5 This line names the data set to be accessed.

Columns 54-65 contain DSNAMEA-ACCOUNT-NO to specify that the data set is A-ACCOUNT-NO.

6 This line begins the input record description of the A-ACCOUNT-NO data set.

7 This line reads the data set, A-ACCOUNT-NO randomly.

Columns 18-27 contain the name of the key field, ACTNO.

Columns 28-32 contain CHAIN to specify a chained read operation.

Columns 33-42 specify the name of the TurboIMAGE data set file, AACCOUNT.

Columns 54-55 contain 60 to turn on indicator 60 when no record is found having the key value contained in ACTNO.

Columns 56-57 contains the default H0 indicator to avoid a compiler warning for the CHAIN operation. The end-of-chain indicator has no meaning for master data sets, since they cannot have duplicate keys.

8 This line starts the description of the output record for OUTPUT. The record contains messages that indicate whether or not the account exists in the master data set.

Updating a TurboIMAGE Data Set

This section explains how to modify records that already exist in a TurboIMAGE data set.

For information on adding and deleting records, see the next two sections. To learn about database security provisions that can be incorporated into your RPG programs, see the section titled "Providing Security for TurboIMAGE Databases and Data Sets."

Figure 3-28 shows how to access records in the TurboIMAGE data set D-ACCOUNTS (see the schema for this data set in Figure 3-23) by its key, and change the TYPEC field in those records.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
1    FINPUT   IP  F  16  16            $STDIN
2    FDACCOUNTUC  F 200 200R 6AM     3 DISC
3    F                                          KIMAGE MARKET25
4    F                                          KLEVEL WRITER
5    F                                          KITEM  ACCOUNT-NO
6    F                                          KDSNAMED-ACCOUNTS

7    IINPUT   NS  01
     I                                      1   6 ACTNO
     I                                      7   8 TYPEC
8    IDACCOUNTNS  02

9    C           ACTNO     CHAINDACCOUNT             60

10   ODACCOUNTD        02
     O                         TYPEC     154
```

Figure 3-28. Randomly Updating Records in a TurboIMAGE Data Set

Comments

1 This line defines the file, INPUT.

2 This line defines the TurboIMAGE data set file, DACCOUNT.

Column 15 is U to indicate that DACCOUNT is updated.

Column 16 is C for chained (random) access.

Column 32 is M to specify that DACCOUNT is a TurboIMAGE file.

3 This line specifies that DACCOUNT is part of the MARKET database.

Columns 54-65 identify MARKET as the TurboIMAGE database that is used.

Column 66 is 2 to indicate Open Mode 2 - Update-Shared Access.

Column 67 is 5 to specify Input/Output Mode 5 - Chained Read.

4 This line defines the password for the database.

Columns 54-65 contain LEVEL WRITER to specify the password, WRITER. This password establishes a user class identification that permits write access to the database.

5 This line identifies the key field for the data set.

Columns 54-65 contain ITEM ACCOUNT-NO to specify that the key for the data set is ACCOUNT-NO.

6 This line names the data set to be accessed.

Columns 54-65 contain DSNAMED-ACCOUNTS to specify that the data set is D-ACCOUNTS.

7 This line begins the input record description of the INPUT file.

8 This line begins the input record description of the DACCOUNT file.

9 This line reads the data set, D-ACCOUNTS, randomly.

Columns 18-27 contain ACTNO to specify the key field for reading the data set.

Columns 28-32 contain CHAIN to specify a chained read operation.

Columns 33-42 specify the name of the TurboIMAGE data set file, DACCOUNT.

Columns 54-55 contain 60 to turn on indicator 60 when a record is not found for the account number in D-ACCOUNTS. Even though 60 is not used in the program, an indicator must be entered in this field to prevent the halt indicator (H0) from being turned on.

10 This line begins the output record description for the DACCOUNT file.

Adding Records to a TurboIMAGE Data Set

This section explains how to add records to a TurboIMAGE data set. Records are added randomly and do not have to be ordered by key value.

To learn about database security provisions that can be incorporated into your RPG programs, see the section titled "Providing Security for TurboIMAGE Databases and Data Sets."

Figure 3-29 shows how to add records to our sample data set, D-ACCOUNTS. (See the schema for the MARKET database in the section titled "Defining a TurboIMAGE Database Schema.") Records that are added come from the file, IDACCT, which has the same format as the D-ACCOUNTS data set. The ACTNO field in IDACCT is used as the key when reading D-ACCOUNTS, but the input records are not in sequence by this key. Before a record is added to D-ACCOUNTS, a check is made to ensure that the ACTNO record does not already exist.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
1    FIDACCT  IP  F 200 200            DISC
2    FDACCOUNTUC  F 200 200R 6AM     3 DISC                              A
3    F                                                  KIMAGE MARKET95
4    F                                                  KLEVEL WRITER
5    F                                                  KITEM  ACCOUNT-NO
6    F                                                  KDSNAMED-ACCOUNTS

7    IIDACCT  NS  01
     I                                          3   8 ACTNO
     I                                          1 200 RECRD
8    IDACCOUNTNS  02

9    C           ACTNO     CHAINDACCOUNT              80
10   C   80                EXCPT          ADDREC

11   ODACCOUNTEADD             ADDREC
     O                         RECRD    200
```

Figure 3-29. Randomly Adding Records to a TurboIMAGE Data Set

Comments

1 This line defines the IDACCT file.

2 This line defines the TurboIMAGE data set file, DACCOUNT.

Column 15 is U to indicate that DACCOUNT is updated.

Column 16 is C for chained (random) access.

Column 66 is A to indicate that records are added to DACCOUNT.

3 This line specifies that DACCOUNT is part of the MARKET database.

Columns 54-65 identify MARKET as the TurboIMAGE database that is used.

Column 66 is 9 to indicate Open Mode 9 - Data Set Locking per Record.

Column 67 is 5 to specify Input/Output Mode 5 - Chained Read.

4 This line defines the password for the database.

Columns 54-65 contain LEVEL WRITER to specify the password, WRITER. This password establishes a user class identification that permits write access to the database.

5 This line identifies the key field for the data set.

Columns 54-65 contain ITEM ACCOUNT-NO to specify that the key for the data set is ACCOUNT-NO.

6 This line names the data set to be accessed.

Columns 54-65 contain DSNAMED-ACCOUNTS to specify that the data

set is D-ACCOUNTS.

7 This line begins the input record description of the IDACCT file.

8 This line begins the input record description of the DACCOUNT file. (Since records are being added only, you do not need to define input fields.)

9 This line reads the data set, D-ACCOUNTS, randomly.

Columns 18-27 contain ACTNO to specify the key field for reading the data set.

Columns 28-32 contain CHAIN to specify a chained read operation.

Columns 33-42 specify the name of the TurboIMAGE data set file, DACCOUNT.

Columns 54-55 contain 80 to turn on indicator 80 when a record is not found for the account number in D-ACCOUNTS.

10 This line specifies that exception output is performed when indicator 80 is turned on (indicator 80 is turned on when a record is not found for an account number).

11 This line specifies the output operation for adding records to the D-ACCOUNTS data set.

Column 15 is E to identify this record as an exception record.

Columns 16-18 are ADD to add records to the TurboIMAGE data set.

Columns 32-37 are ADDREC to name the EXCPT Name for the record to be added.

Deleting Records from a TurboIMAGE Data Set

This section explains how to delete records from a TurboIMAGE detail data set. The data set must not contain duplicate keys. Records are deleted by matching their key field values to those specified in the program.

To learn about database security provisions that can be incorporated into your RPG programs, see the section titled "Providing Security for TurboIMAGE Databases and Data Sets."

Figure 3-30 shows how to delete records from our sample data set D-ACCOUNTS. (See the schema for the MARKET database in the section titled "Defining a TurboIMAGE Database Schema.") The input file, INPUT, contains the keys for the records to delete, though records in INPUT are not sequenced by this key. Each record to be deleted in D-ACCOUNTS is read first to verify that it exists. (If you need to delete records in a data set containing duplicate keys, enter C (Chained Sequential Read) in the Input/Output Mode for the file. Then CHAIN and delete the records until the indicator in columns 56-57 of the CHAIN operation line turns on.)

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
1    FINPUT   IP  F   8   8           DISC
2    FDACCOUNTUC  F 200 200R 6AM     3 DISC
3    F                                          KIMAGE MARKETS5
4    F                                          KLEVEL WRITER
5    F                                          KITEM  ACCOUNT-NO
6    F                                          KDSNAMED-ACCOUNTS

7    IINPUT   NS  01
     I                                      3   8 ACTNO
8    IDACCOUNTNS  02

9    C           ACTNO     CHAINDACCOUNT             80H0
10   C  N80                 EXCPT          DELREC

11   ODACCOUNTEDEL             DELREC
```

Figure 3-30. Deleting Records from a TurboIMAGE Data Set

Comments

1 This line defines the file, INPUT.

2 This line defines the TurboIMAGE data set file, DACCOUNT.

Column 15 is U to indicate that DACCOUNT is updated.

Column 16 is C for CHAINed (random) access.

3 This line specifies that DACCOUNT is part of the MARKET database.

Columns 54-65 identify MARKET as the TurboIMAGE database that is used.

Column 66 is S to indicate Open Mode S - Data Set Locking for Duration.

Column 67 is 5 to specify Input/Output Mode 5 - Chained Read.

4 This line defines the password to the database.

Columns 54-65 contain LEVEL WRITER to specify the password, WRITER. This password establishes a user class identification that permits write access to the database.

5 This line identifies the key field for the data set.

Columns 54-65 contain ITEM ACCOUNT-NO to specify that the key for the data set is ACCOUNT-NO.

6 This line names the data set to be accessed.

Columns 54-65 contain DSNAMED-ACCOUNTS to specify that the data set is D-ACCOUNTS.

7 This line begins the input record description of the INPUT file.

8 This line begins the input record description of the DACCOUNT file.

9 This line reads the D-ACCOUNTS data set randomly.

Columns 18-27 contain ACTNO to specify the key field.

Columns 28-32 contain CHAIN to specify a chained read operation.

Columns 54-55 contain 80 to turn on indicator 80 when a record is not found for the account number in D-ACCOUNTS.

Columns 56-57 contain the H0 indicator to avoid a compiler warning for the CHAIN operation. The end-of-chain indicator has no meaning since records are accessed randomly.

10 This line performs exception output when the appropriate record is found in D-ACCOUNTS (indicator 80 in not on).

11 This line defines the exception output operation for deleting records.

Column 15 is E to identify this record as an exception record.

Columns 16-18 are DEL to specify that records are deleted from D-ACCOUNTS.

Columns 32-37 are DELREC to name the EXCPT Name for the record to be deleted.

Providing Security for TurboIMAGE Databases and Data Sets

TurboIMAGE lets several users access a database simultaneously.

To ensure that there will be no access conflicts, you should specify the security provisions in each RPG program. You can specify security at the database, data set or data record (item) level. There are two ways for doing this. You can specify that database and data set security be handled automatically by RPG. If you need more flexibility, you can provide this security by locking and unlocking databases or data sets yourself. The next two sections discuss automatic and manual locking and unlocking in detail.

Automatically Locking and Unlocking TurboIMAGE Databases and Data Sets. To have RPG automatically lock and unlock a TurboIMAGE database, data set or data record, enter a locking mode in column 66 of the IMAGE File Description Continuation Specification. The locking modes that you can select are:

Table 3-1.

Column 66	Description
B	Lock the entire database when the program starts and unlock it when the program ends.
S	Lock the data set when the program starts and unlock it when the program ends.
1	Lock the entire database before a record is read, written or updated. Unlock it after the record is read, written or updated.

9	Lock the data set before a record is read, written or updated. Unlock it after the record is read, written or updated.
R	Lock the data record before it is read, written or updated. Unlock it after the record is read, written or updated.

Figure 3-31 shows how to specify automatic locking and unlocking at the data record level for TurboIMAGE data sets (see line 2).

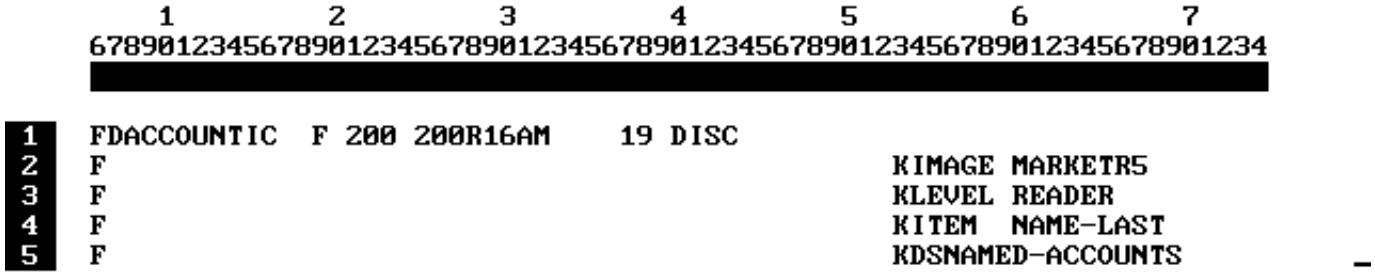


Figure 3-31. Automatically Locking and Unlocking a TurboIMAGE Data Set Record

Comments

1 This line defines the TurboIMAGE data set file, DACCOUNT.

Column 15 is I to indicate that the TurboIMAGE file is input.

Column 16 is C for CHAINed (random) access.

2 This line specifies that DACCOUNT is part of the MARKET database.

Columns 54-65 identify MARKET as the TurboIMAGE database that is used.

Column 66 is R to specify open mode with automatic locking and unlocking at the data record level.

Column 67 is 5 to specify Input/Output Mode 5 - Chained Read.

3 This line defines the password for the database.

Columns 54-65 contain LEVEL READER to specify the password READER. This password establishes a user class identification that permits read access to the database.

4 This line identifies the key field for the data set.

Columns 54-65 contain ITEM NAME-LAST to specify that the key for the data set is NAME-LAST.

5 This line names the data set to be accessed.

Columns 54-65 contain DSNAMED-ACCOUNTS to specify that the data

set is D-ACCOUNTS.

Manually Locking and Unlocking TurboIMAGE Databases and Data Sets. This section explains how you can control the locking and unlocking of TurboIMAGE databases and data sets. You use the LOCK and UNLCK operations in Calculation Specifications.

Figure 3-30 through Figure 3-32 show how to lock and unlock databases, data sets and individual records in the data sets. The examples use the MARKET database whose schema is shown in the section titled "Defining a TurboIMAGE Database Schema."

Figure 3-32 gives an example of adding records to the data set, D-ACCOUNTS. The entire database (MARKET) to which D-ACCOUNTS belongs is locked before a record is added. After the record is added, the database is unlocked.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
1    FIDACCT  IP  F 200 200            DISC
2    FDACCOUNTUC  F 200 200R 6AM     3 DISC                          A
3    F                                                 KIMAGE MARKETLC
4    F                                                 KLEVEL WRITER
5    F                                                 KITEM  ACCOUNT-NO
6    F                                                 KDSNAMED-ACCOUNTS

7    IIDACCT  NS  01
     I                                        3   8 ACTNO
     I                                        1 200 REC
8    IDACCOUNTNS  02

9    C           ACTNO     CHAINDACCOUNT              80H0
10   C   80                LOCK DACCOUNT  MARKET  1    1112
11   C   80                EXCPT          ADDREC
12   C   80                UNLCKDACCOUNT  MARKET  1    1112

13   ODACCOUNTEADD             ADDREC
     O                         REC      200
```

Figure 3-32. Manually Locking and Unlocking a TurboIMAGE Database

Comments

1 This line defines the file, IDACCT.

2 This line defines the TurboIMAGE data set file, DACCOUNT.

Column 15 is U to indicate that DACCOUNT is updated.

Column 16 is C for CHAINed (random) access.

Column 66 is A to indicate that records are added to DACCOUNT.

3 This line specifies that DACCOUNT is part of the MARKET database.

Columns 54-65 identify MARKET as the TurboIMAGE database that is used.

Column 66 is L to indicate user-controlled manual locking.

Column 67 is C to specify chained sequential read mode.

4 This line defines the password for the database.

Columns 54-65 contain LEVEL WRITER to specify password, WRITER. This password establishes a user class identification that permits write access to the database.

5 This line identifies the key field for the data set.

Columns 54-65 contain ITEM ACCOUNT-NO to specify that the key for the data set is ACCOUNT-NO.

6 This line names the data set to be accessed.

Columns 54-65 contain DSNAMEA-ACCOUNT-NO to specify that the data set is D-ACCOUNTS.

7 This line begins the input record description of the IDACCT file.

8 This line begins the input record description of the DACCOUNT file.

9 This line reads the data set, D-ACCOUNTS, randomly.

Columns 18-27 contain ACTNO to specify the key field for reading the data set.

Columns 28-32 contain CHAIN to specify a chained read operation.

Columns 33-42 specify the name of the TurboIMAGE data set file, DACCOUNT.

Columns 54-55 contain 80 to turn on indicator 80 when a record is not found for the account number in D-ACCOUNTS.

Columns 56-57 contain the H0 indicator to avoid a compiler warning for the CHAIN operation. The end-of-chain indicator has no meaning since records are accessed randomly.

10 This line locks the database, MARKET, when indicator 80 is turned on.

Columns 10-11 contain 80 to condition the LOCK operation.

Columns 28-32 contain LOCK to specify the lock operation.

Columns 33-42 contain DACCOUNT to identify the data set to lock.

Columns 43-48 contain MARKET to identify the database to lock. You must enter both a data set (in columns 33-42) and a database in this field to enable locking for the entire database.

Column 51 is 1 to specify the field length (always use 1). This prevents a compiler error.

Columns 54-55 are blank to specify unconditional locking. If the database is already locked by another process, the program suspends until the database is unlocked.

Columns 56-57 contain 11 for the low resulting indicator. An indicator is required in this field. When turned on, this indicator signals a memory manager error.

Columns 58-59 contain 12 for the equal resulting indicator. An indicator is required in this field. When turned on, this indicator signals that the lock request was granted.

11 This line specifies that exception output is performed when indicator 0 is turned on.

12 This line unlocks the database, MARKET, when indicator 80 is turned on.

Columns 10-11 contain 80 to condition the UNLCK operation.

Columns 28-32 contain UNLCK to specify the unlock operation.

Columns 33-42 contain DACCOUNT to identify the data set to unlock.

Columns 43-48 contain MARKET to identify the database to unlock.

Column 51 is 1 for the field length (always use 1).

Columns 56-59 contain the same indicators used in line 10 and they function the same way.

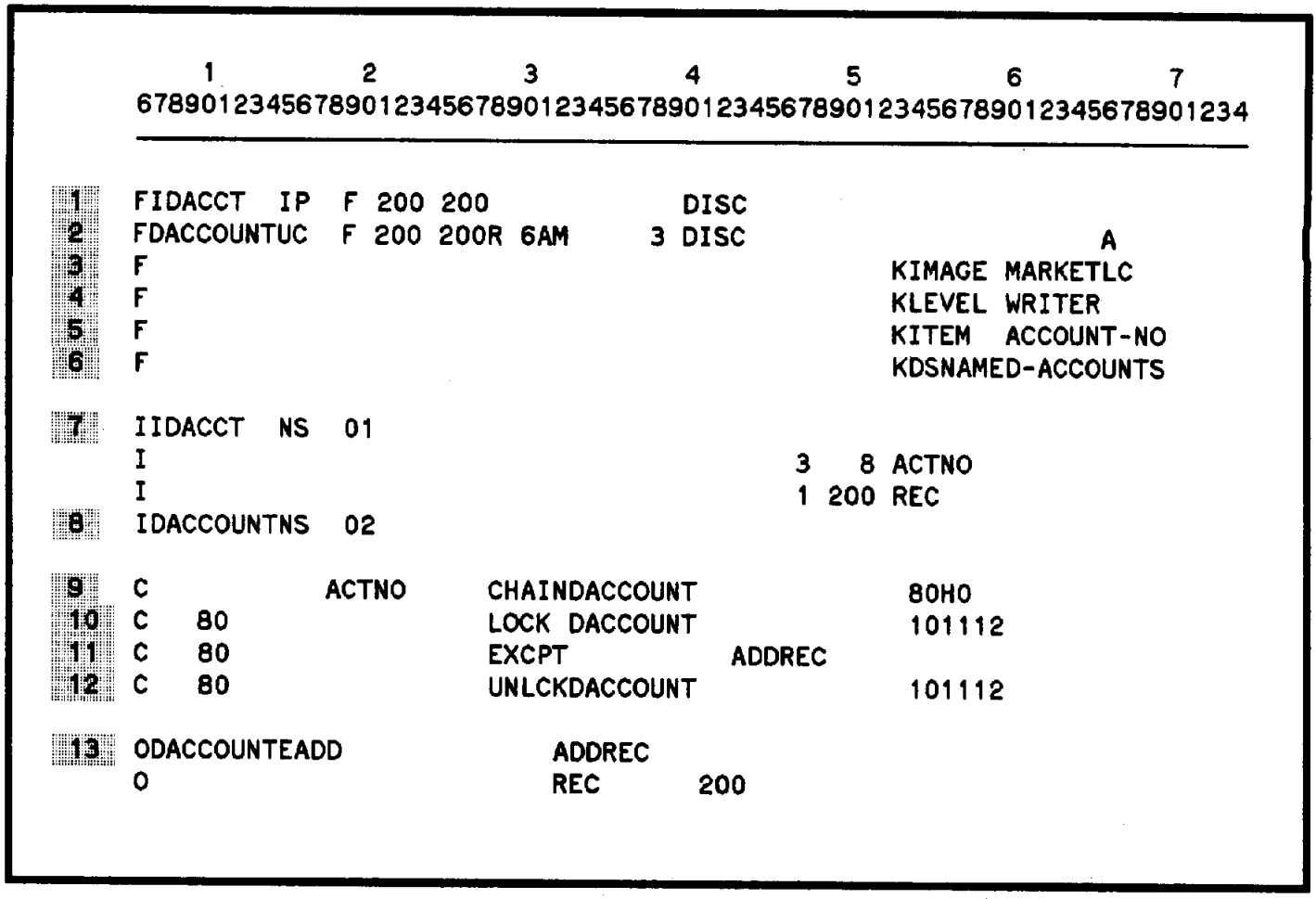
13 This line specifies the output operation for adding records to the D-ACCOUNTS data set.

Column 15 is E to identify this record as an exception record.

Columns 16-18 are ADD to add records to the TurboIMAGE data set.

Columns 32-37 are ADDREC to name the EXCPT Name for the record to be added.

The following figure shows how to lock and unlock the data set, D-ACCOUNTS.



```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-----------------------------------------------------------------------------------------

1    FIDACCT  IP  F 200 200                DISC
2    FDACCOUNTUC  F 200 200R 6AM      3 DISC                                    A
3    F                                                      KIMAGE MARKETLC
4    F                                                      KLEVEL WRITER
5    F                                                      KITEM  ACCOUNT-NO
6    F                                                      KDSNAMED-ACCOUNTS

7    IIDACCT  NS  01
     I                                              3   8 ACTNO
     I                                              1 200 REC
8    IDACCOUNTNS  02

9    C            ACTNO      CHAINDACCOUNT                 80H0
10   C   80                  LOCK DACCOUNT                 101112
11   C   80                  EXCPT           ADDREC
12   C   80                  UNLCKDACCOUNT                 101112

13   ODACCOUNTEADD               ADDREC
     O                           REC       200
```

Figure 3-33. Manually Locking and Unlocking a TurboIMAGE Data Set

Comments

1 This line defines the file, IDACCT.

2 This line defines the TurboIMAGE data set file, DACCOUNT.

Column 15 is U to indicate that DACCOUNT is updated.

Column 16 is C for CHAINed (random) access.

Column 66 is A to indicate that records are added to DACCOUNT.

3 This line specifies that DACCOUNT is part of the MARKET database.

Columns 54-65 identify MARKET as the TurboIMAGE database that is used.

Column 66 is L to indicate user-controlled manual locking.

Column 67 is C to specify chained sequential read mode.

4 This line defines the password for the database.

Columns 54-65 contain LEVEL WRITER to specify the password,

WRITER. This password establishes a user class identification that permits write access to the database.

5 This line identifies the key field for the data set.

Columns 54-65 contain ITEM ACCOUNT-NO to specify that the key for the data set is ACCOUNT-NO.

6 This line names the data set to be accessed.

Columns 54-65 contain DSNAMEA-ACCOUNT-NO to specify that the data set is D-ACCOUNTS.

7 This line begins the input record description of the IDACCT file.

8 This line begins the input record description of the DACCOUNT file.

9 This line reads the data set, D-ACCOUNTS, randomly.

Columns 18-27 contain ACTNO to specify the key field for reading the data set.

Columns 28-32 contain CHAIN to specify a chained read operation.

Columns 33-42 specify the name of the TurboIMAGE data set file, DACCOUNT.

Columns 54-55 contain 80 to turn on indicator 80 when a record is not found for the account number in D-ACCOUNTS.

Columns 56-57 contain the H0 indicator to avoid a compiler warning for the CHAIN operation. The end-of-chain indicator has no meaning since records are accessed randomly.

10 This line locks the data set, D-ACCOUNTS (DACCOUNT), when indicator 80 is turned on.

Columns 10-11 contain 80 to condition the LOCK operation.

Columns 28-32 contain LOCK to specify the lock operation.

Columns 33-42 contain DACCOUNT to identify the data set to lock.

Columns 54-55 contain 10 to specify that indicator 10 be turned on if the database, data set or a record in the data set is already locked by another process. You can enter a TurboIMAGE STATUS array in the File Description Continuation line to get additional information about the lock when indicator 10 is turned on.

Columns 56-57 contain 11 for the low resulting indicator. An indicator is required in this field. When turned on, this indicator signals a memory manager error.

Columns 58-59 contain 12 for the equal resulting indicator. An indicator is required in this field. When turned on, this indicator signals that the lock request was granted.

11 This line specifies that exception output is performed when indicator 80 is turned on (indicator 80 is turned on when a record is not found for an account number).

12 This line unlocks the data set, D-ACCOUNTS, when indicator 80 is turned on.

Columns 10-11 contain 80 to condition the UNLCK operation.

Columns 28-32 contain UNLCK to specify the unlock operation.

Columns 33-42 contain DACCOUNT to identify the data set to unlock.

Columns 56-59 contain the same indicators used in line 10 and they function the same way.

13 This line specifies the output operation for adding records to the D-ACCOUNTS data set.

Column 15 is E to identify this record as an exception record.

Columns 16-18 are ADD to add records to the TurboIMAGE data set.

Columns 32-37 are ADDREC to name the GROUP Name for the record to be added.

The following figure shows how to lock and unlock individual records in the D-ACCOUNTS data set.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
1  FIDACCT  IP  F 200 200            DISC
2  FDACCOUNTUC  F 200 200R 6AM     3 DISC                          A
3  F                                                 KIMAGE MARKETLC
4  F                                                 KLEVEL WRITER
5  F                                                 KITEM  ACCOUNT-NO
6  F                                                 KDSNAMED-ACCOUNTS

7  IIDACCT  NS  01
   I                                          3   8 ACTNO
   I                                          1 200 REC
8  IDACCOUNTNS  02

9  C           ACTNO     CHAINDACCOUNT              80H0
10 C   80      ACTNO     LOCK DACCOUNT              101112
11 C   80                EXCPT          ADDREC
12 C   80      ACTNO     UNLCKDACCOUNT              101112

13 ODACCOUNTEADD             ADDREC
   O                         REC      200
```

Figure 3-34. Manually Locking and Unlocking TurboIMAGE Data Set Records

Comments

1 This line defines the file, IDACCT.

2 This line defines the TurboIMAGE data set file, DACCOUNT.

Column 15 is U to indicate that DACCOUNT is updated.

Column 16 is C for CHAINed (random) access.

Column 66 is A to indicate that records are added to DACCOUNT.

3 This line specifies that DACCOUNT is part of the MARKET database.

Columns 54-65 identify MARKET as the TurboIMAGE database that is used.

Column 66 is L to indicate user-controlled manual locking.

Column 67 is C to specify chained sequential read mode.

4 This line establishes write access to the database.

Columns 54-65 contain LEVEL WRITER to specify write access to the database.

5 This line identifies the key field for the data set.

Columns 54-65 contain ITEM ACCOUNT-NO to specify that the key for the data set is ACCOUNT-NO.

6 This line names the data set to be accessed.

Columns 54-65 contain DSNAMEA-ACCOUNT-NO to specify that the data set is D-ACCOUNTS.

7 This line begins the input record description of the IDACCT file.

8 This line begins the input record description of the DACCOUNT file.

9 This line reads the data set, D-ACCOUNTS, randomly.

Columns 18-27 contain ACTNO to specify the key field for reading the data set.

Columns 28-32 contain CHAIN to specify a chained read operation.

Columns 33-42 specify the name of the TurboIMAGE data set file, DACCOUNT.

Columns 54-55 contain 80 to turn on indicator 80 when a record is not found for the account number in D-ACCOUNTS.

Columns 56-57 contain the H0 indicator to avoid a compiler warning for the CHAIN operation. The end-of-chain indicator has no meaning since records are accessed randomly.

10 This line locks individual records in the data set, D-ACCOUNTS (DACCOUNT).

Columns 18-27 contain ACTNO, which contains the key of the record to lock in DACCOUNT.

Columns 28-32 contain LOCK to specify the lock operation.

Columns 33-42 contain DACCOUNT to identify the data set to lock.

Columns 54-55 contain 10 for the high resulting indicator. An indicator is required in this field.

Columns 56-57 contain 11 for the low resulting indicator. An indicator is required in this field.

Columns 58-59 contain 12 for the equal resulting indicator. An indicator is required in this field.

11 This line specifies that exception output is performed when indicator 80 is turned on (indicator 80 is turned on when a record is not found for an account number).

12 This line unlocks individual records in the data set, D-ACCOUNTS.

Columns 18-27 contain ACTNO, which contains the key for the record to unlock in DACCOUNT.

Columns 28-32 contain UNLCK to specify the unlock operation.

Columns 33-42 contain DACCOUNT to identify the data set to unlock.

Columns 54-55 contain 10 to specify the high resulting indicator. An indicator is required in this field.

Columns 56-57 contain 11 to specify the low resulting indicator. An indicator is required in this field.

Columns 58-59 contain 12 to specify the equal resulting indicator. An indicator is required in this field.

13 This line specifies the output operation for adding records to the D-ACCOUNTS data set.

Column 15 is E to identify this record as an exception record.

Columns 16-18 are ADD to add records to the TurboIMAGE data set.

Columns 32-37 are ADDREC to name the GROUP Name for the record to be added.

Chapter 4 Using a Terminal in an RPG Program

This chapter discusses the two ways that you can use a terminal in an RPG program. The first method, line mode, lets you read and display terminal data field by field. It is useful when you are reading or displaying small amounts of data. The second method, full screen mode, is useful when you have several data fields to process, or when you want the screen to resemble a paper form of some kind.

This figure compares line mode and full screen mode from a user's point of view.

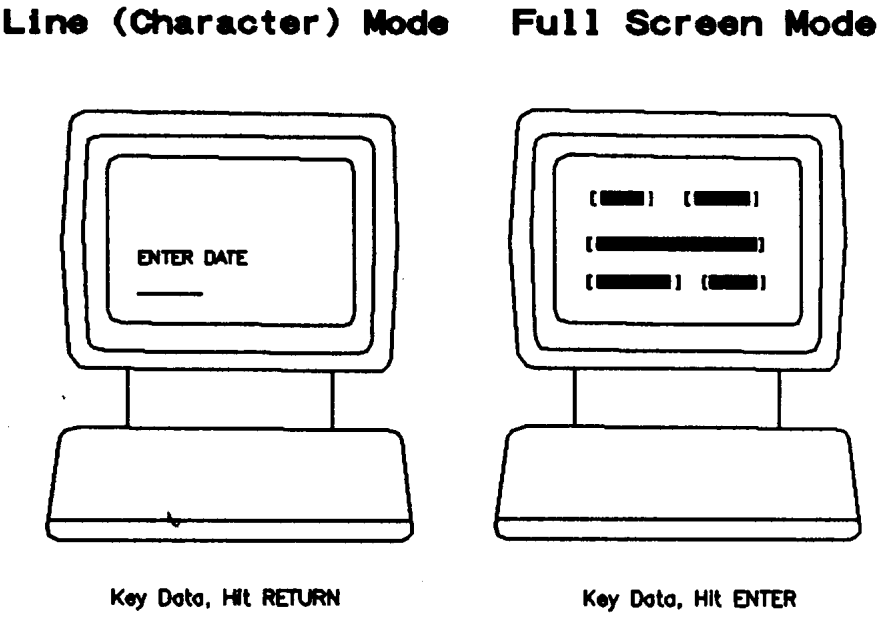


Figure 4-1. A Comparison of Line (Character) and Full Screen Modes

Using a Terminal in Line Mode

Line mode is the easiest way to read or display a small number of fields. For example, line mode can be the simplest way to read a date from the terminal. The "line mode" sections which follow in this chapter explain how to read and display data and how to use function keys and message files in line mode.

When using line mode, you normally use the system-defined files $STDIN for terminal input and $STDLST for terminal output. $STDIN and $STDLST are assigned to the devices shown below:

When running in this mode:	$STDIN, $STDLST have these device assignments:
Session	User terminal
Job	Job stream file, job stream list file

When necessary, you can redirect $STDIN and $STDLST to other devices by using the operating system FILE command.

Reading and Displaying Data

There are two ways to read and display data on a terminal. You can use the Calculation Specification operations READ and EXCPT or you can use

the Calculation Specification operations DSPLY or DSPLM.

The READ/EXCPT method is more flexible but requires more coding. DSPLY and DSPLM are simpler to use and they justify numeric input data. DSPLM also displays data from message files.

Using READ and EXCPT. One method of using a terminal to read and display data is to use the Calculation Specification operations, READ and EXCPT. You can use one READ operation to read several fields at once. EXCPT performs exception output to $STDLST.

If you want to reassign $STDIN and $STDLST to other devices, enter the appropriate operating system FILE equation(s) before running the program. For example, the following FILE equation creates the DATES file on disc and redirects $STDLST to it (see Figure 4-2 for an example of how $STDLST is used in the program),

```
:FILE STDLIST=DATES;SAVE;DEV=DISC;REC=-10,25,F,ASCII;DISC=25
```

Figure 4-2 shows how to use READ and EXCPT to read data from and write data to a terminal. The File Description Specifications assign the files INPUT and OUTPUT to the system-defined files, $STDIN and $STDLST, respectively. The READ operation reads a date in the format, MMDDYY, from the terminal. The date is converted to a YYMMDD format and this converted date is displayed using EXCPT (exception output).

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-----------------------------------------------------------------------------------------

1    FINPUT   ID  F   6   6            $STDIN
2    FOUTPUT  O   F  10  10            $STDLST

3    IINPUT   NS  01
     I                                      1   6OMMDDYY

4    C                    READ INPUT                        LR
     C           10000.01 MULT MMDDYY    YYMMDD  60
5    C                    SETON                        80
6    C                    EXCPT
     C                    SETOF                        80

7    OOUTPUT  E        80NLR
8    O                         YYMMDD    10
```

Figure 4-2. Using READ and EXCPT to Read and Display Terminal Data

Comments

1 This line defines the input file, INPUT, and assigns it to the system-defined file, $STDIN.

2 This line defines the output file, OUTPUT, and assigns it to the system-defined file, $STDLST.

3 This line begins the input record description for the file, INPUT.

4 This line reads 6 characters from the terminal and saves them in the field, MMDDYY.

Columns 58-59 direct RPG to turn on the LR indicator when the user enters an end-of-data signal (:EOD or :).

5 This line turns on the resulting indicator 80.

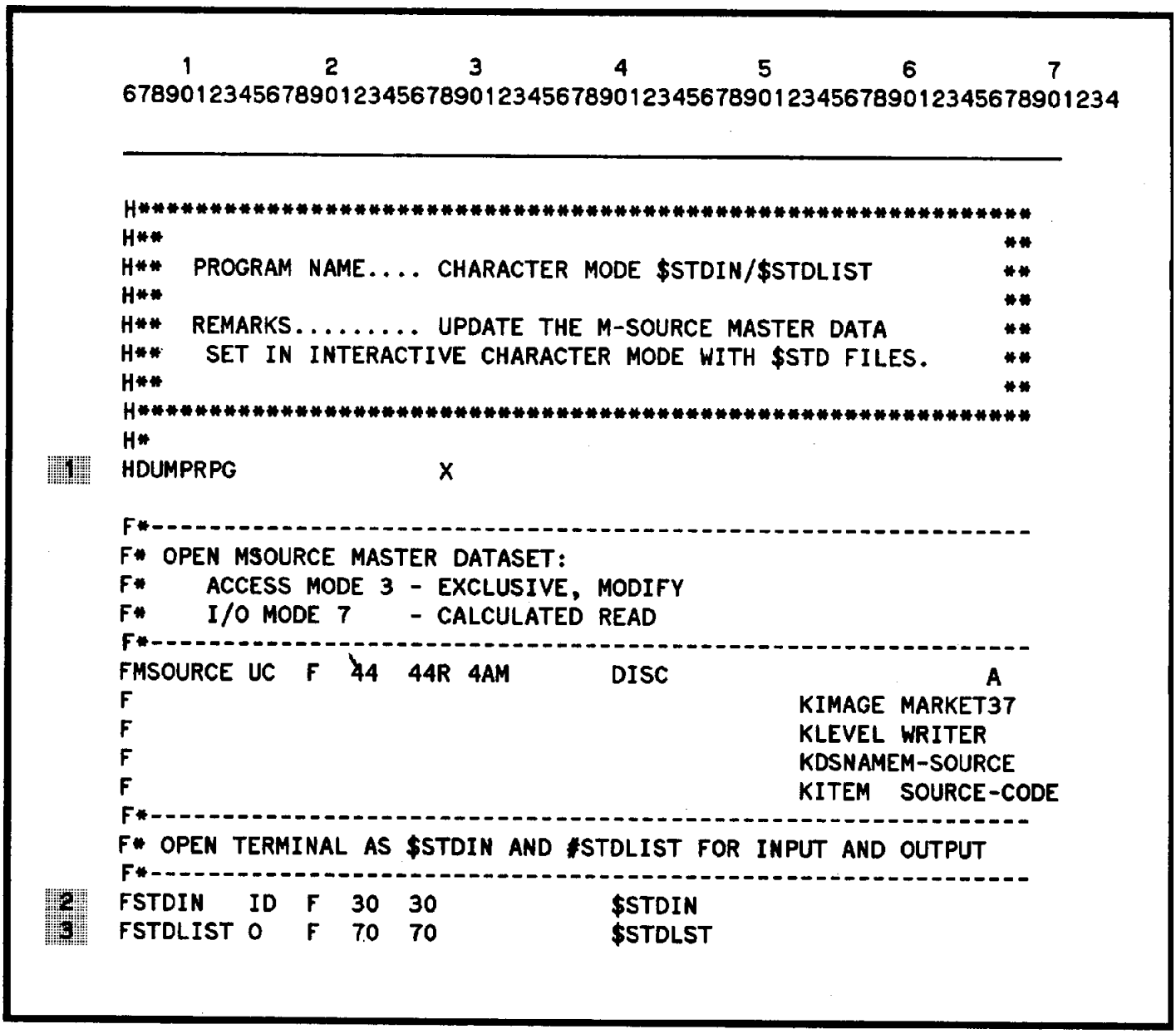
6 This line performs exception output (lines 7 and 8) when the user enters a valid date.

7 This line begins the output record description for the file, OUTPUT.

A Sample Program Using READ and EXCPT

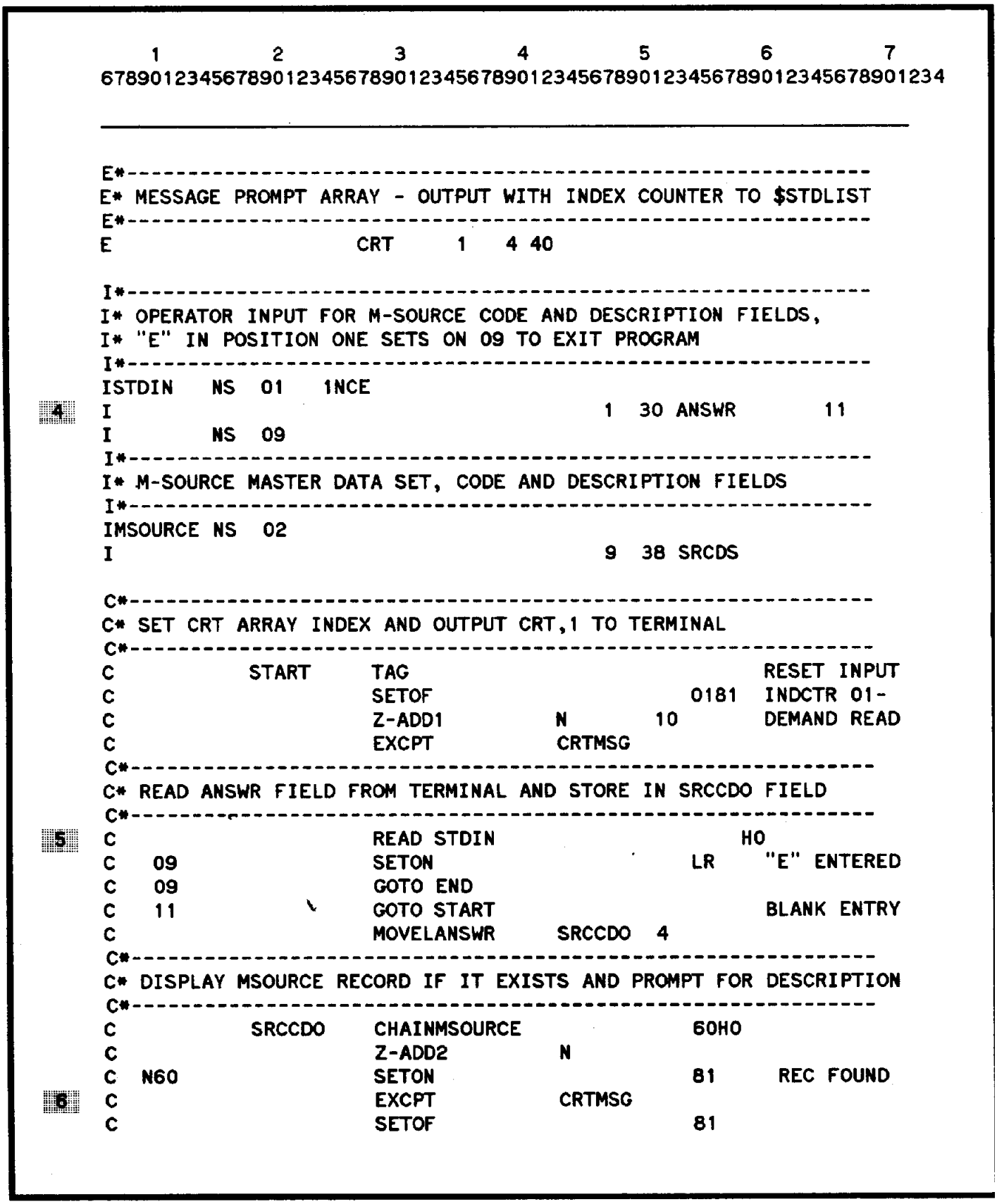
Figure 4-3 lists a program that updates an TurboIMAGE data set using READ and EXCPT. The data set that is updated is M-SOURCE (see the schema for this data set in Figure 3-21). The program prompts a terminal user to enter a source code (SRCCDO). It then reads the source code record in M-SOURCE. If the record exists, the source description field is updated. If the record does not exist, a new record is added for that source code. The next section, "Running the Sample Program," shows what a typical display looks like when the program is executed.

The program in Figure 4-6 is identical to that shown in Figure 4-3, except that Figure 4-6 uses DSPLY instead of READ/EXCPT. Comparing these programs should help you to understand the differences between READ/EXCPT and DSPLY/DSPLM.



```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

H*****************************************************************
H**                                                             **
H**  PROGRAM NAME.... CHARACTER MODE $STDIN/$STDLIST            **
H**                                                             **
H**  REMARKS......... UPDATE THE M-SOURCE MASTER DATA           **
H**   SET IN INTERACTIVE CHARACTER MODE WITH $STD FILES.        **
H**                                                             **
H*****************************************************************
H*
1  HDUMPRPG              X

F*-------------------------------------------------------------
F* OPEN MSOURCE MASTER DATASET:
F*    ACCESS MODE 3 - EXCLUSIVE, MODIFY
F*    I/O MODE 7    - CALCULATED READ
F*-------------------------------------------------------------
FMSOURCE UC  F  44  44R 4AM       DISC                      A
F                                              KIMAGE MARKET37
F                                              KLEVEL WRITER
F                                              KDSNAMEM-SOURCE
F                                              KITEM  SOURCE-CODE
F*-------------------------------------------------------------
F* OPEN TERMINAL AS $STDIN AND #STDLIST FOR INPUT AND OUTPUT
F*-------------------------------------------------------------
2  FSTDIN   ID  F  30  30            $STDIN
3  FSTDLIST O   F  70  70            $STDLST
```

Figure 4-3. Using READ and EXCPT to Update the M-SOURCE Data Set

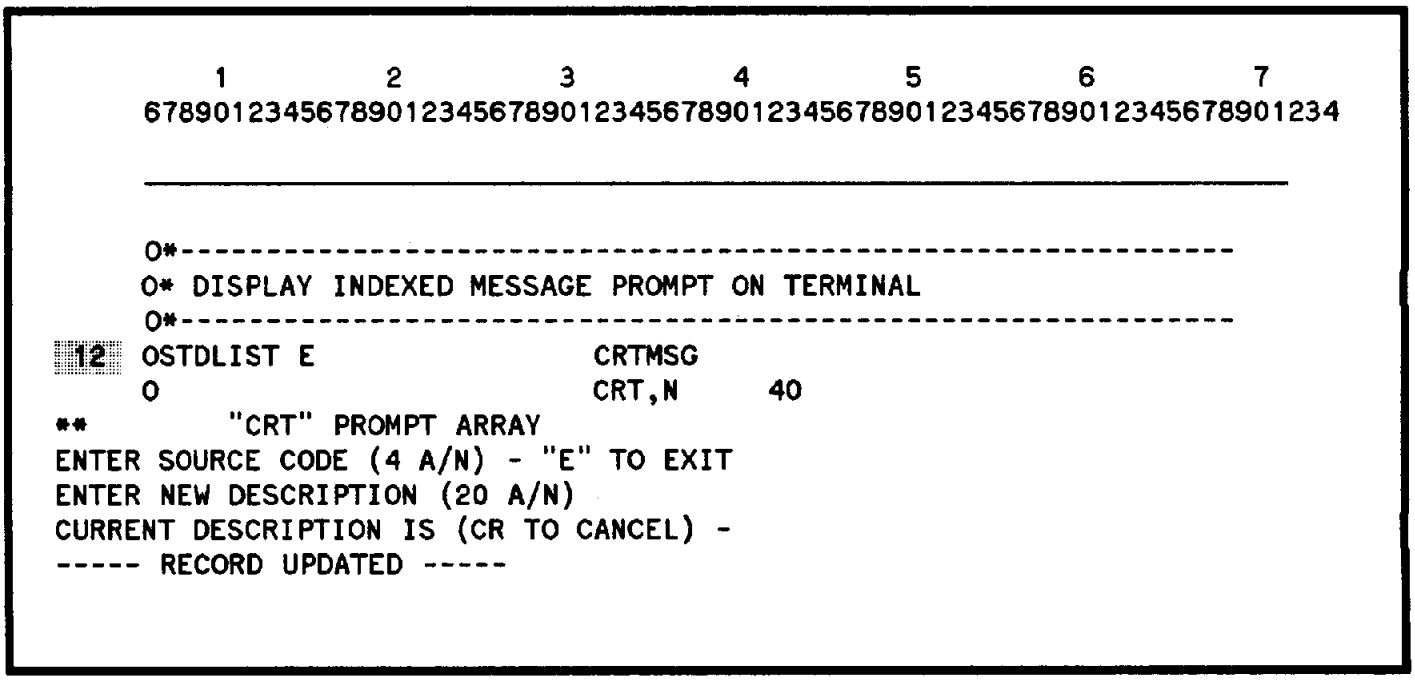


```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

E*-------------------------------------------------------------
E* MESSAGE PROMPT ARRAY - OUTPUT WITH INDEX COUNTER TO $STDLIST
E*-------------------------------------------------------------
E                    CRT     1   4 40

I*-------------------------------------------------------------
I* OPERATOR INPUT FOR M-SOURCE CODE AND DESCRIPTION FIELDS,
I* "E" IN POSITION ONE SETS ON 09 TO EXIT PROGRAM
I*-------------------------------------------------------------
ISTDIN   NS  01   1NCE
4 I                                        1  30 ANSWR        11
I        NS  09
I*-------------------------------------------------------------
I* M-SOURCE MASTER DATA SET, CODE AND DESCRIPTION FIELDS
I*-------------------------------------------------------------
IMSOURCE NS  02
I                                          9  38 SRCDS

C*-------------------------------------------------------------
C* SET CRT ARRAY INDEX AND OUTPUT CRT,1 TO TERMINAL
C*-------------------------------------------------------------
C           START     TAG                                RESET INPUT
C                     SETOF                         0181 INDCTR 01-
C                     Z-ADD1          N       10         DEMAND READ
C                     EXCPT           CRTMSG
C*-------------------------------------------------------------
C* READ ANSWR FIELD FROM TERMINAL AND STORE IN SRCCDO FIELD
C*-------------------------------------------------------------
5 C                   READ STDIN                        HO
C   09                SETON                         LR   "E" ENTERED
C   09                GOTO END
C   11                GOTO START                         BLANK ENTRY
C                     MOVELANSWR      SRCCDO  4
C*-------------------------------------------------------------
C* DISPLAY MSOURCE RECORD IF IT EXISTS AND PROMPT FOR DESCRIPTION
C*-------------------------------------------------------------
C           SRCCDO    CHAINMSOURCE              60HO
C                     Z-ADD2          N
C  N60                SETON                     81     REC FOUND
6 C                   EXCPT           CRTMSG
C                     SETOF                     81
```

Figure 4-3. Using READ and EXCPT to Update the M-SOURCE Data Set (Continued)

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________

     C*---------------------------------------------------------------
     C* READ ANSWR FIELD FROM TERMINAL AND STORE IN SRCDSO FIELD
     C*---------------------------------------------------------------
7    C                     READ STDIN                      HO
     C   11                GOTO START                        BLANK ENTRY
     C                     MOVELANSWR     SRCDSO 30
     C*---------------------------------------------------------------
     C* OUTPUT "RECORD UPDATED" MESSAGE
     C*---------------------------------------------------------------
     C                     Z-ADD4         N       10
8    C                     EXCPT          CRTMSG
     C           END       TAG
     C*---------------------------------------------------------------
     C* DROP THROUGH CALC SPECS TO DETAIL OUTPUT WHICH UPDATES MSOURCE
     C*---------------------------------------------------------------

     O*---------------------------------------------------------------
     O* ADD MSOURCE RECORD IF NONE EXISTED
     O*---------------------------------------------------------------
9    OMSOURCE DADD     01 60
     O                           SRCCDO     4
     O                           ZER09      8B
     O                           SRCDSO    38
     O                           ZER09     42B
     O*---------------------------------------------------------------
     O* UPDATE EXISTING MSOURCE RECORD WITH CHANGED DESCRIPTION
     O*---------------------------------------------------------------
10   OMSOURCE D        01N60
     O                           SRCDSO    38
     O*---------------------------------------------------------------
     O* DISPLAY CURRENT DESCRIPTION ON TERMINAL
     O*---------------------------------------------------------------
11   OSTDLIST E        81        CRTMSG
     O                           CRT,3     40
     O                           SRCDS     70
```

Figure 4-3. Using READ and EXCPT to Update the M-SOURCE Data Set (Continued)



```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234

     ----------------------------------------------------------------

     O*-------------------------------------------------------------
     O* DISPLAY INDEXED MESSAGE PROMPT ON TERMINAL
     O*-------------------------------------------------------------
12   OSTDLIST E                    CRTMSG
     O                             CRT,N      40
**          "CRT" PROMPT ARRAY
ENTER SOURCE CODE (4 A/N) - "E" TO EXIT
ENTER NEW DESCRIPTION (20 A/N)
CURRENT DESCRIPTION IS (CR TO CANCEL) -
----- RECORD UPDATED -----
```

Figure 4-3. Using READ and EXCPT to Update the M-SOURCE Data Set (Continued)

Comments

1 This Header Specification enables buffer-checking for all files.

Column 28 contains X to enable full buffer-checking for the files.

2 This line defines the terminal file, STDIN.

Columns 20-23 contain 30 to provide for the longest input field.

3 This line defines the terminal file, STDLIST.

Columns 20-23 contain 70 to provide enough space for the longest display line. RPG automatically pads with blanks to the end of the line.

4 This line defines the STDIN field, ANSWER.

5 This line reads data from the terminal. The user enters either a source code or an E to end the program.

This READ operation does not turn off indicators set by the previous READ. Indicator 01 must be turned off by a Calculation Specification to prevent last record output when E in column 1 turns on indicator 09.

6 This line performs exception output when a record is found in M-SOURCE.

7 This line reads data from the terminal and saves it in the field, ANSWR.

8 This line performs exception output to the terminal. The message contained in the CRT array, ----- RECORD UPDATED -----, is displayed.

9 This line begins the output record description for records that are added to M-SOURCE. New records are written when indicators 01 and 60 are turned on.

10 This line begins the output record description for records that

are updated in M-SOURCE. Records are updated when indicator 01 is turned on and indicator 60 is not turned on.

11 This line begins the output record description for displaying the source description. This record is displayed when an EXCPT CRTMSG calculation is executed and indicator 81 is turned on.

12 This line begins the output record description for prompting the user to enter data. This record is displayed when an EXCPT CRTMSG calculation is executed.

Running the Sample Program

Figure 4-4 shows what a terminal dialogue might look like when the sample program in Figure 4-3 is executed. A user at a terminal is updating source codes 111A and 112A. Two updates are required for source code 112A because the user made a mistake on the first try. The lines that the user enters are shaded.

```
:RUN MSRCUPD.PROGRAM
ENTER SOURCE CODE (4 A/N) - "E" TO EXIT
111A
ENTER NEW DESCRIPTION (20 A/N)
ADVERTISING - PRINT MEDIA 111
---- RECORD UPDATED ----
ENTER SOURCE CODE (4 A/N) - "E" TO EXIT
112A
ENTER NEW DESCRIPTION (20 A/N)
ADVERTISING PRINT MEDIA 112
---- RECORD UPDATED ----
ENTER SOURCE CODE (4 A/N) - "E" TO EXIT
112A
CURRENT DESCRIPTION IS (CR TO CANCEL) - ADVERTISING - PRINT MEDIA 112
ENTER NEW DESCRIPTION (20 A/N)
ADVERTISING - PRINT MEDIA 112
---- RECORD UPDATED ----
ENTER SOURCE CODE (4 A/N) - "E" TO EXIT
E
END OF PROGRAM
:
```

Figure 4-4. Running the M-SOURCE Update Program

Using DSPLY and DSPLM. The Calculation Specification operations, DSPLY and DSPLM, are alternative ways to read and display data using the terminal. They have two advantages over READ/EXCPT (see the section titled "Using READ and EXCPT"). First, numeric input data is stored right justified and zero-filled; users do not have to enter leading zeros for these fields. Second, one operation (either DSPLY or DSPLM) is used for both reading and displaying data.

DSPLY and DSPLM are used the same way except for the Factor 1 Field. For DSPLY,

Factor 1 contains either a variable or constant. For DSPLM, Factor 1 contains a message identification. DSPLM displays messages from a User Message Catalog (see the section titled "Using Message Files" for more information on how to display messages from a User Message Catalog file).

Figure 4-5 shows how to use DSPLY. A date in the format, YYMMDD, is read from the terminal. This date is converted to MMDDYY format and displayed.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

     H

1    FOUTPUT  O   F  80  80            $STDLST

     C                      Z-ADD0         YYMMDD  60
2    C            "YYMMDD"  DSPLY          YYMMDD
     C*  PROGRAM ENDS WHEN USER ENTERS 999999
     C            YYMMDD    COMP 999999                    01
     C   01                 SETON                          LR
     C*  CONVERT OTHER WAY- YMD TO MDY, E.G.,890318 BECOMES 031889
3    C  NLR       100.0001  MULT YYMMDD    MMDDYY  60
4    C  NLR       MMDDYY    DSPLY

     OOUTPUT  H        1P
     O                                    "DATE CONVERSION:"
```

Figure 4-5. Using DSPLY to Read and Display Terminal Data

Comments

1 This line defines the output file, OUTPUT.

Columns 40-46 contain the device class name, $STDLST.

2 This line displays the prompt, YYMMDD. It then reads 6 characters from the terminal and saves them in the field, YYMMDD.

3 The line converts the date into a MMDDYY format. For example, when 890318

is multiplied by 100.0001, the result is 89031889.0318. When saved in the result field, this number becomes 031889.

4 This line displays the converted date field, MMDDYY, on the terminal.

A Sample Program Using DSPLY

Figure 4-6 lists a program that updates an TurboIMAGE data set using DSPLY. The data set that is updated is M-SOURCE (see the schema for this data set in Figure 3-21). The program gets a source code (SRCCDO) from the terminal user, then reads the corresponding record in M-SOURCE. If the record exists, the source description field is updated. If the

record does not exist, a new record is added for that source code.

The update program in Figure 4-6 is the same as that shown for READ/EXCPT in Figure 4-3. Comparing these programs should help to clarify the differences between READ/EXCPT and DSPLY.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

H*************************************************************************
H**                                                                     **
H**  PROGRAM NAME.... CHARACTER MODE "DSPLY" CALCULATION                **
H**                                                                     **
H**  REMARKS..... UPDATE THE M-SOURCE MASTER DATA                       **
H**   SET IN INTERACTIVE CHARACTER MODE USING "DSPLY".                  **
H**                                                                     **
H*************************************************************************
H*

HDUMPRPG              X
F*-------------------------------------------------------------
F* OPEN MSOURCE MASTER DATA SET:
F*    ACCESS MODE 3 - EXCLUSIVE, MODIFY
F*    I/O MODE 7    - CALCULATED READ
F*-------------------------------------------------------------
FMSOURCE UC  F  44  44R 4AM       DISC                      A
F                                                KIMAGE MARKET37
F                                                KLEVEL WRITER
F                                                KDSNAMEM-SOURCE
F                                                KITEM  SOURCE-CODE
FTERMINALO   F  80  80            $STDLST

E*-------------------------------------------------------------
E* MESSAGE PROMPT ARRAY - OUTPUT WITH INDEX COUNTER TO TERMINAL
E*-------------------------------------------------------------
E                    CRT     1   4 40

I*-------------------------------------------------------------
I* M-SOURCE MASTER DATA SET, CODE AND DESCRIPTION FIELDS
I*-------------------------------------------------------------
```

Figure 4-6. Using DSPLY to Update the M-SOURCE Data Set

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

IMSOURCE NS  02
I                                             9  38 SRCDS
C*-------------------------------------------------------------
C* PROMPT USER WITH CRT ARRAY INDEX SET TO 1
C*-------------------------------------------------------------
C            START     TAG                                 RESET INPUT
C                      SETOF                          01
C                      MOVE *BLANK     SRCCDO  4
1 C            CRT,1     DSPLYTERMINAL  SRCCDO            SOURCE CODE
C            SRCCDO    COMP "E"                          LR EXIT PROG
C   LR                 GOTO END
C            SRCCDO    COMP *BLANK                       11 BLANK ENTRY
C   11                 GOTO START
C*-------------------------------------------------------------
C* DISPLAY MSOURCE RECORD IF IT EXISTS AND PROMPT FOR DESCRIPTION
C*-------------------------------------------------------------
C            SRCCDO    CHAINMSOURCE                60H0
2 C  N60       CRT,3     DSPLY                             REC FOUND
3 C  N60       SRCDS     DSPLY                             REC FOUND
C                      MOVE *BLANK     SRCDSO 30
4 C            CRT,2     DSPLY           SRCDSO
C            SRCDSO    COMP *BLANK                    11
C   11                 GOTO START                        BLANK ENTRY
C*-------------------------------------------------------------
C* OUTPUT "RECORD UPDATED" MESSAGE
C*-------------------------------------------------------------
5 C            CRT,4     DSPLY
C                      SETON                          01
C          END       TAG
C*-------------------------------------------------------------
C* DROP THROUGH CALC SPECS TO DETAIL OUTPUT WHICH UPDATES MSOURCE
C*-------------------------------------------------------------
CLR                    Z-ADD0          ZER10  100

O*-------------------------------------------------------------
O* ADD MSOURCE RECORD IF NONE EXISTED
O*-------------------------------------------------------------
OMSOURCE DADD     01 60
O                          SRCCDO     4
O                          ZER10      8B
O                          SRCDSO    38
O                          ZER10     42B
```

Figure 4-6. Using DSPLY to Update the M-SOURCE Data Set (Continued)

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

    O*-------------------------------------------------------------
    O* UPDATE EXISTING MSOURCE RECORD WITH CHANGED DESCRIPTION
    O*-------------------------------------------------------------
    OMSOURCE D        01N60
    O                           SRCDSO    38
**         "CRT" PROMPT ARRAY
ENTER SOURCE CODE (4 A/N) - "E" TO EXIT
ENTER NEW DESCRIPTION (20 A/N)
CURRENT DESCRIPTION IS (CR TO CANCEL) -
----- RECORD UPDATED -----
```

Figure 4-6. Using DSPLY to Update the M-SOURCE Data Set (Continued)

Comments

1 This line prompts the user to enter a source code. It then reads that source code into the field, SRCCDO.

2 This line displays the third element of the array, CRT. This element is a heading line for the source description.

3 This line displays the source description (SRCDS) retrieved from the M-SOURCE data set.

4 This line displays the second element in the array, CRT. This element prompts the user to enter a new description. Then the new description is saved in the field, SRCDSO.

5 This line displays the fourth element in the array, CRT. This element acknowledges that the M-SOURCE record is updated.

Using Function Keys

This section explains how to use function keys (F1 through F8) in an RPG program. It discusses how to sense them when they are pressed on the keyboard. If you want to display labels for the function keys, see the next section titled "Displaying Function Key Labels."

The Calculation Specifications shown in Figure 4-7 use F8 to end a program. This function key is activated at the beginning of the Calculation Specifications section. When the user presses it, the LR indicator is turned on and the program ends. (See Figure 4-12 for the listing of an entire program that uses function keys.)

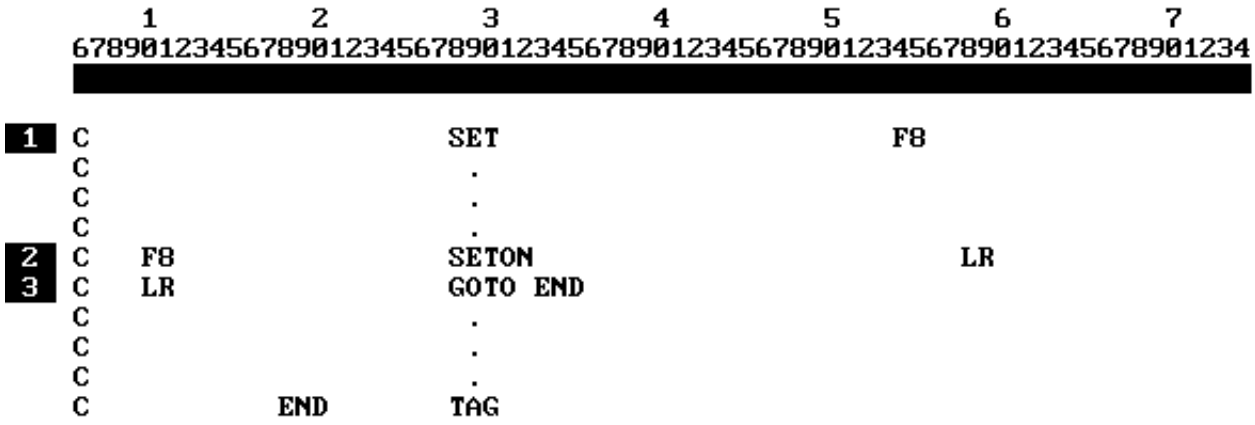


Figure 4-7. Using the Function Keys

Comments

1 This line enables function key F8 (SET F@ enables all of the function keys).

2 This line turns on the LR indicator when function key F8 is pressed.

3 This line directs program execution to the END tag when the LR indicator is turned on.

Displaying Function Key Labels

This section explains how to display function key labels. If you're using function keys in a program (function keys are discussed in the previous section) you may want to display labels for them also. Function key labels are displayed at the bottom of the screen and are highlighted.

Figure 4-8 shows how to activate function keys 4, 7 and 8 (F4 , F7 and F8) and how to display their labels. When these function key labels are displayed, they look like this:

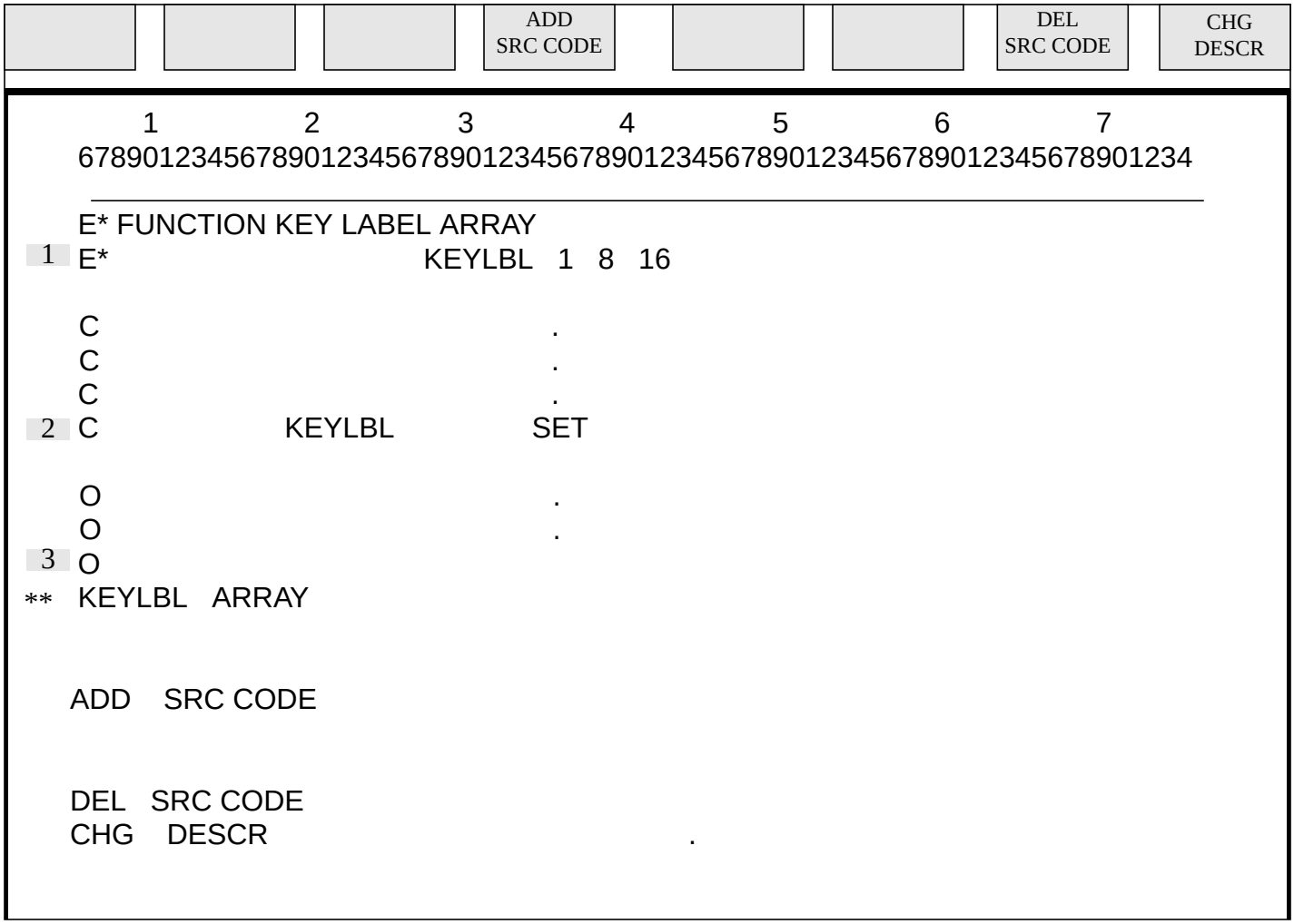


Figure 4-8. Displaying Function Key Labels

Comments

1 This line defines the array, KEYLBL, that contains the function key label text. There are 8 labels in the array, each label containing 16 characters.

2 This line enables the function keys F4 , F7 and F8 . It also displays the function key labels saved in the KEYLBL array.

3 This line is the last line of Output Specifications. Following this line are the contents of the function key labels array (KEYLBL).

Using Escape Sequences

Escape sequences let you perform many terminal-handling functions for which there is no corresponding RPG language facility. For example, you can use escape sequences to set graphics mode, move the cursor and reset the terminal.

The Calculation Specification operation, DSPLY, can be used to "display" escape sequences. You enter an escape sequence literal in the Factor 1 Field of a Calculation Specification. Figure 4-9 shows how to use escape sequences to home the cursor, clear the screen and release a memory lock. Instead of entering the escape sequences as literals in your program, you can enter them into a message file (see the next section for information on message files).

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123
-------------------------------------------------------------------

     C*          HOME CURSOR AND CLEAR SCREEN
1    C   N99      "^h^J"     DSPLY
     C                          .
     C                          .
     C                          .
     C*          TURN OFF MEMORY LOCK
2    C   N99      "^l"       DSPLY
```

Figure 4-9. Using Escape Sequences

Comments

1 This line displays the escape sequences for homing the cursor and clearing the screen. The escape character is indicated here by a ^.

2 This line displays the escape sequence for performing a memory lock. The escape character is indicated here by a ^.

Using Message Files

A message file is a convenient place to keep the text of messages displayed in an RPG program. You can alter the text of these messages when required without recompiling your program. Message files in RPG are called User Message Catalog files.

To use a message file, you must first create a *source* file containing the message text. Then, you use either MAKECAT or GENCAT to convert the source file to a User Message Catalog file. The message files created by MAKECAT and GENCAT are different. GENCAT message files have a compressed format, requiring less disc space, and they can be used with the Native Language Support (NLS) System. Although MAKECAT and GENCAT generate message files having different physical formats, you access messages in them the same way within an RPG program.

Creating a Message File. This section explains how to create a message source file and how to convert it into a User Message Catalog file using GENCAT. For details on using GENCAT, see the *Native Language Programmer's Guide*. For information on using MAKECAT to convert the file, see the *Message Catalogs Programmer's Guide*.

Figure 4-10 gives the terminal dialogue for creating a message source file using EDITOR and for executing GENCAT. (The lines that are shaded are the ones that a user enters.) Although this example uses EDITOR, you can use any editor that produces a standard text file. You organize messages into *sets* (see the $SET lines). Normally, sets contain similar kinds of messages. Within sets, you enter message text and associated message numbers. When a message is long, enter it according to how you want to display it. To display the message as a continuous string of characters, end message text lines with &. If you want to format the lines that are displayed, enter each line of text as you want it displayed, and end each line with a %.

As shown in Figure 4-10, GENCAT converts the message source file, CAT4, into a User Message Catalog file. It is saved with the name, CATALOG. (All User Message Catalog files have the name, CATALOG.)

```
:EDITOR
HP32201A.7.13 EDIT/3000  MON, SEP 16, 1985, 10:58 AM
(C) HEWLETT-PACKARD CO.  1982
/A
   1      $SET 1
   2      0001     DATE CONVERTER (TYPE '999999' TO END)
   3      0002 ENTER DATE (6.0 N) IN FORMAT MMDDYY
   4      0010 CONVERTED FORMAT IS:
   5      $SET 2
   6      0030     DATE CONVERTER (PRESS F8 THEN RETURN TO END)
   7      0031
   8      $SET 3
   9      0001 ENTER SOURCE CODE (R A/N, F8 TO EXIT):
  10      0002 ENTER NEW DESCRIPTION (20 A/N):
  11      0003 CURRENT DESCRIPTION IS (CR-CANCEL, F2-DELETE):
  12      0004 ----RECORD UPDATED----
  13      0005 ----RECORD DELETED----
  14      //
/K CAT4.SOURCE,UNN
/E
END OF SUBSYSTEM
:RUN GENCAT.PUB.SYS
ENTER INDEX OF DESIRED FUNCTION

     0.  EXIT
     1.  HELP
     2.  MODIFY SOURCE CATALOG
     3.  FORMAT SOURCE INTO FORMATTED CATALOG
     4.  EXPAND FORMATTED CATALOG INTO SOURCE
>>3
ENTER NAME OF SOURCE FILE TO BE FORMATTED
 >>CAT4.SOURCE
 FORMATTING....
ENTER NAME FOR NEW FORMATTED FILE
 >>CATALOG
TOTAL NUMBER OF SETS FORMATTED = 3
TOTAL NUMBER OF MESSAGES FORMATTED = 10
 FORMATTING SUCCESSFUL
```

Figure 4-10. Creating a User Message Catalog File

Reading a Message File. This section explains how to use the Calculation Specification operation, DSPLM, to read and display messages contained in a User Message Catalog file (see the previous section for information on how to create a User Message Catalog file).

Although this section does not show how to use the Calculation Specification operation, MSG, you can use it also to access messages in a User Message Catalog file. Instead of displaying a message, MSG saves it in a field specified by the program. MSG is useful when you need to tailor a message before displaying it.

Figure 4-11 shows how the messages shown in Figure 4-10 are accessed and displayed using DSPLM. You enter a message identification number with DSPLM that identifies a message in the User Message Catalog file. RPG retrieves that message and displays it. RPG assumes that the User Message Catalog name is CATALOG. If the file has another name, enter a system FILE command before running the program that equates the name to CATALOG. For instance, to equate the name, CAT4, enter this file equation,

```
FILE CATALOG=CAT4
```

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

     C                      Z-ADD0          MMDDYY  60
1    C            "1:1"     DSPLM
2    C            "2:1"     DSPLM           MMDDYY
     C            MMDDYY    COMP 999999                       LR
     C            10000.01  MULT MMDDYY     YYMMDD  60
3    C            "10:1"    DSPLM
     C            YYMMDD    DSPLY
```

Figure 4-11. Using a User Message Catalog File

Comments

1 This line displays message 1:1 from the User Message Catalog.

Columns 18-27 identify the message (1:1) in the User Message Catalog file (see line 2 in Figure 4-10). The identification number consists of two parts. The first part is the message number (1) within the set and the second part is the set number (1).

Columns 28-32 contain DSPLM to specify that the data to be displayed on the terminal comes from a User Message Catalog file.

2 This line displays message 2:1 from the User Message Catalog and accepts a date entered by the user.

Columns 18-27 identify the message (2:1) in the User Message Catalog file. (See line 1 for an explanation of the message identification.)

Columns 28-32 contain DSPLM to specify that the data to be displayed on the terminal comes from the User Message Catalog file.

Columns 43-48 contain the name of the field, MMDDYY, where the user-entered date is stored.

3 This line displays message 10:1 from the User Message Catalog.

Columns 18-27 identify the message (10:1) in the User Message Catalog file. (See line 1 for an explanation of the message identification.)

Columns 28-32 contain DSPLM to specify that the data to be displayed on the terminal comes from the User Message Catalog file.

A Sample Program Using Message Files. Figure 4-12 lists a program that uses the User Message Catalog file created in Figure 4-10. This message file contains user prompts and informational messages.

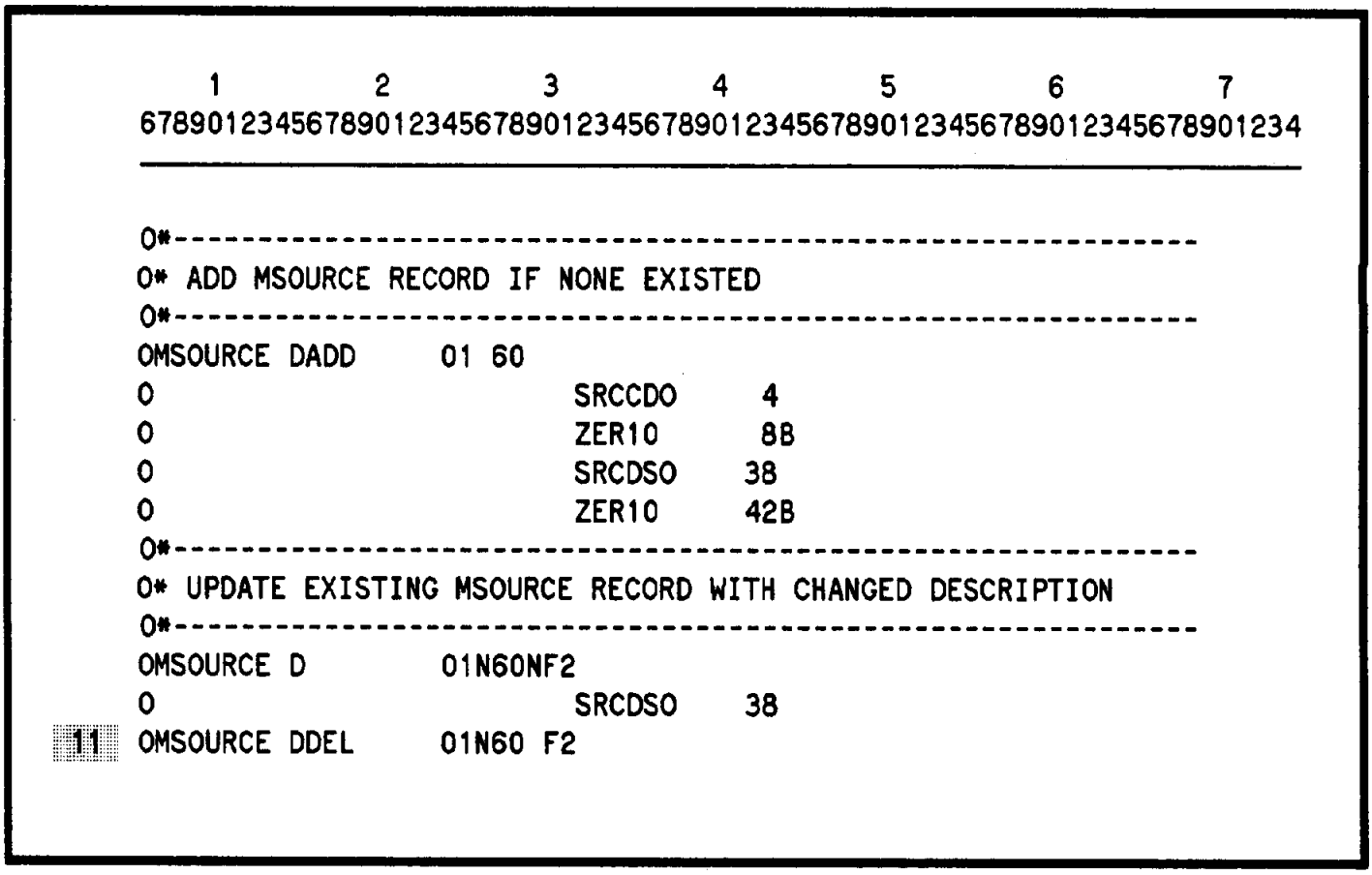
The program updates the TurboIMAGE data set, M-SOURCE. (see the schema for this data set in Figure 3-21). The program gets a source code (SRCCDO) from the terminal user, then reads the corresponding record in M-SOURCE. If the record exists, the source description field is updated. If the record does not exist, a new record is added for that source code. Users delete records by pressing F2 and end the program by pressing F8 . (After entering data or pressing a function key, users must press RETURN .)

```
    1        2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

$CONTROL NAME=SF0412
H*************************************************************************
H**                                                                     **
H**  PROGRAM NAME.... CHARACTER MODE "DSPLM" WITH CATALOG               **
H**                                                                     **
H**  REMARKS..... UPDATE THE M-SOURCE MASTER DATA                       **
H**   SET IN INTERACTIVE CHARACTER MODE USING "DSPLM" AND               **
H**   THE CATALOG FILE CONCEPT.                                         **
H**                                                                     **
H**   ALSO ADD FUNCTION KEY INDICATORS AND DELETE CAPABILITY.           **
H**                                                                     **
H*************************************************************************
H*
1  HDUMPRPG              X                    B   X

F*-------------------------------------------------------------
F* OPEN MSOURCE MASTER DATA SET:
F*    ACCESS MODE 3 - EXCLUSIVE, MODIFY
F*    I/O MODE 7    - CALCULATED READ
F*-------------------------------------------------------------
FMSOURCE UC  F  44  44R 4AM       DISC                      A
F                                                KIMAGE MARKET37
F                                                KLEVEL WRITER
F                                                KDSNAMEM-SOURCE
F                                                KITEM  SOURCE-CODE

I*-------------------------------------------------------------
I* M-SOURCE MASTER DATA SET, CODE AND DESCRIPTION FIELDS
I*-------------------------------------------------------------
IMSOURCE NS  02
```

Figure 4-12. Updating the M-SOURCE Data Set Using a User Message Catalog File and the Function Keys

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
     ------------------------------------------------------------------------------------

     I                                          9  38 SRCDS
     C*-------------------------------------------------------------
     C* 1ST RECORD - ALLOW FUNCTION KEYS F2 AND F8
     C*-------------------------------------------------------------
     C                     MOVE *ZERO     ZER10  100
  2  C  N99                SET                       F2F8
     C  N99                SETON                     99
     C* PROMPT USER WITH "DSPLM" OPERATION; MESSAGE 1, SET 3
     C*-------------------------------------------------------------
     C           START     TAG                             RESET INPUT
  3  C                     SETOF                     01F2
     C                     MOVE *BLANK    SRCCDO  4
  4  C           "01:3"    DSPLM          SRCCDO           SOURCE CODE
     C   F8                SETON                        LR F8 - EXIT
  5  C   LR                GOTO END
     C           SRCCDO    COMP *BLANK                  11 BLANK ENTRY
     C   11                GOTO START                     BLANK ENTRY
     C*-------------------------------------------------------------
     C* DISPLAY MSOURCE RECORD IF IT EXISTS AND PROMPT FOR DESCRIPTION
     C*-------------------------------------------------------------
     C           SRCCDO    CHAINMSOURCE              60H0
  6  C  N60      "03:3"    DSPLM                           REC FOUND
     C  N60      SRCDS     DSPLY                           REC FOUND
     C                     MOVE *BLANK    SRCDSO 30
  7  C           "02:3"    DSPLM          SRCDSO
  8  C  NF2      SRCDSO    COMP *BLANK                 11
     C  NF2 11             GOTO START                     BLANK ENTRY
     C*-------------------------------------------------------------
     C* OUTPUT "RECORD UPDATED" OR "RECORD DELETED" MESSAGE
     C*-------------------------------------------------------------
  9  C  NF2      "04:3"    DSPLM
 10  C   F2      "05:3"    DSPLM
     C                     SETON                     01
     C           END       TAG
     C*-------------------------------------------------------------
     C* DROP THROUGH CALC SPECS TO DETAIL OUTPUT WHICH UPDATES MSOURCE
     C*-------------------------------------------------------------
```

Figure 4-12. Updating the M-SOURCE Data Set Using a User Message Catalog File and the Function Keys (Continued)



```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

0*-------------------------------------------------------------
0* ADD MSOURCE RECORD IF NONE EXISTED
0*-------------------------------------------------------------
OMSOURCE DADD     01 60
0                         SRCCDO     4
0                         ZER10      8B
0                         SRCDSO    38
0                         ZER10     42B
0*-------------------------------------------------------------
0* UPDATE EXISTING MSOURCE RECORD WITH CHANGED DESCRIPTION
0*-------------------------------------------------------------
OMSOURCE D        01N60NF2
0                         SRCDSO    38
OMSOURCE DDEL     01N60 F2
```

Figure 4-12. Updating the M-SOURCE Data Set Using a User Message Catalog File and the Function Keys (Continued)

Comments

1 This line suppresses the display of DSPLY literals and causes all terminal input to be entered on a separate line on the screen.

2 This line enables function keys F2 (delete a record) and F8 (end).

3 This line turns F2 off. (You must specifically turn off function keys.)

4 This line displays message 01:3 from the User Message Catalog file (see line 9 in Figure 4-10).

5 This line ends the program when the user presses F8 .

6 This line displays message 03:3 from the User Message Catalog file (see line 11 in Figure 4-10).

7 This line displays message 02:3 from the User Message Catalog file (see line 10 in Figure 4-10).

8 This line compares SRCDSO (source description) to blanks when F2 is not turned on.

Columns 9-11 contain NF2 to perform the compare when F2 is not pressed. (SRCDSO is blank even when F2 is pressed.)

9 This line displays message 04:3 from the User Message Catalog file when F2 is not pressed (see line 12 in Figure 4-10).

10 This line displays message 05:3 from the User Message Catalog

file when F2 is pressed (see line 13 in Figure 4-10).

11 This line describes the output record format for a delete operation. A record is deleted in M-SOURCE when the user presses F2 .

Using a Terminal in Full Screen Mode

When you have several fields of data to process using a terminal, you may find it easier to work in full screen mode rather than line mode. Full screen (or block) mode lets you read and display entire screens of data at one time.

There are three software tools that let you use the terminal in full screen mode; VPLUS, RPG Screen Interface (RSI) and the RSI CONSOLE facility. VPLUS is a general-purpose forms management system that interfaces with many programming languages. RSI is an RPG utility that creates forms similar to those used on IBM S/34 and S/36. RSI CONSOLE facility lets you use certain RSI functions for processing input files only (CONSOLE files are ideal for simple data entry applications).

The following list compares the features of VPLUS and RSI:

VPLUS	RSI
Uses actions and events to manage input/output	Uses standard RPG specifications to manage input and output
Can be used with other languages and applications	Most easily used with RPG programs
Does not let you use User Message Catalog files for messages	Lets you use User Message Catalog files for messages
Cannot dynamically change screen attributes (for example, blinking and inverse video)	Can dynamically change screen attributes
No automatic conversion to VPLUS	Converts IBM S and D Specifications to RSI-compatible format
No automatic conversion to VPLUS	Automatically generates RPG program specifications from RSI form files

This chapter explains how to use VPLUS, RSI and RSI CONSOLE files in RPG programs. To help you compare VPLUS and RSI, the same program example is used for each (the example differs only in how terminal input and output is handled). The last three section in this chapter, starting with "Enabling the BREAK Key," describe RPG features that you can use with both VPLUS and RSI.

NOTE Although you can use most VPLUS features in an RPG program, some are not available. This chapter and the *HP Reference Manual* describe the features that are available when you use the RPG Interface to VPLUS.

Using VPLUS

When you use VPLUS in an RPG program, follow these procedures:

* **Create a VPLUS form**

  Before using VPLUS to process data full screen mode, use the FORMSPEC facility of VPLUS to create a screen form. When you create a form, you determine the layout of fields on the screen and specify the type of data each field holds. You can also enter titles and field display characteristics.

* **Define a VPLUS form**

When you're finished designing the VPLUS form, enter the following specifications in the RPG program to define the form.

File Description Specifications -

Enter the specifications for a terminal file. This is the forms file you created with FORMSPEC. Specify that it is a WORKSTN file.

Input Specifications -

Enter input records describing the fields in the VPLUS form. Also enter one or more records for processing VPLUS *event* codes. (You use event codes to identify the type of input the user entered.)

Output Specifications -

Define an output record describing the VPLUS form and one describing the message window (the message window is used for displaying one-line messages at the bottom of the screen). You may also need to define a separate output record for processing VPLUS *action* codes. (You use action codes to specify the VPLUS operation to perform.)

* **Perform a VPLUS action or return a VPLUS event**

Calculation Specifications -

For each VPLUS operation you want to perform, enter Calculation Specifications to identify and execute it as follows:

1. Enter a MOVE operation to move the VPLUS action code (for the operation you want to perform) to the first field of the terminal file's output record.

   VPLUS has several action codes that you can use. Here are the ones that are discussed in this chapter:

Action code:	Description:
CHGNXT	Changes VPLUS forms.
EDITS	Performs the data edits specified in FORMSPEC. VPLUS returns an edit error count to the program.
GETDTA	Moves data from the VPLUS buffer to the program's input record.
GETNXT	Gets the next form from the VPLUS forms file.
INIT	Initializes form fields according to the edits specified in FORMSPEC.
PUTDTA	Moves data from the terminal output record to the VPLUS buffer.
RDTERM	Reads screen data into the VPLUS buffer.
SHOMSG	Displays a message in the message window.

   For example, the operation MOVEL "GETDTA" ACTION moves the GETDTA action code to the output field, ACTION.

2. Once you move the VPLUS action code to the output record, perform exception output (EXCPT operation) for that record. VPLUS performs the action and returns control to your program.

3. To complete input actions (for example, GETDTA), enter a READ

operation for the terminal file. VPLUS returns an event code in the first field of the terminal input record. You use this event code to identify the type of input received. Here are some of the codes that may be returned:

Event code:	Description: \00\The user entered data and pressed ENTER .
01-08	The user pressed F1 through F8 .
09	An EDITS action was performed and VPLUS returned the number of fields that failed the VPLUS edits.

For example, if you performed a GETDTA action, you follow it with READ TERMINAL operation.

The VPLUS sections in this chapter explain how to perform the VPLUS procedures introduced above. They do not cover all of the VPLUS actions or the ways to use VPLUS, but they should give you a good start. See the *HP RPG Reference Manual* for rules on using the RPG interface to VPLUS.

Many of the following VPLUS sections in this manual use examples taken from the program listed in the section "A Sample Program Using VPLUS." You may find it helpful to refer to this program to get a clear idea of how the examples can be used together to form a complete program.

Creating a VPLUS Form. Before you can use VPLUS in a program, you must create a screen form using the VPLUS facility, FORMSPEC. Although this section shows a typical screen created by FORMSPEC, it does not explain how to create it. For instructions on how to use FORMSPEC, see the *Data Entry and Forms Management System VPLUS/3000* manual.

Figure 4-13 shows what a FORMSPEC screen form looks like. Titles, captions and headings are capitalized. Fields, used for entering data, are enclosed in brackets.

```
                    MARKET DATA BASE
                  ACCOUNT INFORMATION


ACCOUNT NO [actno ]

LAST NAME  [lname           ]  FIRST NAME [fname     ]
PHONE      [phone     ]          EXTENSION  [phex]

ADDRESS 1   [addr1                        ]
ADDRESS 2   [addr2                        ]
ADDRESS 3   [addr3                        ]
ZIP         [zipcd     ]

LAST CONTACT [adate    ]     TYPE    [ty]
MAIL CODES   [mailc    ]
BEGIN DATE   [bdate    ]     SOURCE  [srcc]
date format - mmddyyyy

 PREV FORM | NEXT FORM |  | REFRESH |  | PREV | NEXT | MAIN/ RESUME | EXIT
```

Figure 4-13. Creating a VPLUS Form Using FORMSPEC

Figure 4-14 shows what the screen form (in Figure 4-13) looks like when displayed from an RPG program. Input fields are highlighted.

The line at the bottom of the screen is the message window. It is used to display one-line messages.

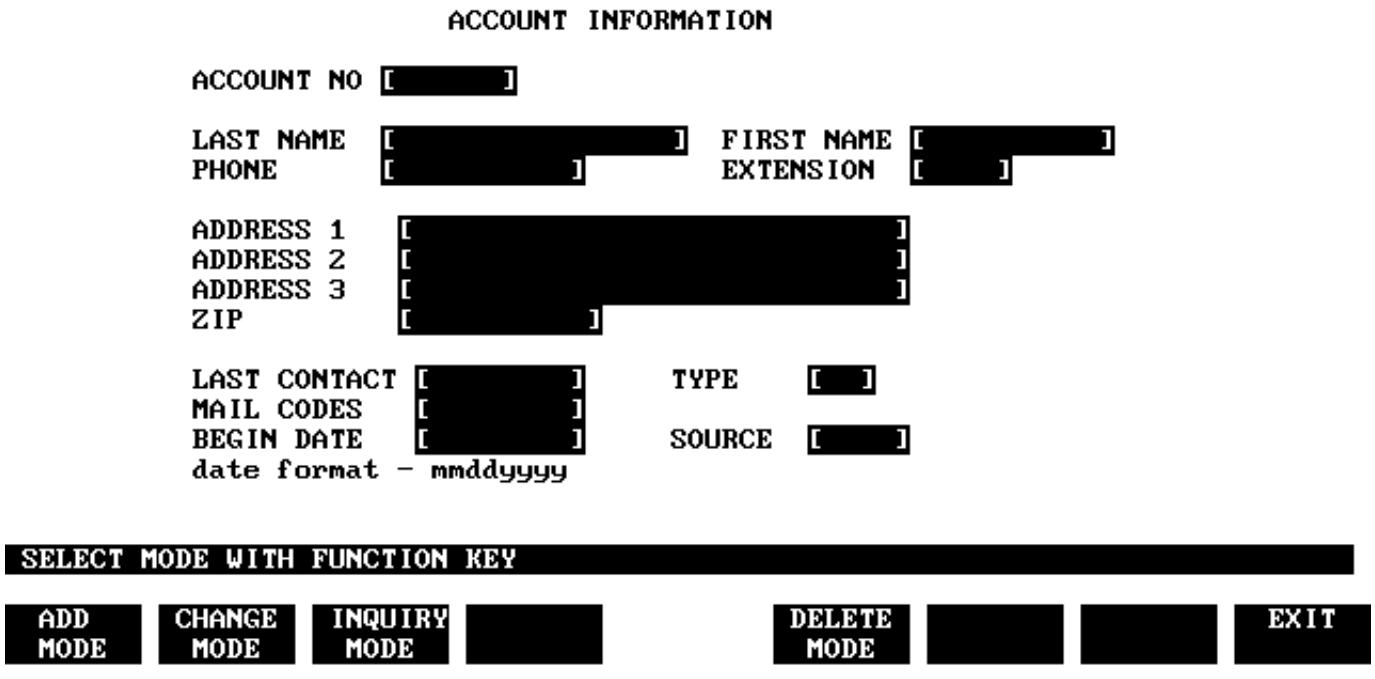


Figure 4-14. Using a VPLUS Form Within an RPG Program

The form shown in Figure 4-14 is used in examples throughout the VPLUS sections in this chapter. The form is used to update the D-ACCOUNTS data set (see the schema for the MARKET database in Figure 3-23.

Defining a VPLUS Form. To use a VPLUS form in an RPG program, enter a File Description Specification for it, defining it as a WORKSTN file. Also enter Input and Output Specifications that describe the form as well as the action and event records that let you communicate with VPLUS. Only define those fields on the form that you're actually using in the program. You can get the field starting and ending locations from the event and action record formats described in the *HP RPG Reference Manual* and the forms file listing produced by FORMSPEC.

Figure 4-15 shows how to define a VPLUS form. The form used is the one in Figure 4-14. Notice that there are two input records for the file, TERMINAL. The first is used for processing event code 09. (To review events, see the section titled "Using VPLUS.") The second input record defines the VPLUS form fields used in the program. There are three output records. The first is used to initiate VPLUS actions other than reading data from the screen. (To review VPLUS actions, see the section titled "Using VPLUS.") The second output record is used for displaying screen data and the third defines the fields in the message window (see the section "Displaying VPLUS Messages" for information about the message window).

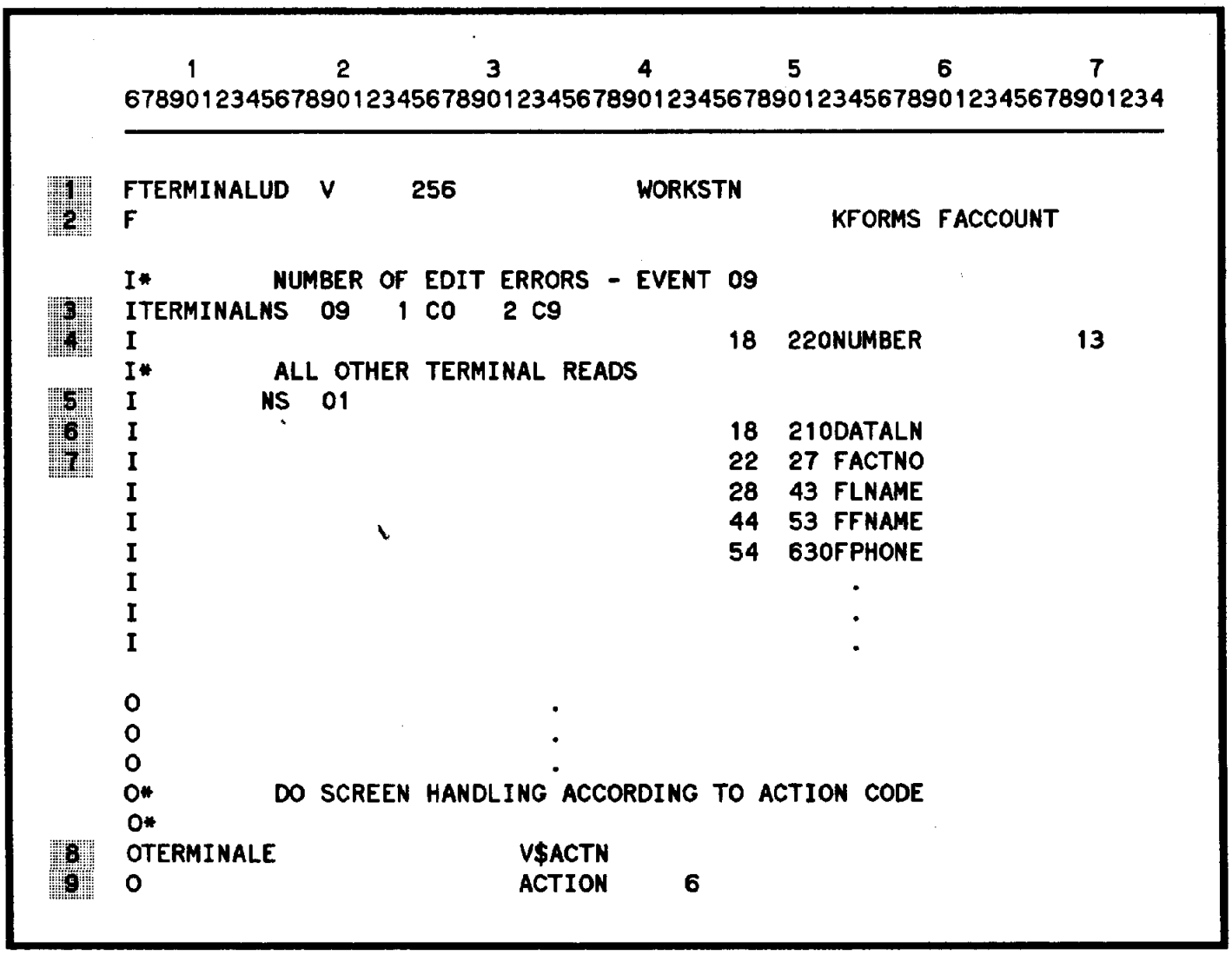
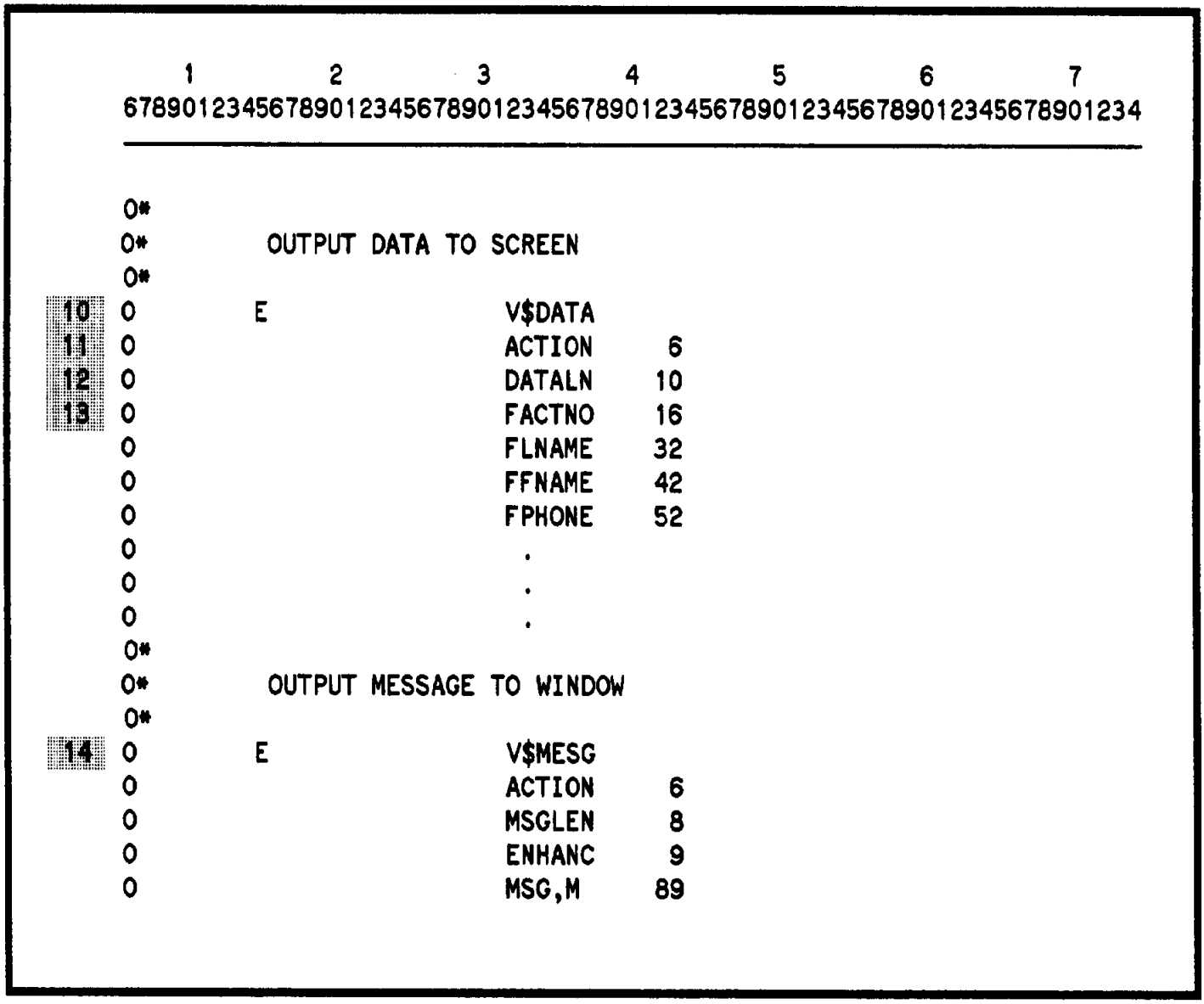


Figure 4-15. Defining a VPLUS Form



```
      1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

     O*
     O*         OUTPUT DATA TO SCREEN
     O*
10   O         E                 V$DATA
11   O                           ACTION     6
12   O                           DATALN    10
13   O                           FACTNO    16
     O                           FLNAME    32
     O                           FFNAME    42
     O                           FPHONE    52
     O                            .
     O                            .
     O                            .
     O*
     O*         OUTPUT MESSAGE TO WINDOW
     O*
14   O         E                 V$MESG
     O                           ACTION     6
     O                           MSGLEN     8
     O                           ENHANC     9
     O                           MSG,M     89
```

Figure 4-15. Defining a VPLUS Form (Continued)

Comments

1 This line defines a WORKSTN (terminal) file.

Column 15 is U to specify update processing.

Column 16 is D to specify that this is a demand file.

Column 19 is V to specify variable length records (you must use V for terminal files).

Columns 24-27 specify that the record length is 256 characters (the record length must be at least 20 characters longer than the longest record in the file).

Columns 40-46 specify that the file is a WORKSTN file.

2 This line identifies the VPLUS forms file used in the program.

Column 53 is K to specify that this is a File Description Continuation line.

Columns 54-59 specify that this line is a FORMS File Description Continuation line.

Columns 60-65 contain the name of the forms file, FACCOUNT. Figure 4-14 shows what this form looks like.

3 This line defines input record type 09. It is used to process event code 09.

Columns 21-24 specify that the record core in columns 1-2 of each record is 09.

4 This line defines the field, NUMBER. NUMBER contains the number of edit errors returned by VPLUS.

Columns 69-70 contain 13 to turn on indicator 13 when there are no edit errors.

5 This line defines input record type 01. This record describes the VPLUS screen form and is used to read data from the terminal.

6 This line defines the field, DATALN, that contains the number of bytes read from the screen.

7 This line starts the descriptions of the fields in the VPLUS. form.

8 This line defines the output record that is written when an EXCPT V$ACTN calculation is executed. This may occur, for example, when you request VPLUS to edit screen data.

9 This line defines the field, ACTION, which contains the VPLUS action code.

10 This line defines the output record to be written when an EXCPT V$DATA calculation is executed. This may occur, for example, when you want to display a form containing data.

11 This line defines the field, ACTION, which contains the VPLUS action code.

12 This line defines the field, DATALN, which contains the number of bytes of data that VPLUS displays on the screen.

13 This line starts the field description for the VPLUS form.

14 This line starts the description of the fields in the message window. See the section "Displaying VPLUS Messages" for information about message window fields.

Displaying an Initialized VPLUS Form. This section explains how to load a form from a VPLUS forms file, initialize it with FORMSPEC values and display it. You must display an initialized form before a user can enter data from the terminal.

The discussion in this section assumes that you are familiar with VPLUS action and event codes. To review these topics, see the third section titled, "Using VPLUS."

Figure 4-16 shows the steps for displaying an initialized VPLUS form. (Refer to Figure 4-15 for descriptions of the terminal input and output records used here.) The first step in displaying an initialized VPLUS form is to set appropriate values in the message window fields, DATALN and ENHANC. The second step is to enter the VPLUS action, GETNXT, to

retrieve the form from the VPLUS forms file. The third step is to enter the VPLUS action, INIT, to place FORMSPEC values in the form fields. The fourth step is to enter the VPLUS action, SHOMSG, to clear the VPLUS message window. And the last step is to use the VPLUS action, SHOW, to display the form with its initial values.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
                                                                                
1    C                    MOVE 256        DATALN  40        INIT DATALN
2    C                    MOVE "J"        ENHANC  1         INIT ENHANC
     C                      .
     C                      .
     C                      .
3    C                    MOVEL"GETNXT"   ACTION  6         GET FORM
4    C                    EXCPT           V$ACTN
5    C                    MOVEL"INIT  "   ACTION            INIT VALUES
6    C                    EXCPT           V$ACTN
7    C                    Z-ADD3          M                 CLEAR
8    C                    MOVEL"SHOMSG"   ACTION            WRITE WINDOW
9    C                    EXCPT           V$MESG
10   C                    MOVEL"SHOW  "   ACTION            DISPLAY FORM
11   C                    EXCPT           V$ACTN
```

Figure 4-16. Displaying a VPLUS Form

Comments

1 This line initializes the DATALN field in the message window record with the screen length. This operation is done once, when the form is first displayed.

2 This line initializes the message window field, ENHANC, with the character J. J is inverse video, half bright. ENHANC is initialized once, when the form is first displayed.

3 This line enters the VPLUS action, GETNXT, into the ACTION output field. GETNXT retrieves the form from the forms file and places it into the VPLUS buffer.

4 This line performs exception output for the record associated with EXCPT Group V$ACTN.

5 This line enters the VPLUS action, INIT, into the ACTION output field. INIT initializes the VPLUS buffer fields with FORMSPEC values.

6 This line performs exception output for the record associated with EXCPT Group V$ACTN.

7 This line resets the message array pointer, M, to 3. Assuming that the third element of the message array is a blank line, the window is cleared.

8 This line enters the VPLUS action, SHOMSG, into the ACTION output field. SHOMSG displays a message in the VPLUS message window.

9 This line performs exception output for the record associated with EXCPT Group V$MESG.

10 This line enters the VPLUS action, SHOW, into the ACTION output field. SHOW displays the form in the VPLUS buffer.

11 This line performs exception output for the record associated with EXCPT Group V$ACTN.

Reading a VPLUS Form. Once you display an initialized VPLUS form (see the previous section), a user can start entering data into it from the terminal. This section explains how to read that data into an RPG program.

Reading screen data involves three steps. The first step is to read screen data into the VPLUS buffer. The second step is to direct VPLUS to edit the data according to the edit values entered with FORMSPEC. The last step is to move the screen data from the VPLUS buffer to fields in the program.

Figure 4-17 shows how to perform these steps. (Refer to Figure 4-15 for a description of the terminal input record associated with EXCPT Group V$DATA.) You read screen data into the VPLUS buffer by using the VPLUS action, RDTERM, followed by a READ operation (READ returns the event code for the action). To perform the second step, editing the data, enter the VPLUS action, EDITS, and a READ operation (READ returns the event code for the action). VPLUS does the editing specified in FORMSPEC. If you do not want VPLUS to perform the edits or you want to edit the data yourself, you may omit this step. When VPLUS finishes the edits and returns to the program, it gives you a count of the number of errors. If there are errors, you redisplay the VPLUS form so that the user can correct mistakes (in this figure, the SHOW action in line 3 is executed). To perform the last step, moving data from the VPLUS buffer to the program's input record, enter the VPLUS action, GETDTA, followed by a READ operation (READ returns the event code for the action).

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
                                                                                
1    ITERMINALNS  09   1 C0   2 C9
2    I                                          18  220NUMBER      13

     C                     .
     C                     .
     C                     .
     C           RDSCR1    TAG
3    C                     MOVEL"SHOW  "  ACTION           DISPLAY FORM
     C                     EXCPT          V$ACTN
     C                     .
     C                     .
     C                     .
4    C                     MOVEL"RDTERM"  ACTION           READ VBUFFER
5    C                     EXCPT          V$ACTN
6    C                     READ TERMINAL                 H0 READ RECORD
7    C  NF0                GOTO RDSCR7
8    C                     MOVEL"EDITS "  ACTION           EDIT DATA
9    C                     EXCPT          V$ACTN
10   C                     READ TERMINAL                 H0
_
```

Figure 4-17. Reading a VPLUS Form

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
                                                                                
11   C   13                GOTO RDSCR1
12   C                     MOVEL"GETDTA"  ACTION           DATA TO PROG
13   C                     EXCPT          V$ACTN
14   C                     READ TERMINAL                H0
     C                     SETON                     F0    SETOF BY F9
     C           RDSCR7    TAG
     C                     .
     C                     .
     C                     .

15   OTERMINALE                V$ACTN
16   O                         ACTION     6
```

Figure 4-17. Reading a VPLUS Form (Continued)

Comments

1 This line defines the input record for processing event code 09 (VPLUS edit errors).

2 This line defines the field, NUMBER, which contains a count of the VPLUS edit errors.

3 This line displays the VPLUS form on the terminal. The SHOW action is executed after the EDITS action (see line 8). It redisplays the form so that a user can correct edit errors.

4 This line enters the VPLUS action, RDTERM, into the output field, ACTION. RDTERM retrieves data from the terminal and saves it in the VPLUS buffer.

5 This line performs exception output for the record associated with EXCPT Group V$ACTN.

6 This line returns the event code for the RDTERM action.

Columns 28-32 contain READ to perform a read to the TERMINAL file.

7 This line directs program execution to RDSCR7 when a user presses a function key instead of entering data. (when F0 is turned on, the user entered data and pressed ENTER .)

8 This line enters the VPLUS action, EDITS, into the output field, ACTION. EDITS performs the field edits specified by FORMSPEC.

Columns 33-42 contain EDITS to specify the VPLUS action.

9 This line performs exception output for the record associated with EXCPT Group V$ACTN.

10 This line returns the event code for the EDITS action.

11 When event 09 occurs (an edit error), this line directs program execution to RDSCR1. RDSCR1 redisplays the form so that a user can enter corrections.

12 This line enters the VPLUS action, GETDTA, into the output field, ACTION. GETDTA moves data fields from the VPLUS buffer to the program.

13 This line performs exception output for the record associated with EXCPT Group V$ACTN.

14 This READ operation moves fields in the VPLUS buffer to the corresponding fields in the terminal input record.

15 This line defines the output record associated with EXCPT Group V$ACTN.

16 This line defines the field, ACTION, which contains the VPLUS action code.

Processing VPLUS Form Data. Once you retrieve screen data from the VPLUS buffer with a GETDTA action followed by a READ operation, the data is moved to the appropriate input record in the program. You process the data fields in the input record as you would normally.

Moving Data to a VPLUS Form and Displaying It. This section explains how to display data that is calculated in the program or that comes from sources such as a disc file. For example, you may want to display master file information in response to a query.

Figure 4-18 shows how to move disc file information to a terminal file output record then display it using a VPLUS PUTDTA action.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
                                                                                
1    C           FACTNO    CHAINDACCOUNT             6061
     C                       .
     C                       .
     C                       .
2    C                     MOVE FNAME     FFNAME
     C                     MOVE LNAME     FLNAME
     C                     MOVE ADDR1     FADDR1
     C                       .
     C                       .
     C                       .
3    C                     MOVEL"PUTDTA"  ACTION           DATA TO BUFF
4    C                     EXCPT          V$DATA
5    C                     MOVEL"SHOW  "  ACTION           DISPLAY BUFF
6    C                     EXCPT          V$ACTN
     C                       .
     C                       .
     C                       .
```

Figure 4-18. Moving Data to a VPLUS Form and Displaying It

Comments

1 This line reads the TurboIMAGE data set file, DACCOUNT. (Information from this data set will be displayed on the terminal.)

2 This line begins the MOVE operations that move the data set fields to the terminal output record. (See Figure 4-15 for a description of the terminal output record associated with indicator 85.)

3 The line enters the VPLUS action, PUTDTA, into the output field, ACTION. PUTDTA moves the terminal output record fields to the VPLUS buffer.

4 This line performs exception output for the record associated with EXCPT Group V$DATA.

5 This line enters the VPLUS action, SHOW, into the output field, ACTION. SHOW displays the form in the VPLUS buffer.

6 This line performs exception output for the record associated with EXCPT Group V$ACTN.

Displaying Messages with VPLUS. When you use VPLUS, you can display one-line messages in the message window located at the bottom of the display. VPLUS automatically displays FORMSPEC data field prompts and edit errors in the window, but occasionally you may need to use the window for other purposes.

Figure 4-19 shows how to display a message in the message window. The

example assumes that the message, SELECT MODE WITH FUNCTION KEY, is placed in the first element of the array, MSG. It also assumes that the message window fields, MSGLEN and ENHANC, are already initialized.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
                                                                                
1    C                   Z-ADD1         M                    MSG 1
2    C                   MOVEL"SHOMSG"  ACTION               WRITE WINDOW
3    C                   EXCPT          V$MESG
     C                     .
     C                     .
     C                     .

     O*
4    O*        OUTPUT MESSAGE TO WINDOW
     O*
     O         E                V$MESG
     O                          ACTION     6
     O                          MSGLEN     8
     O                          ENHANC     9
5    O                          MSG,M     89
-
```

Figure 4-19. Displaying Messages in the VPLUS Message Window

Comments

1 This line sets the index of the MSG array tp 1.

Columns 43-58 contain the name of the index field, M.

2 This line places the VPLUS action, SHOMSG, into the first field of the message window record. SHOWMSG displays messages in the VPLUS message window.

3 This line performs exception output for the record associated with EXCPT Group V$MESG. This is the message window record.

4 This line begins the message window record description.

5 This line specifies that the window message is in the array, MSG. Since the value of M is one, the first element of the array is displayed in the message window.

Specifying the VPLUS Error Message Display Interval. RPG run-time error messages related to VPLUS files (except edit errors or messages you display in the message window) are displayed for 3 seconds in the message window.

You can change the time interval, if necessary, by using column 51 of the File Description Specification.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
1    FTERMINALUD  V     256            WORKSTN    8
     F                                             KFORMS FACCOUNT

-
```

Figure 4-20. Specifying the Error Message Display Interval

Comments

1 This line defines the terminal file, TERMINAL.

Column 51 contains the number of seconds (8) to pause when displaying errors. You can enter a number from 0 to 9. If you enter 0, all messages are suppressed.

Changing VPLUS Forms. This section tells you how to display a second VPLUS form in a program.

The number of forms that you can use is limited only by the number of forms that can be placed in a forms file. The second and successive forms are retrieved differently from the first. The first form displayed in a program is retrieved using the GETNXT action. To retrieve additional forms, use the CHGNXT action instead of GETNXT. Also enter Input and Output record descriptions for the new form(s).

Figure 4-21 shows how to retrieve a form from the forms file, FORM2. (The form is only retrieved, not displayed.) For instructions on how to display the form, see the section "Displaying an Initialized VPLUS Form." You display a form the same way shown in that section except that you omit the GETNXT action. You also perform exception output for the new form rather than the original.

See the *HP RPG Reference Manual* for details about the fields used in the TERMINAL action output record.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
                                                                                
1    C                   MOVE "0"        RPTAPP  1
2    C                   MOVE "0"        FRZAPP  1
3    C                   MOVEL"FORM2"    NXTFRM 15
4    C                   MOVEL"CHGNXT"   ACTION
5    C                   EXCPT           V$ACTN
     C                     .
     C                     .
     C                     .

     O*
     O*        DO SCREEN HANDLING ACCORDING TO ACTION CODE
     O*
6    OTERMINALE                V$ACTN
     O                         ACTION     6
     O                         NXTFRM    21
     O                         RPTAPP    22
     O                         FRZAPP    23
-
```

Figure 4-21. Changing a VPLUS Form

Comments

1 This line sets the repeat/append code (RPTAPP) code to 0. Code 0 does not repeat the current form.

2 This line sets the freeze/append code (FRZAPP) code to 0. Code 0 clears the current form before displaying the next one.

3 This line enters the name of the next form (FORM2) in to the output field, NXTFRM.

4 This line enters the VPLUS action, CHGNXT, into the action field. CHGNXT directs VPLUS to retrieve another forms file and place it into the VPLUS buffer.

5 This line performs exception output for the record associated with EXCPT Group V$ACTN.

6 This line begins the description of the TERMINAL output record associated with EXCPT Group V$ACTN.

Using Function Keys with VPLUS. This section explains how to use function keys and function key labels with VPLUS.

You can use function keys to turn on record identification indicators instead of using event codes to do this. (For each event code returned by VPLUS, a corresponding function key is turned on also.) For example, when the user enters data and presses ENTER , VPLUS returns event code 00 and turns on F0. An advantage of using function keys is that they are automatically reset for each event. Indicators set by event codes are not reset by READ operations and can cause erroneous results if you do not use them properly in the program.

Figure 4-22 shows how to enable the function keys and how to use them to identify the type of data the user enters on the terminal.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
1  FTERMINALUD  V     256            WORKSTN   L
   F                                              KFORMS FACCOUNT

2  E                    LBL     1   8 16                FUNCKEY LBLS

3  C            LBL       SET                          F@
   C                        .
   C                        .
   C                        .
   C*            FUNCTION KEYS   F1 = ADD NEW ACCOUNT
4  C   F1                  EXSR ADD
   C                        .
   C                        .
   C                        .
5  C   F0                  GOTO RDSCR7
   C                        .
   C                        .
   C                        .
```

Figure 4-22. Using Function Keys with VPLUS

Comments

1 This line defines the terminal file, TERMINAL.

Column 50 contains L to enable the function key labels.

2 This line defines the function key label array, LBL. There are 8 labels in array, each label containing 16 characters.

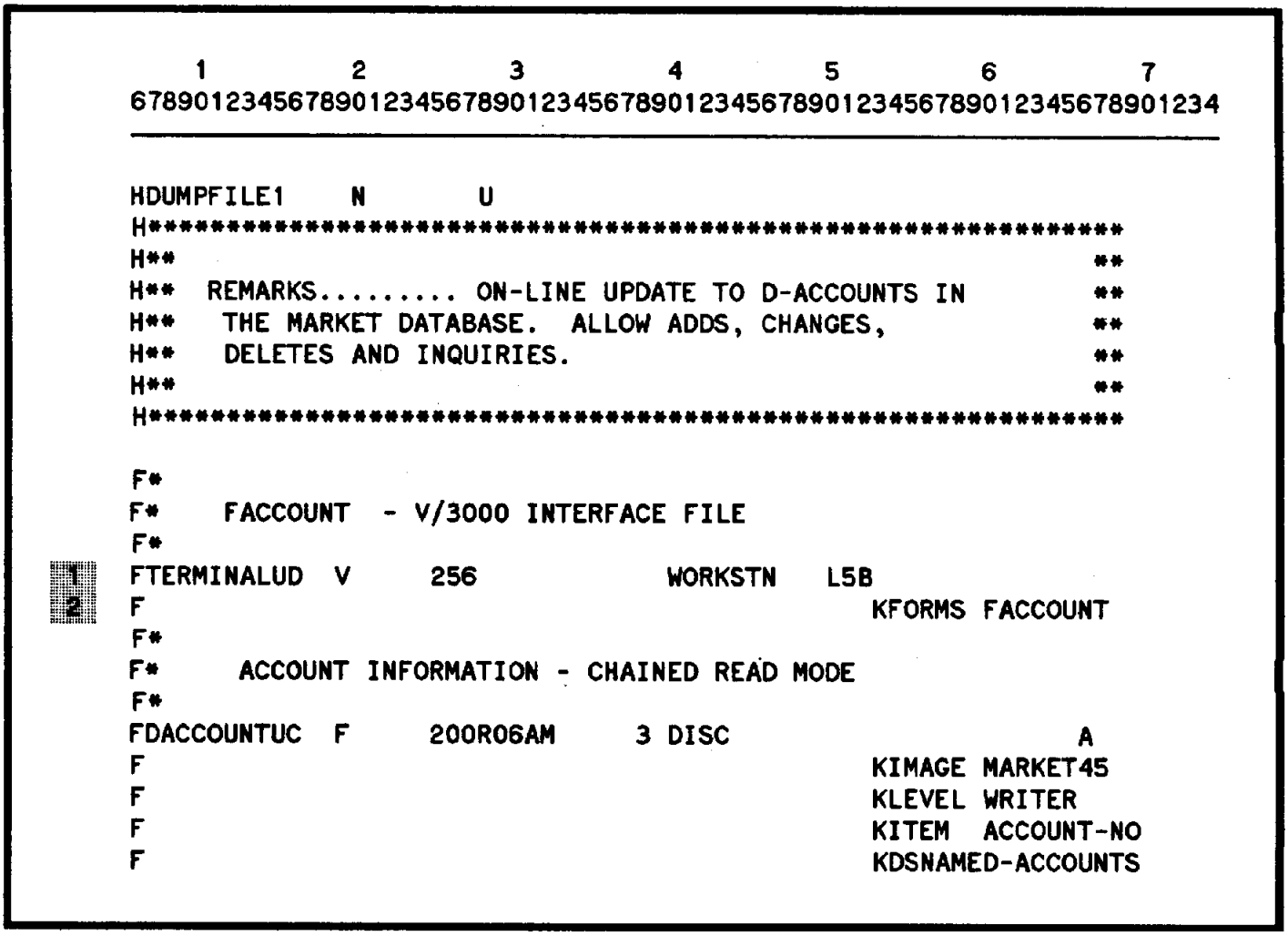
3 This line enables all function keys.

4 This line uses the setting of F1 to direct program execution. The subroutine, ADD, is executed when F1 is turned on.

5 This line uses the setting of function key F0 to direct program execution. Program execution goes to RDSCR1 when F0 is turned on. F0 is turned on when a user enters data into the VPLUS form and then presses ENTER .

A Sample Program Using VPLUS. This section lists a complete program that processes screen data using VPLUS forms.

The program in Figure 4-23 updates the D-ACCOUNTS data set. This data set keeps customer information such as account number, name and address. The data set is part of the TurboIMAGE MARKET database whose schema is shown in Figure 3-23. Users access customer records by the customer's account number and select the type of update operation by pressing one of the function keys, F1 , F2 , F3 or F5 (add, change, query, delete). The program uses VPLUS to handle all screen input and output.



```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

HDUMPFILE1    N       U
H*****************************************************************
H**                                                              **
H**  REMARKS..... ON-LINE UPDATE TO D-ACCOUNTS IN                **
H**   THE MARKET DATABASE.  ALLOW ADDS, CHANGES,                 **
H**   DELETES AND INQUIRIES.                                     **
H**                                                              **
H*****************************************************************

F*
F*    FACCOUNT  - V/3000 INTERFACE FILE
F*
FTERMINALUD  V     256           WORKSTN   L5B
F                                          KFORMS FACCOUNT
F*
F*     ACCOUNT INFORMATION - CHAINED READ MODE
F*
FDACCOUNTUC  F     200R06AM     3 DISC                    A
F                                         KIMAGE MARKET45
F                                         KLEVEL WRITER
F                                         KITEM  ACCOUNT-NO
F                                         KDSNAMED-ACCOUNTS
```

1
2

Figure 4-23. Program to Update D-ACCOUNTS Using VPLUS

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

E*-------------------------------------------------------------
E                    MSG     1  20 79               MESSAGES
E                    LBL     1   8 16               FUNC KEY LBLS
E*-------------------------------------------------------------

I*
I*        NUMBER OF EDIT ERRORS - EVENT 09
3  ITERMINALNS  09   1 C0   2 C9
4  I                                        18  220NUMBER      13
I*        ALL OTHER TERMINAL READS
5  I        NS  01
I                                          18  210DATALN
I                                          22   27 FACTNO
I                                          28   43 FLNAME
I                                          44   53 FFNAME
I                                          54  630FPHONE
I                                          64  670FPHEXT
I                                          68   95 FADDR1
I                                          96  123 FADDR2
I                                         124  151 FADDR3
I                                         152  161 FZIPCD
I                                         162  169 FADATE
I                                         170  171 FTYPEC
I                                         172  179 FMAILC
I                                         180  1870FBDATE
I                                         188  191 FSRCCD
I*     DATABASE FILE
IDACCOUNTNS  12
I                                           3    8 ACTNO
I                                           9   18 FNAME
I                                          19   34 LNAME
I                                          35   62 ADDR1
I                                          63   90 ADDR2
I                                          91  118 ADDR3
I                                         119  128 ZIPCD
I                                       P 129  1340PHONE
I                                       B 135  1360PHEXT
I                                         137  144 MAILC
I                                       B 145  1480BDATE
I                                         149  152 SRCCD
I                                         153  154 TYPEC
I                                         155  162 ADATE
```

Figure 4-23. Program to Update D-ACCOUNTS Using VPLUS (Continued)

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

C*-------------------------------------------------------------
C*-------------------------------------------------------------
C*     MAINLINE CALCULATIONS
C*-------------------------------------------------------------
C*-------------------------------------------------------------
C  N50                EXSR ONETIM
C   LR                GOTO ENDMN
C*
C*-----------------------------------------------------
C*  DIRECT TO SUBROUTINE DEPENDING ON FUNCTION KEY PRESSED
C*            FUNCTION KEYS   F1 = ADD NEW ACCOUNT
C*                            F2 = CHANGE ACCOUNT
C*                            F3 = INQUIRY
C*                            F5 = DELETE ACCOUNT
C*                            F8 = END
C*                            F4, 6, 7 = WRONG
C*-----------------------------------------------------
C   F1                EXSR ADD
C   F2                EXSR CHANGE
C   F3                EXSR INQRY
C   F5                EXSR DELETE
C           ENDMN     TAG
C********************************************************* SUBR ADD
C*    ADD NEW ACCOUNTS
C*************************************************************************
CSR         ADD       BEGSR
CSR         ADD1      TAG
CSR                   EXSR GETSCR                      REFRESH SCRN
CSR         ADD3      TAG
CSR                   EXSR RDSCR                       READ TERM
CSRNF0                EXSR CHGMOD                      FUNC KEY CHK
CSR 51                GOTO ADD1
CSR LR
CORNF0                GOTO ADD9
C*-----------------------------------------------------
C*        CHECK D-ACCOUNTS - ID MUST NOT EXIST
C*-----------------------------------------------------
CSR         FACTNO    CHAINDACCOUNT             6061
CSRN60                Z-ADD6          M                NO SUCH
CSRN60                EXSR DSPMSG                      ACCT #
CSRN60                GOTO ADD3
```

Figure 4-23. Program to Update D-ACCOUNTS Using VPLUS (Continued)

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
-----------------------------------------------------------------------

C*-------------------------------------------------
C*        ADD NEW RECORD TO D-ACCOUNTS
C*-------------------------------------------------
CSR                     EXCPT          $ADD
CSR                     GOTO ADD1
CSR           ADD9      ENDSR
C************************************************** SUBR CHANGE
C*    CHANGE EXISTING ACCOUNTS
C*************************************************************************
CSR           CHANGE    BEGSR
CSR           CHANG1    TAG
CSR                     SETOF                      52    CLR ERROR
CSR                     EXSR GETSCR                      REFRESH SCRN
CSR           CHANG3    TAG
CSR                     EXSR RDSCR                       GET ACCT#
CSR 52 F7               GOTO CHANG1                      RESTART
CSR 52NF7               GOTO CHANG3                      RETRY F7
CSRNFO                  EXSR CHGMOD                      FUNC KEY CHK
CSR 51                  GOTO CHANG1
CSR LR
CORNFO                  GOTO CHANG9
CSR                     EXSR RDFIL
C*-------------------------------------------------
C*        IND 60 - INVALID ID
C*-------------------------------------------------
CSR 60                  Z-ADD7         M
CSR 60                  EXSR DSPMSG
CSR 60                  GOTO CHANG3
CSR                     Z-ADD2         M
CSR                     EXSR DSPMSG
CSR                     MOVE FACTNO    HACTNO  6         MOVE TO
CSR                     MOVE FLNAME    HLNAME 16         HOLD FLDS
CSR                     EXSR RDSCR                       GET CHANGES
CSR F7                  GOTO CHANG1                      CANCEL CHNG
CSRNFO                  EXSR CHGMOD                      FUNC KEY CHK
CSR 51                  GOTO CHANG1                      ERROR
CSR LR
CORNFO                  GOTO CHANG9
C*-------------------------------------------------
C*        CAN'T CHANGE ACCT NO OR LAST NAME
C*-------------------------------------------------
CSR           FACTNO    COMP HACTNO                5252
```

Figure 4-23. Program to Update D-ACCOUNTS Using VPLUS (Continued)

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

CSRN52      FLNAME    COMP HLNAME              5252
CSR 52                Z-ADD4         M
CSR 52                EXSR DSPMSG
CSR 52                GOTO CHANG3
C*-------------------------------------------------
C*        UPDATE D-ACCOUNTS RECORD
C*-------------------------------------------------
CSR                   EXCPT          $UPD
CSR                   GOTO CHANG1
CSR         CHANG9    ENDSR
C************************************************** SUBR INQRY
C*    DISPLAY EXISTING ACCOUNTS TO SCREEN
C*****************************************************************
CSR         INQRY     BEGSR
CSR         INQRY1    TAG
CSR                   EXSR GETSCR                   REFRESH SCRN
C*-------------------------------------------------
C*        CLEAR SCREEN AND DISPLAY ERROR IF NON-EXISTENT CUSTOMER
C*-------------------------------------------------
CSRN60                GOTO INQRY3
CSR                   Z-ADD7         M              FORMAT ERROR
CSR                   MOVELFACTNO    FACT50 50      MSG W/ INVLD
CSR                   MOVE FACT50    MSG,7          ACCT#
CSR                   EXSR DSPMSG
CSR                   MOVE BLNK50    MSG,7
CSR         INQRY3    TAG
CSR                   EXSR RDSCR                    READ TERM
CSRNF0                EXSR CHGMOD                   FUNC KEY CHK
CSR 51                GOTO INQRY1                   ERROR
CSR LR
CORNF0                GOTO INQRY9
CSR 60                Z-ADD3         M              CLEAR ERROR
CSR 60                EXSR DSPMSG                   FROM BEFORE
CSR                   EXSR RDFIL
CSR 60                GOTO INQRY1                   INVLDACCT#
CSR                   GOTO INQRY3
CSR         INQRY9    ENDSR
C************************************************** SUBR DELETE
C*    DELETE EXISTING ACCOUNTS
C*****************************************************************
CSR         DELETE    BEGSR
CSR         DELET1    TAG
```

Figure 4-23. Program to Update D-ACCOUNTS Using VPLUS (Continued)

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

CSRN60                EXSR GETSCR                     REFRESH SCRN
CSR                   EXSR RDSCR                      GET ACCT#
CSRNF0                EXSR CHGMOD                     FUNC KEY CHK
CSR 51                GOTO DELET1                     ERROR
CSR LR
CORNF0                GOTO DELET9
CSR                   EXSR RDFIL
C*-------------------------------------------------
C*        IND 60 - INVALID ID
C*-------------------------------------------------
CSR 60                Z-ADD7          M
CSR 60                EXSR DSPMSG
CSR 60                GOTO DELET1
C*-------------------------------------------------
C*        RECORD FOUND, PRESS F5 AGAIN TO CONFIRM DELETE
C*-------------------------------------------------
CSR                   Z-ADD8          M
CSR                   EXSR DSPMSG
CSR                   EXSR RDSCR                      GET CONFIRM
CSR F7                GOTO DELET1                     CANCEL
CSR F5                GOTO DELET5
CSRNF0                EXSR CHGMOD                     FUNC KEY CHK
CSR F0
COR 51                GOTO DELET1                     ERROR
CSR LR
CORNF0                GOTO DELET9
C*-------------------------------------------------
C*        DELETE D-ACCOUNTS RECORD
C*-------------------------------------------------
CSR         DELET5    TAG
CSR                   EXCPT           $DEL
CSR                   Z-ADD3          M               CLEAR
CSR                   EXSR DSPMSG                     WINDOW
CSR                   GOTO DELET1
CSR         DELET9    ENDSR
C************************************************** SUBR RDFIL
C*     RETRIEVE CURRENT CUSTOMER INFO & DISPLAY TO SCREEN
C*****************************************************************
CSR         RDFIL     BEGSR
```

Figure 4-23. Program to Update D-ACCOUNTS Using VPLUS (Continued)

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

C*-----------------------------------------------------
C*        CHECK DACCOUNT - ID MUST EXIST
C*-----------------------------------------------------
CSR         FACTNO    CHAINDACCOUNT             6061
CSR 60                GOTO RDFIL9                     INVLDACCT#
CSR                   MOVE FNAME      FFNAME
CSR                   MOVE LNAME      FLNAME
CSR                   MOVE ADDR1      FADDR1
CSR                   MOVE ADDR2      FADDR2
CSR                   MOVE ZIPCD      FZIPCD
CSR                   MOVE PHONE      FPHONE
CSR                   MOVE PHEXT      FPHEXT
CSR                   MOVE MAILC      FMAILC
CSR                   MOVE BDATE      FBDATE
CSR                   MOVE SRCCD      FSRCCD
CSR                   MOVE TYPEC      FTYPEC
CSR                   MOVE ADATE      FADATE
CSR                   MOVEL"PUTDTA"   ACTION
CSR                   EXCPT           V$DATA
CSR         RDFIL9    ENDSR
C*************************************************** SUBR CHGMOD
C*     VALIDATE FUNCTION KEY, DISPLAY ANY ERRORS, AND
C*     REWRITE THE FUNC KEY LABELS - CHANGE MODE.
C*******************************************************************
7   CSR         CHGMOD    BEGSR
CSR                   SETOF                     60
CSR F8                SETON                     LR
CSR F8                GOTO CHGMO9
C*-----------------------------------------------------
C*        DISPLAY ERROR MESSAGE IF F4, F6 OR F7 PRESSED,
C*        REQUIRE F7 TO CLEAR ERROR AND CONTINUE
C*-----------------------------------------------------
CSR F4
COR F6
COR F7                SETON                     51
CSRN51                GOTO CHGMO5
CSR         CHGMO3    TAG
CSR                   Z-ADD5          M
CSR                   EXSR DSPMSG                     SHOW ERR MSG
CSR                   EXSR RDSCR                      GET F7
CSRNF7                GOTO CHGMO3                     TRY AGAIN
CSR                   GOTO CHGMO9
```

Figure 4-23. Program to Update D-ACCOUNTS Using VPLUS (Continued)

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
C*------------------------------------------------
C*         DISPLAY FUNC KEY LABELS
C*------------------------------------------------
CSR          CHGMO5    TAG
CSR F1                 Z-ADD1          F
CSR F2                 Z-ADD2          F
CSR F3                 Z-ADD3          F
CSR F5                 Z-ADD5          F
C*         RESET ALL MODE FLAGS TO BLANK
CSR                    MOVE " "        LBL
C*------------------------------------------------
C*         SET CURRENT MODE FLAG WITH ASTERISK
C*------------------------------------------------
CSR                    MOVE "*"        LBL,F
CSR          LBL       SET                       F@
CSR          CHGMO9    ENDSR
C************************************************** SUBR GETSCR
C*     GET NEXT FORM AND PERFORM INIT PHASE
C*************************************************************************
8    CSR          GETSCR    BEGSR
CSR                    SETOF                     51    INIT ERR IND
CSR                    MOVEL"GETNXT"   ACTION  6       GET FORM
CSR                    EXCPT           V$ACTN
CSR                    MOVEL"INIT  "   ACTION          INIT VALUES
CSR                    EXCPT           V$ACTN
CSR                    Z-ADD3          M               CLEAR
CSR                    EXSR DSPMSG                     WINDOW
CSR                    ENDSR
C************************************************** SUBR RDSCR
C*     DISPLAY NEXT FORM AND READ FROM TERMINAL
C*************************************************************************
9    CSR          RDSCR     BEGSR
CSR          RDSCR1    TAG
CSR                    MOVEL"SHOW  "   ACTION          DISPLAY FORM
CSR                    EXCPT           V$ACTN
CSR                    MOVEL"RDTERM"   ACTION          READ VBUFFER
CSR                    EXCPT           V$ACTN
CSR                    READ TERMINAL                 HOREAD RECORD
CSRNF0                 GOTO RDSCR7
```

Figure 4-23. Program to Update D-ACCOUNTS Using VPLUS (Continued)

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

     C*-------------------------------------------------
     C*        CHECK FOR VPLUS ERRORS, LOOP IF ANY
     C*-------------------------------------------------
10   CSR                   MOVEL"EDITS "  ACTION           EDIT DATA
     CSR                   EXCPT          V$ACTN
     CSR                   READ TERMINAL                 H0
     CSR 13                GOTO RDSCR1
     CSR                   MOVEL"GETDTA"  ACTION           DATA TO PROG
     CSR                   EXCPT          V$ACTN
     CSR                   READ TERMINAL                 H0
     CSR                   SETON                      F0    SETOF BY F9
     CSR          RDSCR7   TAG
     CSR                   ENDSR
     C************************************************** SUBR DSPMSG
     C*     DISPLAY MESSAGE TO WINDOW
     C*********************************************************************
11   CSR          DSPMSG   BEGSR
     CSR                   MOVEL"SHOMSG"  ACTION           WRITE WINDOW
     CSR                   EXCPT          V$MESG
     CSR                   ENDSR
     C************************************************** SUBR ONETIM
     C*     INITIALIZE FIELDS AND GET ADD/CHANGE/INQUIRY/DELETE MODE
     C*********************************************************************
     CSR          ONETIM   BEGSR
     CSR                   MOVE 256       DATALN  40
     CSR                   MOVE "J"       ENHANC  1
     CSR                   Z-ADD79        MSGLEN  20
     CSR                   Z-ADD0         BADFLD  50
     CSR                   MOVE *BLANK    BLNK50  50
     CSR                   Z-ADD0         M       10
     CSR                   Z-ADD0         F       10
     CSR          LBL      SET                       F@
     CSR          ONETI5   TAG
     CSR                   EXSR GETSCR                   NEW SCREEN
     CSR                   Z-ADD1         M
     CSR                   EXSR DSPMSG
     CSR                   EXSR RDSCR                    GET F KEY
     CSR                   EXSR CHGMOD                   FUNC KEY CHK
     CSR F0
     COR 51                GOTO ONETI5
     CSR                   SETON                      50
     CSR                   ENDSR
```

Figure 4-23. Program to Update D-ACCOUNTS Using VPLUS (Continued)

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

0*-------------------------------------------------------------
0*
0*  V/3000 OUTPUT TO FILE TERMINAL
0*
0*-------------------------------------------------------------
0*
0*         DO SCREEN HANDLING ACCORDING TO ACTION CODE
0*
OTERMINALE                V$ACTN
0                         ACTION     6
0*
0*          OUTPUT DATA TO SCREEN
0*
12 0        E             V$DATA
0                         ACTION     6
0                         DATALN    10
0                         FACTNO    16
0                         FLNAME    32
0                         FFNAME    42
0                         FPHONE    52
0                         FPHEXT    56
0                         FADDR1    84
0                         FADDR2   112
0                         FADDR3   140
0                         FZIPCD   150
0                         FADATE   158
0                         FTYPEC   160
0                         FMAILC   168
0                         FBDATE   176
0                         FSRCCD   180
0*
0*          OUTPUT MESSAGE TO WINDOW
0*
13 0        E             V$MESG
0                         ACTION     6
0                         MSGLEN     8
0                         ENHANC     9
0                         MSG,M     89
```

Figure 4-23. Program to Update D-ACCOUNTS Using VPLUS (Continued)

```
    1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------

O*---------------------------------------------------------------
O*
O*         OUTPUT TO DATABASE
O*
O*---------------------------------------------------------------
ODACCOUNTEADD             $ADD
O                         FACTNO     8
O                         FFNAME    18
O                         FLNAME    34
O                         FADDR1    62
O                         FADDR2    90
O                         FADDR3   118
O                         FZIPCD   128
O                         FPHONE   134P
O                         FPHEXT   136B
O                         FMAILC   144
O                         FBDATE   148B
O                         FSRCCD   152
O                         FTYPEC   154
O                         FADATE   162
O*
O         E               $UPD
O                         FACTNO     8
O                         FFNAME    18
O                         FLNAME    34
O                         FADDR1    62
O                         FADDR2    90
O                         FADDR3   118
O                         FZIPCD   128
O                         FPHONE   134P
O                         FPHEXT   136B
O                         FMAILC   144
O                         FBDATE   148B
O                         FSRCCD   152
O                         FTYPEC   154
O                         FADATE   162
O*
O         EDEL            $DEL
O*---------------------------------------------------------------
O*           END OF OUTPUT SPECS
O*---------------------------------------------------------------
```

Figure 4-23. Program to Update D-ACCOUNTS Using VPLUS (Continued)

```
      1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-----------------------------------------------------------------------------------------

14 O*   ARRAY ENTRIES FOLLOW
**                    MSG - MESSAGE ARRAY
SELECT MODE WITH FUNCTION KEY
MAKE CHANGES AND HIT ENTER (F7 TO CANCEL CHANGE)

YOU MAY NOT CHANGE ACCOUNT NUMBER OR LAST NAME  (F7 TO CLEAR)
INVALID KEY (F7 TO CLEAR)
EXISTING ACCOUNT NUMBER
INVALID ACCOUNT NUMBER
PRESS F5 AGAIN TO CONFIRM DELETE (F7 TO CANCEL DELETE)
**                    LBL - FUNC KEY LABEL ARRAY
ADD     MODE
CHANGE  MODE
INQUIRY MODE

DELETE  MODE


EXIT
```

Figure 4-23. Program to Update D-ACCOUNTS Using VPLUS (Continued)

Comments

1 This line defines the terminal as a WORKSTN file and enables the function keys.

2 This line identifies the VPLUS forms file, FACCOUNT, that is used.

3 This line associates event code 09 with the input record indicator 09. Event code 09 reports edit errors to the program.

4 This line describes the field, NUMBER, which contains the edit error count.

5 This line begins the description of the record that handles all terminal input except edit error counts (line 4).

6 This line executes the ADD subroutine when the user presses F1 .

7 This subroutine checks to see if the user pressed a valid function key (F1 , F2 , F3 , F5 or F8).

8 This line begins the subroutine that retrieves the form from the forms file.

9 This subroutine displays the VPLUS form retrieved in the subroutine, GETSCR.

10 This line begins the operations that check for VPLUS edit errors.

11 This subroutine displays messages in the VPLUS message window.

12 This is the output record for displaying VPLUS form data.

13 This is the output record for displaying messages in the VPLUS message window.

14 This is the last line of Output Specifications. Following it are the values for the MSG and LBL arrays.

Using the RPG Screen Interface (RSI)

RSI is a unique forms handling facility designed to work exclusively with RPG. In many cases, you can have RSI perform screen handling automatically during normal logic cycle processing. If you're using the RSI file for a simple input application, you may prefer to use RSI CONSOLE files instead of the regular RSI files that use the full capabilities of RSI (for information on RSI CONSOLE files, see "Using RSI CONSOLE Files").

To use RSI in an RPG program, perform the following steps (the steps are discussed in detail in the following sections in this chapter):

* **Create the RSI form**

  Before using RSI to process data in full screen mode, use SIGEDITOR (see the *RPG Utilities Reference Manual*) to create a screen form. When creating the form, you specify headings and titles, where the data fields are located on the screen, the type of data each field holds and which function and command keys are used.

* **Define an RSI form**

  When you use SIGEDITOR to define an RSI form, you can have SIGEDITOR generate the RPG specifications for the form. You then include those specifications in the RPG program, modifying them when necessary. Whether or not you or SIGEDITOR creates the specifications, they must be entered as described below:

  File Description Specifications -

  These specifications define the file as an RSI WORKSTNR file and, optionally, the name of the forms file.

  Input Specifications -

  These specifications define the fields that are used for input.

  Output Specifications -

  These specifications name the RSI form (in the forms file) to be used and the data for the fields to be displayed. You can also include a field for errors and other messages.

* **Display and Read the RSI form**

  Calculation Specifications -

You can display and read the form in one of two ways. If you define the file as a primary file, RSI performs input and output automatically during the normal logic cycle. If you define the file as a demand file, you control input and output by entering the Calculation Specification operations EXCPT and READ.

In addition, to determine which command keys the user presses at run time, enter Calculation Specifications to test the command key indicators. If you need to know which function keys the user presses, enter one or more Calculation Specifications to test the values in the RSI STATUS array.

Creating an RSI Form. Figure 4-24 shows how a form is defined using SIGEDITOR. This form is used in the following sections to illustrate how RSI is used. The form is used to update the D-ACCOUNTS data set (see the schema for the MARKET database in Figure 3-23).

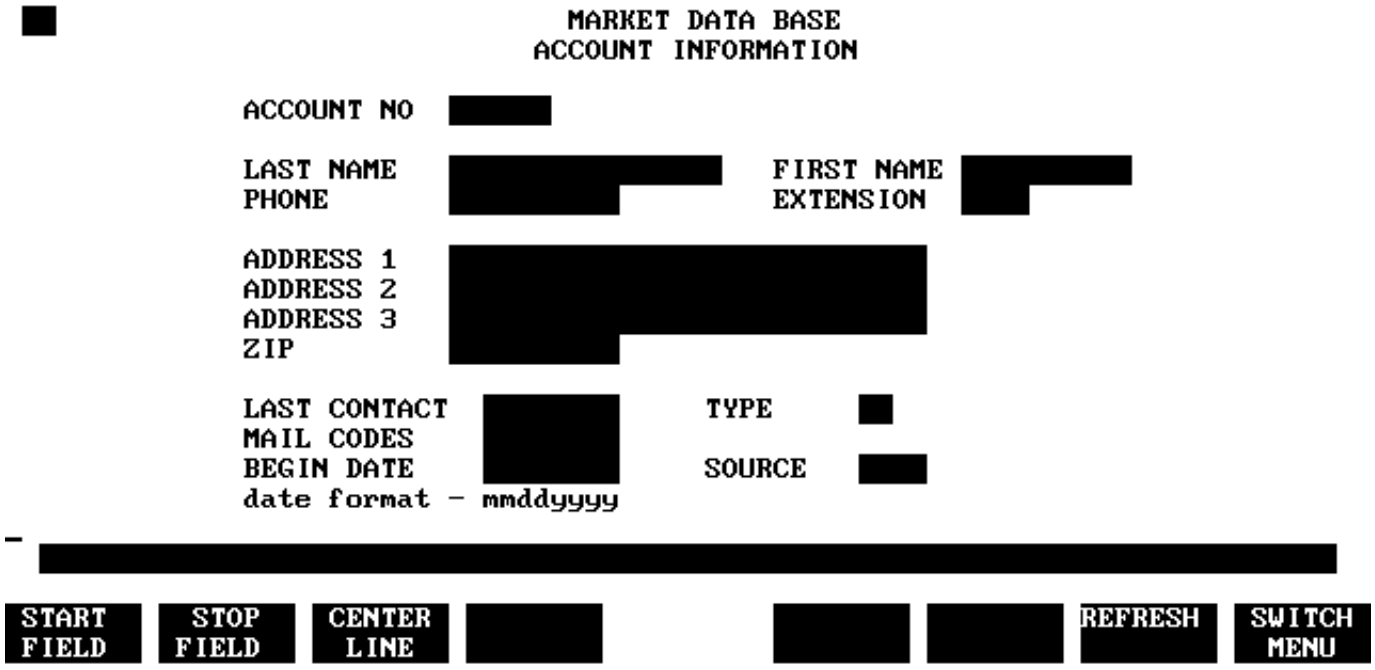


Figure 4-24. Creating an RSI Form Using SIGEDITOR

In addition to defining the layout of the form, you must also assign a name to the forms file and define certain processing requirements. For example, you may want to process the form using exception output rather than letting the processing be handled automatically by the RPG logic cycle.

Figure 4-25 shows the SIGEDITOR Forms Specification screen that lets you define these requirements.

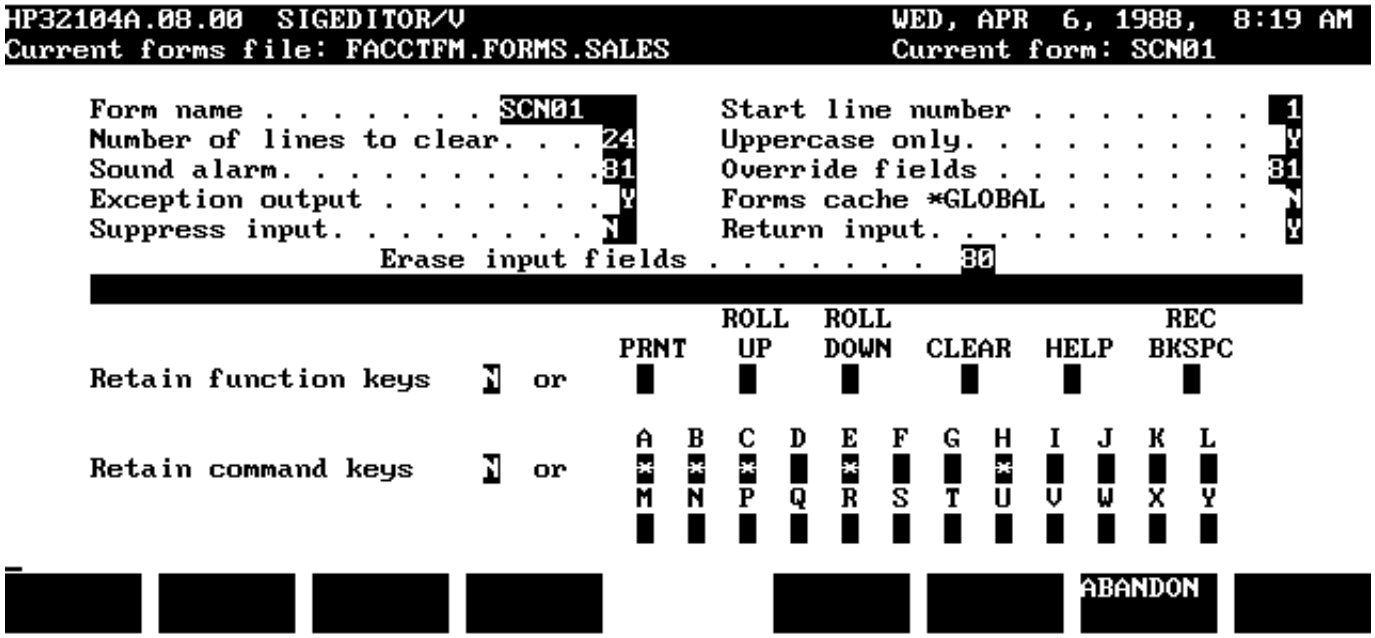


Figure 4-25. Using the SIGEDITOR Forms Specification Screen

The lower part of the SIGEDITOR Forms Specification screen lets you enable one or more of the function keys (F2 - F7). To enable a function key, enter an asterisk for it on this screen.

The function keys F3 - F8 let you simulate the function keys of the same name on IBM 3x systems. To use them in an RPG program, you must enable them in on this screen. The function keys are:

Function Key	Description
PRNT (PRINT SCREEN) (F2)	When enabled, the function key label PRINT SCREEN appears and lets you simulate the print screen key on IBM3X systems. When not enabled, the label *PRINT SCREEN appears at run time and the screen contents are written to RSIPRINT (DEV=LP).
ROLL UP (F3)	Lets you simulate the ROLL UP key on IBM 3x systems.
ROLL DOWN (F4)	Lets you simulate the ROLL DOWN key on IBM 3x systems.
CLEAR (F5)	Lets you simulate the CLEAR key on IBM 3x systems.
HELP (F6)	When enabled, the function key label HELP appears and lets you simulate the HELP key on IBM 3x systems. When not enabled, the label *HELP appears and the application hep facility is activated when the key is pressed.
REC BKSPC (RECORD BACKSPACE) (F7)	Lets you simulate the REC BKSPC key on IBM 3x systems.

The last part of the SIGEDITOR Forms Specification screen lets you enable one or more command keys. You must enable the command keys before using the corresponding command key indicators in an RPG program. Figure 4-25 shows that all of the command keys are disabled except for command keys 1, 2, 3, 5 and 8 (which correspond to command key indicators KA, KB, KC,

KE and KH).

At run time, the form defined in Figure 4-24 and Figure 4-25 appears as shown in Figure 4-26. Data is entered in the form to show how the fields are used.

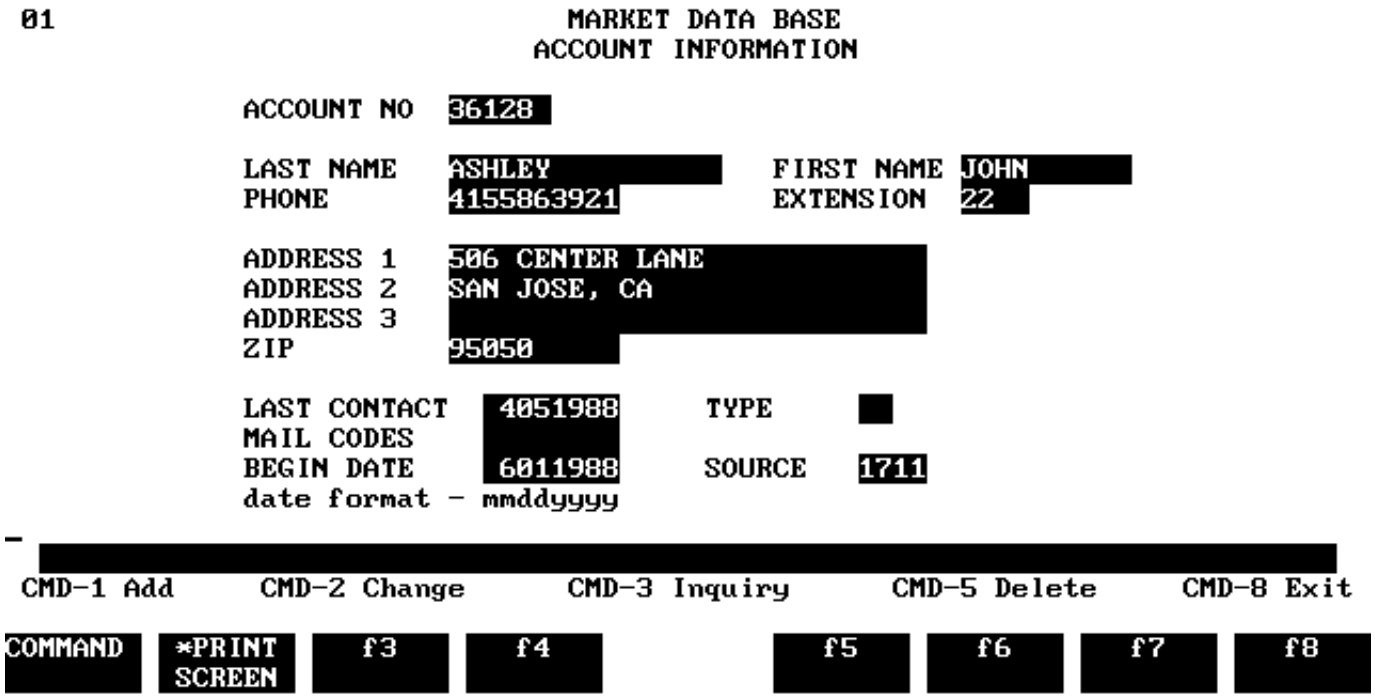


Figure 4-26. Using an RSI Form Within an RPG Program

Defining an RSI Form. Figure 4-27 shows the specifications generated by SIGEDITOR for the screen in Figure 4-24. You can create the specifications yourself, if you prefer or you can modify the ones generated by SIGEDITOR.

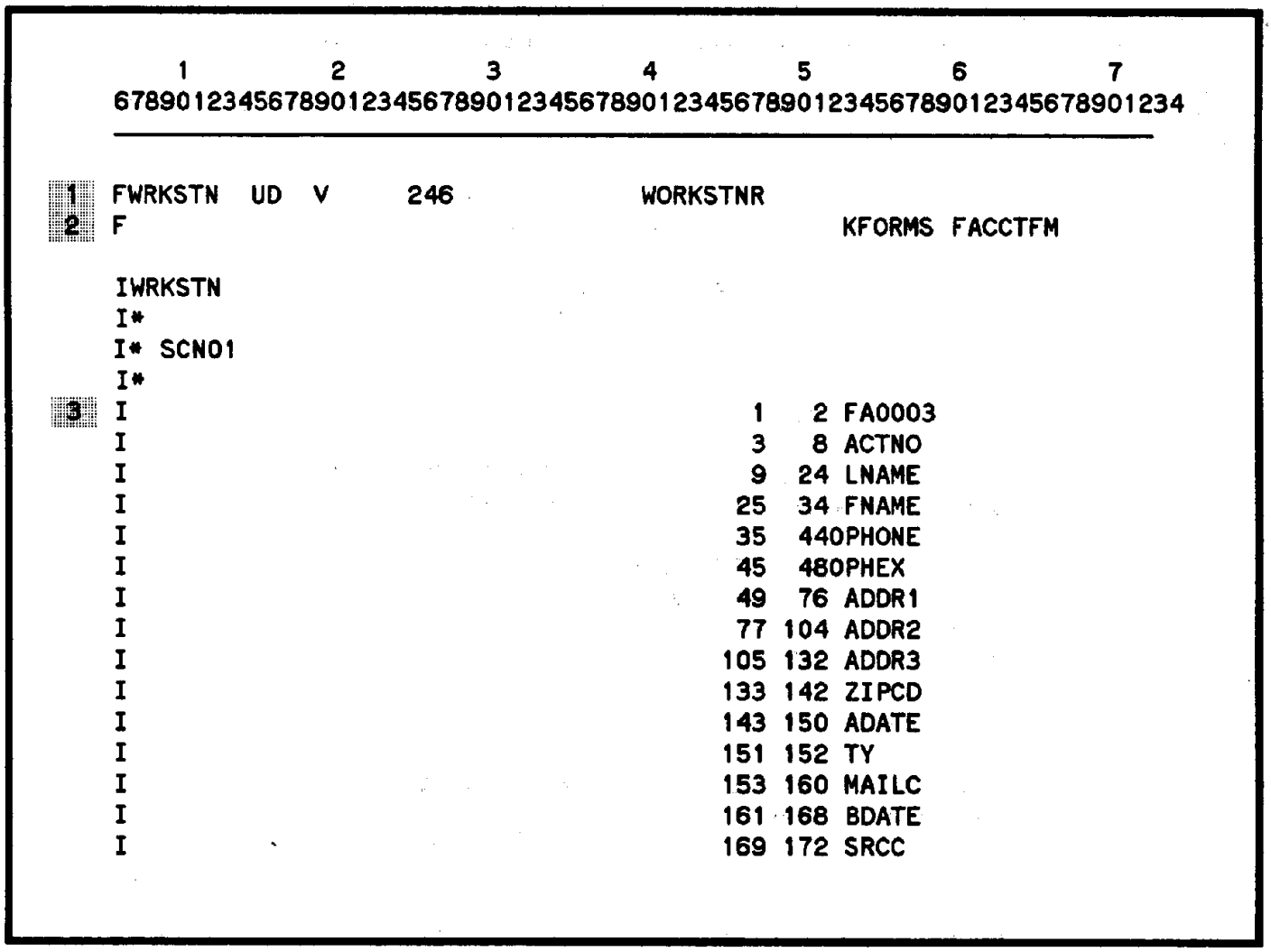
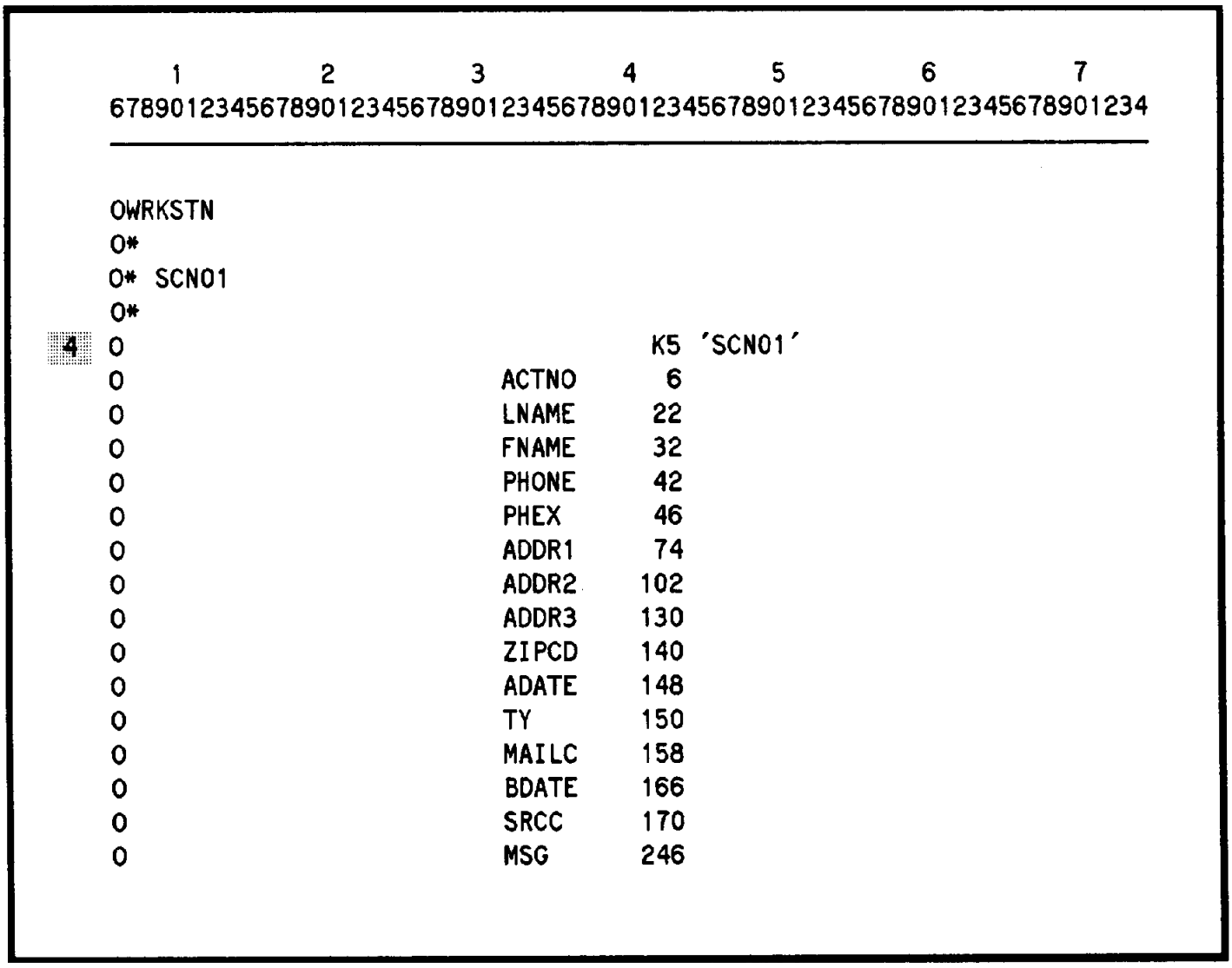


Figure 4-27. RPG Specifications Generated by SIGEDITOR



```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------

     OWRKSTN
     O*
     O* SCN01
     O*
4    O                                    K5 'SCN01'
     O                         ACTNO       6
     O                         LNAME      22
     O                         FNAME      32
     O                         PHONE      42
     O                         PHEX       46
     O                         ADDR1      74
     O                         ADDR2     102
     O                         ADDR3     130
     O                         ZIPCD     140
     O                         ADATE     148
     O                         TY        150
     O                         MAILC     158
     O                         BDATE     166
     O                         SRCC      170
     O                         MSG       246
```

Figure 4-27. RPG Specifications Generated by SIGEDITOR (Continued)

Comments

1 This line defines the file WRKSTN as a WORKSTNR (RSI) file.

Column 15 is U to specify update file processing.

Column 16 is D to specify that this is a demand file.

Column 19 is V to specify variable length records (you must use V for terminal files).

Columns 24-27 specify that the record length is 246 characters.

2 This line names the RSI forms file, FACCTFM, to be used in the program.

3 This line begins the definition of the fields that are used for input.

4 This line begins the definition of the fields that are used for output.

Displaying and Reading an RSI Form. You can enter the Calculation Specifications operations EXCPT and READ to display, then read and RSI form. You do not need to enter Calculation Specifications to display and read an RSI form if you entered "N" for Exception Output in SIGEDITOR because the form is processed automatically during normal logic cycle processing.

Figure 4-28 shows how to display and read an RSI form using Calculation Specifications.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
                                                                                
1    IWORKSTN NS  01   1 C0   2 C1
     I                                          3   8 ACTNO
     I                                              .
     I                                              .
     I                                              .

2    C                   EXCPT          R$DATA
3    C                   READ WORKSTN                    H0

4    OWORKSTN E              R$DATA
     O                                  K8 "SCN01    "
     O                       ACTNO       6
     O                       LNAME      22
     O                         .
     O                         .
     O                         .
-
```

Figure 4-28. Displaying and Reading an RSI Form Using Calculation Specifications

Comments

1 This line begins the definition of the input fields in the RSI form.

2 This line displays the form SCN01 whose definition starts in line 4.

3 This line reads the data entered into the form by the user and stores it in the Input Specification fields. When this line is executed, all of the data entered by the user is available for processing by the program.

4 This line defines the fields that are displayed in form SCN01. Field values can be specified as defaults in SIGEDITOR or you can move the values into the output fields using Calculation Specifications operations.

Displaying Messages with RSI. There are no special areas of the screen or special fields for messages. If you want to display a message (for example, an error message), you must define a field for it. Once defined, there are several ways you can use the field to display messages. You can place a message in the field yourself or, if the message is contained in a User Message Catalog, you can specify the message number in SIGEDITOR. In that case, the message is read and placed into the field automatically. Also, you can associate an indicator with the field in SIGEDITOR, then display the field by turning the indicator

ON at run time.

You cannot use the Calculation Specification DSPLM operation with RSI to display messages.

Figure 4-29 shows how to display a message contained in the compile-time array MSG.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

     C                      .
     C                      .
     C                      .
1    C                    Z-ADD02         M#
2    C                    SETON                     82
3    C                    EXCPT           R$DATA
     C                      .
     C                      .
     C                      .

     OWORKSTN E             R$DATA
     O                                   K8 "SCN01    "
     O                      ACTNO         6
     O                        .
     O                        .
     O                        .
4    O                 82   MSG,M#     246

**                      MESSAGE ARRAY
                 ------- SELECT MODE WITH COMMAND KEY -------
5       ********** END OF JOB -------------------- PRESS ENTER **********
                          EXISTING ACCOUNT NUMBER
                          INVALID ACCOUNT NUMBER
                        PRESS CMD-5 TO CONFIRM DELETE
```

Figure 4-29. Displaying Messages in an RSI Form

Comments

1 This line sets the index (M#) to the array MSG to the value 2.

2 This line turns ON indicator 82 which is defined in the "Output Field" attribute for this field in SIGEDITOR.

3 This line displays the RSI form SCN01.

4 This line defines the message field as part of the form SCN01. It consists of a single element of the array MSG.

5 This is the second element of the array MSG and is displayed in line 3.

Using Function Keys with RSI. When you enable any or all of the function keys F3 - F8 in SIGEDITOR, their labels (as shown below) appear. If you do not enable them, the labels F3-F8 appear. Even though the function keys are enabled, RSI does not perform any action when the user presses them. You must perform the processing yourself by entering Calculation Specifications.

The function keys are:

Function Key	Description
F1 (COMMAND)	Used in conjunction with one of the keys on the top row of the keyboard to turn on the corresponding command key indicator.
F2 (PRINT SCREEN)	If enabled, the function key label PRINT SCREEN appears though no printing takes place. You must enter Calculation Specifications to perform this function. If not enabled, the label *PRINT SCREEN appears and the contents of the screen are written to RSIPRINT (DEV=LP).
F3 (ROLL UP)	If enabled, you must enter the Calculation Specifications to perform this function.
F4 (ROLL DOWN)	If enabled, you must enter the Calculation Specifications to perform this function.
F5 (CLEAR)	If enabled, you must enter the Calculation Specifications to perform this function.
F6 (HELP)	If enabled, the function lable HELP appears. You must enter the Calculation Specifications to perform this function. If not enabled, the label *HELP appears and the application help facility is activated when the key is pressed.
F7 (RECORD BACKSPACE)	If enabled, you must enter the Calculation Specifications to perform this function.
F8 (DUPLICATION)	If enabled in the Attribute Specification screen of SIGEDITOR, this key fills the field (from the cursor position to the end of the field) with the hexadecimal character 1C (^) and moves the cursor to the next field. You can check for these characters and take appropriate action. For example, you may want to reuse the field's previous value.

If the user presses a function key that is not enabled, the message "FUNCTION KEY NOT ENABLED AT THIS TIME" is displayed momentarily.

To determine which function key was pressed, check the value in the first element of the RSI STATUS array. (Refer to the *HP RPG Reference Manual* for these values.) Figure 4-30 shows how to test the array STAT to determine if the user pressed F4 (ROLL DOWN).

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
                                                                                
1    C                     READ WORKSTN                    H0
     C                      .
     C                      .
     C                      .
2    C           STAT,1    COMP 1123                       49 *F4 PRESSED

-
```

Figure 4-30. Using Function Keys with RSI

Comments

1 This line reads the RSI form and the settings of the function keys.

2 This line tests the first element of STAT for the value 1123 (which indicates that F4 was pressed).

Using Command Key Indicators with RSI. If you enabled a command key when you defined the form using SIGEDITOR, you can use the associated command key indicator in the program to set indicators and to condition operations. See "Creating an RSI Form" at the beginning of this chapter for information on enabling the command keys.

If the user presses a command key that you did not enable, the message "COMMAND KEY NOT ENABLED AT THIS TIME", is displayed momentarily.

Figure 4-31 shows how to use command key 3 (command key indicator KC) to condition an EXSR operation.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
                                                                                
C    KC                 EXSR ADD
```

Figure 4-31. Using the RSI Command Keys

Changing Field Attributes with RSI. From time to time, you may need to define a field's attributes one way in SIGEDITOR, but change them dynamically at run time. You can change video and protection attributes. To do this, you must associate an indicator with the attribute in SIGEDITOR, then turn the indicator ON before the form is displayed.

Figure 4-32 shows how to associate an indicator (41) with the "High Intense" attribute of a field in SIGEDITOR.

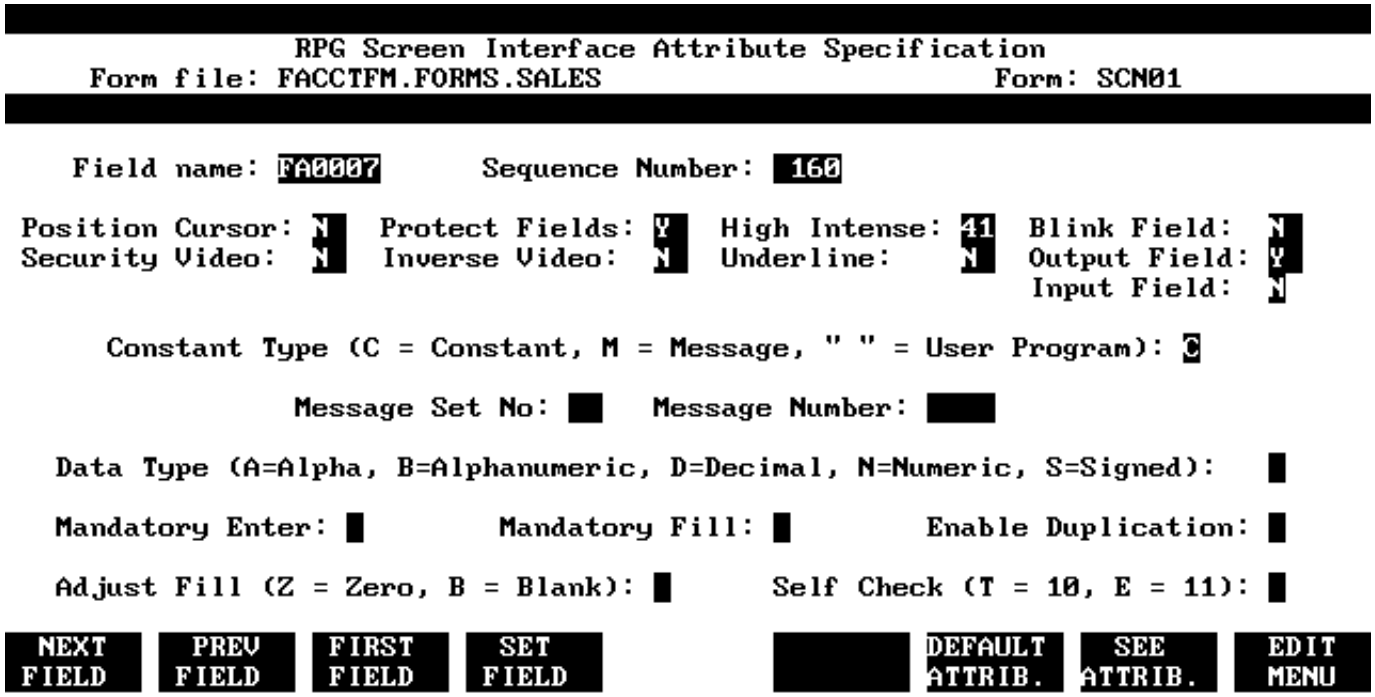


Figure 4-32. Assigning an Indicator to a Field's Attribute

Assuming that the field shown in Figure 4-32 is a field in the output record R$DATA, Figure 4-33 shows how to display it in high intensity.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
1  C                   SETON                       41
2  C                   EXCPT          R$DATA

    OWORKSTN E              R$DATA
    O                         .
    O                         .
    O                         .
```

Figure 4-33. Changing a Field's High Intense Attribute

Comments

1 This line turns ON indicator 41.

2 This line displays the RSI form. All fields having indicator 41 associated with them in SIGEDITOR are highlighted when the form is displayed.

A Sample Program Using RSI. This section lists a complete program that processes data using RSI forms.

The program updates the D-ACCOUNTS data set. This data set keeps customer information such as account numbers, names and addresses. The data set is part of the TurboIMAGE MARKET database whose schema is shown in Figure 3-23. The RSI form is the one shown in Figure 4-24.

Customer records are accessed by the customer's account number. To add a record, the user presses command key 1 then enters an account number. "CMD-1 Add" is highlighted and the user can then proceed to add data for the new customer. When input is complete, the user presses ENTER . Similarly, to modify an existing record, the user presses command key 2 then enters an account number. "CMD-2 Change" is highlighted and the customer information in the D-ACCOUNTS data set is displayed. The user can then change that information, pressing ENTER when input is complete. To display existing account information for a customer, the user presses command key 3 then enters the customer's account number. To delete a customer record, the user presses command key 5 then enters an account number. The user is asked to confirm the delete, then the record is purged from the data set.

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

$CONTROL USLINIT,NAME=FACCT,MAP,LINES=84
HDUMPFILE1    N       U
H*************************************************************************
H*
H*  REMARKS..... ON-LINE UPDATE TO D-ACCOUNTS IN
H*   THE MARKET DATABASE.  ALLOW ADDS, CHANGES,
H*   DELETES AND INQUIRIES - RPG SCREEN INTERFACE (RSI).
H*
H*
H*  INDICATOR USAGE:
H*                01  FILE RECORD IDENTIFIER
H*                02  FILE RECORD IDENTIFIER
H*                40  ERROR INDICATOR
H*          41 - 45  RSI SCREEN HIGHLIGHT INDICATORS
H*                41  ADD MODE - CMD-1
H*                42  CHANGE MODE - CMD-2
H*                43  INQUIRY MODE - CMD-3
H*                45  DELETE MODE - CMD-5
H*                49  ENTER PRESSED INDICATOR
H*                50  SCREEN ADVANCE INDICATOR
H*                    (ALSO RSI PROTECT/POSITION FIELD INDICATOR)
H*                51  RSI PROTECT FIELD INDICATOR DURING INQUIRY
H*                60  NO RECORD FOUND ON CHAIN
```

Figure 4-34. Program to Update D-ACCOUNTS Using RSI

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

     H*               61  END OF CHAIN FOR CURRENT GROUP
     H*               80  RSI ERASE INPUT
     H*               81  RSI OVERRIDE/SOUND ALARM
     H*               82  MESSAGE OUTPUT CONDITIONING
     H*               89  BLANK MESSAGE INDICATOR
     H*               99  ONE TIME INDICATOR
     H*               H0  RSI SCREEN READ ERROR
     H*               LR  END OF JOB
     H*
     H*****************************************************************************
1    FWORKSTN UD  V     256            WORKSTNR
     F                                            KFORMS FACCTFM
     F                                            KSTATUSSTAT
     F*
     F*     ACCOUNT INFORMATION - CHAINED READ MODE
     F*
     FDACCOUNTUC  F     200R06AM     3 DISC                    A
     F                                            KIMAGE MARKET45
     F                                            KLEVEL WRITER
     F                                            KITEM  ACCOUNT-NO
     F                                            KDSNAMED-ACCOUNTS

     E                    MSG     1   5 76              MESSAGES

2    IWORKSTN NS  01   1 C0   2 C1
     I* DACCOUNT DATA SCREEN
     I*                                        1   2 RECID
     I                                         3   8 ACTNO
     I                                         9  24 LNAME
     I                                        25  34 FNAME
     I                                        35  440PHONE
     I                                        45  480PHEXT
     I                                        49  76 ADDR1
     I                                        77 104 ADDR2
     I                                       105 132 ADDR3
     I                                       133 142 ZIPCD
     I                                       143 1500ADATE
     I                                       151 152 TYPEC
     I                                       153 160 MAILC
     I                                       161 1680BDATE
     I                                       169 172 SRCCD
     IDACCOUNTNS  02
     I                                         3   8 ACTNO
```

Figure 4-34. Program to Update D-ACCOUNTS Using RSI (Continued)

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

I                                            9  18 FNAME
I                                           19  34 LNAME
I                                           35  62 ADDR1
I                                           63  90 ADDR2
I                                           91 118 ADDR3
I                                          119 128 ZIPCD
I                                        P 129 1340DPHNE
I                                        4 135 1360PHEXT
I                                          137 144 MAILC
I                                        9 145 1480DBDTE
I                                          149 152 SRCCD
I                                          153 154 TYPEC
I                                          155 1620ADATE
I             DS
I                                            1  110DPHNE
I                                            2  110PHONE
I                                           12  200DBDTE
I                                           13  200BDATE

C*****************************
C* MAINLINE CALCULATIONS
C*****************************
C  N99                EXSR ONETIM
C   KA                EXSR ADD
C   KB                EXSR CHANGE
C   KC                EXSR INQRY
C   KE                EXSR DELETE
C   KH                EXSR ENDJOB
C*************************************************** SUBR ADD
C* ADD RECORD TO DACCOUNT FILE
C*********************************************************************************
C           ADD       BEGSR
C           ADDRST    TAG
C                     EXSR RESET
C                     SETON                     4180  *ADD MODE
C                     EXCPT          R$DATA
C           ADDMSG    TAG
C                     EXSR RDSCRN
C  N49                GOTO ENDADD                    *NOT ENTER
C*-------------------------------------------------
C* CHECK IF RECORD ALREADY EXISTS - IF NOT, ADD RECORD
C*-------------------------------------------------
C                     EXSR RECINP
```

Figure 4-34. Program to Update D-ACCOUNTS Using RSI (Continued)

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

C   40                EXCPT          R$DATA
C   40                GOTO ADDMSG                      *REC EXISTS
C                     EXCPT          $ADD              *ADD RECORD
C                     GOTO ADDRST
C             ENDADD  ENDSR
C*************************************************************** SUBR CHANGE
C* CHANGE RECORD IN DACCOUNT FILE
C*****************************************************************************
5 C             CHANGE  BEGSR
C               CNGRST  TAG
C                       EXSR RESET
C                       SETON                      4280  *CHANGE MODE
C                       EXCPT          R$DATA
C               CNGMSG  TAG
C                       EXSR RDSCRN
C  N49                  GOTO ENDCNG                      *NOT ENTER
C*-----------------------------------------------------
C* IF SCREEN ADVANCE INDICATOR ON - UPDATE RECORD
C*-----------------------------------------------------
C  N50                  GOTO SKPCNG
C                       EXCPT          $UPD              *CHANGE RECORD
C                       GOTO CNGRST
C*-----------------------------------------------------
C* CHECK IF RECORD ALREADY EXISTS - IF SO, DISPLAY RECORD
C*-----------------------------------------------------
C               SKPCNG  TAG
C                       EXSR RECINP
C  N40                  SETON                      50
C  N40                  SETOF                        80
C                       EXCPT          R$DATA
C                       GOTO CNGMSG
C               ENDCNG  ENDSR
C*************************************************************** SUBR INQRY
C* INQUIRY RECORD FROM DACCOUNT FILE
C*****************************************************************************
C               INQRY   BEGSR
C               INQRST  TAG
C                       EXSR RESET
C                       SETON                      4380  *INQUIRY MODE
C                       EXCPT          R$DATA
C               INQMSG  TAG
```

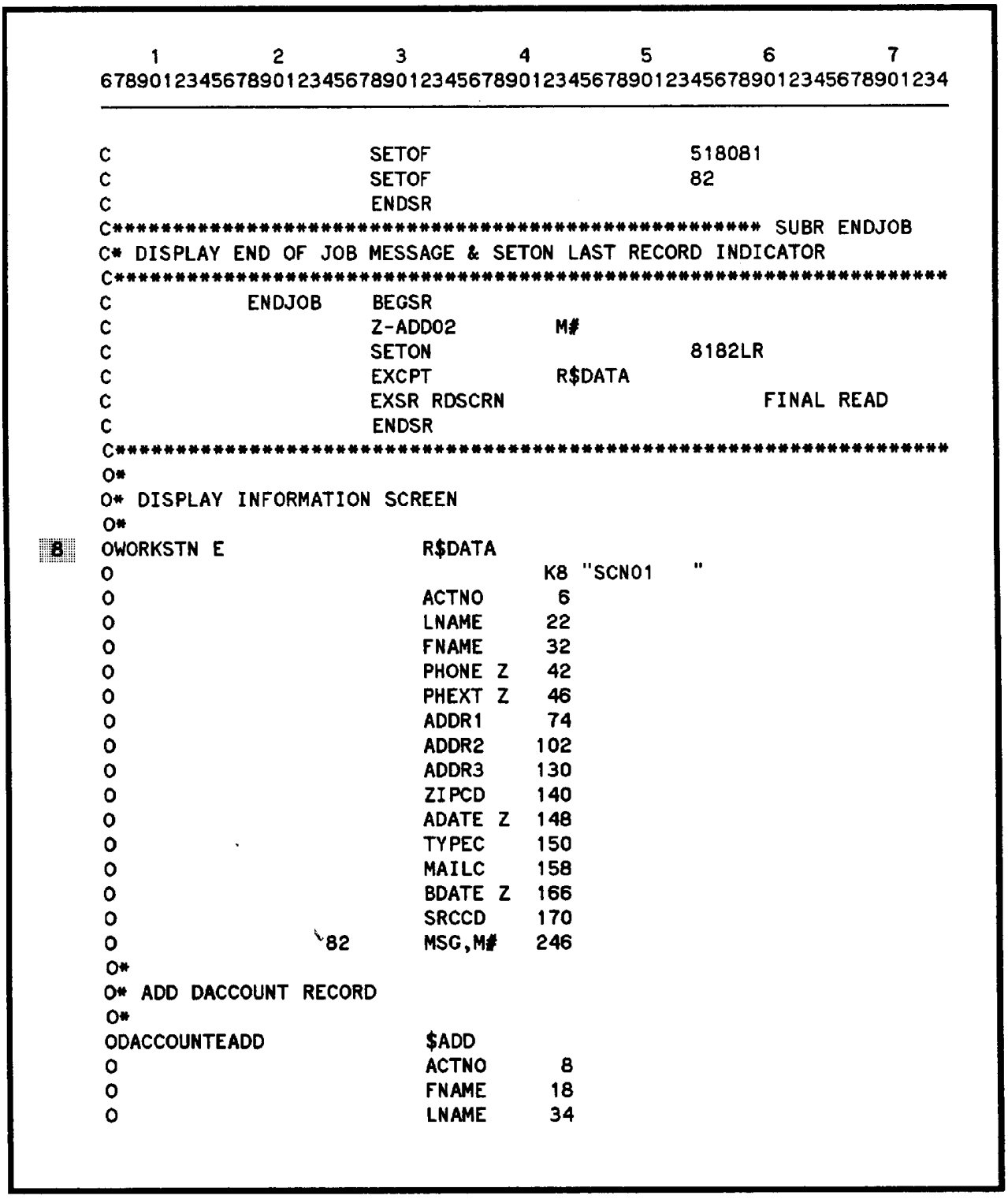
Figure 4-34. Program to Update D-ACCOUNTS Using RSI (Continued)

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

C                    EXSR RDSCRN
C  N49               GOTO ENDINQ
C   50               GOTO INQRST
C*-------------------------------------------------
C* CHECK IF RECORD ALREADY EXISTS - IF SO, DISPLAY RECORD
C*-------------------------------------------------
C                    EXSR RECINP
C  N40               SETOF                      80
C  N40               SETON                    5051  *SCREEN ADVNCE
C                    EXCPT          R$DATA
C                    GOTO INQMSG
C           ENDINQ   ENDSR
C************************************************** SUBR DELETE
C* DELETE RECORD IN DACCOUNT FILE
C*************************************************************************
C           DELETE   BEGSR
C           DELRST   TAG
C                    EXSR RESET
C                    SETON                    4580  *DELETE MODE
C                    EXCPT          R$DATA
C           DELMSG   TAG
C                    EXSR RDSCRN
C  N49NKE                                           *NOT ENTER
COR    KEN50         GOTO ENDDEL                    *UNLS DELETE
C*-------------------------------------------------
C* IF SCREEN ADVANCE INDICATOR ON (AND CMD-5 PRESSED) - DELETE RECD
C*-------------------------------------------------
C   50 KE            EXCPT          $DEL           *DELETE RECORD
C   50 KE            GOTO DELRST
C*-------------------------------------------------
C* CHECK IF RECORD ALREADY EXISTS - IF NOT DISPLAY ERROR
C*-------------------------------------------------
C  N50               EXSR RECINP
C  N40               SETON                    5051
C  N40               SETOF                      80
C                    EXCPT          R$DATA
C   40               GOTO DELMSG
C                    Z-ADD05        M#
C                    SETON                    81  82
```

Figure 4-34. Program to Update D-ACCOUNTS Using RSI (Continued)

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

C                    EXCPT          R$DATA
C                    GOTO DELMSG
C           ENDDEL   ENDSR
C************************************************** SUBR RECINP
C* CHAIN RECORD FROM DACCOUNT FILE
C*********************************************************************************
6   C           RECINP   BEGSR
C*
C* CHECK FOR EXISTENCE OF RECORD - DISPLAY ERROR ACCORDING TO MODE
C*
C           ACTNO    CHAINDACCOUNT             6061
C   41N60            Z-ADD03        M#         40    *REC EXISTS
C  N41 60            Z-ADD04        M#         40    *NO REC FOUND
C    40              SETON                     81  82
C                    ENDSR
C************************************************** SUBR ONETIM
C* FIRST CYCLE CALCULATIONS
C*********************************************************************************
C           ONETIM   BEGSR
C           ONETAG   TAG
C                    SETON                     505180
C                    SETON                       82
C                    Z-ADD1         M#      20
C                    EXCPT          R$DATA
C                    EXSR RDSCRN
C    49              GOTO ONETAG
C                    SETON                     99
C                    ENDSR
C************************************************** SUBR RDSCRN
C* READ INFO FROM SCREEN AND CHECK COMMAND KEYS PRESSED
C*********************************************************************************
7   C           RDSCRN   BEGSR
C                    SETOF                     4049
C                    READ WORKSTN                 HO
C           STAT,1   COMP 0                       49 *ENTER PRESSED
C           ENDCHK   ENDSR
C************************************************** SUBR RESET
C* RESET ERROR/MODE/ADVANCE INDICATORS
C*********************************************************************************
C           RESET    BEGSR
C                    SETOF                     404142
C                    SETOF                     434550
```

Figure 4-34. Program to Update D-ACCOUNTS Using RSI (Continued)



```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

C                    SETOF                          518081
C                    SETOF                          82
C                    ENDSR
C*********************************************************** SUBR ENDJOB
C* DISPLAY END OF JOB MESSAGE & SETON LAST RECORD INDICATOR
C*************************************************************************
C           ENDJOB   BEGSR
C                    Z-ADD02        M#
C                    SETON                          8182LR
C                    EXCPT          R$DATA
C                    EXSR RDSCRN                          FINAL READ
C                    ENDSR
C*************************************************************************
O*
O* DISPLAY INFORMATION SCREEN
O*
8  OWORKSTN E                R$DATA
O                                     K8 "SCN01    "
O                            ACTNO     6
O                            LNAME    22
O                            FNAME    32
O                            PHONE Z  42
O                            PHEXT Z  46
O                            ADDR1    74
O                            ADDR2   102
O                            ADDR3   130
O                            ZIPCD   140
O                            ADATE Z 148
O                            TYPEC   150
O                            MAILC   158
O                            BDATE Z 166
O                            SRCCD   170
O                  82        MSG,M#  246
O*
O* ADD DACCOUNT RECORD
O*
ODACCOUNTEADD                $ADD
O                            ACTNO     8
O                            FNAME    18
O                            LNAME    34
```

Figure 4-34. Program to Update D-ACCOUNTS Using RSI (Continued)

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

O                          ADDR1     62
O                          ADDR2     90
O                          ADDR3    118
O                          ZIPCD    128
O                          PHONE    134P
O                          PHEXT    136B
O                          MAILC    144
O                          BDATE    148B
O                          SRCCD    152
O                          TYPEC    154
O                          ADATE    162
O*
O* CHANGE DACCOUNT RECORD
O*
O         E                $UPD
O                          FNAME     18
O                          LNAME     34
O                          ADDR1     62
O                          ADDR2     90
O                          ADDR3    118
O                          ZIPCD    128
O                          PHONE    134P
O                          PHEXT    136B
O                          MAILC    144
O                          BDATE    148B
O                          SRCCD    152
O                          TYPEC    154
O                          ADATE    162
O*
O* DELETE DACCOUNT RECORD
O*
O         EDEL             $DEL
**                     MESSAGE ARRAY
                 ------ SELECT MODE WITH COMMAND KEY -------
          ********** END OF JOB -------------------- PRESS ENTER **********
                          EXISTING ACCOUNT NUMBER
                          INVALID ACCOUNT NUMBER
                        PRESS CMD-5 TO CONFIRM DELETE
```

Figure 4-34. Program to Update D-ACCOUNTS Using RSI (Continued)

Comments

1 This line begins the definition of the RSI WORKSTN file WORKSTN. The forms file for WORKSTN is FACCTFM and the STATUS array is STAT. (STAT is used to determine if the user pressed ENTER .)

2 This line begins the description of the record used for all terminal input. (The lines were generated by SIGEDITOR, then modified.)

3 This line begins the mainline portion of the program. The subroutine ONETIM is executed only at the beginning of the first logic cycle. The other subroutines are executed when the user presses the associated command key (for example, CMD-2 Change).

4 This line begins the subroutine that adds records to the D-ACCOUNTS data set. The subroutine first resets (in the RESET subroutine) all indicators used in the previous cycle. Next, indicators 41 and 80 are turned ON. Indicator 41 is used to change the video attribute for the constant "CMD-1 Add" so that the user knows what the current mode is. Indicator 80 is the "erase input fields" indicator. It causes all input fields to be cleared before the form is displayed. (This saves having to initialize each field yourself before displaying the form.)

The form is displayed by the EXCPT R$DATA operation. Then, the RDSCRN subroutine reads the form and checks to see if the user pressed the ENTER key or some other enabled command key. Control remains in the subroutine until the user presses another command key. When this happens, control proceeds to the end of the subroutine. (The mainline portion of the program processes the command key request during the next logic cycle.)

The RECINP subroutine checks to see if the data record already exists in D-ACCOUNTS. If it does, the account number field is cleared, and the form is redisplayed with the error message "EXISTING ACCOUNT NUMBER" (override is in effect so that the user can enter the correct account number). If there are no errors (indicator 40 is OFF), a record is added to the data set by the EXCPT $ADD operation and control goes back to the beginning of this step.

5 This line begins the subroutine that updates D-ACCOUNTS data set records. The RESET subroutine turns OFF indicators that were turned ON during the previous logic cycle, then indicators 42 and 80 are turned ON. The RSI form is displayed, then read. If the user pressed another command key (indicator 49 is OFF) control skips to the end of the subroutine.

The RECINP subroutine reads the record from the D-ACCOUNTS data set. If the record exists, indicator 40 is turned ON and the form containing account information is displayed. Control skips to CNGMSG where changes are read. If the record does not exist, indicator 80 is turned OFF so that the "Erase input fields" attribute will not be in effect. The error message "INVALID ACCOUNT NUMBER" is displayed and control skips back to CNGMSG to accept

the new account information.

After the new information is read, the record in D-ACCOUNTS is updated by the EXCPT $UPD operation. Control remains in the subroutine until the user changes modes (presses a different command key).

6 This line begins the subroutine that locates records in the D-ACCOUNTS data set and prepares error messages for display. The CHAIN operation reads D-ACCOUNTS using ACTNO as its key field. If the record is not found, indicator 60 is turned ON.

If add mode is in effect and the record is found, or if other modes are in effect and the record is not found, an error results. The index of the appropriate error message in MSG is placed in the variable M# and indicator 82 is turned ON.

Indicator 81 is turned ON to allow override and to sound the alarm (beep). When override is in effect, data on the screen remains unaltered except for attributes conditioned by indicators and output fields conditioned by indicators that are ON.

7 This line begins the subroutine that reads the form and checks to see if the user pressed a function key. Indicator H0 is turned ON by the READ WORKSTN operation when a terminal read error occurs.

When READ is executed, a code that identifies which function key (if any) was pressed, is placed in the first element of the STAT array. If the ENTER key was pressed this element equals 0 and indicator 49 is turned ON. Since none of the function keys were enabled on the form, the only other possible value is 2 (which indicates that a command key was pressed).

8 This line begins the output record description for the RSI form. The specifications were generated by SIGEDITOR, then modified. The form name is entered as a constant in the first line. It has a length of 8 and the prefix "K". The field used for messages, MSG, is indexed by M# (M# contains the number of the message). MSG is conditioned by indicator 82 and the SIGEDITOR "Output Data" attribute for MSG also contains 82.

Using RSI CONSOLE Files

RSI CONSOLE files are RSI files that are used for input only and whose forms file is generated automatically during compilation. RSI handles all file input and output during run time.

To use an RSI CONSOLE file in an RPG program, perform the following steps (the steps are discussed in detail in the following sections in this chapter):

* **Define the RSI CONSOLE form**

  Enter the File Description and Input Specifications that describe the form, then compile the program. The forms file is generated automatically. If you wish, you can use SIGEDITOR to modify the form.

  File Description Specifications -

  These specifications define the file as an RSI WORKSTNC file.

  Input Specifications -

  These specifications define the fields that are used for input.

* **Process data entered in the form**

  Calculation Specifications -

  Since RSI handles all of the input and output for the CONSOLE form, you do not enter Calculation Specification operations to do this. However, you must enter the usual operations to process the data once it is read.

Defining an RSI CONSOLE File. Figure 4-35 shows the File Description and Input Specifications that define the RSI CONSOLE file SCRNFILE. SCRNFILE is defined as a WORKSTNC file (line 1) which consists of three different forms. The first form is FORM01 and is defined starting with line 2. (The form name is "FORM" suffixed by the record-identifying indicator for the record type.) Lines 3 and 4 start the definitions for the second and third forms, respectively.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
1 FSCRNFILEIP  V      80            WORKSTNC    B

2 ISCRNFILE011 01   1 CH   2 CD
  I* HEADER RECORD
  I                                        3   80INVNUM
  I                                        9  140INVDAT
  I                                       15   34 INVNAM
  I                                       35   54 INVADR
  I                                       55   74 INVCTY
  I                                       75  790INVZIP
3 I        02N 02   1 CD   2 CE
  I* DETAIL RECORD
  I                                        3   22 INVDES
  I                                       23  312INVAMT
  I                                       32  370INVGLN
4 I        031 03   1 CS   2 CU
  I* SUMMARY RECORD
  I                                        3   40INVCNT
  I                                        5  132INVTOT
  I                                       14  190INVPAY
```

Figure 4-35. Defining an RSI CONSOLE Form

When the forms FORM01, FORM02 and FORM03 are displayed at run time, they appear as shown in Figure 4-36, Figure 4-37 and Figure 4-38.

Figure 4-36. FORM01 Displayed at Run Time

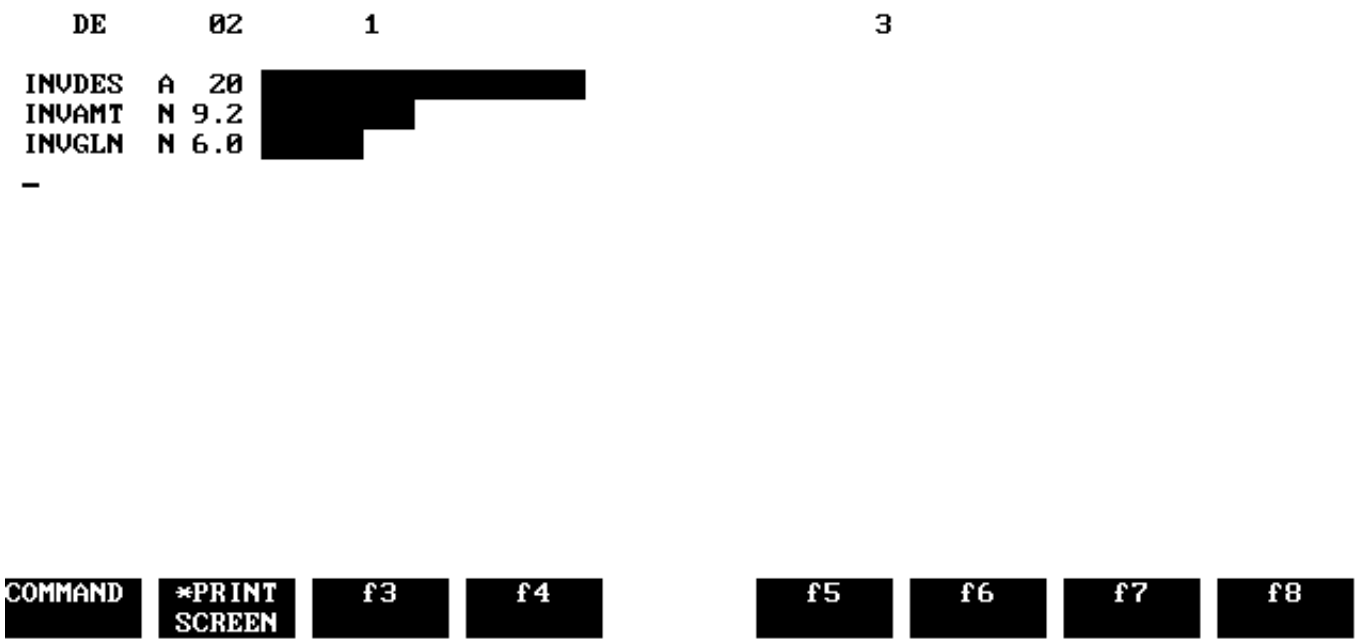


Figure 4-37. FORM02 Displayed at Run Time

Figure 4-38. FORM03 Displayed at Run Time

You can modify the forms generated by RSI yourself or by using SIGEDITOR. However, do not change the top (status) line, the data type and length of the fields or the order of the fields.

Figure 4-39 shows how you can modify FORM01 in Figure 4-35 to make it

easier to understand and use.

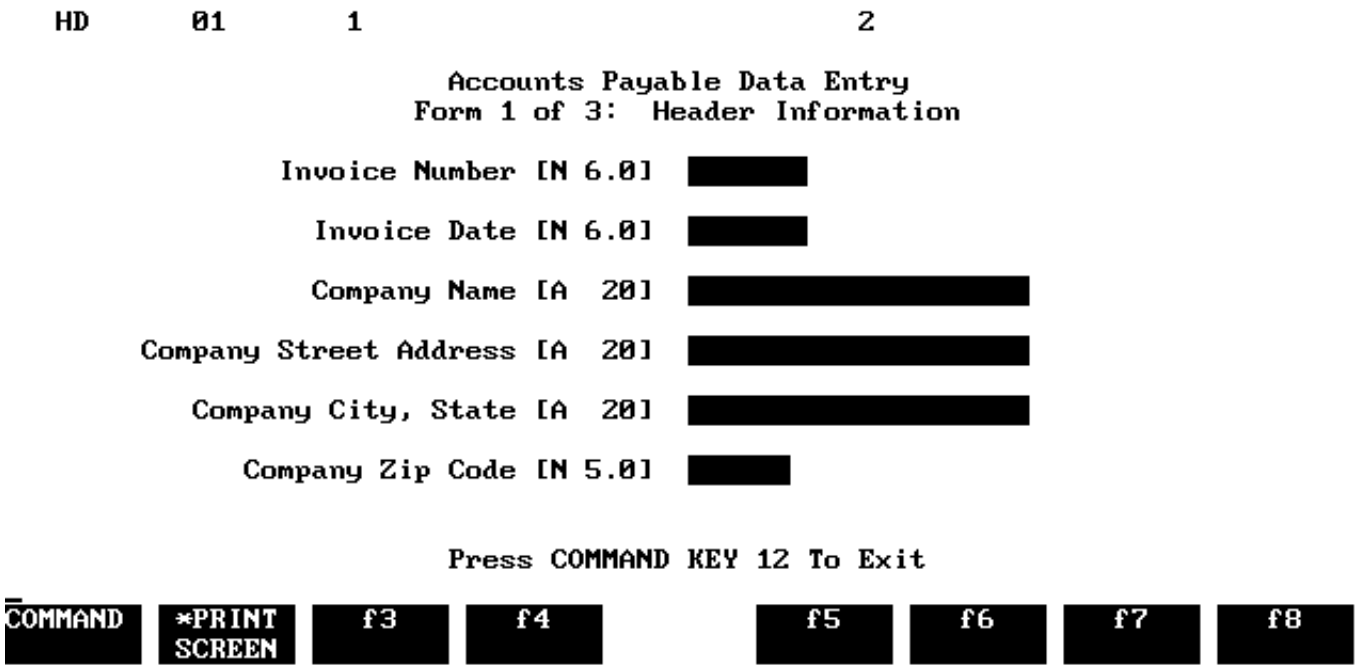


Figure 4-39. Modifying an RSI CONSOLE Forms File

Displaying and Reading an RSI CONSOLE File. You do not need to enter Calculation Specifications to display and read an RSI form, the input and output are handles automatically by RSI during normal logic cycle processing. Once the data is read, you may enter Calculation Specifications to process the data the same way you do other file data.

Displaying Messages with an RSI CONSOLE File. You cannot display messages when using RSI CONSOLE files, since CONSOLE forms are used for input only. Simple input editing is handled by RSI based on each field's data type. If the user enters invalid data, RSI displays an error message and allows the user to correct the data.

Using Function Keys with an RSI CONSOLE File. There are just two function keys that are used with RSI CONSOLE files and they are described below. You cannot change the actions that they perform.

Function Key	Description
F1 (COMMAND)	Used in conjunction with one of the keys on the top row of the keyboard to select a command key (see the next section).
F2 (PRINT SCREEN)	Copies the contents of the screen to the print file RSIPRINT (DEV=LP).

If the user presses another function key, the message "FUNCTION KEY NOT ENABLED AT THIS TIME" is displayed momentarily.

Using Command Keys with an RSI CONSOLE File. If you defined more than one record type in the CONSOLE file, the user must press a command key at run time to select the type. For example, to select record type 3 (SUMMARY RECORD) in Figure 4-35, the user presses command key 3. The command keys do not turn on the command key indicators, as they do with

regular RSI files. Therefore, you cannot use the indicators in the normal way to condition Calculation Specification operations.

If the user presses a command key not associated with any record type, the message "FORM CHANGE OR EXIT NOT ACCEPTABLE" is displayed momentarily.

A Sample Program Using an RSI CONSOLE File. This section lists a complete program that uses RSI CONSOLE files to read and balance invoices. Each invoice can have three or more records, where each record belongs to a different record type (the record types are sequenced). Each invoice must have a header record that gives the customer name and address, one or more detail records that list individual items on the invoice and a summary record that gives the total invoice amount. The user enters data for the header, detail an summary records in that order. Each group of invoice records are written to disc along with any errors.

When the program is executed, the first form (FORM01) is displayed. When the user finishes entering data on the form and presses ENTER , the second form (FORM02) is displayed. When ENTER is pressed again, the second form is redisplayed because it can occur more than once in a group. The user can continue entering detail records or can select the third form by pressing command key 3 (the record-identifying indicator for the last record type). When input is complete for FORM03, the user presses ENTER and the first form is displayed again, starting the sequence over. The user presses command key 12 to end the program.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

     $CONTROL NAME=INV36S
     HDUMPFILE JF           X     B       B N     P1 1

1    FSCRNFILEIP  V      80             WORKSTNC    B
     FOUTDATA O   F      80             DISC

2    ISCRNFILE011 01   1 CH   2 CD
     I* HEADER RECORD
     I                                          3   8OINVNUM
     I                                          9  14OINVDAT
     I                                         15  34 INVNAM
     I                                         35  54 INVADR
     I                                         55  74 INVCTY
     I                                         75  79OINVZIP
3    I        02N 02   1 CD   2 CE
     I* DETAIL RECORD
     I                                          3  22 INVDES
     I                                         23  31 2INVAMT
     I                                         32  37OINVGLN
4    I        031 03   1 CS   2 CU
     I* SUMMARY RECORD
     I                                          3   4OINVCNT
     I                                          5  13 2INVTOT
     I                                         14  19OINVPAY
```

Figure 4-40. Invoice-Balancing Program that Uses an RSI CONSOLE File

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

5  C      01              SETOF                          909192
   C      01              Z-ADD0          SUMAMT  92
   C      01              Z-ADD0          SUMCNT  20
   C      02              ADD  INVAMT     SUMAMT
   C      02              ADD  1          SUMCNT
   C      03   SUMAMT     COMP 0                            90
   C  N90 03   SUMAMT     COMP INVTOT                   9191
   C      03   SUMCNT     COMP INVCNT                   9292

   OOUTDATA D             01
   O                                       3 "HDR"
   O                           INVNUM      9
   O                           INVDAT     15
   O                           INVNAM     35
   O                           INVADR     55
   O                           INVCTY     75
   O                           INVZIP     80
   O         D             02
   O                                       3 "DET"
   O                           INVDES     23
   O                           INVAMT     32
   O                           INVGLN     38
   O         D             03
   O                                       3 "SUM"
   O                           INVCNT      5
   O                           INVTOT     14
   O                           INVPAY     20
   O                       91  SUMAMT     29
   O                       92  SUMCNT     31
   O                       91             55 "INVOICE AMOUNT ERROR    "
   O                       92             55 "INVOICE LINE COUNT ERROR"
   O                   91    92           55 "INV AMOUNT/LINE CNT ERR "
```

Figure 4-40. Invoice-Balancing Program that Uses an RSI CONSOLE File (Continued)

Comments

1 This line begins the definition of the RSI WORKSTNC file SCRNFILE. The forms file for SCRNFILE is INV36SFM which is the program name in the $CONTROL subsystem command suffixed by "FM".

2 This line begins the description of the invoice header records.

Columns 15-16 contain the record sequence (01) in the group.

Column 17 is 1 to indicate that there is only one header record per invoice.

Column 18 is blank to indicate that this header record is

required.

Columns 19-20 contain the record-identifying indicator (01) for the form. The indicator is appended to "FORM" to create the form name FORM01.

Columns 21-41 contain the record identification code (HD) for the record. (The code can be one or two characters but must begin in the first position of the record.)

3 This line begins the description of the record used for detail invoice records. The form name is FORM02, the record-identifying indicator is 02 and the record identification code is DE.

Column 17 is N to specify that there can be more than one detail record per invoice.

4 This line begins the description of the record that summarizes each invoice. The form name is FORM03, the record-identifying indicator is 03 and the record identification code is SU.

5 This line begins the Calculation Specification operations that do invoice-balancing. It also flags invoices that are out-of-balance.

Enabling the BREAK Key

On 264X terminals, the BREAK key is physically positioned near ENTER . As a result, it is easy to press BREAK accidentally. Since it is difficult to recover from a break, this key is disabled automatically whenever you use a WORKSTN file.

If you want to use the BREAK key, indicate this in column 52 of the terminal's File Description Specification as shown below.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
                                                                        
1  FTERMINALUD  V     256            WORKSTN     B
F                                                  KFORMS FACCOUNT

-
```

Figure 4-41. Enabling the BREAK Key

Comments

1 This line defines the terminal file, TERMINAL.

Column 52 contains B to enable the BREAK key.

Handling Run-Time Errors

When you run a program that uses VPLUS or RSI, run-time errors are handled differently from programs that do not use them. When using VPLUS, an error message is displayed in the message window and for RSI, it is displayed on a separate screen. VPLUS and RSI then reset the function keys so that the user can select an error response. The function keys and their responses are:

Function Key:	Response
F1	Continue execution
F2	Skip the input record containing the error and continue execution
F3	Terminate the program by executing the normal termination code
F4	Terminate the program immediately
F5	Terminate normally and print an error dump
F6	Terminate immediately and print an error dump

You can specify the function key responses within your program instead of letting the user select them. Use the Header Specification to do this. For each error you anticipate, enter the response in columns 56-71. Figure 4-42 shows how to suppress the error message for arithmetic overflow and to continue program execution when arithmetic overflow occurs.

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
1  H                                                   N         0
```

Figure 4-42. Specifying a Run-Time Error Response

Comments

1 This line specifies special error handling for arithmetic overflow errors.

Column 55 contains N to suppress the display of arithmetic overflow error messages.

Column 65 contains 0 to suppress arithmetic overflow messages and to continue program execution.

Chapter 5 Processing Data in an RPG Program

This chapter discusses ways to manipulate data once it is read into a program. The RPG language features that make it easier to handle data in a program are tables, arrays, and data structures. You can also use subroutines to manipulate data in a program. Internal and external subroutines are discussed in this chapter.

This chapter does not address all of the ways to work with data internally in a program. For information on indicators, numeric editing, changing the hexadecimal value of characters and other features, see the *HP RPG Reference Manual*.

Using Tables and Arrays

Tables and arrays are the most convenient and efficient way to organize elements of related data so that you can reference them individually or as a group. Each element in a table or array has the same length, the same data type (numeric or alphanumeric) and, if numeric, the same number of decimal positions. Tables and arrays are defined using the File Extension Specification.

You commonly use tables to keep rates, constants or data of any kind that remains the same during program execution. For instance, you can enter a table for state tax rates.

Normally, you use arrays to save data accumulated during program execution. However, arrays and tables can often be used interchangeably.

Differences Between Tables and Arrays

Tables are loaded either during program compilation or before a program is executed. Tables can only be processed one element at a time; you cannot perform operations on an entire table in one operation. Table names start with TAB. When you use a table name, RPG selects the table element found during the last lookup (LOKUP) operation.

Arrays are loaded either during compilation, or before or during program execution. You can process arrays one element at a time. You can also process all elements of an array in one operation by using the array name.

Kinds of Tables and Arrays

There are three kinds of tables and arrays: compile-time, preexecution-time and execution-time. They are grouped into these categories according to when they are loaded or created by a program.

The kind of table or array that you use depends how you're using it in your program:

This kind of table or array:	Is used when:
Compile-Time	The table or array elements change infrequently, if at all.
Preexecution-Time	The table or array elements change frequently.
Execution-Time	The array elements change frequently or they are created by the program.

Compile-time, preexecution-time and execution-time tables and arrays are discussed in the next three sections.

Compile-Time Tables and Arrays. Compile-time table and array values are loaded during program compilation. These values can be kept as part of the source program or they can be saved in separate files on disc. Once compiled, compile-time tables and arrays become a part of the object program. To permanently change elements in them, you must recompile the program.

Figure 5-1 shows what compile-time array values look like when saved in a disc file. The name of this particular disc file is FLBL1. It contains the function key labels for the program shown in Figure 5-2.

```
ADD        MODE
CHANGE     MODE
INQUIRY    MODE

DELETE     MODE


EXIT
```

Figure 5-1. Compile-Time Array Entries on Disc

Figure 5-2 shows segments of a program that uses two compile-time arrays, LBL and MSG (the program from which the segments are taken is listed in Figure 4-23). LBL is an array that contains function key labels. The labels come from the file, FLBL1, shown in Figure 5-1. The MSG array contains messages that are displayed on the terminal during program execution. The actual messages for the MSG array follow the Output Specifications in Figure 5-2.

```
      1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

1     E                     LBL     1   8 16                FUNC KEY LBLS
2     E                     MSG     1  20 79                MESSAGES

3     C           LBL       SET                         F@
      C                       .
      C                       .
      C                       .
4     C                     Z-ADD5          M
5     C                     MOVEL"SHOMSG"   ACTION            WRITE WINDOW
6     C                     EXCPT           V$MESG
      C                       .
      C                       .
      C                       .

7     O*         OUTPUT MESSAGE TO WINDOW
      O*
      OTERMINALE                  V$MESG
      O                           ACTION     6
      O                           MSGLEN     8
      O                           ENHANC     9
8     O                           MSG,M     89

9     AFLBL1
**                    MSG - MESSAGE ARRAY
SELECT MODE WITH FUNCTION KEY
MAKE CHANGES AND HIT ENTER (F7 TO CANCEL CHANGE)

YOU MAY NOT CHANGE ACCOUNT NUMBER OR LAST NAME  (F7 TO CLEAR)
INVALID KEY (F7 TO CLEAR)
EXISTING ACCOUNT NUMBER
INVALID ACCOUNT NUMBER
PRESS F5 AGAIN TO CONFIRM DELETE (F7 TO CANCEL DELETE)
```

Figure 5-2. Using Compile-Time Arrays

Comments

1 This line defines the LBL array.

Columns 33-35 contain 1 to specify that there is one function key label per record in the array file on disc.

Columns 36-39 contain 8 to specify the number of elements in the array.

Columns 40-42 contain 16 to specify the length of each element in the array.

2 This line defines the MSG array.

Columns 33-35 contain 1 to specify that there is one message per record in the array.

Columns 36-39 contain 20 to specify the maximum number of elements in the array.

Columns 40-42 contain 79 to specify the length of each element in the array.

3 This line enables the function keys and displays their labels from the LBL array.

4 This operation sets the value of the index (M) in the MSG array to 5.

5 This line enters the VPLUS action, SHOMSG, into the output field, ACTION.

6 This performs exception output for the record associated with EXCPT Group V$MESG.

7 This line begins the description of the output record used for displaying VPLUS messages.

8 This line specifies that the fourth field in the output record is an element (selected by M) from the MSG array.

9 This line is a File Name record. It identifies the disc file (FLBL1) which contains values for the LBL array. The contents of FLBL1 are loaded and compiled as part of the program. Figure 5-1 shows the contents of this file.

The MSG array entries follow this line.

Preexecution-Time Tables and Arrays. Preexecution-time tables and arrays are loaded before a program begins the RPG program cycle. Elements in these tables and arrays are available before files are read and before calculations are performed.

Preexecution-time tables and arrays can reside on the same disc device or on different disc devices. They are loaded in the same order as their File Extension Specifications are listed.

Table 5-1 is a table of postal rates that is used to calculate item shipping charges. (Only the first 8 and the last 3 entries are shown. Since the last entry is 30 pounds, the items are assumed to weigh no more than 30 pounds.) The postal charges are based on an item's weight, with fractions of pounds putting the item in the next higher weight category.

Table 5-1. Weights/Postal Charges

Weight (lbs.)	Postal Charges
1	$0.45
2	.50
3	.50
4	.55

5	.55
6	.55
7	.60
8	.65
.	.
.	.
.	.
28	1.00
29	1.05
30	1.05

Figure 5-3 shows the contents of the postal rates table as it is saved on disc. The name of this particular disc table file is WGHRATE. It is used in the program shown in Figure 5-4.

```
010045
020050
030050
040055
050055
060055
070060
080060
   .
   .
   .
280100
290105
300105
```

Figure 5-3. Preexecution-Time Table Entries on Disc

Figure 5-4 shows program segments that look up postal rates contained in the file, WGHRATE (see Figure 5-3). The program uses the rates to calculate invoice shipping charges. The program first reads an item from the ORDER file. It then searches the WGHRATE table (line 7) to find a weight less than or equal to the item's WEIGHT. When a weight is found, the corresponding rate is moved to the invoice record field, TABRAT.

```
     1         2         3         4         5         6         7
    6789012345678901234567890123456789012345678901234567890123456789012345678901234
1   FORDER   IP  F     100            DISC
2   FWGHRATE IT  F      80           EDISC
3   FINVOICE O          132            PRINTER

4   E    WGHRATE          TABWGT  1  30  3 1 TABRAT  3 2

5   IORDER   NS  01   1 CO
    I                                      3   60CUSTNO  M1
    I                                      7  110ITEM
6   I                                     12  141WEIGHT

    C                      .
    C                      .
    C                      .
7   C   02      WEIGHT    LOKUPTABWGT    TABRAT       2323FIND POST RATE
    C                      .
    C                      .
    C                      .

8   OINVOICE D        23 02   TABRAT    45 "$ .  "
```

Figure 5-4. Using a Preexecution-Time Table

Comments

1 This line defines the ORDER file containing the items to be printed on the invoices.

2 This line defines the WGHRATE table file on disc. The File Extension Specification on line 4 describes the table in detail.

3 This line defines the INVOICE file. It is a report file assigned to the printer.

4 This line defines the WGHRATE table.

Columns 11-18 contain WGHRATE to identify the table file. WGHRATE is loaded into the program before it is executed.

Columns 27-32 contain TABWGT to define the weight table.

Columns 33-35 contain 1 to specify that there is one weight table entry and one rate table entry per record.

Columns 36-39 contain 30 to specify that there are 30 elements in each table.

Columns 40-42 contain 3 to specify that the length of each weight table element is 3 positions.

Column 44 contains 1 to specify that each weight table element has one decimal place (fractions of a pound are possible).

Columns 46-51 contain TABRAT to define the rate table as an alternating table.

Columns 52-54 contain 3 to specify the length of the rate table element.

Column 56 contains 2 to specify that each rate table element has 2 decimal places.

5 This line begins the description of record 01 in the ORDER file.

6 This line defines the WEIGHT field in the ORDER file. This field will be used as the search field to find the correct postal rate in the WGHRATE table.

7 This line specifies a LOKUP operation which searches the WGHRATE table to find the postal rate corresponding to each item's weight.

Columns 10-11 contain 02 to specify that this operation is performed when indicator 02 is turned on (assume that indicator 02 is turned on when an item requires postal delivery).

Columns 18-27 contain WEIGHT to specify the search field.

Columns 28-32 contain LOKUP to specify the table look-up operation.

Columns 33-42 contain TABWGT to specify the table field to be searched.

Columns 43-48 contain TABRAT to specify the table field to be selected when WEIGHT is equal to or less than TABWGT.

Columns 56-59 contain 2323 to specify that indicator 23 is turned on when a weight is found in TABWGT equal to or less than WEIGHT.

8 This line defines the INVOICE record that prints the postal rate found by a successful search of the WGHRATE table. The rate table name, TABRAT, selects the element in the table (not the entire table) found by the last LOKUP operation.

Execution-Time Arrays. Execution-time arrays are loaded from files named in File Description and File Extension Specifications. They can also be created by Calculation Specifications. There are no execution-time tables.

Figure 5-5 lists a program that uses an execution-time array, AR. This program is an online program that lets users query an employee master file by department number. When the department number is found, the last name of each employee in the department is saved in the AR array. When the array is filled with ten names, they are displayed on the terminal.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-----------------------------------------------------------------------------------------

1    FMASTFL  IF  F     256L 4AI     1 DISC
2    FTERMINALUD  V     373            WORKSTN   L5B
     F                                            KFORMS FACCOUNT

3    E                    AR          10 29

4    IMASTFL  AA  01  80 CA
     I                                          1   4 DEPT
     I                                         11  39 AR,I
5    ITERMINALNS  01
     I                                          1   4 DEPTNO

     C                    SETOF                     21
     C                      .
     C                      .
     C                      .
6    C                    READ TERMINAL                  LR READ RECORD
     C   LR               GOTO END
     C                    MOVE *BLANKS    AR
     C                    Z-ADD0          I       20
     C           LOOP     TAG
7    C           DEPTNO   CHAINMASTFL             6060
     C   60               GOTO END
8    C                    ADD  1          I
9    C           I        COMP 11                      21
     C   21               GOTO END
     C                    READ MASTFL                  90
     C   90               GOTO END
     C                    GOTO LOOP
     C           END      TAG

10   OTERMINALE       21
     O                         DEPT       8
11   O                         AR       373
```

Figure 5-5. Using an Execution-Time Array

Comments

1 This line defines the MASTFL file containing employee records.

2 This line defines the TERMINAL file that is processed in line (character) mode.

3 This line defines the array AR. It is used to store the last

names of employees matching a selected department number.

Columns 27-32 contain AR which is the name of the last names array.

Columns 36-39 contain 10 to specify that there are 10 elements in the array.

Columns 40-42 contain 29 to specify that each element in the array is 29 characters long.

4 This line begins the input record description for the employee master file, MASTFL.

5 This line begins the input record description for the terminal file, TERMINAL.

6 This line reads a department number entered from the terminal.

7 This line reads a record (for the department) in the employee master file, MASTFL.

8 This line increments the index, I, to the AR array. (An employee record matched the department entered by the user.)

9 This line compares the AR index I to its maximum value. If the array is full, indicator 21 is turned on.

10 This line begins the output record description for the teminal file, TERMINAL. This record is displayed when indicator 21 is turned on.

11 This line specifies that the second field to be displayed in the TERMINAL output record is contained in the AR array.

Using Data Structures

A data structure is an area of storage that contains subfields. You can use data structures to describe input areas in more than one way. You may want to do this because:

* You need to process records in the same file and they have different formats.
* You need to reference subfields of fields as well as the fields themselves.
* You can process input fields more conveniently if they are rearranged.

Data structures can also be used to exchange information between other RPG programs.

You use Input Specifications to define a data structure. Place data structures at the end of all of the other Input Specifications in the program.

The "data structure" sections which follow in this chapter discuss the ways you can use data structures to accomplish the objectives listed above.

NOTE HP implements data structures differently from some RPG vendors. Data structures occupy separate memory areas from the areas they redefine. Thus, you should not use data structures simply to reduce memory requirements in your program.

Using Data Structures to Define Data More Than One Way

This section explains how to use data structures to describe the same data areas in more than one way. This enables you to use the same data several different ways. For example, you can use data structures to

describe different record formats within a file. Or you can use data structures to redefine numeric fields in order to process them differently.

Figure 5-6 and Figure 5-7 show how to describe three different input records in the same file. The example in Figure 5-6 does not use data structures and the example in Figure 5-7 does. In Figure 5-6, the second and third records redescribe the first, using different field names and definitions. This method of defining data more than one way keeps field definitions near the fields being redefined which improves readability and maintainability.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

     FTRANFILEIP  F     100            DISC

     ITRANFILENS  01  91 CS  92 CR
1    I                                        1  32 SREC
     I*SALES RECORD FIELDS
     I                                        1   9 ORDNO
     I                                       11  16 PARTNO
     I                                       17  20QTY1
     I                                       21  26 2UNITCT
     I                                       27  32 2TOTCST
     I         NS  02  91 CP  92 CC
2    I                                        1  35 PREC
     I*PURCHASE RECORD FIELDS
     I                                        1   6 PARTNO
     I                                        7  12 0QTYOH
     I                                       13  17 0QTYORD
     I                                       18  23 2COST
     I                                       24  29 ORDATE
     I                                       30  35 2AMTDUE
     I         NS  03  91 CT  92 CF
3    I                                        1  40 TREC
     I*TRANS RECORD FIELDS
     I                                        1   6 PARTNO
     I                                        7  10 0QTY2
     I                                       11  20 FRWHSE
     I                                       21  30 TOWHSE
     I                                       31  40 TIMDAT
```

Figure 5-6. Defining Different Record Formats Without Using Data Structures

Comments

1 This line defines the first record, SREC, in the file. SREC is

described in detail in the lines that follow.

2 This line defines the second record, PREC. PREC is described in detail in the lines that follow.

3 This line defines the third record, TREC. TREC is described in detail in the lines that follow.

Figure 5-7 is the same example as Figure 5-6 except that it uses data structures to describe the three input records.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-----------------------------------------------------------------------------------------

     FTRANFILEIP  F     100            DISC

     ITRANFILENS  01  91 CS  92 CR
1    I                                        1  32 SREC
     I            02  91 CP  92 CC
2    I                                        1  35 PREC
     I            03  91 CT  92 CF
3    I                                        1  40 TREC
     I*
4    ISREC        DS
     I*SALES RECORD FIELDS
     I                                        1   9 ORDNO
     I                                       11  16 PART1
     I                                       17  200QTY1
     I                                       21  262UNITCT
     I                                       27  322TOTCST
5    IPREC        DS
     I*PURCHASE RECORD FIELDS
     I                                        1   6 PART2
     I                                        7  120QTYOH
     I                                       13  170QTYORD
     I                                       18  232COST
     I                                       24  29 ORDATE
     I                                       30  352AMTDUE
6    ITREC        DS
     I*TRANS RECORD FIELDS
     I                                        1   6 PART3
     I                                        7  100QTY2
     I                                       11  20 FRWHSE
     I                                       21  30 TOWHSE
     I                                       31  40 TIMDAT
```

Figure 5-7. Defining Different Record Formats Using Data Structures

Comments

1 This line defines the sales record, SREC. SREC is described in detail starting with line 4. There are 32 characters in SREC.

2 This line defines the purchase record, PREC. PREC is described in detail starting with line 5. There are 35 characters in PREC.

3 This line defines the transaction record, TREC. TREC is described in detail starting with line 6. There are 40 characters in TREC.

4 This line names SREC as the first record to be redefined as a data structure.

5 This line names PREC as the second record to be redefined as a data structure.

6 This line names TREC as the third record to be redefined as a data structure.

Using Data Structures to Define Subfields Within a Field

This section tells you how to use data structures to divide a field or array into subfields. You can then refer to the subfields in Calculation and Output Specifications. In addition, you can use data structure names to refer to the subfields as one unit.

Figure 5-8 and Figure 5-9 show how to define subfields two different ways. Figure 5-8 shows an input area that uses two data structures. The first data structure, PRODID, defines the subfields in positions 31-40 of the input TRANS record. The second data structure, CATLOG, is a nested data structure. That is, it is a data structure defining subfields within another data structure (in this case, PRODID).

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
ITRANS   NS  01
I                                                .
I                                                .
I                                                .
1 I                                           31  40 PRODID
I                                             41  50 DESC
I                                             51  582PRICE

2 IPRODID      DS
3 I                                            1   2 CATLOG
I                                              3   5 VENDOR
I                                              6  10 PRDNO
4 ICATLOG      DS
5 I                                            1   1 CLASS
6 I                                            2   2 REFNO
```

Figure 5-8. Using Data Structures to Define Subfields within a Field

Comments

1 This line defines the field PRODID. It occupies positions 31 to 40 in the input record.

2 This line marks the beginning of the data structure PRODID. The next three lines define the subfields of the PRODID field.

3 This line defines the first subfield of PRODID. Notice that CATLOG is defined further by the next data structure (line 4).

4 This line is the beginning of the data structure CATLOG. The next two lines define the subfields of the CATLOG field.

5 The first subfield of CATLOG is CLASS. It is a one-character field starting with the first position of CATLOG.

6 The second subfield of CATLOG is REFNO. It is also a one-character field starting with the second position of CATLOG.

NOTE You can nest data structures to the number of levels necessary. This feature is unavailable in most implementations of RPG.

Figure 5-9 shows subfields defined in a different way from those shown in the previous figure. However, the end results of the two examples are identical.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
                                                                              
ITRANS   NS  01
I                                                    .
I                                                    .
I                                                    .
1 I                                                31  40 PRODID
I                                                  41  50 DESC
I                                                  51  582PRICE
2 IPRODID      DS
3 I                                                 1   2 CATLOG
4 I                                                 1   1 CLASS
5 I                                                 2   2 REFNO
I                                                   3   5 VENDOR
I                                                   6  10 PRDNO
```

Figure 5-9. Alternate Way to Define Subfields within a Field

Comments

1 This line defines the input field, PRODID, whose subfields are

defined in the data structure starting on line 2. PRODID occupies positions 31-40 of the input record.

2 This line starts the data structure, PRODID. It describes the subfields of PRODID in the input record.

3 This line defines the field, CATLOG, that occupies positions 1 and 2 of the field PRODID. Notice that the next two fields are subfields of CATLOG.

4 This lines defines the field, CLASS, that occupies position 1 of the field PRODID.

5 This line defines the field, REFNO, that occupies position 2 of the field PRODID.

Using Data Structures to Reorganize Fields in an Input Record

This section explains how data structures can be used to rearrange input record fields so that you can use them more readily in a program.

Two examples are presented. The first shows how to reorder one field and the second shows how to reorganize several fields at once.

Figure 5-10 shows how to reorder the field, KEY05, so that it is the second part of a key field. The key field will be used for reading a KSAM file. This example initializes the data structure field, ALPHA, to A at line 5. When the TRANS file is READ, KEY05 is automatically moved to the input field and to the data structure. Line 6 shows the KSAM disc read operation that uses the new key, KEY06.

```
     1         2         3         4         5         6         7
    6789012345678901234567890123456789012345678901234567890123456789012345678901234
    ITRANS   NS  01
    I                                                .
    I                                                .
    I                                                .
1   I                                            71  75 KEY05
2   IKEY06       DS
3   I                                             1   1 ALPHA
4   I                                             2   6 KEY05

5   C N99                MOVE "A"       ALPHA
    C N99                SETON                        99
    C                      .
    C                      .
    C                      .
6   C           KEY06    CHAINFILE1                   60
```

Figure 5-10. Using a Data Structure to Build a Key Field

Comments

1 This line defines the input field, KEY05. It occupies positions 71 to 75 in the input record, TRANS.

2 This line begins the data structure, KEY06.

3 This line defines the first subfield, ALPHA, in the data structure. ALPHA occupies the first position of KEY06.

4 This line defines the position in the data structure that KEY05 occupies. It is the second field and occupies positions 2 through 6.

5 This line is part of the initialization operations. It sets the field, ALPHA, to A (it remains A for the duration of the program).

6 This line reads the file, FILE1, by key value. The specific key value used has already been placed into the data structure, KEY06, by previous operations.

Figure 5-11 shows how to reorganize an input record so that its fields closely match those of an output record. When TRANS is read, RPG automatically moves its input fields to the appropriate areas in the KEYDS data structure. Notice that the input fields, QTY and CODE, are not included in the data structure. The PRTKEY field is optional. It can be used to refer to all 16 positions of the KEYDS data structure (KEYDS also refers to these positions).

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
                                                                        
ITRANS   NS  01                                                       
I                                                        .              
I                                                        .              
I                                                        .              
I                                                    3  10 PARTNO       
I                                                   11  16 QTY          
I                                                   17  20 TYPE         
I                                                   21  21 CODE         
I                                                   22  25 LOCATN       
1 IKEYDS       DS                                                       
2 I                                                  1   4 LOCATN       
3 I                                                  5  12 PARTNO       
4 I                                                 13  16 TYPE         
5 I                                                  1  16 PRTKEY       
```

Figure 5-11. Using a Data Structure to Reorganize Input Fields

Comments

1 This line begins the description of the data structure, KEYDS.

2 This line defines LOCATN as the first field in KEYDS. Since LOCATN is also defined in the input record, its value is moved

here when the file is read.

3 This line specifies that the second field in the data structure is PARTNO and its value comes from the field by the same name in the input record, TRANS.

4 This line defines the third field, TYPE, in the data structure. Its value comes from the field by the same name in the input record, TRANS.

5 This line defines the field, PRTKEY. PRTKEY refers to all of the fields in the data structure (positions 1-16).

Using Data Structures for Interprogram Communication

This section discusses how to receive information from other programs and how to pass information to them using a standard RPG file, the Local Data Area file (LDAFILE). To use a Local Data Area file, you must define it using a data structure.

When you use a Local Data Area in your program, RPG begins by reading the file into your data structure. You can use that data during the first cycle (1P) output. When your program ends, RPG automatically writes the data structure back to the Local Data Area file.

Before you can use a Local Data Area file in a program, you must create it using the RPGINIT utility (see the *RPG Utilities Reference Manual* for more detailed information).

Figure 5-12 lists a logon User Defined Command (UDC) that creates a Local Data Area file automatically. The UDC creates a file consisting of two records, each containing 256 characters. The records are initialized to blanks. The FCOPY command copies the file, SYSCTL, to the newly-created Local Data Area file. SYSCTL contains one record with the company name in positions 1 to 40. The company name is used in the next example as a report heading.

```
START
OPTION  LOGON
COMMENT  INITIALIZE  LOCAL  DATA  AREA  AND  WRITE
COMMENT  COMPANY  NAME  TO  POSITION  ONE
COMMENT  OF  LDA  FOR  USE  BY  REPORT  PROGRAMS
RUN RPGINIT.PUB.SYS
FCOPY FROM=SYSCTL;TO=LDAFILE
```

Figure 5-12. Creating a Local Data Area File (LDAFILE) with a UDC

Once you create a Local Data Area file, you can use it in any RPG program. Figure 5-13 shows how to process the Local Data Area file created in Figure 5-12. The company name (CNAME), contained in the Local Data Area file, is used in a report heading. Also, a transaction record count is accumulated in the field, COUNT. This field is also saved in the Local Data Area file.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
H*   READ THE COMPANY NAME FROM THE LDA,
H*   INITIALIZED BY LOGON UDC.

1 FSTDLIST O   F  40  40            $STDLST

2 ILDA         UDS
3 I                                      1  40 CNAME
4 I                                    101 105 0COUNT

5 C                      ADD  1         COUNT

6 OSTDLIST H               1P
  O                          CNAME     40
```

Figure 5-13. Using a Local Data Area File (LDAFILE)

Comments

1 This line defines the file used for printing the report.

2 This line defines the Local Data Area file and begins the data structure description for it. Notice that you do not use a File Description Specification for a Local Data Area file.

Columns 7-14 contain LDA to identify this structure as a Local Data Area structure. You must use LDA; other names are ignored.

Column 18 contains U to specify that this is a User Data Structure (the data structure for the Local Data Area file).

Columns 19-20 contain DS to identify this as a data structure.

3 This line defines the first field, CNAME, of the data structure. It is 40 characters long and starts in the first position of the record. CNAME contains the company name.

4 This line defines the second field, COUNT, of the data structure. It is 5 positions long and starts in position 101. It contains the count of transactions.

5 This line increments the COUNT field.

6 This line begins the Output Specification that writes the company name, CNAME, to the report file.

Using Subroutines

When there are a set of operations that you perform repeatedly from different parts of your program, you should code them once as a subroutine. Then, when you want to execute the subroutine, you enter a

Calculation Specification to perform it.

RPG has two kinds of subroutines. Internal subroutines are part of an RPG program. External subroutines are independent load modules that are written in Business BASIC, C, Pascal, SPL or COBOL.

Internal Subroutines

To perform an internal subroutine, enter an EXSR Calculation Specification operation. Place the subroutine, itself, after the last Calculation Specification in the program. The BEGSR operation marks the beginning of an internal subroutine and always has a tag associated with it. You use this tag to execute the subroutine. To end an internal subroutine, follow the last line in the subroutine with an ENDSR operation.

Figure 5-14 shows how internal subroutines are coded. In this example, the subroutine is named, INQRY.

```
     1         2         3         4         5         6         7
    6789012345678901234567890123456789012345678901234567890123456789012345678901234
1   C   F3                EXSR INQRY
    C                       .
    C                       .
    C                       .
    C*************************************************************************
    C*    INTERNAL SUBROUTINE - INQRY
    C*************************************************************************
2   CSR          INQRY     BEGSR
    CSR                     .
    CSR                     .
    CSR                     .
3   CSR                   GOTO INQRY9
    CSR                     .
    CSR                     .
    CSR                     .
4   CSR          INQRY9    ENDSR
```

Figure 5-14. Using an Internal Subroutine

Comments

1 This line performs the internal subroutine INQRY when a user presses function key F3. (Conditioning the EXSR operation is optional.)

Columns 28-32 contain EXSR to execute an internal subroutine.

Columns 33-38 contain the name of the internal subroutine, INQRY.

2 This line begins the INQRY internal subroutine.

Columns 7-8 contain SR to indicate that this is a subroutine line (SR is optional on this and subsequent subroutine lines).

Columns 18-23 contain the name of the internal subroutine INQRY.

Columns 28-32 contain BEGSR to mark the beginning of the internal subroutine.

3 This line transfers control to the end of the internal subroutine. You cannot use GOTO operations to transfer control out of the subroutine.

4 This line ends the internal subroutine.

Columns 18-23 contain the tag, INQRY9. (A tag is used in this particular example, though it is optional.)

Columns 28-32 contain ENDSR to end the internal subroutine.

External Subroutines

External subroutines are separate procedures; you do not code them as part of an RPG program. For example, you can code an external subroutine in Business BASIC. You then compile it, and place it in an executable library using HP Link Editor/XL. To execute the external subroutine from an RPG program, enter an EXIT Calculation Specification operation. The *HP RPG Reference Manual* contains information on how to create external subroutines in COBOL and other languages.

There are two ways to pass information to external subroutines. The first method uses the Calculation Specification operation, RLABL, to name the fields, arrays, tables or indicators that you want to pass. The second method uses the Calculation Specification operation, PARM, to name the fields, arrays and tables to pass. PARM is more limited than RLABL because you cannot pass indicators. Also, PARM values are available only to the subroutine executed with the PARM operation(s). The next two sections explain how to use external subroutines and how to pass parameters to them using RLABL and PARM.

Using RLABL. RLABL passes field names, tables, arrays and indicators to external subroutines. Values passed by RLABL are available to all external subroutines in the program.

Figure 5-15 shows a portion of a C procedure that is used as an external subroutine. Just the statements defining the data that is passed to the procedure (the field, PNAME, and the indicator, IN20) are shown. Figure 5-16 shows how to execute this C procedure from an RPG program.

```
VOID  SUB01()
{
    EXTERN  CHAR  PNAME[];
    EXTERN  INT IN20;
          .
          .
          .
}
```

Figure 5-15. An External Subroutine written in C

Figure 5-16 lists the Specifications that execute the external subroutine, SUB01, shown in the previous figure. The RPG program passes indicator 20 (IN20) and the field (PNAME) to the subroutine.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
1 C                     RLABL          IN20
2 C                     RLABL          PNAME
  C                       .
  C                       .
  C                       .
3 C                     EXIT SUB01
```

Figure 5-16. Using RLABL to Pass Information to an External Subroutine

Comments

1 This line makes indicator 20 available to an external subroutine.

Columns 43-48 specifies that indicator 20 is passed to the external subroutine. (Prefix indicator names by IN.)

2 This line makes the field, PNAME, available to an external subroutine.

3 This line executes the external subroutine, SUB01.

Using PARM. PARM passes field names, tables and arrays to external subroutines. PARM does not pass indicators. Also, you must code PARM operations for each external subroutine called. PARM values are not globally available to other external subroutines in a program. (See Figure 8-4 for a complete program that uses external subroutines and PARM.)

RPG variables used with PARM are passed to external subroutines as byte values by reference. The corresponding external subroutine variables must be declared accordingly. (All RPG numeric fields have packed decimal formats.)

Figure 5-17 shows part of a C procedure that is executed as an external subroutine. Only the statements defining the data that is passed to the procedure are shown. Figure 5-18 shows how this C procedure is executed from an RPG program.

```
VOID  SUB01  (PNAME,  STATUS)
     CHAR  *PNAME;
     CHAR  *STATUS;
{
          .
          .
          .
}
```

Figure 5-17. An External Subroutine Written in C

Figure 5-18 shows how to execute the C procedure shown in Figure 5-16. The RPG program passes two fields to the C procedure, PNAME and STATUS. The C procedure may alter the value in STATUS, indicating that an error occurred in the external subroutine.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
C                      .
C                      .
C                      .
C                    MOVEL"RPT10"   PNAME  26
C                    MOVE "000"     STATUS  3
C                    EXIT SUB01
C                    PARM           PNAME
C                    PARM           STATUS
C             STATUS COMP "000"                      10
C   10               GOTO RUNEND
C                      .
C                      .
C                      .
C             RUNEND   .
```

Figure 5-18. Using PARM to Pass Information to an External Subroutine

Comments

1 This line moves a program name, RPT10, to the first parameter field, PNAME. This field is passed to the external subroutine.

2 This line sets the field, STATUS, to zeros. This field is passed to the external subroutine. The subroutine returns an error code in this field.

3 This line executes the external subroutine, SUB01. SUB01 is the C procedure shown in Figure 5-17.

4 This line passes the field, PNAME, to the external subroutine. (You enter PARM operations immediately after the EXIT operation.)

5 This line passes the field, STATUS, to the external subroutine.

6 This line tests the value in STATUS. The external subroutine can modify this field.

Chapter 6 Compiling an RPG Program

This chapter gives you the information you need to know in order to successfully compile an RPG program. It explains how to use source libraries and compiler options. It gives you specific instructions on how to compile under the MPE XL operating system. The compiler listings are discussed in detail as well as how to recover from compilation errors.

Using Source Libraries

Source libraries help to standardize parts of RPG programs and reduce the manual effort in coding programs. You enter commonly-used source code into a source library, then copy that code into programs rather than recoding it in each of them.

An RPG source library is a standard text file containing RPG specifications. You can create a source library using any text processor, such as EDITOR. Enter the RPG specifications exactly as they should appear in a source program. When you want to use the source libraries in a program, enter a $COPY statement at the beginning of the program. $COPY enables the source library facility of RPG. At the point in the program where you want to insert the source library text, enter an $INCLUDE statement.

Figure 6-1 lists a program that extracts records from the D-ACCOUNTS data set (see the MARKET schema in Figure 3-23). It prints the extracted records and writes them to the file EXTRACT. There are two $INCLUDE statements in the program. The first (line 3) copies the input field definitions for D-ACCOUNTS. The second (line 5) copies the output field definitions for the detail report record. Figure 6-2 and Figure 6-3 list the contents of the source library files and Figure 6-4 shows what the compiled source program listing looks like.

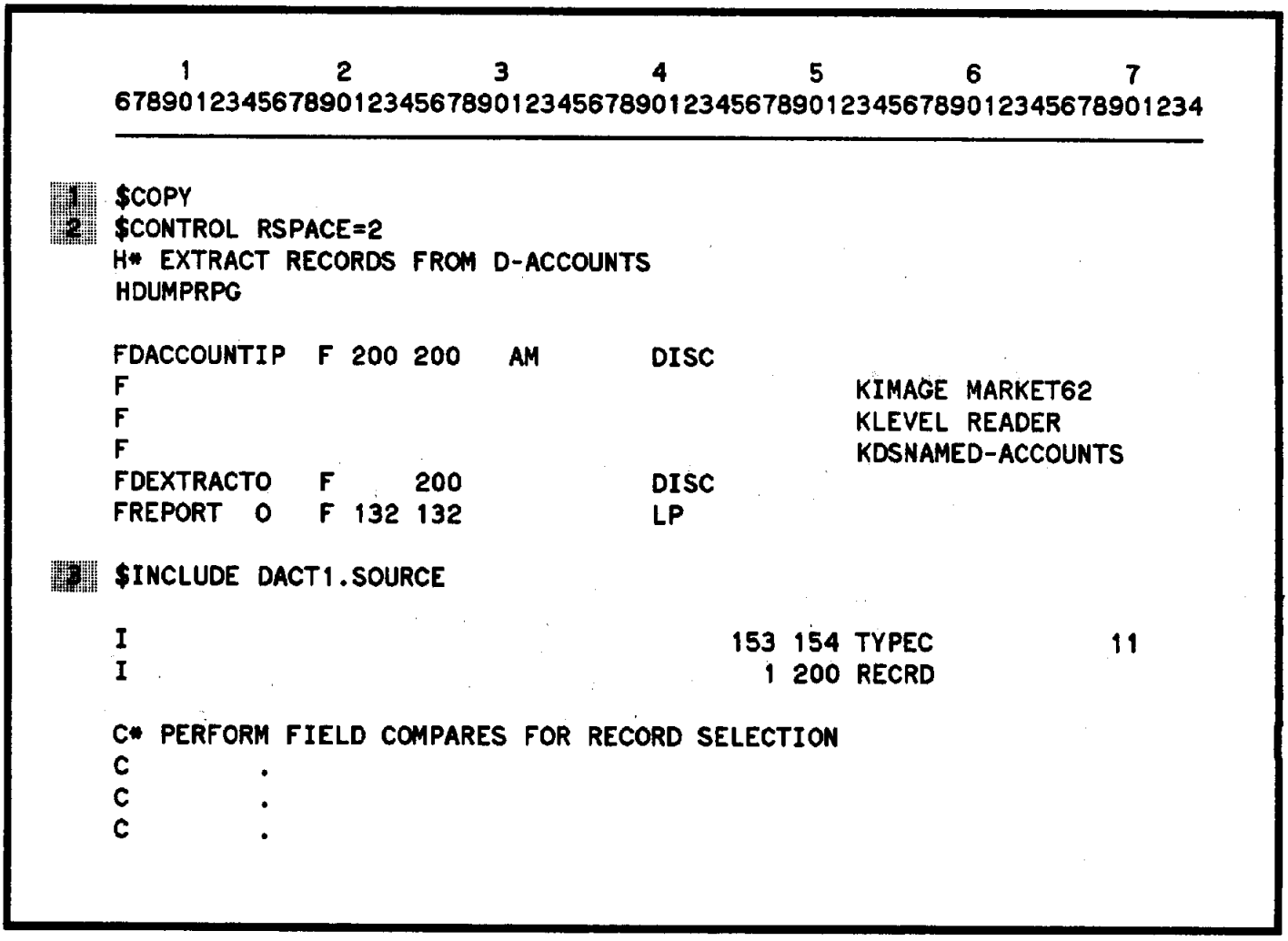


Figure 6-1. A Source Program Before Compilation

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

     ODEXTRACTD         01 20
     O                             RECRD    200
4    OREPORT  H         1P
     O                                        "ACCT #"
     O                                        "FIRST NAME"
     O                                        "LAST NAME       "
     O                                   + 19 "ADDRESS 1   "
     O                                   + 19 "ADDRESS 2   "
     O                                        "ZIP CODE   "
     O         D        01
5    $INCLUDE DACT2.SOURCE
```

Figure 6-1. A Source Program Before Compilation (Continued)

Comments

1 This $COPY subsystem command specifies that the preprocessor utility (RPGCOPY.PUB.SYS) is used during compilation. RPGCOPY processes $INCLUDE statements. $COPY must be the first statement in the program.

2 This $CONTROL subsystem command contains the RSPACE option. RSPACE directs the compiler to add 2 spaces between each output field having relative end positions.

3 This $INCLUDE subsystem command directs the compiler to insert source library text at this point in the program. The text comes from the file, DACT1, in group SOURCE.

4 This line begins the specifications for the report headings.

5 This $INCLUDE command directs the compiler to insert source library text at this point in the program. The text comes from the file, DACT2, in group SOURCE.

Figure 6-2 lists the entries in the source library file, DACT1.

```
|                                                    |
|     IDACCOUNTNS   01             3    8 ACTNO|
|     I                            9   18 FNAME|
|     I                           19   34 LNAME|
|     I                           35   62 ADDR1|
|     I                           63   90 ADDR2|
|     I                           91  118 ADDR3|
|     I                          119  128 ZIPCD|
|     I                          129  134OPHONE|
|     I                          135  136OPHEXT|
|     I                          137  144 MAILC|
|     I                          145  148OBDATE|
|     I                          149  152 SRCCD|
|     I                          153  154 TYPEC|
|     I                          155  162 ADATE|
|                                                    |
```

Figure 6-2. Contents of the Source Library DACT1.SOURCE

Figure 6-3 lists the entries in the source library file, DACT2.

```
|                                  |
|     O                ACTNO       |
|     O                FNAME       |
|     O                LNAME       |
|     O                ADDR1       |
|     O                ADDR2       |
|     O                ZIPCD       |
|                                  |
```

Figure 6-3. Contents of the Source Library DACT2.SOURCE

Figure 6-4 shows, in effect, what the program looks like after the source text is inserted. Source library text is inserted during the compilation process; it is not incorporated as a permanent part of the source program. In the listing, a "C" precedes the text that has been copied and $COPY and $INCLUDE are changed to *COPY and *INCLUDE.

```
    *COPY
    $CONTROL  RSPACE=2
    H*  EXTRACT  RECORDS  FROM  D-ACCOUNTS
    HDUMPRPG
    FDACCOUNTIP  F  200  200    AM    DISC
    F                                      KIMAGE MARKET62
    F                                      KLEVEL READER
    F                                      KDSNAMED-ACCOUNTS
    FDEXTRACTO   F       200          DISC
    FREPORT  O   F  132  132          LP
    *INCLUDE DACT1
 1  CIDACCOUNTNS 01
    CI                                      3    8 ACTNO
    CI                                      9   18 FNAME
    CI                                     19   34 LNAME
    CI                                     35   62 ADDR1
    CI                                     63   90 ADDR2
    CI                                     91  118 ADDR3
    CI                                    119  128 ZIPCD
    CI                                    129  134OPHONE
    CI                                    135  136OPHEXT
    CI                                    137  144 MAILC
    CI                                    145  148OBDATE
    CI                                    149  152 SRCCD
 2  MI                                    153  154 TYPEC
    CI                                    155  162 ADATE
 3  I                                      1  200 RECORD
     C*  PERFORM  FIELD  COMPARES  FOR  RECORD  SELECTION
     C*        .
     C*        .
     C*        .
```

Figure 6-4. A Source Program After Compilation

```
     ODEXTRACTD     01  20
     O                          RECRD    200
     OREPORT   H    1P
     O                                   6 "ACCT #"         R
     O                                  18 "FIRST NAME"     R
     O                                  36 "LAST NAME  "    R
     O                                  66 "ADDRESS 1  "    +019
     O                                  96 "ADDRESS 2  "    +019
```

```
          O         D    01                  108 "ZIP CODE  "     R
          O
          *INCLUDE  DACT2
      4   CO                        ACTNO       6                 R
          CO                        FNAME      18                 R
          CO                        LNAME      36                 R
          CO                        ADDR1      66                 R
          CO                        ADDR2      96                 R
          CO                        ZIPCO     108                 R
```

Figure 6-4. A Source Program After Compilation (Continued)

Comments

1 This is the first line copied from source library, DACT1. Each of the lines in DACT1, except for lines 2 and 3, are copied.

2 This line remains the same as that in the original program. Fields with the same names remain unaltered.

3 This field does not exist in the source library; therefore, it remains unaltered.

4 This is the first line copied from source library, DACT2.

Compiling Only

This section explains how to compile a program without linking or executing it. You may want to do this, for instance, when you're in the initial stages of developing a program and you want to know where the compiler errors are. You *must* compile-only when the program contains external subroutines because they must be linked to the program in a separate step using the LINK command (see the LINK command in the *HP Link Editor/XL Reference Manual*).

There are two ways to compile a program. You can use the MPE XL command RPGXL or, if you're using RISE to develop a program, you can use the RISE VERIFY command to compile it. (For information on RISE, see the *RPG Utilities Reference Manual*.)

Figure 6-5 shows how to compile a program from a job file. The job file is named GL50.JOB and contains an RPGXL command that compiles the program, GL50S.SOURCE. RPGXL directs the compiler to create the *relocatable object file* GL50O.OBJECT for the program. (The relocatable object file is not executable; it must be converted to an executable program file using the HP Link Editor/XL LINK command. See the *HP Link Editor/XL Reference Manual* for details about creating an executable program file.)

To start the compile job shown in Figure 6-5, enter the command,

```
:STREAM GL50.JOB
```

```
|                                                                |
|     !JOB  GL50,MGR.ACCTG                                       |
|     !RPGXL  GL50S.SOURCE,GL500.OBJECT                          |
|     !TELL  MGR.ACCTG;PROGRAM  GL50  COMPILED  SUCCESSFULLY|
|     !EOJ                                                       |
|                                                                |
```

Figure 6-5. Compiling an RPG Program From a Job File

To compile GL50S.SOURCE in session mode, enter this command,

```
:RPGXL GL50S.SOURCE,GL50O.OBJECT
```

In both the job and session examples above, the program listing is written to $STDLIST.

Compiling and Linking

You can compile an RPG program and prepare it for execution in one step. You may want to do this when you need to compile the program but execute it at a later time.

Figure 6-5 shows how to compile and link a program using the RPGXLLK command. RPGXLLK is contained in the job file GL50.JOB. The compiler reads the source file GL50S.SOURCE and produces the *executable program file* GL50P.PROGRAM. (Using RPGXLLK is equivalent to entering RPGXL followed by LINK.)

To start the compile and link job shown in Figure 6-6, enter the command,

```
:STREAM GL50.JOB
```

```
!JOB  GL50,MGR.ACCTG
!RPGXLLK  GL50S.SOURCE,GL50P.PROGRAM,$NULL
!TELL MGR.ACCTG;PROGRAM  GL50  COMPILED  SUCCESSFULLY
!EOJ
```

Figure 6-6. Compiling and Linking an RPG Program From a Job File

To compile and link GL50S.SOURCE in session mode, enter this command,

```
:RPGXLLK GL50S.SOURCE,GL50P.PROGRAM,$NULL
```

In both the job and session examples above, $NULL suppresses the program listing.

Compiling, Linking and Executing

You can compile, link and execute an RPG program in one operation. This comes in handy when you're testing a program.

To compile, link and execute an RPG program, enter the MPE XL RPGXLGO command. Figure 6-7 shows how to use RPGXLGO in a job file (GL50.JOB) that compiles and executes the RPG program, GL50S.SOURCE. The RPGXLGO command creates a temporary executable program file $OLDPASS. $OLDPASS is overwritten by the next RPGXLGO command and is purged when you log off.

To execute the job file in the following figure, enter the command,

```
:STREAM GL50.JOB
```

```
!JOB  GL50,MGR.ACCTG
!RPGXLGO  GL50S.SOURCE
!TELL  MGR.ACCTG;PROGRAM  GL50  COMPILED  SUCCESSFULLY
!EOJ
```

Figure 6-7. Compiling, Linking and Executing an RPG Program From a Job File

To compile, link and execute GL50S.SOURCE in session mode, enter the command,

```
:RPGXLGO GL50S.SOURCE
```

In both the job and session mode examples above, the program listing is written to $STDLIST.

Changing the RPG Compiler Defaults

When you compile a program, RPG makes certain assumptions about the compile and run-time options that you're using. For instance, when a run-time error occurs, RPG displays a message and the operator chooses an appropriate response or action.

You can change the defaults for some of the compiler options, by using one or all of the following: the Header Specification, the $CONTROL statement or the $TITLE statement. The next three sections discuss the options that these statements control and how to use them. For a discussion of all of the compiler subsystem commands, see the *HP RPG Reference Manual*.

The Header Specification

The Header Specification is used to specify these common compile-time and run-time options:

* Where to write the run-time Error Dump
* Whether the RPG DEBUG facility is used
* What search method to use for tables and arrays
* How DSPLY and DSPLM work
* Whether or not to print a Cross-Reference listing
* How to handle run-time error messages

For a complete list of the Header Specification options, see the *HP RPG Reference Manual*.

Figure 6-8 shows how to change the defaults for the options listed above.

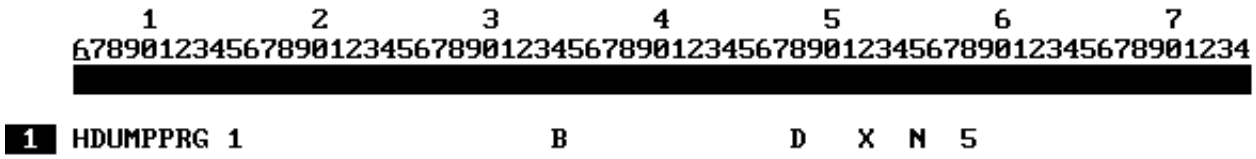


Figure 6-8. Using the Header Specification to Change Compiler Defaults

Comments

1 Columns 7-14 contain the name of the disc file, DUMPRPG, that contains the run-time Error Dump.

Column 15 contains 1 to enable the RPG DEBUG facility. (For information on DEBUG, see "Using RPG DEBUG" in Chapter 7.)

Column 34 is B to specify that tables and arrays are searched in a binary fashion.

Column 48 is D to display literals for DSPLY and DSPLM and to prompt for user input on the same line as the literals.

Column 52 contains X to print a Cross-Reference listing.

Column 55 contains N to suppress run-time error messages for the errors having responses entered in columns 56-71.

Column 58 contains 5 to terminate the program when a run-time input file sequence error occurs. An Error Dump is also printed.

$CONTROL

The $CONTROL statement is a compiler subsystem command that lets you specify many compile-time options including:

* The name of the source program file (this overrides the name in columns 75-80 of the Header Specification)
* The delimiter character for non-numeric literals
* The number of lines per page to print on the compiler listings
* Which lines to include in the source program and Cross-Reference listings

Normally, if you use $CONTROL, it is the first statement in the program. However, you can use $CONTROL statements throughout a program. For instance, you may want to turn compiler options on and off at different points in the program.

Figure 6-9 shows how to enter a $CONTROL statement to change the compiler options listed above.

```
     1         2         3         4         5         6         7
    67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
    ----------------------------------------------------------------------------------------

1   $CONTROL NAME=PAYRL5,QUOTE='',LINES=55,NOLIST
    H*************************************************************
    H*    PROGRAM TO CALCULATE STATE TAX
    H*************************************************************
    HDUMPRG                                                 X
                .
                .
                .
2   $CONTROL LIST,MAP
    C*************************************************************
    C*    SUBROUTINE TO LOOKUP TAX RATE                  TXRTE
    C*************************************************************
    C           .
    C           .
    C           .
    C*************************************************************
    C*    END SUBROUTINE                                 TXRTE
    C*************************************************************
3   $CONTROL NOLIST,NOMAP
                .
```

Figure 6-9. Using $CONTROL to Change Compiler Defaults

Comments

1 This $CONTROL statement specifies that: the program name is PAYRL5; the delimiter character for non-numeric literals is a single quote ('); the maximum number of lines per page in the compiler listings is 55; and the source program and Cross-Reference listings will not be printed.

2 This $CONTROL statement turns on the source program and Cross-Reference listing options.

3 This $CONTROL statement turns off the source program and Cross-Reference listing options.

$TITLE

$TITLE is used to change the title that appears on the top of each

compiler listing page. You can use it to print the program name on each page, for example, or you can alter it from page to page to print subroutine names.

To alter the title for the entire compiler listing, place a $TITLE statement at the beginning of the program after the $CONTROL statement, if there is one. When you use $TITLE, the top line of each page (except the first) looks like this:

PAGE nnnn ($TITLE text)

Figure 6-10 shows how to use $TITLE to print a description of a program, along with its name, at the top of each page of a compiler listing.

```
    1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
                                                                        

$CONTROL QUOTE=',NAME=PAYRL5,LINES=55,NOLIST
1 $TITLE "Calculate State Tax - PAYRL5"
H*************************************************************
H*    PROGRAM TO CALCULATE STATE TAX
H*************************************************************
HDUMPRG                                                  X

             .
             .
             .
```

Figure 6-10. Using $TITLE to Change the Compiler Listing Title

Comments

1 This line specifies that the title, Calculate State Tax - PAYRL5, appear at the top of each compiler listing page.

Using the RPG Compiler Listings

When you compile a program, one or more of the following listings is produced:

* A listing of the source program. (Optionally, you can print just those statements containing errors.)
* A Symbol Table listing.
* A Cross-Reference listing.

Each of these listings is discussed in detail in the following sections.

The Source Program Listing

When you compile a program, a full source listing is printed automatically. It is printed using the device assigned to $STDLIST (or RPGLIST). Source lines that contain errors are stamped with error numbers. If the source lines contain potential errors, they are stamped with a warning numbers. These warning and error numbers are printed after the Symbol Table (or Cross-Reference listing, if there is one) along with short explanations (see "Understanding RPG Compiler Messages"). At the end of the source listing, indicators that are defined but not referenced in the program, are listed.

You can turn off the source program listing anywhere in the program by entering a $CONTROL statement with the NOLIST option. (see "$CONTROL").

Doing this causes only those source statements containing errors to be printed. Suppressing the source program listing in this manner is useful when you're compiling new or large programs.

Figure 6-11 shows what a source program listing looks like. It shows the first part of the program listed in Figure 4-23. Sequence numbers have been added to the source program to illustrate how they appear on the listing.

```
PAGE 0001  RPG/XL  HP30318A.00.00  COPYRIGHT HEWLETT-PACKARD CO. 1987
  THU, MAR 10, 1988,  2:15 PM

0001    0010 HDUMPFILE     N       U
0002    0015 H*************************************************************
0003    0020 H**                                                          ***
0004    0025 H**  REMARKS..... ON-LINE UPDATE TO D-ACCOUNTS IN            ***
0005    0030 H**   THE MARKET DATABASE.  ALLOW ADDS, CHANGES,             ***
0006    0035 H**   DELETES AND INQUIRIES.                                 ***
0007    0040 H**                                                          ***
0008    0045 H*************************************************************
0009

0010    0050 F*
0011    0055 F*    FACCOUNT  - V/3000 INTERFACE FILE
0012    0060 F*
0013    0065 FTERMINALUD  V     256            WORKSTN   L5B
0014    0070 F                                             KFORMS FACCOUNT
0015    0075 F*
0016    0080 F*     ACCOUNT INFORMATION - CHAINED READ MODE
0017    0085 F*
0018    0090 FDACCOUNTUC  F     200R06AM     3 DISC                     A
0019    0095 F                                             KIMAGE MARKET45
0020    0100 F                                             KLEVEL WRITER
0021    0105 F                                             KITEM  ACCOUNT-NO
0022    0110 F                                             KDSNAMED-ACCOUNTS
0023

0024    0115 E*-------------------------------------------------------------
0025    0120 E                    MSG     1  20 79              MESSAGES
0026    0125 E                    LBL     1   8 16              FUNC KEY LBLS
0027    0130 E*-------------------------------------------------------------
0028

0029    0135 I*     VPLUS TERMINAL FILE
0029    0140 I*
0030    0145 I*        NUMBER OF EDIT ERRORS - EVENT 09
0031    0150 ITERMINALNS  09   1 C0   2 C9
0032    0155 I                                        18  22ONUMBER      13
0033    0160 I*        ALL OTHER TERMINAL READS
0034    0165 I        NS  01
0035    0170 I                                        18  21ODATALN
0036    0175 I                                        22  27 FACTNO
```

Figure 6-11. A Source Program Listing

Comments

PAGE 0001... This is the standard RPG heading. It contains the page number, RPG compiler version number and the current date.

0001	This number (like others in this column) is a source sequence number generated by the RPG compiler. References to source code lines in the compiler listings use this number.
0010	This is the sequence number contained in the actual source line (columns 1-5).

Figure 6-12 shows the unreferenced indicators for the VPLUS program shown in Figure 4-23. Unreferenced indicators, if they exist, are printed at the end of the source listing.

```
------------------------------------------------------------

UNREFERENCED INDICATOR = 01
UNREFERENCED INDICATOR = 09
UNREFERENCED INDICATOR = 12
UNREFERENCED INDICATOR = 61
------------------------------------------------------------
```

Figure 6-12. A Source Program Listing (Unreferenced Indicators)

The Symbol Table Listing

The Symbol Table listing gives the storage allocations for data fields, tables, arrays, subroutines and calculations having tag references. The names are listed in alphabetical order. You may find a Symbol Table listing useful when debugging a program.

The Symbol Table listing is produced automatically when you compile a program (it is suppressed when you use the NOLIST option of the $CONTROL compiler subsystem command). Figure 6-13 shows what the Symbol Table listing looks like for the VPLUS program shown in Figure 4-23.

SYMBOL	SZ/TP	ADDR	SYMBOL	SZ/TP	ADDR	SYMBOL	SZ/TP	ADDR
ACTION	6	000002a4	DELET9	TAG		HLNAME	16	00000a96
ACTNO	6	00000000	DELETE	SUBR		INQRY	SUBR	
ADATE	8	00000a4b	DSPMSG	SUBR		INQRY1	TAG	
ADD	SUBR		ENDMN	TAG		INQRY3	TAG	
ADD1	TAG		ENHANC	1	00000a40	INQRY9	TAG	
ADD3	TAG		F	1.0	00000acc	LBL	16	0000033eA
ADD9	TAG		FACT50	50	00000a53	LNAME	16	0000028e
ADDR1	28	000003c8	FACTNO	6	00000288	M	1.0	00000ad6
ADDR2	28	000003e4	FADATE	8	00000404	MAILC	8	00000aa6
ADDR3	28	00000000	FADDR1	28	000002b8	MSG	79	0000040cA
BADFLD	5.0	00000ac6	FADDR2	29	00000306	MSGLEN	2.0	00000abe
BDATE	10.0	00000adc	FADDR3	28	00000322	NUMBER	5.0	00000000
BLNK50	50	00000256	FBDATE	8.0	00000acf	ONETI5	TAG	
CHANG1	TAG		FFNAME	10	00000a41	ONETIM	SUBR	
CHANG3	TAG		FLNAME	16	00000a85	PHEXT	5.0	00000ad7
CHANG9	TAG		FMAILC	8	00000a38	PHONE	11.0	00000ac0
CHANGE	SUBR		FNAME	10	0000024c	RDFIL	SUBR	
CHGMO3	TAG		FPHEXT	4.0	00000ac9	RDFIL9	TAG	
CHGMO5	TAG		FPHONE	10.0	00000ab4	RDSCR	SUBR	
CHGMO9	TAG		FSRCCD	4	00000aae	RDSCR1	TAG	
CHGMOD	SUBR		FTYPEC	2	00000400	RDSCR7	TAG	
DATALN	4.0	00000ae2	FZIPCD	10	000002ae	SRCCD	4	000002aa
DELET1	TAG		GETSCR	SUBR		TYPEC	2	00000402
DELET5	TAG		HACTNO	6	0000029e	ZIPCD	10	000003be

Figure 6-13. A Symbol Table Listing

	Comments
ACTION	This SYMBOL name is the name of a field. Other names appearing in the SYMBOL column are names of table items, array elements, subroutines and TAG operation labels.
6	This is the length of the SYMBOL field, ACTION. If the SYMBOL name is a numeric field, the number of digits and decimals in the field are printed. For example, 10.0 indicates that there are 10 digits and no decimals. If the SYMBOL name is a label for a TAG operation, TAG appears in this column. If the SYMBOL name is a subroutine name, SUBR (subroutine) is printed.
000002a4	This is the starting memory location (in hexadecimal) for the field, ACTION. It is relative to the beginning of the run-time memory area for the program. Storage locations are printed for fields, table and array elements. Storage locations are not printed for SYMBOL names that are labels for TAG operations or subroutines. The last character in the ADDR column indicates whether the SYMBOL name is a table or array. If the SYMBOL name is a table, T is printed. If the SYMBOL name is an array, A is printed. If the SYMBOL name is not a table or array, this position is blank (see LBL, for example).

The Cross-Reference Listing

The Cross-Reference listing shows each field, tag, indicator, subroutine, table and array name along with each reference to it in the program. You should request this listing when you're debugging a program.

You request the Cross-Reference listing using the Header Specification (see "The Header Specification" in this chapter). You can select which portions to cross-reference, if you like, by using the MAP and NOMAP options of the $CONTROL statement (see "$CONTROL" in this chapter).

Figure 6-14 through Figure 6-16 show the three parts of a Cross-Reference listing (the program used is the one shown in Figure 4-23). Figure 6-14 shows indicator references. Figure 6-15 shows field references and Figure 6-16 shows file references.

```
PAGE 0011       INDICATORS USED

 *01 INDICATOR DEFINED     0034
          NOT REFERENCED
 *09 INDICATOR DEFINED     0031
          NOT REFERENCED
 *12 INDICATOR DEFINED     0051
          NOT REFERENCED
  13 INDICATOR DEFINED     0032
          REFERENCED        0025
  50 INDICATOR DEFINED     0359
          REFERENCED        0073
  51 INDICATOR DEFINED     0270 0299
          REFERENCED        0099 0128 0145 0181 0199 0219 0271 0358
  52 INDICATOR DEFINED     0121 0151 0153
          REFERENCED        0125 0126 0153 0154 0155 0156
  60 INDICATOR DEFINED     0105 0239 0261
          REFERENCED        0106 0107 0108 0135 0136 0137 0172 0184 0185
                            0186 0196 0206 0207 0208 0240
 *61 INDICATOR DEFINED     0105 0239
          NOT REFERENCED
  H0 INDICATOR DEFINED     0316 0324 0328
          RPG-REFERENCED
  LR INDICATOR DEFINED     0262
          REFERENCED        0074 0100 0129 0146 0182 0200 0220
  F0 INDICATOR DEFINED     0329
          REFERENCED        0098 0101 0127 0130 0144 0147 0180 0183
                            0198 0201 0217 0218 0221 0317 0357
  F1 INDICATOR RPG-DEFINED
          REFERENCED        0085 0283
  F2 INDICATOR RPG-DEFINED
          REFERENCED        0086 0284
  F3 INDICATOR RPG-DEFINED
          REFERENCED        0087 0285
  F4 INDICATOR RPG-DEFINED
          REFERENCED        0268
             .
             .
             .

 * 4 INDICATOR(S) DEFINED, BUT NOT REFERENCED.
```

Figure 6-14. A Cross-Reference Listing (Indicators Used)

	Comments
01	This is the indicator name.
INDICATOR DEFINED	This message precedes the line numbers where the indicator is defined.
0034	This is the line number where indicator 01 is defined (0034 is the compiler-generated line number).
NOT REFERENCED	This message appears for indicators that are not referenced in the program.
	REFERENCED appears for indicators that are used in the program.
	RPG-REFERENCED appears for indicators used by the RPG logic cycle.

```
PAGE 0012              FIELD NAMES USED

**$ADD     EXCPT)            NOT DEFINED
           REFERENCED            0113 0406
  $DEL     EXCPT)                0226
           REFERENCED            0438
**$UPD     EXCPT)            NOT DEFINED
           REFERENCED            0160 0422
  ACTION   (FIELD)       6      0300
           REFERENCED            0253 0302 0312 0314 0322 0326 0336 0371 0376 0396
 *ACTNO    (FIELD)       6      0052
           NOT REFERENCED
  ADATE    (FIELD)       8      0065
           REFERENCED            0252
  ADD      (SUBR)                0093
           REFERENCED            0085
  ADD1     (TAG)                 0094
           REFERENCED            0099 0114
  ADD3     (TAG)                 0096
           REFERENCED            0108
  ADD9     (TAG)                 0115
           REFERENCED            0101
  ADDR1    (FIELD)      28       0055
           REFERENCED            0243
  ADDR2    (FIELD)      28       0056
           REFERENCED            0244
 *ADDR3    (FIELD)      28       0057
           NOT REFERENCED
  BADFLD   (FIELD)     5.0       0346
           RPG-REFERENCED
                .
                .
                .
  * 3 FIELD NAME(S) DEFINED, BUT NOT REFERENCED.
 ** 4 FIELD NAME(S) REFERENCED, BUT NOT DEFINED.
```

Figure 6-15. A Cross-Reference Listing (Field Names Used)

	Comments
ACTION	This is a field name. Other names appearing in this column are table or array element names, labels for TAG operations and subroutine names.
(FIELD)	This describes ACTION's data element type. TAG appears for TAG operation labels. SUBR appears for subroutine names.
6	This is the length of the field, ACTION. If the field is a numeric field, the number appearing in this column shows the number of digits and decimals in the field. For example, 5.2 indicates that the field has 5 digits and 2 decimals. If this is not a data field, no number appears.
0300	This number is the compiler-generated line number where the field is defined.
REFERENCED	This message precedes the line numbers that specify where ACTION is used. NOT REFERENCED appears when the field is not used in the program.
0253...	This begins the line numbers that specify where the field ACTION is used.

```
PAGE 0015              FILE NAMES USED

  ACCOUNT(UPDATE - CHAIN)     0018
          REFERENCED          0051 0105 0239 0406
  TERMINAL(UPDATE - DEMAND)   0013
          REFERENCED          0031 0316 0324 0328 0370

_
```

Figure 6-16. A Cross-Reference Listing (File Names Used)

	Comments
DACCOUNT	This is the name of a file used in the program.
(UPDATE - CHAIN)	This message describes DACCOUNT as File Type UPDATE with File Designation CHAIN. (File Type and Designation are defined in columns 15 and 16 of the

	File Description Specification.)
0018	This is the line number where DACCOUNT is defined.
REFERENCED	This message precedes the line numbers that specify where DACCOUNT is used in the program. NOT REFERENCED is printed when the file is not referenced in the program.
0051...	This begins the compiler-generated line numbers that specify where DACCOUNT is referenced in the program.

Understanding RPG Compiler Messages

There are three kinds of compiler messages: informational messages, warnings and errors. An informational message reminds you of potential error situations that you may have overlooked. A warning indicates that there may be an error, but it does not prevent a successful compilation. An error means that the problem prevents a successful compilation. You must correct errors. See Appendix A in the *HP RPG Reference Manual* for explanations of compiler messages and how to correct errors.

Informational messages, warnings and errors are shown at the end of the compiler listing. Figure 6-17 shows what informational messages look like.

```
PAGE 0016   RPGOBJ          ERRORS DURING COMPILATION

9001I    INDICATOR DEFINED BUT NOT REFERENCED                    0443
9014I    FIELD NAME(S) DEFINED BUT NOT REFERENCED.               0443

 NO. SERIOUS ERRORS 000     NO. WARNINGS 000
 PROCESSOR TIME 00:00:03    ELAPSED TIME 00:00:04
```

Figure 6-17. Compiler Informational Messages

	Comments
9001	This is the number which identifies the message. Use this number to look up the message in Appendix A of the *HP RPG Reference Manual*.
I	This is the letter code for Informational (I) messages. Warnings have a W letter code and terminal errors have a T letter code. You must correct Terminal errors before you can successfully compile

	the program.
INDICATOR...	This is a brief description of the message. For a complete description and corrective action, see Appendix A in the *HP RPG Reference Manual*.
0443	This is the compiler-generated line number to which the message corresponds.

Chapter 7 Executing an RPG Program

This chapter describes how to execute an RPG program and how to use the RPG DEBUG feature during execution. It also explains how to interpret various run-time messages that you may encounter and what to do about them. And finally, it gives general tips on how to avoid run-time errors.

There are three ways to execute an RPG program:

Use this (these) MPE XL Command(s):	When you want to:
RPGXLGO	Compile, link and execute the program in one step. This method is handy during initial program testing. For an explanation of this method, see "Compiling, Linking and Executing" in Chapter 6.
RPGXLLK RUN	Compile and link the program in one step and execute it in another. If you do not enter a file name for the executable program file, $OLDPASS is used. To run the executable program file, follow RPGXLLK with the RUN command (see "RUN" below).
RUN	Run an executable program file that was created previously by a RPGXLLK command or by a HP Link Editor/XL LINK command. You normally execute production programs or programs with minor changes using the RUN command. You can also run an executable program file by omitting the word "RUN" and simply typing the executable program file name at the operating system prompt.

Figure 7-1 shows how to use RPGXLLK and RUN. This example compiles, links and executes a program. The program source file is GL50S.SOURCE and the job file name is GL50.JOB. To execute this job file, enter the operating system command,

```
:STREAM GL50.JOB
```

```
!JOB  GL50,MGR.ACCTG
!RPGXLLK  GL50S.SOURCE,  GL50P.PROGRAM
!RUN  GL50P.PROGRAM
!EOJ
```

Figure 7-1. Compiling, Linking and Executing an RPG Program

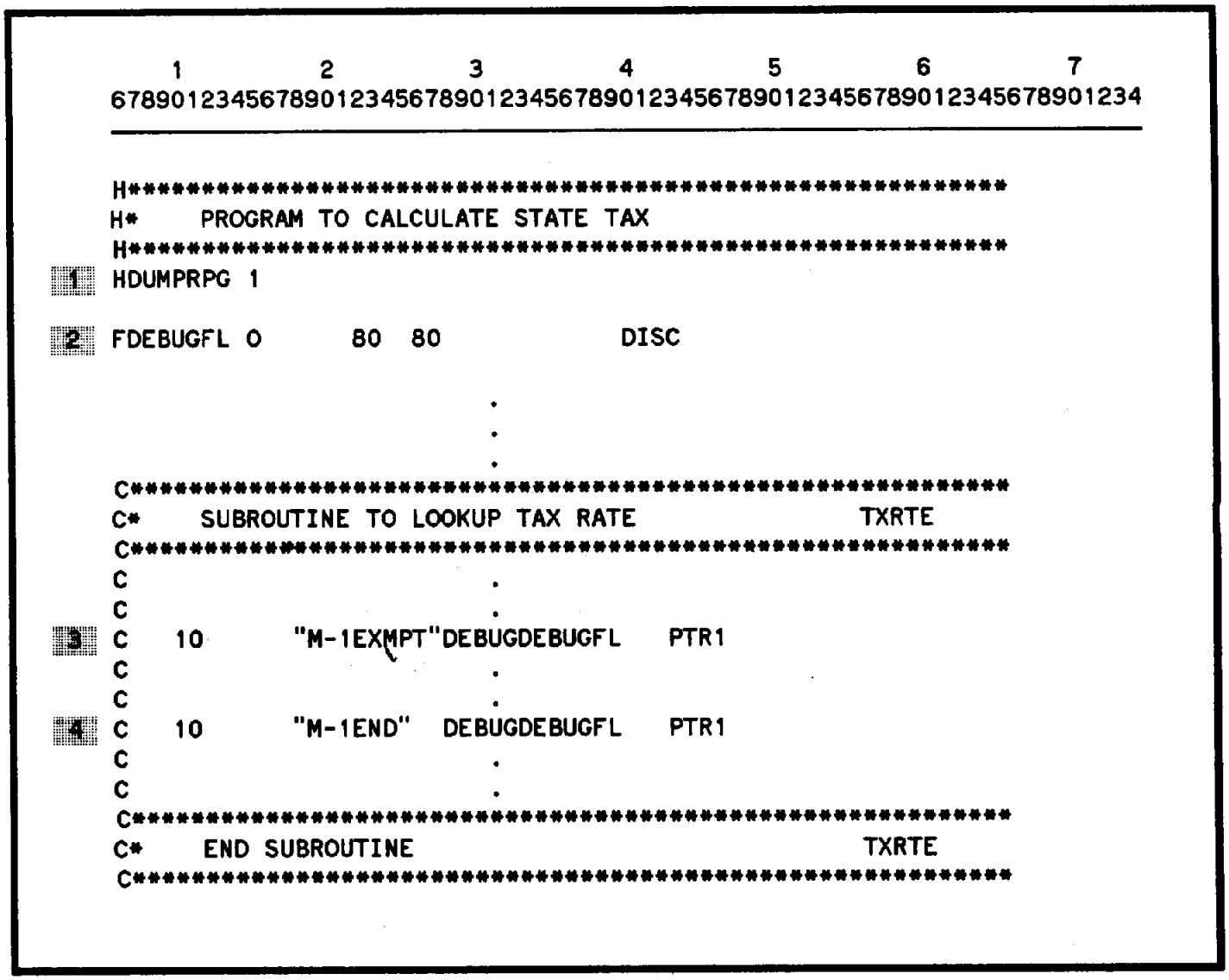
Using RPG DEBUG

RPG has a facility, DEBUG, that you can use in your program as a debugging tool. DEBUG is an operation used with Calculation Specifications. It lets you monitor the status of indicators and the contents of fields, tables and arrays while a program is executing. You place DEBUG statements before and after the operations that you want to monitor.

DEBUG information is usually displayed on your terminal (session mode) or printed along with your job listing (job mode). If you wish, you can

have this information written to a disc file instead.

Figure 7-2 shows how to use DEBUG. To enable DEBUG, you must enter a 1 in column 15 of the Header Specification. If you leave this column blank, DEBUG statements in your program are ignored (treated as comments).



```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
H*****************************************************************
H*    PROGRAM TO CALCULATE STATE TAX
H*****************************************************************
1    HDUMPRPG 1

2    FDEBUGFL O      80  80            DISC

                              .
                              .
                              .
C*****************************************************************
C*    SUBROUTINE TO LOOKUP TAX RATE                  TXRTE
C*****************************************************************
C                             .
C                             .
3    C   10       "M-1EXMPT"DEBUGDEBUGFL    PTR1
C                             .
C                             .
4    C   10       "M-1END"  DEBUGDEBUGFL    PTR1
C                             .
C                             .
C*****************************************************************
C*    END SUBROUTINE                                 TXRTE
C*****************************************************************
```

Figure 7-2. Using DEBUG

Comments

1 This Header Specification enables the DEBUG feature of RPG.

Column 15 is 1 to enable DEBUG.

2 This line defines the disc file that holds the DEBUG information.

3 This line logs DEBUG information to a disc file.

Columns 18-27 contain a tag, M-1EXMPT, that identifies this DEBUG operation data in the disc file.

Columns 28-32 specifies the DEBUG operation.

Columns 33-42 name the disc file, DEBUGFL, in which the DEBUG data is saved.

Columns 43-48 specify that the contents of the field, PTR1, are also written to the disc file.

4 This line logs DEBUG information to a disc file.

Columns 18-27 contain a tag, M-1END, that identifies this DEBUG operation data in the disc file.

Handling Execution Errors

If an error occurs while you're executing an RPG program, an error message (and often an error number) is displayed. If the error is a file error, the File Information Display is produced automatically along with an Error Dump (you can request error dumps independently, if you wish). The sections which follow in this chapter discuss error messages, Error Dumps and the File Information Display in detail.

If you're executing a program in session mode, the error message and File Information Display are shown on your terminal. In job mode, error messages are displayed on the operator's console and the File Information Display is saved with the job listing. It is a good idea, when running in job mode, to capture the messages and file information in a disc file. This gives you more flexibility in scheduling your jobs and lets you examine results at a later time. The following RUN command executes the program GL50 and redirects error information to the existing disc file, MESSFL:

```
:RUN GL50;STDLIST=MESSFL
```

Error Messages

There are four types of error messages: RPG, IMAGE, USWITCH and BUFCHK. RPG errors are general errors such as arithmetic overflow and matching record sequence errors. TurboIMAGE errors originate in the TurboIMAGE subsystem and you may see these errors if you're using an IMAGE database in your program. You may see USWITCH and BUFCHK errors if you're using the USWITCH Source or the BUFCHK features of RPG, respectively.

Each type of error message has a different format and is documented separately. RPG, USWITCH and BUFCHK errors are documented in Appendix B of the *HP RPG Reference Manual*. TurboIMAGE errors are explained in the *TurboIMAGE/XL Database Management System manual*. It is important, therefore, to learn to recognize these three types of message formats so that you will know where to find explanations and remedial procedures for them.

RPG Errors. Most of the error messages you see when running an RPG program are RPG error messages. RPG errors are detected by the software that performs the basic RPG processing. They also include errors occurring in software that RPG interfaces with, such as VPLUS, RSI, and KSAM.

Appendix B of the *HP RPG Reference Manual* contains a complete description of the RPG errors and actions that you can take to correct the errors.

Identifying RPG Errors

Figure 7-3 shows a typical RPG error message. Lines relating to RPG errors are shaded. You identify an RPG error by the line,

```
*** RPG ERROR ***
```

You identify the specific error by the error number line,

```
1.    FATAL FILE ERROR,  FILENAME=DACCOUNT
```

```
Program:RPGOBJ = RPGPROG.RPGXL.RPG      WED, FEB 5, 1986, 5:10 PM
Pgm-Err:RPGOBJ = RPGPROG.RPGXL.RPG      WED, FEB 5, 1986, 5:10 PM
*** RPG ERROR ***
RPG Error: File not available for access.
MPE Error: NONEXISTENT PERMANENT FILE  (FSERR 52)

                    .
                    .
                    .

1.    FATAL FILE ERROR,  FILENAME=DACCOUNT

 ACTION WAS 5 = IMMEDIATE TERMINATION WITH DUMP
```

Figure 7-3. An RPG Execution Error

Changing the Way RPG Errors Are Handled

Normally, when an RPG error occurs, a message is displayed and the user (or console operator) selects a particular action to take. The operator can ignore the error and continue, skip the line in error and continue or terminate the program. If you do not want the operator or user to make these choices for specific error situations, you can enter the actions to take within your program. You can do this using either (or both) the Header Specification or the *ERROR field.

Use the Header Specification to enter the responses you want RPG to make when an error occurs. For instance, you might want the operator to handle errors except when an input file sequence error occurs. When input sequence errors occur, you want to continue processing and use your own error handling instead. (Use the *ERROR field to find out the error number, then code your own RPG operations to handle the error.) Figure 7-4 shows a Header Specification that suppresses normal RPG processing for input file sequence errors (the program includes code to handle these errors).

```
     1         2         3         4         5         6         7
    6789012345678901234567890123456789012345678901234567890123456789012345678901234
                                                                                
1   HDUMPRPG                                                N  0
```

Figure 7-4. Providing a Response to RPG Errors Using the Header Specification

Comments

1 This Header Specification changes the default error handling for input file sequence errors.

Column 55 contains N to specify that the error message for sequence errors be suppressed. Instead of the normal response to sequence errors, take the action entered in column 58.

Column 58 contains 0 to ignore input file sequence errors and to continue program execution.

TurboIMAGE Errors. If you're using an TurboIMAGE database, you may see TurboIMAGE errors. To find the explanation and remedies for TurboIMAGE errors, see the *TurboIMAGE/XL Database Management System manual*.

Figure 7-5 shows what an TurboIMAGE read error message looks like.

```
Program:RPGOBJ = RPGPROG.RPGXL.RPG      WED, FEB 5, 1986, 5:10 PM
Pgm-Err:RPGOBJ = RPGPROG.RPGXL.RPG      WED, FEB 5, 1986, 5:10 PM
******FILE ERROR******
READ ERROR ON DACCOUNT.PUB.RPG

                .
                .
                .
```

Figure 7-5. An TurboIMAGE Error

USWITCH Errors. USWITCH errors may occur when you use the USWITCH Source feature (column 16 in the Header Specification). Appendix B of the *HP RPG Reference Manual* contains a complete description of the USWITCH errors and recovery procedures for them.

These are the USWITCH error messages you may encounter:

I/O ERROR ON $STDLIST

I/O ERROR ON $STDIN

INVALID INPUT DATA

UNEXPECTED END OF FILE

Figure 7-6 shows what a typical USWITCH error display looks like.

```
Program:RPGOBJ = RPGPROG.RPGXL.RPG      WED, FEB 5, 1986, 5:10 PM
Pgm-Err:RPGOBJ = RPGPROG.RPGXL.RPG      WED, FEB 5, 1986, 5:10 PM
INVALID INPUT DATA

                    .
                    .
                    .
```

Figure 7-6. A USWITCH Error

BUFCHK Errors. BUFCHK errors are data buffer handling errors that may occur when you use the BUFCHK Defaults feature (column 28 in the Header Specification). Appendix B of the *HP RPG Reference Manual* contains a complete description of the BUFCHK errors and recovery procedures for them.

These are the BUFCHK error messages you may encounter:

INTERNAL OR INTRINSIC ERROR

ATTEMPTED UPDATE BEFORE INPUT OF FIRST RECORD.

ATTEMPTED UPDATE ON SAME RECORD OR ON AN INTERVENING "ADD" RECORD.
ATTEMPTED LOCKING FILE AT BOTH RECORD-LEVEL AND FILE-LEVEL

Figure 7-7 shows what a typical BUFCHK error display looks like.

```
Program:RPGOBJ = RPGPROG.RPGXL.RPG      WED, FEB 5, 1986, 5:10 PM
Pgm-Err:RPGOBJ = RPGPROG.RPGXL.RPG      WED, FEB 5, 1986, 5:10 PM
ATTEMPTED UPDATE ON SAME RECORD OR ON AN INTERVENING
_         "ADD" RECORD.

                    .
                    .
                    .
```

Figure 7-7. A BUFCHK -4 Error

The File Information Display

You see the File Information Display (often referred to as "tombstone") when an error occurs that involves a file. File Information Displays give additional information about the error. File Information Displays are discussed in the operating system intrinsics and error messages manuals.

The File Information Display is shown in short form when the file cannot be opened for processing. This happens, for example, when the file does not exist in the group or account specified. The shaded portion of Figure 7-8 shows what the short form looks like.

```
Program:RPGOBJ = RPGPROG.RPGXL.RPG     WED, FEB 5, 1986, 3:54 PM
Pgm-Err:RPGOBJ = RPGPROG.RPGXL.RPG     WED, FEB 5, 1986, 3:54 PM
*** RPG ERROR ***
RPG Error: File not available for access.
MPE Error: NONEXISTENT PERMANENT FILE  (FSERR 52)

+-F-I-L-E---I-N-F-O-R-M-A-T-I-O-N---D-I-S-P-L-A-Y+
|  ERROR NUMBER: 52     RESIDUE:0                |
|  BLOCK NUMBER: 0              NUMREC:0         |
+----------------------------------------------+

1.    FATAL FILE ERROR,  FILENAME=ZSYSCTL
ACTION WAS 5 = IMMEDIATE TERMINATION WITH DUMP
```

Figure 7-8. Short Form of the File Information Display

Once a file is opened for processing and an error occurs, additional file information is given in the File Information Display. This information should help in clarifying the exact cause of the error. In Figure 7-9, for example, RPG error number 1 occurred for the file, DACCOUNT. The File Information Display gives detailed information about DACCOUNT.

```
Program:RPGOBJ = RPGPROG.RPGXL.RPG      WED, FEB 5, 1986, 5:10 PM
Pgm-Err:RPGOBJ = RPGPROG.RPGXL.RPG      WED, FEB 5, 1986, 5:10 PM
*** RPG ERROR ***
RPG Error: ?????
MPE Error: END OF FILE  (FSERR 0)

+-F-I-L-E---I-N-F-O-R-M-A-T-I-O-N---D-I-S-P-L-A-Y-+
|  FILE NAME IS DACCOUNT.PUB.RPG                   |
|  FOPTIONS: SYS,ASCII,FORMAL,F,NOCCTL,REQ,KSM     |
|            NOLABEL                               |
|  AOPTIONS: IN/OUT,NOMR,NOLOCK,DEF,NOBU           |
|            NOMULTI,WAIT,NOCOPY                   |
|  DEVICE TYPE:0      DEVICE SUBTYPE: 9            |
|  LDEV: 2       DRT: 4       UNIT:1               |
|  RECORD SIZE: 200   BLOCK SIZE: 1200  (BYTES)    |
|  EXTENT SIZE: 10    MAX EXTENTS: 8               |
|  RECPTR: 2          RECLIMIT: 90                 |
|  LOGCOUNT: 84       PHYSCOUNT: 14                |
|  EOF AT: 90         LABEL ADDR: %00200020273     |
|  FILE CODE: 0       ID IS MGR   ULABELS: 0       |
|  ERROR NUMBER: 0    RESIDUE: 0                   |
|  BLOCK NUMBER: 0    NUMREC:0                     |
+--------------------------------------------------+

1.    FATAL FILE ERROR,  FILENAME=DACCOUNT

 ACTION WAS 5 = IMMEDIATE TERMINATION WITH DUMP
```

Figure 7-9. Long Form of the File Information Display

The Error Dump

The Error Dump shows the contents of certain areas in memory when a program terminates due to execution errors. For example, the Error Dump shows fields and their contents and the settings of indicators. Executable program code is not listed in the Error Dump.

You get the Error Dump automatically when file errors occur. Otherwise, you must specifically request it. You should always request dumps for production programs. The next four sections explain how to request an Error Dump and the information it contains.

Creating an Error Dump File. It is usually more convenient to write the Error Dump to a disc file rather than display it. Before executing the program, create an empty disc file for the Error Dump.

The following operating system command creates the disc file, DUMPRPG, that is in a format that can be used to save Error Dumps,

```
:BUILD DUMPRPG;REC=-80,16,F,ASCII;DISC=512,8
```

Requesting an Error Dump. You can get an Error Dump automatically by entering an S in the Error Log Field (column 55) of the Header Specification. The dump is produced whenever an RPG error occurs. If you only want certain errors to trigger a dump, put an N in column 55 or leave it blank. Then enter an error response for those errors (which include a dump) into the Error Response Field (columns 56-71). (Alternatively, if you leave the Error Log Field blank and an error occurs, the operator can select response 5. Response 5 terminates the program and produces a dump.)

This figure shows a Header Specification that requests an Error Dump whenever an RPG error occurs.

```
     1         2         3         4         5         6         7
    67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
1   HDUMPRPG                                                    S
```

Figure 7-10. Requesting an Error Dump

Comments

1 This Header Specification requests an Error Dump and directs it to a disc file.

Columns 7-14 name the disc file, DUMPRPG, that contains the Error Dump.

Column 55 is S to request an Error Dump if an error occurs. The program terminates immediately. When you use S, you cannot use the Response Field (columns 56-71) to enter error responses.

Parts of the Error Dump. Program information is shown in the Error Dump in logical groupings with appropriate titles. The dump is easy to read. Figure 7-11 (spread over several pages) shows an Error Dump of the program in Figure 4-23 (see the compiler listing for this program in Figure 6-11 through Figure 6-16).

The first part of the dump, LIBRARY POINTERS, shows the addresses of the pointers to various tables and storage areas used by the program. Each pointer contains the address of the first word of the table or storage area.

```
*** LIBRARY POINTERS ***
lib_pointer pointer = 40200010
run-time globals pointer = 40200078
indicators pointer = 40200170
work area pointer = 40200230
alpha field pointer = 4020025C
numeric field pointer = 40200AC4
LDA buffer pointer = 00000000
UDATE field pointer = 40200ACA
UDAY field pointer = 40200ADD
UMONTH field pointer = 40200AEA
UYEAR field pointer = 40200AE4
ERROR field pointer = 40200AA5
alt collating seq tbl pointer = 00000000
file translation tables pointer = 00000000
file extension tables pointer = 40200DF4
workstation table pointer = 40200AF8
first file table pointer = 40200ED0
first record buffer pointer = 402010F0
level info pointer = 00000000
matching record info pointer = 00000000
first chain table pointer = 00000000
table/array tables pointer = 402012D0
first RLABL pointer = 00000000
terminal control table pointer = 00000000
current input proc. pointer = 40100081
get_rec() proc. pointer = 401000A1
```

Figure 7-11. The Error Dump - LIBRARY POINTERS

After the LIBRARY POINTERS section, the RUN-TIME GLOBALS DATA AREA is displayed. This area contains run-time control and status information. For example, one piece of information is the file number for a message file used in the program. The first part of the globals area for the program in Figure 4-11 is shown below.

```
*** RUN-TIME GLOBALS DATA AREA ***
H Col 16: user switch source = 0
H Col 17: UDATE source = 0
H Col 49: message control on record length error = 0
Date and number format: 0
Packed Decimal -1: OOOOOO1D
Packed Decimal +0: OOOOOOOC
Packed Decimal +1: OOOOOO1C
Run-time File Table count: 2
Run-time File Table size (bytes): 260
Table/Array Table count: 2
Table/Array Table size (bytes): 68
local data area size (bytes): 0
$STDLIST output record length (bytes): 81
User Message File number: 0
Current Line number: 504
Loop Counter: 0
Min size: 0
H Col 41: Request Page Alignment = 0
H Col 47: Don't skip to new page = 0
H Col 44: OK to move spec. chars = 0
in RPG logic cycle: 1
Match Descending in record matching: 2
Match Primary in record selection: 0
Match Records in record selection: 0
OK to end program: 0
In block mode: 1
Running Interactively: 1
NLS Catalog opened: 0
DS move done: 0
perform numeric validation: 1
uswitches used flag: 0
error parameter: 0
```

```
temporary storage: 0
$CONTROL EXCQUIT flag: 0
LDA file number: 0
$CONTROL NEWSAVE flag: 0
          .
          .
          .
```

Figure 7-11. The Error Dump - RUN-TIME GLOBALS DATA AREA

The next two figures show the contents of the File Table Areas for each file processed in the program. The first file shown is TERMINAL and the second is DACCOUNT (these names are shaded in the figures).

The first section of each figure shows general information about the file. For example, TERMINAL is a WORKSTN file whose designation is DEMAND. Following the general information is the contents of the file buffer (RECORD BUFFER (ASCII)). Then, the contents of the FILE TABLE are listed in hexadecimal notation. The FILE TABLE contains all of the control information used to process the file and may prove helpful in determining the cause of file errors. The WORKSTATION EXTENSION TABLE follows the FILE TABLE if the file is a WORKSTN file; the IMAGE EXTENSION TABLE follows if the file is a TurboIMAGE file. They give information relevant to those file types.

Refer to Table 7-1 through Table 7-3 for descriptions of positions in the FILE, IMAGE EXTENSION and WORKSTATION EXTENSION TABLES. (A "word" in Table 7-1 through Table 7-3 is 32 bits long.)

```
*** FILE TABLE AREA ***
FILE NAME = TERMINAL

MPEXL FILE NUMBER = -1         DEVTYPE = WORKSTN (ACTUAL = DISC)

FILE TYPE   UPDATE             FILE DESIGNATION   DEMAND
RECORD FORMAT        256B, IR/B, VARIABLE SEQUENTIAL

RECORD BUFFER (ASCII)

FILE TABLE
40200ED0:   FFFFFFFF   5445524D   494E414C   574F524B   53544E20
40200EE4:   00000000   00000000   00000000   00000002   00000006
40200EF8:   00000000   00000000   00000000   00000000   00000000
40200F0C:   00000000   00000000   00000000   00000001   00000000
40200F20:   00000000   00000000   00000100   00000001   00000000
40200F34:   402010F0   00000000   00000000   00000000   00000000
40200F48:   00000000   00000000   00000000   00000000   00000000
40200F5C:   40200AF8   00000000   00000000   00000000   00000000
40200F70:   00000000   00000000   00000000   00000000   00000000
40200F84:   00000000   00000000   00000000   00000001   01000000
40200F98:   00000000   00000000   00000000   00000000   00000000
4O2OOFAC:   00000000   00000000   00000000   00000000   00000000
40200FC0:   00000000   00000000   00000000   00000000   00000000

WORKSTATION EXTENSION TABLE
40200AF8:   00000000   00000000   00000000   00000000   00000000
40200B0C:   00000001   00000000   00000000   00000001   00000001
40200B20:   00000001   00000000   00000000   00000000   00000000
40200B34:   00000000   00000000   0000000B   00000000   FFFFFFF7
40200B48:   00000000   40A00000   40200ED0   00000000   00000000
               .
               .
               .
```

Figure 7-11. The Error Dump - TERMINAL File and Workstation Tables

```
FILE NAME = DACCOUNT

MPEXL FILE NUMBER = 1          DEVTYPE = DISC     (ACTUAL = DISC)

FILE TYPE   UPDATE             FILE DESIGNATION    CHAINING
RECORD FORMAT        200B, IR/B, FIXED RANDOM

RECORD BUFFER (ASCII)

FILE TABLE
```

```
40200FD4:   00000001   44414343   4F554E54   44495343   20202000
40200FE8:   00000000   00000000   00000000   00000002   00000004
4O2OOFFC:   0000000F   00000001   00000000   00000000   00000006
40201010:   00000000   00000002   00000000   00000000   00000000
40201024:   00000000   00000000   000000C8   00000001   00000000
40201038:   40201208   00010100   00000001   00000001   00000000
4020104C:   00000000   00000000   00000000   00000000   00000000
40201060:   00000000   40200DF4   00000000   00000000   00000000
40201074:   00000000   00000000   00000000   00000000   00000000
40201088:   00000000   00000000   00000000   00000101   01000000
4020109C:   00000000   00000000   00000000   00000000   00000000
402010B0:   00000000   00000000   00000000   00000000   00000000
402010C4:   00000000   00000000   00000000   00000000   00000000

IMAGE EXTENSION TABLE
40200DF4:   00014D41   524B4554   20202020   57524954   45522020
40200E08:   4143434F   554E542D   4E4F2020   20202020   20202020
40200E1C:   20202020   442D4143   434F554E   54532020   20202020
40200E30:   00000011   3CAElOOO   00004192   40200DF4   00CA0000
40200E44:   00000000   00000000   00000000   00000000   00000000
40200E58:   00000004   00000005   00000040   00000001   00000000
40200E6C:   00000000   00000000   00000000   00000000   00000000
40200E80:   00000000   00000001   00000003   00000000   00000000
40200E94:   00000000   00000000   00000000   00000000   00000000
40200EA8:   40200EC4   40200ECA   00000000   00000000   00000000
4O2OOEBC:   00000000   00000000   00000000   00000000   00000000
40200ED0:   FFFFFFFF   5445524D   494E414C   574F524B   53544E20
40200EE4:   00000000   00000000   00000000   00000002   00000006
40200EF8:   00000000   00000000   00000000   00000000   00000000
                .
                .
                .
```

Figure 7-11. The Error Dump - DACCOUNT File and Extension Tables

After the File Table area, all indicator settings are listed as shown in this figure.

```
***  LEVEL  INDICATORS  ***

LO   =   ON    Ll   =   OFF   L2   =   OFF   L3   =   OFF   L4   =   OFF
LS   =   OFF   L6   =   OFF   L7   =   OFF   L8   =   OFF   L9   =   OFF

***  FUNCTION  KEY  INDICATORS  ***

FO   =   OFF   Fl   =   OFF   F2   =   OFF   F3   =   OFF   F4   =   OFF
FS   =   OFF   F6   =   OFF   F7   =   OFF   F8   =   OFF   F9   =   OFF

***  COMMAND  KEY  INDICATORS  ***

KA   =   OFF   KB   =   OFF   KC   =   OFF   KD   =   OFF   KE   =   OFF
KF   =   OFF   KG   =   OFF   KH   =   OFF   KI   =   OFF   KJ   =   OFF
KK   =   OFF   KL   =   OFF   KM   =   OFF   KN   =   OFF   KP   =   OFF
KQ   =   OFF   KR   =   OFF   KS   =   OFF   KT   =   OFF   KU   =   OFF
KV   =   OFF   KW   =   OFF   KX   =   OFF   KY   =   OFF

***  HALT  INDICATORS  ***

HO   =   OFF   Hl   =   OFF   H2   =   OFF   H3   =   OFF   H4   =   OFF
H5   =   OFF   H6   =   OFF   H7   =   OFF   H8   =   OFF   H9   =   OFF

***  OVERFLOW  INDICATORS  ***

OA   =   OFF   OB   =   OFF   OC   =   OFF   OD   =   OFF
OE   =   OFF   OF   =   OFF   OG   =   OFF   OV   =   OFF
LR   =   OFF   MR   =   OFF   IP   =   OFF

***   USER INDICATORS ***

Ul   =   OFF   U2   =   OFF   U3   =   OFF   U4   =   OFF   U5   =   OFF
U6   =   OFF   U7   =   OFF   U8   =   OFF

***   OTHER   INDICATORS ***

                01   =   OFF   02   =   OFF   03   =   OFF   04   =   OFF
```

```
OS   =   OFF   06   =   OFF   07   =   OFF   08   =   OFF   09   =   OFF
```

Figure 7-11. The Error Dump - INDICATORS

The contents of alphanumeric and numeric fields are listed next, as shown in the following figure. To see the contents of a particular field, find the field's address in the compiler Symbol Table listing, then locate that address using the address column at the left margin. For example, the ADDR column in the Symbol Table listing for the field ACTION contains 000002a4 (in hexadecimal). Adding the offset address (40200010) of lib_pointer in the LIBRARY POINTERS section of the Error Dump, the actual address for ACTION is 402002B4. Since ACTION is an alphanumeric field, it takes up six bytes. The shaded area in the Alphanumeric Fields section shows the contents of ACTION (it is blank).

```
*** ALPHANUMERIC FIELDS ***

4020025C:
4020029D:
402002DE:
4020031F:                                                    ADD     MODE      CH
40200360: ANGE   MODE    INQUIRY MODE                        DELETE  MODE
402003A1:                                EXIT
402003E2:                                                                SELECT
40200423: MODE WITH FUNCTION KEY
40200464: MAKE CHANGES AND HIT ENTER (F7 TO CANCEL CHANGE)
402004A5:
402004E6:                                          YOU MAY NOT CHANGE ACCOUNT NUM
40200527: BER OR LAST NAME    (F7 TO CLEAR)                  INVALID KEY (F7
40200568: TO CLEAR)                                                        EX
402005A9: ISTING ACCOUNT NUMBER
402005EA:               INVALID ACCOUNT NUMBER
4020062B:                              PRESS F5 AGAIN TO CONFIRM DELETE (F7 TO
4020066C:   CANCEL DELETE)
402006AD:
402006EE:
4020072F:
    .
    .
    .

*** NUMERIC FIELDS ***

40200AC4: 0000000000000030988F000000000000000000000000000000009F0000000000
40200AE4: 088F00000000003F00000000000000000000
```

Figure 7-11. The Error Dump - ALPHANUMERIC and NUMERIC FIELDS

The last part of the formatted dump lists table and array information. Each table and array is listed in the order in which it appears in the program. For each table and array, control information is printed first followed by an ASCII dump of the table or array's contents. In the following figure, the arrays MSG and LBL are printed (they are defined in lines 25 and 26 of the source program).

```
** TABLES AND ARRAYS ***

ARRAY DEFINED ON LINE NUMBER = 25
COMPILE TIME ARRAY, NO SEQUENCE, ALPHAMERIC, NOT ALTERNATING
ENTRY LENGTH 79           POINTER TO ARRAY =   4020041C    NO. OF ENTRIES = 20

HEX DUMP OF T/A CONTROL INFORMATION
402012D0:  403BBC08  00000002  00000000  00000000  00000000
402012E4:  00000000  00000000  00000001  00000001  0000004F
402012F8:  00000014  0000004F  00000019  4020041C  00000000
4020130C:  00000000  00000000

ASCII DUMP OF ARRAY
4020041C :
SELECT MODE WITH FUNCTION KEY
4020045D :
               MAKE CHANGES AND HIT ENTER (F7 TO CANCEL CHANGE)
4020049E :
    .
```

```
    .
    .
ARRAY DEFINED ON LINE NUMBER = 26
COMPILE TIME ARRAY, NO SEQUENCE, ALPHAMERIC, NOT ALTERNATING
ENTRY LENGTH 16          POINTER TO ARRAY =   4020034E     NO. OF ENTRIES = 8

HEX DUMP OF T/A CONTROL INFORMATION
40201314:  00000000  00000002  00000000  00000000  00000000
40201328:  00000000  00000000  00000000  00000001  00000010
4020133C:  00000008  00000010  0000001A  4020034E  00000000
40201350:  00000000  00000000

ASCII DUMP OF ARRAY
4020034E :
ADD MODE         CHANGE  MODE     INQUIRY MODE                    D
4020038F :
ELETE MODE                                        EXIT
```

Figure 7-11. The Error Dump - TABLES AND ARRAYS

Table 7-1. The File Table

Word	Byte	Meaning
0		Number returned by FOPEN intrinsic
1-2		File name
3-4		Device type name
5-7		Line number for line printer channel control
8		File type: 0 = Input file 1 = Output file 2 = Update file 3 = Display file 4 = Combined file
9		File designation: 0 = Secondary input file 1 = Blank 2 = Primary input file 3 = Record address file 4 = Chaining file 5 = Table or array file 6 = Demand file
10		File organization: 1-7 = Sequential or direct access file (allocate 1-7 buffers) 8-9 = Sequential or direct access file (allocate 2 buffers) 00 = Neither an ADDROUT nor a direct-access file 10 = An ADDROUT file 11 = A direct-access file 13 = A KSAM file 14 = An ISAM file 15 = A TurboIMAGE file
11		Processing mode: 0 = A sequential file 1 = A random-access file 2 = An indexed file processed sequentially between limits
12		Overflow indicator (if this is a printer file): 122-128 = OA-OG 129 = OV
13		File conditioner: 0 = File can be used unconditionally 179-186 = U1-U8
14		The key length (in bytes) of a RAF or TurboIMAGE file
15		Line printer carriage control
16		Record address type: 0 = A sequential file or a direct access file not processed by chaining or a RAF 1 = A file processed by chaining or a RAF 2 = A KSAM, ISAM or TurboIMAGE file processed by alphanumeric key 3 = A KSAM, ISAM or TurboIMAGE file processed by packed numeric key
17		Number of user labels: 0 = Standard label only 1-9 = Standard label with 1-9 user labels
18		File format: 0 = Fixed length records 1 = Variable length records

Table 7-1. The File Table (Continued)

Word	Byte	Meaning
19		Number of extents
20		Type of file translation table: 0 = None 1 = User-defined 2 = EBCDIC 3 = EBCDIK
21		New page starting line number for line printer files; or record number adjust for direct file
22		Record length in bytes
23		Block length in bytes
24		Pointer to translation table
25		Pointer to file record buffer
26	0	Enable current data check flag
26	1	Enable new record check flag
26	2	Enable update record check
26	3	Look-ahead file flag: 0 = Not used as a look-ahead file 1 = Used as a look-ahead file
27	0	Terminate method flag: 0 = End-of-file field in File Description Specification is blank 1 = End-of-file field in File Description Specification contains (all records must be processed before program termination)
27	1	Run-time SPECIAL file flag: 0 = Not a SPECIAL file 1 = A SPECIAL file
27	2	Overflow line passed flag
27	3	Append to end-of-file flag
28	0	NOLOCK Continuation line flag
28	1	Lock/unlock option flag
28	2	Printing mode flag
29		Pointer to label exit name or SPECIAL file processing routine
30		Pointer to error exit subroutine
31		Pointer to BYPASS field name
32		Pointer to RDEXIT routine name
33		Pointer to *CONTD field for RAF
34		Pointer to RAF chain table
35		Pointer to WORKSTN Extension Table (Table 7-3)
36		Pointer to TurboIMAGE Extension Table (Table 7-2)
37		Pointer to control level table or overflow line number (if this is a pinter file)

Table 7-1. The File Table (Continued)

Word	Byte	Meaning
38		Matching Field table pointer or form length (number of lines) if this is a printer file
39 thru 41		Matching or control level field in current record: For control level fields: Word 39, byte 1 = 1 (Level 1 is present) Word 39, byte 2 = 1 (Level 2 is present) Word 39, byte 3 = 1 (Level 3 is present) Word 40, byte 0 = 1 (Level 4 is present) Word 40, byte 1 = 1 (Level 5 is present) Word 40, byte 2 = 1 (Level 6 is present) Word 40, byte 3 = 1 (Level 7 is present) Word 41, byte 0 = 1 (Level 8 is present) Word 41, byte 1 = 1 (Level 9 is present) For matching fields: Word 39, byte 0 = 1 (match fields are present) Word 39, bytes 1-3 = Unused
42	0	ISAM flag: 0 = File opened is not a ISAM file 1 = File opened is an ISAM file
42	1	KSAM flag: 0 = File opened is not a KSAM file 1 = File opened is a KSAM file
42	2	Print file flag
42	3	Other file open flag (equals 1 if this file references another opened file)
43	0	Run-time end-of-file flag (equals 1 when end-of-file detected)
43	1	KSAM file open flag (equals 1 if file currently opened is a KSAM file)
43	2	Record number flag (equals 1 when the first record number is 1)
43	3	OK-to-use flag (equals 1 when the file is not inhibited by U1-U8)
44	0,1	Multiname file usage flag
44	2	Locking flag (equals 1 when locking specified)
45		Binary key for KSAM or TurboIMAGE file
46		Record-identifying indicator
47		Sequence number from Input Specification (columns 15-16)
48		Start of first trailer field
49		Offset from header to current trailer field
50		Trailer field length
51		File parameters
52		File access options
53		Pointer to move-fields routine

Table 7-2. The TurboIMAGE Extension Table

Word	Byte	Meaning
0-2		TurboIMAGE database name
3-4		Password (LEVEL) identification (8 bytes)
5-8		First (ITEM) key name (16 bytes)
9-10		New key file (KEYFL) name
11-14		Data set name (16 bytes)
15-24		TurboIMAGE communication area (10 STATUS words)
25		Open mode: 1 = Read/Write Shared mode 2 = Update Shared mode 3 = Exclusive mode
26		I/O mode: 2 = Serial read 3 = Backward serial read 4 = Directed read 5 = Chained read 6 = Backward chained read 7 = Associative read 8 = Primary associative read 9 = Sequential read 10 = Backward sequential read 11 = Chained sequential read 12 = Backward chained sequential read
27	0	First record flag (blank, zero or 1)
27	1	Duplicate flag (blank or zero)
27	2	Random flag (blank, C or R)
28		Database check (equals 1 if database is open)
		The next seven items (words 29-35) are used with the BUFCHK option:
29		BUFCHK specification
30		CDC response
31		NRC response
32		UPC response
33		Must repoint
34		Dirty here
35		Needs reset
36		TurboIMAGE type: 0 = Non-TurboIMAGE 1 = TurboIMAGE 2 = HP
37		Starting location of key field
38		Dslock duration
39		Generic key length (in bytes)

Table 7-2. The TurboIMAGE Extension Table (Continued)

Word	Byte	Meaning
40		Generic relation: 0 = Equal 1 = Greater than 2 = Greater than or equal to
41		Indicator index for generic relation
42		BUFCHK record number
43		BUFCHK address
44		Unused
45		Unused
46		Pointer to user status array
47		Pointer to current key value
48		Pointer to previous key value
49		Pointer to File Table

Table 7-3. The WORKSTN Extension Table

Word	Byte	Meaning
0		Forms file used
1		Batch file used
2		Trace file used
3		Block mode flag (equals 1 when block mode is in effect)
4		Function key labels flag (equals 1 when function keys are enabled)
5		BREAK key flag (equals 1 when BREAK key enabled)
6		Batch search direction: 0 = Reverse 2 = Forward
7		Auto-read done flag (equals 1 when auto-read is finished)
8		Release WORKSTN flag
9		WORKSTN type: 0 = VPLUS 1 = RSI 2 = CONSOLE
10		Number of forms to download
11		Trace file open number
12		Input event number (less than or equal to 12)
13		Form field number
14		Form field length
15		Activity or event number
16		Previous batch record number
17		Error message display interval
18		Pointer to user status array
19		Pointer to RSI start field
20		Pointer to terminal identification field
21		Pointer to main File Table for WORKSTN
22-25		Saved file name
26-29		Forms file name
30-33		Batch file name
34-37		Trac file name
38-57		Error message array
58 thru 157		VPLUS communications area

Troubleshooting

This section suggests ways to determine the cause of an execution error and also how to expedite the debugging process.

When you have trouble understanding why a particular error occurs, try one or more of the following:

* Request an Error Dump (using column 55 of the Header Specification).
* Organize your test files into a separate group. If your test files are on tape under another group (a production group, for example), you can use RESTORE to load them into your test group.
* If you're testing, try running your program with various files empty. This may be helpful in isolating bugs. To specify that a file exists, but has no data, equate the file to $NULL. For example, to equate the file GLMAST to $NULL, enter the command,

```
:FILE GLMAST=$NULL
```

* Use good programming practices and structured code in your programs.

Chapter 8 Communicating with MPE and Other RPG Programs

RPG has certain unique language features that make it easy to communicate with other RPG programs. It also contains features that give RPG programs access to certain operating system areas and routines. Overall, these features extend the functionality of RPG and allow you to gain more control over the execution of application programs.

Some of the most common ways to use the operating system and to communicate with other RPG programs are described in this chapter. Where appropriate, you are directed to operating system and RPG manuals for additional information.

Using the System Date and Time

There are two ways to retrieve the current date in an RPG program.

You can use the UDATE special field in Calculation and Output Specifications. UDATE is automatically initialized to the current date when a program begins execution (the date is not reinitialized when programs continue execution past midnight).

If you want to retrieve the date as well as the current time, use the TIME (or TIME2) Calculation Specification operation. Since TIME is executed dynamically, the date and time are always current. You may want to use TIME, for example, to stamp transaction records before they are written to tape. TIME gets the time and, optionally, the date and stores them in the Result Field in the format: hhmmssmmddyy (or hhmmssddmmyy in Foreign format). For example, the date 10/29/86 and time 3:13:25 is returned as 031325102986. TIME2 returns the date and time in a 40-character formatted string. The format for this string is shown below along with an example of a date and time that might be returned:

```
day, mon dd, year, hh:mm xM   JULIAN:nnn

WED, OCT 29, 1986,  3:13 AM   JULIAN:304
```

You can specify which parts of the date and time to retrieve, if you wish, by entering their positions in the Factor2 and Result Fields.

Figure 8-1 shows how to use the TIME2 operation to stamp output records with the most current date and time. Only the current time and the day of the month are used in this example.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234

FTERM    O   F      80            $STDLST

1   I            DS
2   I                                      20  27 CLOCK
3   I                                       1   3 DAYWK
4   I                                       1  27 STAMP

    C*  GET TIME JUST BEFORE RECORD OUTPUT
5   C                    TIME21          STAMP
    C                    EXCPT

    OTERM    E
    O                        CLOCK       8
    O                        DAYWK      11
    O                          .
    O                          .
    O                          .
```

Figure 8-1. Getting the System Date and Time

Comments

1 This line begins the data structure that holds the system date and time.

2 This line specifies that the first field in the data structure contains the time. The time is located in positions 20-27 of the date and time string returned by TIME2 (see line 5).

3 This line specifies that the second field in data structure contains the day of the week. The day of the week is located in positions 1-3 of the date and time string returned by TIME2 (see line 5).

4 The line defines the third field in the data structure. STAMP includes all 27 positions of the data structure.

5 This line returns a string containing the system date and time.

Columns 28-32 contain TIME2 to return a formatted date and time string.

Columns 33-42 contain 1 to specify the starting location in the system date and time string where extraction is to begin.

Columns 43-48 name the field, STAMP, where the system date and time string is stored.

Using System UDCs

A User-Defined Command (UDC) is an operating system command saved in a file. UDCs are similar to job files except that they can be executed interactively in session mode.

You can use UDCs to do many things. For instance, you can set up a test UDC that loads an RPG program, then executes it. This test UDC can contain file equations, program parameters, etc. Logon UDCs are also handy. You set them up so that when a user logs onto the system, the UDC is executed automatically. Logon UDCs are used primarily for displaying

application menus to users. They ensure that users execute only the programs that they are authorized to use. UDCs are discussed in length in the operating system commands reference manual.

You can create a UDC file with any standard text processor. Once you create it, you must add it to the system command Directory with the system SETCATALOG command (see your operating system commands reference manual for more information on SETCATALOG). To execute a UDC, type its name at the operating system colon prompt. For example, Figure 8-2 shows a UDC file containing three UDCs: DAILY, MONTHLY and YEARLY. These UDCs produce daily, monthly and year-end reports. To print only the year-end report, type:

```
:YEARLY
```

```
|                                        |
|     DAILY                              |
|     COMMENT  -  DAILY  REPORT          |
|     RUN  RPT10.PROGRAM                 |
|     **                                 |
|     MONTHLY                            |
|     COMMENT  -  MONTH-END  REPORT      |
|     FILE  DETAIL=ALLMONTH              |
|     RUN  RPT10.PROGRAM                 |
|     RESET  DETAIL                      |
|     **                                 |
|     YEARLY                             |
|     COMMENT  -  YEAR-END  REPORT       |
|     STREAM  RPT20.JOB                  |
|     **                                 |
|                                        |
```

Figure 8-2. A Simple UDC

Figure 8-3 expands the UDC shown in Figure 8-2 and makes it more flexible. One UDC, REPORT, replaces the three UDCs in Figure 8-2. REPORT displays a menu of available reports. When the user selects a report (DAILY, MONTH-END or YEAR-END), the UDC validates the choice. If the selection is valid, the UDC starts the appropriate report program.

REPORT uses the logic command, IF, to validate the report choice and start the appropriate report program. It also uses a User-Defined JCW (OPTJCW) for storing the user's report choices. For information on the User-Defined JCWs, see "Communicating File Information" or refer to the *MPE XL Intrinsics Reference Manual*.

```
REPORT !OPT=0
OPTION NOLIST
COMMENT - SET THE OPTJCW ACCORDING TO THE MENU OPTION CHOSEN
COMMENT - NO CHOICE DEFAULTS TO OPTJCW=0 AND MENU WILL DISPLAY
SETJCW OPTJCW=!OPT
IF OPTJCW=0 THEN
     MSG NO OPTION WAS ENTERED...
     MSG VALID OPTIONS ARE:
     MSG >>> 1. DAILY REPORT
     MSG >>> 2. MONTH-END REPORT
     MSG >>> 3. YEAR-END REPORT
ENDIF
IF OPTJCW=1 THEN
     COMMENT - DAILY REPORT
     RUN RPT10.PROGRAM
ENDIF
IF OPTJCW=2 THEN
     COMMENT - MONTH-END REPORT
     FILE DETAIL=ALLMONTH
     RUN RPT10.PROGRAM
     RESET DETAIL
ENDIF
IF OPTJCW=3 THEN
     COMMENT - YEAR-END REPORT
     STREAM RPT20.JOB
ENDIF
**
MSG P1=" " P2=" " P3=" " P4=" " P5=" " P6=" " P7=" " P8=" " P9=" "
OPTION LIST
COMMENT-> !P1 !P2 !P3 !P4 !P5 !P6 !P7 !P8 !P9
**
```

Figure 8-3. A More Complex UDC

Using System Intrinsics

System intrinsics are operating system routines that perform specific tasks. For instance, the PAUSE system intrinsic temporarily stops a program. This intrinsic may be used, for example, to give an operator time to mount a tape. It is often useful and convenient to use intrinsics when you need to perform similar functions in your program. Intrinsics provide a way to gain more control over processing and they simplify a program. Although you cannot use system intrinsics directly from an RPG program, you can use external subroutines to access them. Intrinsics are described thoroughly in the *MPE XL Intrinsics Reference*

Manual.

Figure 8-4 lists a report menu program. It executes the appropriate report program selected by the user. The menu program contains two external subroutines, RUNPGM and COMAND. They are Pascal procedures that call certain system intrinsics directly.

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

H*****************************************************************
H**                                                             **
H**  PROGRAM NAME.... RPG MENU DRIVER                           **
H**                                                             **
H**  REMARKS..... USE PROCESS HANDLING WITH EXTERNAL            **
H**   SUBROUTINES TO RUN PROGRAMS FROM A DISPLAYED MENU.        **
H**                                                             **
H**  PREP COMMAND IS...                                         **
H** :PREP $OLDPASS,DEMO6A.PROGRAM;RL=REALRL.PROGRAM;CAP=IA,PH**
H*****************************************************************
H*
HDUMPRPG
F*-------------------------------------------------------------
F* OPEN TERMINAL AS $STDIN AND $STDLIST FOR INPUT AND OUTPUT
F*-------------------------------------------------------------
FSTDIN   ID  F   1   1            $STDIN
FSTDLIST O   F  79  79            $STDLST
E*-------------------------------------------------------------
E* MESSAGE PROMPT ARRAY - OUTPUT WITH INDEX COUNTER TO $STDLIST
E*-------------------------------------------------------------
E                    CRT     1   9 40
I*-------------------------------------------------------------
I* OPERATOR INPUT TO TYPE MENU CHOICE
I*-------------------------------------------------------------
ISTDIN   NS  01
```

Figure 8-4. An RPG Menu Program

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

I                                              1    10NUMBR
C*-------------------------------------------------------------------
C*                 --- CALCULATION MAINLINE ---
C*-------------------------------------------------------------------
C                     EXSR MSHOW                          SHOW MENU
C                     EXSR MREAD                          READ MENU
C   21                EXSR OPT01
C   22                EXSR OPT02
C   23                EXSR OPT03
C   24                SETON                         LR
C*-------------------------------------------------------------------
C* SET CRT ARRAY INDEX AND OUTPUT MENU WITH CRT,N TO TERMINAL
C*-------------------------------------------------------------------
C           MSHOW     BEGSR
C                     Z-ADD0         N       10
C           SLOOP     TAG
C                     ADD  1         N
C           N         COMP 6                        80  LESS
C   80                EXCPT
C   80                GOTO SLOOP
C                     ENDSR
C*-------------------------------------------------------------------
C* READ ANSWR FIELD FROM TERMINAL AND STORE IN SRCCDO FIELD
C*-------------------------------------------------------------------
C           MREAD     BEGSR
C                     READ STDIN                    HO
C           NUMBR     COMP 1                        21
C           NUMBR     COMP 2                        22
C           NUMBR     COMP 3                        23
C           NUMBR     COMP 4                        24
C                     ENDSR
C*-------------------------------------------------------------------
C* OPT01 - MENU OPTION 1 - RUN RPT10 (DAILY)
C*-------------------------------------------------------------------
C           OPT01     BEGSR
C                     MOVEL"RPT10"   PNAME  26
C                     EXSR RUNSUB
C                     ENDSR
C*-------------------------------------------------------------------
C* OPT02 - MENU OPTION 2 - SET FILE EQUATION AND RUN RPT20 (EOM)
C*-------------------------------------------------------------------
C           OPT02     BEGSR
```

Figure 8-4. An RPG Menu Program (Continued)

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123
-------------------------------------------------------------------

C                    MOVE *BLANK    CWORK  19
C                    MOVEL"FILE DAY"CWORK
C                    MOVE "=MONTH"  CWORK
C                    MOVELCWORK     CBUF   72
C                    EXSR CMDSUB
C                    MOVEL"RPT20"   PNAME
C                    EXSR RUNSUB
C                    ENDSR
C*------------------------------------------------------------
C* OPT03 - MENU OPTION 3 -INITIATE JOB STREAM (EOY)
C*------------------------------------------------------------
C           OPT03    BEGSR
C                    MOVE *BLANK    CWORK
C                    MOVEL"STREAM"  CWORK
C                    MOVE "JRPT30"  CWORK
C                    MOVELCWORK     CBUF
C                    EXSR CMDSUB
C                    ENDSR
C*------------------------------------------------------------
C* RUNSUB - SUBROUTINE TO CALL CREATE-PROCESS INTRINSIC
C*------------------------------------------------------------
C           RUNSUB   BEGSR
C                    MOVE "000"     STATUS  3
1  C                 EXIT RUNPGM
C                    PARM           PNAME
C                    PARM           STATUS
C           STATUS   COMP "000"                    10
C   10               GOTO RUNEND
C*     DISPLAY ERROR IF STATUS NOT ZERO
C                    Z-ADD8         N
C                    SETON                  8581
C                    EXCPT
C                    SETOF                  8581
C           RUNEND   ENDSR
C*------------------------------------------------------------
C* CMDSUB - SUBROUTINE TO CALL THE COMMAND INTRINSIC
C*------------------------------------------------------------
C           CMDSUB   BEGSR
C                    MOVE "000"     STATUS  3
2  C                 EXIT COMAND
C                    PARM           CBUF
C                    PARM           STATUS
```

Figure 8-4. An RPG Menu Program (Continued)

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

C            STATUS    COMP "000"                      10
C   10                 GOTO CMDEND
C*     DISPLAY ERROR IF STATUS NOT ZERO
C                      Z-ADD9         N
C                      SETON                      8582
C                      EXCPT
C                      SETOF                      8582
C            CMDEND    ENDSR
O*-------------------------------------------------------------
O* DISPLAY MENU TO TERMINAL
O*-------------------------------------------------------------
OSTDLIST E        80
O                           CRT,N      40
O        E        85
O                 81       PNAME      26
O                 82       CBUF       72
O                           CRT,N      78
**              "CRT" PROMPT ARRAY
       SAMPLE REPORT MENU
1. REPORT - DAILY
2. REPORT - END-OF-MONTH
3. REPORT - END-OF-YEAR
4. END REPORT MENU
  (ENTER MENU OPTION)

PROGRAM NOT EXECUTED --
COMMAND NOT EXECUTED --
```

Figure 8-4. An RPG Menu Program (Continued)

Comments

1 This line calls an external subroutine, RUNPGM. RUNPGM is a Pascal procedure that directly calls the operating system intrinsic RUN. RUN creates a process for the report program selected.

2 This line calls an external subroutine, COMAND. COMAND is a Pascal procedure that directly executes the selected operating system command (for example, FILE).

Communicating with Other RPG Programs

The remaining sections in this chapter explain how to use certain operating system and RPG facilities to communicate information to other RPG programs. The sections discuss ways to communicate:

* Switches (user indicators U1-U8)

* System file information (retrieved from the operating system LISTF command)
* Data (Local Data Area)

Communicating Switches

When the information you want to pass to other programs or receive from them can be put in the form of yes/no or on/off states, you may want to use user indicators (U1-U8) in your RPG program. User indicators can be saved and passed to other RPG programs by either:

* Saving the switches in a USWITCH file, or
* Saving the switches in the System-Defined JCW

Both of these methods are discussed in the following sections.

Using USWITCH Files. A USWITCH file is a standard text file that contains the settings of the user indicators (U1-U8). This file can be created, read and updated by RPG programs. If you're running in job mode, you do not need to enter the indicator settings in a USWITCH file. You can include them as part of the job file. For more details about using USWITCH files, see the USWITCH sections in the *HP RPG Reference Manual*.

You can read a USWITCH file only, or you can read and update it in a program. The next two sections show how to do this.

Reading a USWITCH File

Figure 8-5 shows how to read the USWITCH file. In this particular example, user indicator U1 signals that the program should terminate immediately. User indicator U2 means that the program can access the master file, GLMAST. These indicators are set by the update program shown in Figure 8-6. They are read by each application program to ensure that they do not execute concurrently with the update program or process the GLMAST file concurrently.

To read a USWITCH file, enter a Header Specification indicating that the USWITCH file is being used. Then use the user indicator(s) in File Description, Input, Calculation and Output Specifications.

```
     1         2         3         4         5         6         7
    67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
    H* THIS PROGRAM RUNS WHEN PROGRAM GL1000.PROGRAM IS NOT RUNNING
1   HDUMPRPG  F

2   FGLMAST   IP  F 256 256           DISC                           U2
    F                       .
    F                       .
    F                       .

3   C   U1                EXSR DISPLY                      TELL OPERATOR
    C   U1                SETON          LR
    C   U1                GOTO END
    C                       .
    C                       .
    C                       .
```

Figure 8-5. Reading User Indicators

Comments

1 This line enables the USWITCH facility.

Column 16 contains F to specify that a USWITCH file is used.

2 This line defines the GLMAST file. It is opened only when user indicator U2 is turned on.

3 This line displays a message indicating that the update program in Figure 8-6 is executing and the program must be run later.

Columns 10-11 contain U1 to display the message only when user indicator U1 is turned on.

Updating a USWITCH File

Figure 8-6 shows how an update program uses the USWITCH file to prevent other programs from running at the same time. When the program starts, it turns on user indicator U1. At the end of the update program, when the last record (LR) indicator is turned on, U1 is turned off and user indicator U2 is turned on.

Other programs (as shown in Figure 8-5) test U1 to determine whether to continue and test U2 to determine whether to process the master file, GLMAST.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------

     H* WRITE SWITCH SETTING RECORD TO USWITCH FILE
     HDUMPRPG

     FGLMAST  UP  F 256 256           DISC
  1  FUSWITCH OC  F  72  72R 1        DISC
     F                                          KLOCK
     F                        .
     F                        .
     F                        .

  2  IUSWITCH NS  02
     I                        .
     I                        .
     I                        .

  3  C                   SETON                   20
     C   20
  4  COR LR      *ZERO    CHAINUSWITCH            HO
     C                        .
     C                        .
     C                        .

  5  OUSWITCH D          20
  6  O                                 17 "USWITCH: 10XXXXXX"
  7  O       T           LR
  8  O                                 17 "USWITCH: 01XXXXXX"
```

Figure 8-6. Setting User Indicators to Prevent Simultaneous Program Execution

Comments

1 This line defines the USWITCH file.

Column 16 specifies that this file is a chained file.

Column 30 indicates that the key field length is 1.

2 This line defines the input USWITCH record.

3 This line (at the beginning of the program) turns on indicator 20. Indicator 20 triggers an update of the USWITCH file (U1 is turned on).

4 This line reads a USWITCH record whenever indicator 20 or the last record indicator (LR) are turned on.

5 This line describes the USWITCH output record that is written when indicator 20 is turned on.

6 This line defines the user indicator settings that are written to the USWITCH file when indicator 20 is turned on.

Columns 45-63 contain "USWITCH: 10XXXXXX" to specify the settings for the user indicators. User indicator U1 is turned on, U2 is turned off and the others remain unchanged.

7 This line defines the USWITCH output record that is written when the LR indicator is turned on.

8 This line defines the user indicator settings that are written to the USWITCH file at the end of the program.

Columns 45-63 contain "USWITCH: 01XXXXXX" to specify the settings for the user indicators. User indicator U1 is turned off, U2 is turned on and the others remain unchanged.

Using the System-Defined Job Control Word (JCW). The System-Defined Job Control Word (JCW) is a two-byte data area that is part of the operating system. It provides a way to pass indicators among operating system routines. Application programs can use the rightmost byte of the JCW for indicators. The rightmost byte provides up to eight User Switches, one switch per bit. When these switches are used in RPG, they are called user indicators (U1-U8).

The System-Defined JCW is easier to use than a USWITCH file. You do not enter File Description and Input Specifications for the JCW and it is updated automatically by RPG. The System-Defined JCW has one disadvantage. Since the system software also uses it, you cannot always count on switch values remaining unaltered. For more details on JCWs, see the *MPE XL Intrinsics Reference Manual*.

Figure 8-7 shows the structure of the System-Defined JCW.

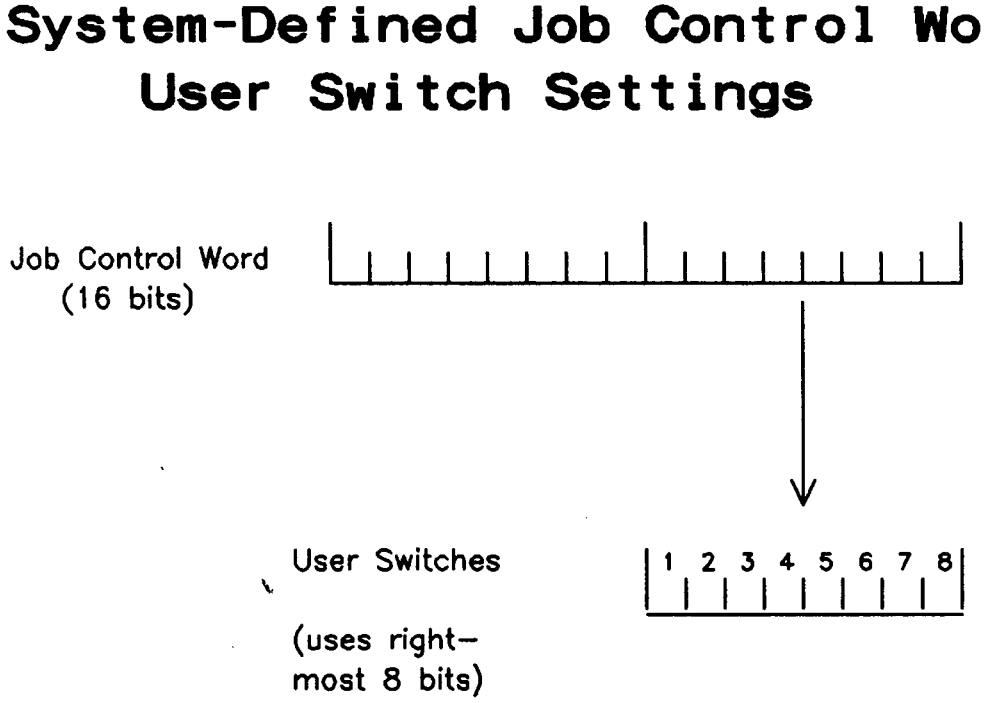


Figure 8-7. The System-Defined Job Control Word (JCW)

User Switches are assigned values depending on their relative bit positions in the Job Control Word (see Figure 8-8). You can test and change User Switch settings using decimal or octal notation. For example, to see if user indicator U1 is turned on, you can test JCW=128 (decimal) or JCW=%200 (octal).

Calculating the JCW

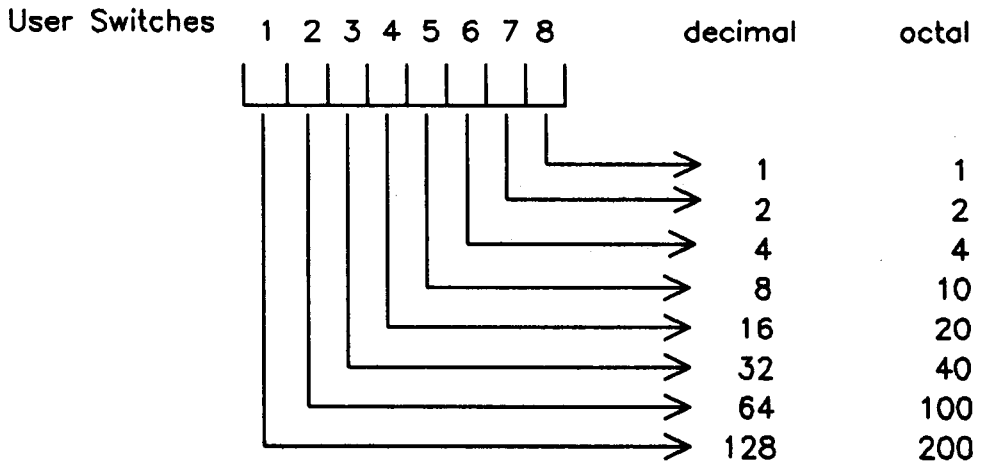


Figure 8-8. Setting User Switches in the System-Defined Job Control Word (JCW)

Figure 8-9 shows a UDC that uses a System-Defined JCW. The UDC initializes the JCW (SETJCW JCW=0), then calls a menu program, RMENU. RMENU prompts the user to select the reports to print. These selections are saved in the JCW. When RMENU ends, the UDC executes the report programs indicated by the User Switches in the JCW.

```
REPORT
SETJCW JCW=0
RUN RMENU.PROGRAM
IF JCW=%200
   COMMENT - DAILY REPORT,  U1 ON
   RUN RPT10.PROGRAM
ENDIF
IF JCW=%100
   COMMENT - MONTH-END REPORT,  U2 ON
   FILE DETAIL=ALLMONTH
   RUN RPT10.PROGRAM
   RESET DETAIL
ENDIF
IF JCW=%40
   COMMENT - YEAR-END REPORT,  U3 ON
   STREAM RPT20.JOB
ENDIF
SETJCW JCW=0
**
-
```

Figure 8-9. Using the System-Defined JCW in a UDC

Portions of the program, RMENU (see Figure 8-9), that set User Switches are shown in Figure 8-10. RMENU prompts the user for report selections and saves the selections in the System-Defined JCW.

```
           1         2         3         4         5         6         7
     678901234567890123456789012345678901234567890123456789012345678901234

  1  H         J

     C                        .
     C                        .
     C                        .
     C*  DISPLAY MENU AND PROMPT FOR OPTION.  OPTION 1 TURNS
     C*  ON INDICATOR 01, OPTION 2 TURNS ON INDICATOR 2, 3=03
  2  C   01                    SETON                         U1    DAILY
  3  C   02                    SETON                         U2    MONTH
  4  C   03                    SETON                         U3    YEAR
     C                        .
     C                        .
     C                        .
```

Figure 8-10. Using the System-Defined JCW

Comments

1 This line specifies where the values for the user indicators originate.

Column 16 contains J to specify that user indicator values come from the System-Defined JCW.

2 This line turns on user indicator U1 when indicator 01 is turned on. (Indicator 01 is turned on when the user requests the DAILY report.)

3 This line turns on user indicator U2 when indicator 02 is turned on. (Indicator 02 is turned on when the user requests the MONTH report.)

4 This line turns on user indicator U3 when indicator 03 is turned on. (Indicator 03 is turned on when the user requests the YEAR report.)

Communicating File Information

This section explains how to use a User-Defined Job Control Word (JCW) to pass file information between RPG programs. For example, you may need to exchange the maximum record count for a file with other programs, so that they will not write beyond the file's limits. You can use a User-Defined JCW to exchange any information, not just file information.

A User-Defined JCW is a 16-bit logical word located within the operating system. You can create and use as many User-Defined JCWs as necessary. When you create them, you assign identifying names to them (the first character in each name must be a letter). User-Defined JCWs are used exclusively by application programs. The operating system software does not use them. Because of this, you may prefer to use them instead of System-Defined JCWs (see the section "Using a System-Defined Job Control Word (JCW)" in this chapter). User-Defined JCWs are discussed in detail in your *MPE XL Intrinsics Reference Manual*.

The next three figures illustrate how to use a User-Defined JCW. The first figure shows a UDC that runs program, SIZER. It then creates a BATCHOUT file based on the record count of an input BATCHIN file computed by SIZER. Then the UDC executes the program, UPDATE. UPDATE uses the maximum record count for BATCHOUT to ensure that it does not write beyond the file's limits.

Two JCWs are used; JCWEOF and JCWLIM. JCWEOF contains the record count of the BATCHIN file. JCWLIM contains the maximum number of records that can be written to the output batch file, BATCHOUT.

```
COMMENT - REDIRECT :LISTF OUTPUT TO THE DISC FILE "LIMITS"
PURGE LIMITS.TEMP
FILE LIMITS;REC=-80,3,F,ASCII;NOCCTL;DEV=DISC;DISC=10
LISTF BATCHIN,2;*LIMITS
RUN SIZER.PROGRAM
COMMENT - BUILD BATCHOUT ACCORDING TO SIZE IN JCWEOF
IF JCWEOF>5000 THEN
     BUILD BATCHOUT;DISC=10000
     ELSE BUILD BATCHOUT;DISC=5000
ENDIF
      .
      .
      .
RUN UPDATE.PROGRAM
-
```

Figure 8-11. Setting a User-Defined JCW - UDC

Figure 8-12 lists segments of the program, SIZER. SIZER updates the User-Defined JCWs, JCWEOF and JCWLIM with the values placed in the LIMITS file (LIMITS is created by the UDC in Figure 8-11.)

```
    1         2         3         4         5         6         7
    6789012345678901234567890123456789012345678901234567890123456789012345678901234
1   FLIMITS   IP   F  80  80
    F                          .
    F                          .
    F                          .

    I*  :LISTF RECORD TYPE WITH RECSIZE 'B' FOR BYTE IN COLUMN 27
2   ILIMITS   NS       22 CB
3   I                                        33  380IEOF
4   I                                        44  490ILIMIT
    I         NS
    I                          .
    I                          .
    I                          .

5   C            IEOF       PUTJW"JCWEOF"                    20
6   C            ILIMIT     PUTJW"JCWLIM"                    20
    C                          .
    C                          .
    C                          .
```

Figure 8-12. Setting a User-Defined JCW - Program SIZER Comments

Comments

1 This line defines the file, LIMITS, created by the UDC.

2 This line describes the input record in the LIMITS file.

Column 27 contains B to specify the record code.

3 This line defines the first field, IEOF.

4 This line defines the second field, ILIMIT.

5 This line creates a User-Defined JCW, JCWEOF, containing the IEOF field information.

Columns 18-27 names the field, IEOF, that is written to JCWEOF.

Columns 28-32 contain PUTJW to update the User-Defined JCW (this operation uses the intrinsic, PUTJCW).

Columns 33-42 name the User-Defined JCW, JCWEOF.

Columns 58-59 contain 20 to turn indicator 20 on when the PUTJW operation is successful.

6 This line creates a User-Defined JCW, JCWLIM, containing the ILIMIT field information.

Columns 18-27 names the field, ILIMIT, that is written to JCWLIM.

Columns 28-32 contain PUTJW to update the User-Defined JCW (this operation uses the intrinsic, PUTJCW).

Columns 33-42 name the User-Defined JCW, JCWLIM.

Columns 58-59 contain 20 to turn indicator 20 on when the PUTJW operation is successful.

Segments of the program, UPDATE, are shown below. UPDATE reads the User-Defined JCWs (JCWEOF and JCWLIM) created in Figure 8-12. UPDATE increments the record count (from JCWEOF) each time a record is written

to BATCHOUT. This number is tested against the file limit in JCWLIM. If there is no more room to write records, the program ends.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
1 C  N99                FNDJW"JCWEOF"  EOF     60 H1
2 C  N99                FNDJW"JCWLIM"  LIMIT   60 H2
  C                     SETON                     99
  C*  INCREMENT EOF AND TEST IT BEFORE OUTPUT
3 C                     ADD  1         EOF
4 C           EOF       COMP LIMIT                20
5 C   20                EXSR WARN
6 C   20                SETON                  LR
  C                       .
  C                       .
  C                       .
```

Figure 8-13. Reading a User-Defined JCW - Program UPDATE

Comments

1 This line reads the User-Defined JCW, JCWEOF.

Columns 9-11 contain N99 to perform this Calculation operation when indicator 99 is turned off (at the start of the program).

Columns 28-32 contain FNDJW to read a User-Defined JCW (this operation uses the FINDJCW intrinsic).

Columns 33-42 contain the name of the User-Defined JCW, JCWEOF.

Columns 43-48 contain the name of the field, EOF, where the JCWEOF information is stored.

2 This line reads the User-Defined JCW, JCWLIM.

Columns 9-11 contain N99 to perform this Calculation operation when indicator 99 is turned off (at the beginning of the program).

Columns 28-32 contain FNDJW to read a User-Defined JCW (this operation uses the FINDJCW intrinsic).

Columns 33-42 contain the name of the User-Defined JCW, JCWLIM.

Columns 43-48 contain the name of the field, LIMIT, where the JCWLIM information is stored.

3 Before writing a record, this line increments the output record count in EOF.

4 This line compares the number of records in the BATCHOUT file with the maximum number of records it holds. The maximum is stored in LIMIT. Indicator 20 is turned on when EOF reaches LIMIT.

5 This line executes a subroutine that displays a warning message.

6 This line turns on the last record (LR) indicator when the maximum record limit is reached (indicator 20 is turned on).

Communicating Data

A Local Data Area File (LDAFILE) is a special RPG file that you can use to pass data to other programs and to receive data from them. You can use an LDA file in a program without entering a File Description Specification for it. An LDA is defined in an Input Specification as a User Data Structure.

RPG loads the LDA data into your program when it starts executing and that data is available at the first cycle (1P) output. When your program ends, the LDA data is written back to the LDA file automatically.

You create an LDA using the RPGINIT utility (see the *RPG Utilities Reference Manual*).

Chapter 5 discusses LDAs in the context of data structures. Refer to Figure 5-12 and Figure 5-13 for an example of how to use LDAs.

Chapter 9 Writing More Efficient RPG Programs

This chapter discusses some of the ways you can increase the efficiency of your RPG programs. More efficient programs execute faster and require fewer system resources and memory.

Establishing and Using Standards

Standards provide the framework for building efficient RPG programs. Programming standards decrease development and maintenance time and improve the legibility of programs. System standards, such as file naming conventions and account and group standards make it easier to build and maintain multi-program applications.

The next six sections in this chapter suggest general RPG programming standards you may want to adopt.

Using Comments

This section contains tips on how to use comments in an RPG program. A general rule of thumb is that a well-documented program is one-third comments.

* Enter comments for all Input and Output record types.
* Enter comments for all Calculation Specification GOTO operations.
* Enter notes in columns 60-74 on important Calculation Specifications.
* Maintain a "Date-Modified" comment line at the beginning of each program. Also develop a scheme to keep track of program modifications. For example, you can use the Sequence Number Field (columns 1-5) to document changes. For lines added on 02/28, enter A0228. For lines that are modified, enter C0228. "Comment out" deleted lines and enter D0228 on them.
* Develop consistent standards and style.

Using Structured Programming Techniques

Structured programs are easier to develop, test and maintain. They are also easier to understand.

* Replace repetitive code segments with subroutines (EXSR).
* Stick to sequential logic in the Calculation Specifications. Try to avoid GOTO operations. Instead, use the combinations: IF-THEN-ELSE, IF-THEN, DO-WHILE, DO-UNTIL.

Standardizing Field Names

Use field names that are:

* Five characters long

  A 5-character name can be prefixed or suffixed. For example, if a last name field is called LNAME, it can be prefixed by T when used in an Output Specification for a Terminal file.

* Meaningful

  Names that relate to the contents or purpose of a field are easier to understand and make programs more readable. Also use similar names for arrays and their associated counters. For example, the counter for array AMT can be A.

* Used consistently in other programs

You can use the $COPY/$INCLUDE statements to get field definitions from a source library (see "Using Source Libraries" in Chapter 6). For RPG programs running under MPE V, fields that are not used in a program increase the size of the data space at both compile-time and run-time.

Standardizing the Use of Indicators

Use this indicator:	For:
01-09	Input Specification records
10	Input Specification flush indicator
11-19	Input Specification fields
20-59	General-purpose Calculation Specification operations
60-69	CHAIN or READ resulting indicators
70-79	External subroutines.
80-89	Exception output
90-98	Work (reusable) indicators
99	First-time indicator

Standardizing File Names

Use this prefix (or suffix):	For this type of file:
B	Batch
C	Catalog
D	Data
I	Input
J	Job
K	KSAM key
N	Notes
P	Executable program
R	RAF
S	Source programs or sorted output
T	Temporary
W	Work

Standardizing MPE Group Names

Use this Group name:	For:
DATA	MPE, KSAM, TurboIMAGE files; operator logon group
DOC	User documentation
FORMS	VPLUS and SIGEDITOR forms
JOB	Job streams
PROGRAM	Executable programs
PUB	UDCs, user notes, temporary and scratch files
SOURCE	Source programs, schemas and program documentation

Entering Efficient Specifications

The next five sections suggest ways to code RPG Specifications that will reduce the size of your program and make it execute faster.

Header Specification

* Use the Error Dump File Name (columns 7-14) to avoid lengthy program dumps to the terminal.

* Use N for the Line # Option (column 20) to decrease code segment size. (Leave this field blank during testing.)

File Description and File Extension Specifications

* When you don't need to process a KSAM file in key order, put a C in column 32 of the File Description Specification. This accesses a file chronologically and does not use the key file. For information on accessing a KSAM file chronologically, see "Reading a KSAM File Chronologically" in Chapter 3.

* Use the DSNAME facility to access a file more than one way, rather than using file equations. DSNAME files share a common file buffer. For information on the DSNAME feature, see "Reading a KSAM File Randomly and Sequentially Using Different Keys" in Chapter 3.

* Use the file blocking guidelines for MPE and KSAM files and TurboIMAGE data bases in this chapter.

Input Specifications

* When files have more than one record type, start with the one that occurs most often.

* Define only those record types in a file that are used in the program.

* Define only those fields in a record that are used in the program.

* Define record identification codes as characters (C in column 26, 33 or 40) rather than zones or digits. This speeds up the processing of those records.

* Only describe a field as numeric when it is used in arithmetic operations or is edited for numeric contents.

* Avoid describing a numeric field as unpacked numeric or binary. RPG does all arithmetic in packed decimal format, so it must first pack non-packed fields.

* Use data structures to reformat fields instead of Calculation Specification MOVE operations.

* Do not define fields that are not used.

Calculation Specifications

* Minimize the number of work fields by reusing them.

* Minimize the number of indicators by reusing them.

* Define a work field with the minimum size possible.

* Use *BLANK and *ZERO to initialize fields instead of work fields containing blanks or zeros.

* Only define a work field as numeric when it is used in an arithmetic operation.

* CLOSE files as soon as you're finished with them. This frees system resources and makes the files available to other programs.

* Use GOTO operations to branch around calculation lines rather than conditioning the calculation lines with indicators.

* In MULT operations, enter the smallest field in the Factor 2 field.

* To zero a field, subtract the field from itself. This is more efficient than using Z-ADD or moving zeros to the field.

* When doing table searches, decide whether the table should be searched sequentially or in a binary fashion. If most of the searches in the table are for a few items, put those items first in the table and search it sequentially (column 34 in the Header Specification is blank). If the table consists of a large number of

items that are searched with about the same frequency, place the items in the table in either ascending or descending sequence and perform a binary search (column 34 in the Header Specification is B). Correctly placing items in a table and selecting the right search method reduces the average time to find an item.

* Minimize the number and length of table entries. This reduces the size of the run-time data segments.

* Get rid of indicators that are not used. They cause warnings to be printed during compilation.

* If a file has only one input record type defined, do not condition Calculation Specification operations with its associated indicator.

* Wherever possible, use subroutines (EXSR) to avoid duplicate code. The execution of subroutines can be conditioned by indicators. Use external subroutines for operations that can be handled more efficiently in other languages (for example, string manipulation).

Output Specifications

* Put the most frequently used Output Specifications first.

* When conditioning output, place the most frequently used indicator first in the OR lines.

* Use record lengths that match the number of characters actually written. For example, if a report requires only 60 characters, enter 60 as the record length rather than 132 (maximum length).

* If terminal output ($STDLST) includes long and short messages, define a file for each type. Terminal output records are padded with blanks before being displayed. Longer record lengths require more memory and transmission time.

Improving Input/Output Performance

You can decrease the time it takes to process disc files by observing a few simple guidelines. These relate to the physical placement of files on disc and their blocking factors. The next four sections discuss the efficiencies to employ whenever you're working with disc files. (Refer to operating system manuals, such as *Accessing Files Programmer's Guide*, for more information on the disc files used with RPG.)

MPE Files

MPE files are sequential and random files. They are not KSAM files or TurboIMAGE databases. When you use MPE files, follow these guidelines:

* Use block lengths that are multiples of 256 bytes.

  This ensures efficient use of input/output buffers.

* If the file is processed sequentially, make the block length as large as possible.

  This reduces the number of physical reads to the disc. Since disc I/O is the primary factor in throughput, choose the block length carefully. For example, you might enter a record length of 80 bytes and a block length of 32 records as follows:

  ```
  :BUILD DFILE;REC=-80,32,F,ASCII;DISC=24000
  ```

  A block length of 32 records requires 750 disc accesses to process the entire file of 24,000 records. A block length of 3 requires 8,000 disc accesses.

* Use address-out sorts when available disc space is limited.

  Address-out sorts produce output files containing 4-byte record addresses. You use these record addresses to retrieve data records in sorted order.

  The RPG utility, XSORT, can be used to create address-out (ADDROUT) files (for detailed information on XSORT, see the *RPG Utilities*

Reference Manual). Figure 9-1 and Figure 9-2 show how to use XSORT and how to process the address-out file in an RPG program. Figure 9-1 lists the job file that performs the sort and runs the RPG program. The input file, SEQ1, is sorted by zip code, last name and first name. Only records whose zip codes equal 95501 and whose Type Codes equal AA are sorted. The XSORT parameters are placed after the !RUN XSORT.PUB.SYS command in the job file and the output (address-out) file is SEQ2.

```
!JOB SORT,MGR.RPG;OUTCLASS=LP,7
!FILE XSORTIN=SEQ1
!FILE XSORTOUT=SEQ2
!RUN XSORT.PUB.SYS
00001HSORTA    36A        3X
000020 C 119 123EQC95501              ZIP OMIT
00003I C 153 154EQCAA                 TYPE INCLUDE
00004FNC 119 128                      ZIPCD
00005FNC  19  34                      LNAME
00006FNC   9  18                      FNAME
!EOD
!SAVE SEQ2
!RUN PROG1.LOAD
!EOJ
-
```

Figure 9-1. Creating an address-out File With XSORT

When XSORT finishes, the job file executes program, PROG1. PROG1 reads the ADDROUT file, SEQ2, created by XSORT and uses its record addresses to access the data records in SEQ1. Figure 9-2 lists the File Description and File Extension Specifications that process these files.

```
     1         2         3         4         5         6         7
    6789012345678901234567890123456789012345678901234567890123456789012345678901234
    ████████████████████████████████████████████████████████████████████████████████

1   FSEQ1    IP  F 200 200R  I        DISC
2   FSEQ2    IR  F   4   4  4 T      EDISC

3   E    SEQ2    SEQ1

                .
                .
                .
```

Figure 9-2. Using an address-out File - Program PROG1

Comments

1 This line defines the input file, SEQ1. This is the data file that was input to XSORT.

Column 31 contains I to specify that this file is processed by relative record number.

2 This line defines the ADDROUT file, SEQ2, produced by XSORT.

Column 16 contains R to specify that this file is a Record Address (address-out) file.

Column 32 contains T to specify that this is an address-out file.

3 This line identifies SEQ2 as the address-out file containing addresses for records in SEQ1.

KSAM Files

When you use KSAM files, follow these guidelines:

* Try to use no more than two keys.

  When you use more than two, system performance degrades as the file is updated. See the *KSAM/3000 Reference Manual* for details on using KSAM files efficiently.

* When you do not need to access records in order by key, read the file chronologically. See "Reading a KSAM File Chronologically" in Chapter 3.

* When accessing a file sequentially, use large block lengths.

* When accessing a file randomly, use small block lengths.

* When appropriate, temporarily override a file's block length via the BUF parameter of the operating system FILE command.

  If a file was created with a large block length and you are processing it randomly, enter 1 for the BUF parameter. Conversely, if a file was created with a small block length and you need to process it sequentially, enter a large number for the BUF parameter. The following FILE command overrides 2, which is the default number of buffers, and requests that 4 buffers be used for the file, DFILE. The program, GL050, is executed next. When is processes DFILE, 4 buffers are used. Finally, the number of buffers is reset to 2 by the RESET command:

```
:FILE DFILE;BUF=4
:RUN GL050
:RESET DFILE
```

* For systems with two or more discs, allocate the KSAM key file to one disc and the data file to the other.

  The following example assigns the KSAM data file DFILE to device DATADISC (a symbolic name for disc drive 1). The KSAM key file is assigned to device KEYDISC (a symbolic name for disc drive 2).

```
:RUN KSAMUTIL.PUB.SYS
>BUILD DFILE;DEV=DATADISC;BUILD KEYFILE;KEYDEV=KEYDISC;....
```

* Use multiple file extents to optimize disc space.

  When you divide a file into extents, you specify how many equal parts it can be divided into when stored on disc. Extents enable the system software to allocate disc space as it is needed. Extents also help prevent file overflow errors. For large files, use a large number of extents. The following BUILD command divides the file GLDETAIL into 24 extents. It also specifies that the file contains a maximum of 24,000 records.

```
:BUILD GLDETAIL;DISC=24000,24;....
```

* Use address-out sorts when available disc space is limited.

XSORT is an RPG utility sort that can be used to produce address-out files. See "Reading a KSAM File Sequentially by a Non-Key Field" under "Reading a KSAM File Sequentially" in Chapter 3 for an example of how to use XSORT with a KSAM file.

TurboIMAGE Databases

When you use TurboIMAGE databases, follow these guidelines:

* When desirable, change the block length of a data set via the $CONTROL schema statement (for details about $CONTROL, see the *TurboIMAGE/XL Database Management System* manual).

  When you create a database using TurboIMAGE, it automatically optimizes file blocking. In some instances, you may want to change those block lengths. For example, if you're using a small data set to validate account numbers, you may want to read all of the account numbers into memory at one time (one block).

  There are two steps to changing the block length of an TurboIMAGE data set. First, you create the database letting TurboIMAGE compute the block lengths. Second, using the information computed by TurboIMAGE, you calculate a new block length. And finally, you recreate the database using your new block length figure. For example, assume that the M-SOURCE data set (see the schema for this data set in Figure 3-23) contains 29 records and all 29 records must be read into memory to validate source codes. The database (MARKET) containing this data set must first be created using the following $CONTROL statement. This statement directs TurboIMAGE to compute the optimum blocks lengths automatically:

  ```
  $CONTROL TABLE,LIST
  ```

  Once the database is created, TurboIMAGE produces a schema listing that gives the computed block lengths. Figure 9-3 shows that the computed block length for the M-SOURCE data set is 506.

```
 DATA SET          TYPE  FLD  PT  ENTR  MED   CAPACITY   BLK BLK  DISC
   NAME                  CNT  CT  LGTH  REC              FAC LGTH SPACE

A-LAST-NAME        A     1    1   8     18    97         28  506  20
A-ACCOUNT-NO       A     1    1   3     13    97         36  471  16
M-SOURCE           M     5    0   22    27    29         16  433  12
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
D-ACCOUNTS         D    16    2   95    103   100        4   413  104
                                TOTAL DISC SECTORS INCLUDING ROOT: 164
NUMBER OF ERROR MESSAGES: 0
ITEM NAME COUNT: 20     DATA SET COUNT:4
ROOT LENGTH:574     BUFFER LENGTH: 506     TRAILER LENGTH: 256
_
```

Figure 9-3. The Block Length in an TurboIMAGE Schema Listing

The computed block length of 506 accommodates only 16 (BLK FAC) records from the M-SOURCE data set. Since the program must read 29 records, the computed block length must be changed. To do this, calculate the new block length as follows:

```
M-SOURCE MED REC * M-SOURCE CAPACITY + 2
     (27)        *       (29)        + 2 = 785
```

Now, recreate the MARKET database using this $CONTROL statement:

```
$CONTROL TABLE,BLOCKMAX=785,LIST
```

NOTE Using BLOCKMAX changes the block length for all data sets in the database.

* For small master data sets that you want to access sequentially by key, read them into an array, then use SORTA to sequence the array.

  Master data sets cannot be accessed sequentially by key. You must sort the data set to do this. Figure 9-4 shows how to put the M-SOURCE data set in sequence by SRCCD (SOURCE-CODE). The program reads each M-SOURCE record saving the SOURCE-CODE in the SRC array. When all records have been stored, the SORTA operation sorts the array. Finally, the program uses the sorted array to output the ordered SOURCE-CODES.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

     F*  ACCESS MODE 6 - SHARED READ ACCESS; I/O MODE 2 - SERIAL READ
     FMSOURCE IP  F  20  20
     F                                              KIMAGE MARKET62
     F                                              KLEVEL READER
     F                                              KDSNAMEM-SOURCE

1    E                    SRC        29  4              SRC CODES

     IMSOURCE NS  01
2    I                                        1   4  SRCCD

     C   01                ADD  1         I       20       INDEX COUNTER
3    C   01                MOVE SRCCD     SRC,I            FILL ARRAY
4    CLR                   SORTASRC                        ASCENDING SORT
     CLR                   SUB I          I                RESET I TO 0
5    CLR          ELOOP    TAG                             EXCPT OUTPUT
     CLR                   ADD  1         I                IN SORTED ORDER
     CLR                   EXCPT
     CLR          I        COMP 29                    80  LESS
     CLR 80                GOTO ELOOP

     O                          .
     O                          .
     O                          .
6    O          E      80
     O                     SRC,I     20
     O                          .
     O                          .
     O                          .
```

Figure 9-4. Using SORTA to Sort an TurboIMAGE Master Data Set

Comments

1 This line defines the array, SRC, that contains the SOURCE-CODEs from the M-SOURCE data set.

Column 45 is blank to indicate that the array elements are in ascending sequence (D indicates descending).

2 This line defines the SOURCE-CODE field, SRCCD, in the M-SOURCE data set.

3 This line fills the SRC array with SOURCE-CODEs from the M-SOURCE data set.

4 This line sorts the entries in the SRC array when all of the SOURCE-CODEs have been retrieved from the M-SOURCE data set.

5 This is the first line in the loop that accesses each (sorted) SOURCE-CODE in the SRC array.

6 This line defines the exception output record written by the loop starting on line 5. SOURCE-CODEs are written in ascending sequence, one per output record.

Releasing Printer Files

When you want to print data from a printer file before a program finishes, release the data using column 16 of the Output Specification (enter R into column 16 of the last Output Specification line you want to print).

Normally, when you use a printer file, information accumulates during program execution and is actually printed when the program ends. There are occasions, however, when you may want to print data immediately, but continue with the program. For example, when customers deposit money at a bank, they get deposit receipts immediately. The online program that processes deposits must "release" the printer file in order to print these receipts.

Figure 9-5 shows how to release a printer file. The last line to be printed (line 3) contains an R in column 16.

```
     1         2         3         4         5         6         7
    67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
1   FONEPAGE O   F  80  80            SLOWLP

    O                                  .
    O                                  .
    O                                  .
2   OONEPAGE H
    O*  THE LAST OUTPUT LINE RELEASES THE PRINTER FILE
3   O       DR
4   O                          TOTAL J   70
```

Figure 9-5. Releasing a Printer File

Comments

1 This line defines the printer file, ONEPAGE.

Columns 40-46 contain SLOWLP which is a symbolic name for the printer device.

2 This line starts the output definition of the ONEPAGE record.

3 This line starts the detail print record.

Column 16 is R to specify that once this detail record is processed, the printer file will be released, reopened and the program will continue.

4 This line defines the last detail field in the printer record.

Appendix A Migrating to HP RPG

This chapter gives information about converting existing RPG programs to run under the MPE XL operating system.

If your RPG programs are currently running on an IBM system, read the section that follows titled "Migrating from IBM RPG". If your RPG programs are currently running under the HP MPE V operating system, read the section that follows titled "Migrating from MPE V RPG".

Migrating from IBM RPG

Converting IBM S/34 and S/36 RPG programs to run under MPE XL consists of four steps. First, you convert the programs on a system that runs MPE V by using the HP product, TRANSFORM. Next, you manually convert certain features that are not converted by TRANSFORM. Then, you recompile the programs under MPE V. Finally, you must STORE the programs onto tape, then RESTORE them onto the system that runs MPE XL. These steps are discussed in more detail below:

Step:	Description:
1.	TRANSFORM performs the following primary tasks when converting IBM RPG S/34 and S/36 programs to run on an MPE V system: * It translates IBM RPG source programs to HP RPG. * It generates HP RPG source files for IBM DFU files. * It translates IBM procedure files into PROCMON procedure files. * It translates IBM message files into a format compatible with the HP message file produced by MAKECAT. * It transfers data files to the HP 3000, converting sequential and ISAM files to MPE and KSAM file formats. For complete information about TRANSFORM, see the *TRANSFORM/3000 Reference Manual*.
2.	Since HP RPG is approximately 95% compatible with IBM RPG, TRANSFORM can convert most features automatically. However, for those DFU features that are not converted, you must manually modify the generated source programs to include those features. The *TRANSFORM/3000 Reference Manual* lists the features that you must convert manually.
3.	Compile the HP RPG programs generated by TRANSFORM (compile them on the system running MPE V).
4.	STORE the compiled programs onto tape, and then RESTORE the files onto the system running MPE XL. (See the instructions in the MPE XL migration guides for information on how to do this.) HP RPG programs run under MPE XL in compatibility mode only.

RPG Features That Are New

HP RPG extends the features found in IBM System/3 and System/360 DOS RPG II to include:

External Subroutine Call Parameters

You can pass parameters with an EXIT Calculation Specification operation (the parameters are identified by PARM operation(s) immediately following EXIT). EXIT makes it easier to use subroutines written in Business BASIC, C, Pascal, COBOL and FORTRAN.

Run-Time Error Options

RPG provides three methods for handling run-time errors:

1. Using the Header Specification to specify whether to ignore the error or whether to abort the program.
2. Letting the operator determine, at run-time, how to handle the error.
3. Using Calculation Specification to handle the error. (this method provides an individualized way of handling the error.)

Cross-Reference Listing

The Cross-Reference listing shows all line references to file names, indicators, and field names. You must specifically request this listing using the Header Specification or the MAP option of the $CONTROL compiler subsystem command.

EBCDIC/ASCII Translation

You can have HP RPG automatically generate file translation tables for EBCDIC-to-ASCII, EBCDIK to JIS, or ASCII-to-EBCDIC, JIS to EBCDIK conversions. Or, you can use the EBCDIC or EBCDIK alternate collating sequence.

Partial Field Translation

You can have HP RPG translate just alphanumeric and unpacked numeric fields, leaving packed numeric and binary fields unchanged.

Combined Input/Output (Terminal) File

HP RPG lets you use a terminal as a single file. This allows both read and write operations for the file.

Calculation Indicator Repetition

HP RPG lets you enter conditioning indicators once in Calculation Specifications and repeat them on successive lines.

Compile-Time Tables/Arrays on Separate Disc Files

If you're using a compile-time table or array in more than one program, select one program in which to define it, then include an Array/Table File Name Specification (A) in each of the remaining programs. The table or array is saved on disc and each Array/Table File Name Specification references that disc file.

Structured Programming Constructs

The following Calculation Specification operations let you use standard structured programming techniques: DO-WHILE, DO-UNTIL and IF-THEN-ELSE.

EXCPT Group Names

You can name a group of Output Specifications with the EXCPT Calculation Specification operation.

Full Procedural Files

The READE Calculation Specification operation reads the next sequential record in a demand or full procedural file whose key matches a specified key field. The READP operations reads the previous record for that key field.

RPG Screen Interface (RSI)

RSI lets you use screen (terminal) files in an RPG program. RSI forms processing provides:

* Full screen capability using standard RPG specifications.
* Lets you use message files and dynamically change screen attributes.
* Generates an RSI forms file from the file's Input Specifications (RSI CONSOLE files).

RPG Features That Are Different

The following HP RPG features are different from IBM RPG:

Printer Files

For programs that use carriage control tape channels other than Channel 1 for printer files, add Line Counter Specifications to equate each channel to a particular line number. Otherwise, the compiler equates Channel 1 to line 6 and the overflow line to line 60. Also, Channels 2-12 are equated to the line numbers obtained by multiplying the channel numbers by 5.

Edit Words

HP RPG handles edit words as follows:

* All blanks in edit words are replaceable characters. (To print the blank character, included an ampersand (&) in the edit word.)
* Constants are allowed to the right of an edit word.
* The floating dollar sign is not a replaceable character.
* Fields containing all zeros are positive.
* The edit field can be smaller or larger than the number of replaceable characters in the edit word. Extra characters are truncated or leading zeros are added.
* All constants that follow a significant digit are printed (except for a minus or credit sign following a positive number).

Differences in Character Codes

HP RPG uses the American Standard Code for Information Interchange (ASCII) Character Set/Collating Sequence. If your system has Katakana characters installed, you use the Japanese Industrial Standard (JIS). IBM System/3 and System 360 use Extended Binary Coded Decimal Interchange Code (EBCDIC). In Japan, this is EBCDIK. The items below summarize the differences in these character sets:

* Alphabetic characters are higher in the ASCII collating sequence than numeric characters. This affects compare and matching field operations. You can use the Header Specification (column 26) to generate an EBCDIC or EBCDIK alternate collating sequence table. Also, you can use columns 54-59 in the File Description Continuation line to generate file translation tables for EBCDIC-TO-ASCII (EBCDIK to JIS) or ASCII-to-EBCDIC (JIS to EBCDIK) conversion.
* You must convert all existing translation tables (including alternate collating sequence and file translation tables) to ASCII (JIS) equivalents.
* Move Zone operations using ASCII (JIS) may yield results that are different from the same operations using EBCDIC or EBCDIK (these differences involve special characters).

Device Class Names

HP device class names have no established values except for a few reserved names such as WORKSTN, STDIN and STDLIST. They are defined then the operating system is installed. Therefore, HP RPG accepts all values for device class names. When entering a device class name in the File Description Specification, be sure to use the same name assigned during system generation.

Rewind Operations

HP RPG does not support all tape-rewinding operations, since the operating system performs these tape operations.

Quotation Marks

Because other HP languages use the double quotation mark as the delimiter for constants and edit words, HP RPG also uses them for the same purpose. To continue to use the single quotation mark as the delimiter character for constants and edit words, use the $CONTROL compiler subsystem command with the QUOTE= parameter. Compiler subsystem commands are discussed in the *HP RPG Reference Manual*.

File and Program Names

File and program names must begin with a letter (A-Z), followed by letters (A-Z) or digits (0-9). File names may contain up to eight characters. If the file is in a different group or account, you must enter a :FILE command to identify its location. Append the group and account to the file name. For instance, to qualify the file named FILENAME, enter FILENAME.GROUPNAME.ACCOUNTNAME. See the *MPE XL General User's Reference Manual* for more information on qualifying file names. Program names may contain up to six characters.

RPG Features That Are Not Supported

HP RPG does not support the following IBM RPG features:

Sterling Notation

The Sterling currency (pounds, shillings, pence) specifications (Header, Input, and Output) must be changed.

Telecommunications

HP RPG does not currently support telecommunications specifications.

ULABL Operation

RPG does not support ULABL calculation operations that make external subroutine fields accessible to an RPG program. HP RPG does, however, support the RLABL and PARM operations. RLABL makes fields, tables, arrays, and indicators in RPG programs available to external subroutines. PARM passes parameters to and from external subroutines.

Card Reader/Punch/Interpreter

HP RPG does not support punched card devices. You must manually convert code related to these devices.

Header Specification Features

The following Header Specification features are not supported:

* Variable memory sizes during compilation and execution.
* Destination device of a compiled object program.
* Inquiry request option (to allow/disallow interruption and roll-out of a running program, followed by roll-in of a new program).
* Normal halt bypass when the object program transmits an unrecognizable character to an output device.
* Sharing of a single output area by all disc files.

Migrating from MPE V RPG

This section lists the features that are not supported when you compile and run a program under MPE XL. To run RPG programs in compatibility mode under MPE XL, STORE them onto tape then RESTORE them onto the MPE XL system. (See the MPE XL migration guides for complete information about how to do this.)

To convert RPG V programs to RPG XL, first STORE them onto tape. Then make any changes to the program to reflect the features that are no longer supported (see the next section). Finally, recompile the programs

using the RPG XL compiler.

RPG MPE V Features That Are Not Supported by RPG XL

Certain RPG features that are supported under MPE V are not supported under MPE XL. They are:

* KSAM record-level locking

  If a program contains KSAM record-level locking, RPG XL emits a warning and defaults to file locking.

* ISAM simulation using TurboIMAGE

  This facility provided indexed sequential file access capability before KSAM was available. You must manually convert ISAM simulation code to other file access methods before compiling under MPE XL.

* Card Reader/Punch Options

  Punched card devices are no longer supported. You must manually convert code related to these devices before compiling under MPE XL.

* Automatic Program Segmentation

  HP RPG does not segment object programs.

* Record Number Conversions

  The RPGCV, ERPGC and EXTCV Calculation Specification operations are not available.

* $EDIT Compiler Subsystem Command

  The $EDIT compiler subsystem command is not available in RPG XL. RPG XL will emit a warning and ignore the record.

* $CONTROL Compiler System Command Options

  Certain $CONTROL compiler system command options are not supported by RPG XL. RPG XL will emit a warning message when an unsupported command option is encountered and will ignore the option. The unsupported command options are KSAM, OPT1, OPT2, SEG= and USLINIT.

  For programs that contain these options, run the RPGCONV conversion utility. RPGCONV removes the options and produces an updated source program. To run RPGCONV, enter the following command using the appropriated filenames,

  :RPGCONV *old_source_prog new_source_prog*

* External Subroutines

  External subroutines called from RPG XL must be written in a language that can be compiled in native mode (HP C or HP Pascal, for example). Refer to the *HP RPG/XL Reference Manual* for information on linking these procedures into your program.

* User-ISAM

  This feature allowed users to write their own ISAM routine and place place it in the RPG library. User-ISAM is not available in MPE XL.