SPERRY UNIVAC
COMPUTER SYSTEMS

PUBLICATIONS RELEASE

Operating System/3 (OS/3)

Information Management System (IMS) Action Programming in COBOL and Basic Assembly Language (BAL)

User Guide

UP-9207

This Library Memo announces the release and availability of "SPERRY UNIVAC® Operating System/3 (OS/3) Information Management System (IMS) Action Programming in COBOL and Basic Assembly Language (BAL) User Guide", UP-9207.

The Information Management System (IMS) Action Programming in COBOL and Basic Assembly Language (BAL) User Guide is one of five books replacing the IMS 90 Applications User Guide/Programmer Reference, UP-8614 Rev. 1. Other manuals replacing UP-8614 are:

— IMS Concepts and Facilities, UP-9205

— IMS Action Programming in RPG II User Guide, UP-9206

— IMS Terminal Users Guide, UP-9208

— IMS Data Definition and UNIQUE User Guide, UP-9209

This manual describes and illustrates how to write COBOL and basic assembly language action programs. It is presented in six parts as follows:

1. INTRODUCTION

   Section 1. Transaction Processing in the IMS Environment

2. BASIC IMS ACTION PROGRAMMING

   Section 2. General Rules for Coding Action Programs

   Section 3. Communicating with IMS

   Section 4. Receiving Input Messages

   Section 5. Processing Data Files

   Section 6. Sending Output Messages

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ and MZ	Mailing Lists A00, A07, A08, B00, B07, 18, 18U, 19, 19U, 20, 20U, 21, 21U, 28U, 29U, 75, 75U, 76, and 76U (Cover and 585 pages)	Library Memo for UP-9207 RELEASE DATE: September, 1982

The complete titles and ordering numbers of the books that form the IMS library are:

- Information Management System (IMS) System Support Functions User Guide, UP-8364, Rev. 7
- Information Management System (IMS) Concepts and Facilities, UP-9205
- Information Management System (IMS) Action Programming in RPG II User Guide, UP-9206
- Information Management System (IMS) Action Programming in COBOL and Basic Assembly Language (BAL) User Guide, UP-9207
- Information Management System (IMS) Terminal Users Guide, UP-9208
- Information Management System (IMS) Data Definition and UNIQUE User Guide, UP-9209
- IMS/DMS Interface User Guide, UP-8748, Rev. 1

Destruction Notice: This release supersedes and replaces "SPERRY UNIVAC Operating System/3 (OS/3) Information Management System 90 (IMS 90) Applications User Guide/Programmer Reference" UP-8614 Rev. 1, released on Library Memo dated October 1980. Also destroyed are Updating Package A, UP-8614 Rev. 1–A, released on Library Memo dated September, 1981 and Updating Package B, UP-8614 Rev. 1–B, released on Library Memo dated December, 1981. Please destroy all copies of UP-8614 Rev. 1, UP-8614 Rev. 1–A, UP-8614 Rev. 1–B, and their Library Memos.

Additional copies may be ordered by your local Sperry Univac representative.

Information Management System (IMS)

Action Programming in COBOL and Basic Assembly Language (BAL)

OS/3

User Guide

H

SPERRY UNIVAC

UP-9207

This document was prepared by Systems Publications using the SPERRY UNIVAC UTS 400 Text Editor. It was printed and distributed by the Customer Information Distribution Center (CIDC), 555 Henderson Rd., King of Prussia, Pa., 19406.

PAGE STATUS SUMMARY

ISSUE: UP-9207
RELEASE LEVEL: 8.0

Part/Section	Page Number	Update Level
Cover/Disclaimer		
PSS	1	
Acknowledgment	1	
Preface	1 thru 5	
Contents	1 thru 15	
PART 1	Title Page	
1	1 thru 14	
PART 2	Title Page	
2	1 thru 15	
3	1 thru 41	
4	1 thru 19	
5	1 thru 62	
6	1 thru 52	
PART 3	Title Page	
7	1 thru 30	
8	1 thru 10	
9	1 thru 16	
10	1 thru 14	
PART 4	Title Page	
11	1 thru 12	
PART 5	Title Page	
12	1 thru 51	
PART 6	Title Page	
Appendix A	1, 2	
Appendix B	1 thru 64	
Appendix C	1 thru 70	
Appendix D	1 thru 7	
Appendix E	1 thru 25	
Appendix F	1 thru 21	
Appendix G	1 thru 8	
Index	1 thru 22	
User Comment Sheet		

All the technical changes are denoted by an arrow (➛) in the margin. A downward pointing arrow (↓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (↑) is found. A horizontal arrow (➛) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

Acknowledgment

We are indebted to the many systems analysts and staff members of Sperry Univac branch offices and customer organizations who helped us develop the OS/3 IMS library. They gave us suggestions, answered numerous questions, reviewed the manuals, and provided us with "real-life" programming examples. The customer organizations assisting us include:

- Gay and Taylor Insurance Adjustors, Winston-Salem, NC
- Penn Ventilator Company, Philadelphia, PA
- Victor Valley Community College District, Victorville, CA

The Sperry Univac organizations assisting us include:

- Los Angeles Access Center, Customer Support Services, Los Angeles, CA
- Charlotte Commercial Branch, Raleigh Office, Raleigh, NC
- Charlotte Commercial Branch, Greensboro Office, Greensboro, NC
- Minneapolis Marketing Branch, Minneapolis, MN
- Wellesley General Branch, Wellesley, MA
- Philadelphia Manufacturing Branch, Wayne, PA
- Des Moines Marketing Branch, West Des Moines, IA
- System 80 Benchmark and Demonstration Services, Blue Bell, PA

Preface

This manual is one of a series designed to instruct and guide you in using the SPERRY UNIVAC Information Management System (IMS) for Operating System/3 (OS/3). It describes all aspects of writing action programs in COBOL and Basic Assembly Language (BAL).

Before you start writing action programs, you should understand basic IMS concepts as described in the information management system (IMS) concepts and facilities, UP-9205 (current version). You should also be able to code standard COBOL or BAL programs. For more information on programming in these two languages, see the current versions of:

- Extended COBOL supplementary reference, UP-8059
- 1974 American National Standard COBOL programmer reference, UP-8613
- Assembler programmer reference, UP-8227

Information in this manual is divided into six parts:

PART 1. INTRODUCTION

- Section 1. Transaction Processing in the IMS Environment

  Introduces COBOL and BAL programmers to action programs and their interface with IMS. Also previews actions, transaction structures, action program termination, succession, and single-thread and multithread environments.

PART 2. BASIC IMS ACTION PROGRAMMING

- Section 2. General Rules for Coding Action Programs

  Discusses COBOL and BAL action program structures and compares them to regular COBOL and BAL program structures. Describes the activation record, its contents, structure, and use.

- Section 3. Communicating with IMS

  Provides a more detailed description of the COBOL and BAL program information blocks including formats, contents, and use.

- Section 4. Receiving Input Messages

  Describes the input message area including the formats, contents, and use of the input message control header format for COBOL and BAL programs and the description of input message text.

- Section 5. Processing Data Files

  Tells how to access and update data files.

- Section 6. Sending Output Messages

  Covers all aspects of output messages including the formats, contents, and use of the output message control header for COBOL and BAL programs; the use of the SEND function for multiple output or message switching; the use of a work area for output messages; continuous output; and, output-for-input queueing.

PART 3. USING IMS SPECIAL FEATURES

- Section 7. Using Screen Format Services to Format Messages

  Discusses and shows examples of how to display a screen format, display a replenish screen or error format; handle error returns; receive formatted input in a successor program; display a screen format on an auxiliary device; and use screen formats in a distributed data processing environment.

- Section 8. Calling Subprograms from Action Programs

  Describes how to call subprograms from COBOL or BAL action programs and illustrates the use of a subprogram.

- Section 9. Action Programming in a Distributed Data Processing Environment

  Presents basic distributed data processing terminology, defines and illustrates directory, operator, and action program routing of transactions, and describes how to initiate a remote transaction and how to process a transaction initiated by a remote system.

- Section 10. Additional Special Features

  Describes the downline load feature and how to write your own downline load program. Also describes how to disconnect a single-station dial-in line from an action program and how to initiate batch jobs from your action program using the RUN function.

PART 4. PREPARING ACTION PROGRAMS FOR EXECUTION

- Section 11. Compiling, Linking, and Storing Action Programs

  Provides control streams needed to compile and link your action programs and describes how to store them in load libraries.

PART 5. DUMP ANALYSIS

- Section 12. Debugging Action Programs

  Discusses all portions of termination and CALL 'SNAP' dump and provides examples and a step-by-step explanation of how to interpret them.

PART 6. APPENDIXES

- Appendix A. Statement Conventions

  Describes the format conventions used in this manual.

- Appendix B. COBOL Action Programming Examples

  Contains complete compiler listings with accompanying flowcharts of sample COBOL action programs discussed throughout the manual. Examples include simple and dialog transactions, external and immediate internal succession, screen format services, sending a message to another terminal, output-for-input queueing, and continuous output.

- Appendix C. Basic Assembly Language (BAL) Action Programming Examples

  Contains complete compiler listing with accompanying flowcharts of sample BAL action programs discussed throughout the manual.

- Appendix D. Status and Detailed Status Codes

  Provides status codes and detailed status codes returned after execution of function calls issued by action programs.

- Appendix E. Generating Edit Tables

  Discusses the edit table generator including coding rules, parameter values that describe the edit table, edit table execution, and error processing. Shows how input messages entered at the terminal are edited. Includes a sample action program that uses an edit table.

- Appendix F. Device Independent Control Expressions and Field Control Characters

  Explains device independent control expressions (DICE), their values, interpretation, how to create them via the DICE macroinstructions, and when to use them.

- Appendix G. Differences Between Extended COBOL and 1974 American National Standard COBOL

  Describes the minor differences between using the extended COBOL and 1974 COBOL compilers to compile action programs.

As one of a series, this manual is designed to guide you in programming and using the OS/3 information management system. Depending on your need, you should also refer to the current versions of other manuals in the series. Complete manual names, their ordering numbers, and a general description of their contents and use are as follows:

- Information management system (IMS) concepts and facilities, UP-9205

  Describes the basic concepts of IMS and the facilities that IMS offers.

- Information management system (IMS) system support functions user guide, UP-8364

  Describes the procedures to generate, initiate, and recover an online IMS system.

- Information management system (IMS) action programming in RPG II user guide, UP-9206

  Describes how to write action programs in RPG II, with extensive examples.

- Information management system (IMS) data definition and UNIQUE user guide, UP-9209

  Describes data definitions for use with the uniform inquiry update element (UNIQUE) or your action programs and explains how to use UNIQUE.

- Information management system (IMS) terminal users guide, UP-9208

  Describes terminal operating procedures, standard and master terminal commands, and special purpose IMS transaction codes. Also includes UNIQUE command formats with brief descriptions. The manual is in easel format for ease of use at the terminal.

- IMS/DMS interface user guide, UP-8748

  Describes how to access a data base management system (DMS) data base from IMS.

Contents

TABLES

PART 1

INTRODUCTION

1. Transaction Processing in the IMS Environment

1.1. INTRODUCING IMS

The SPERRY UNIVAC Information Management System (IMS) is an interactive, transaction-oriented file processing system. It is interactive because it carries on a conversation with the terminal operator; it is transaction-oriented because for each input message, the terminal operator receives a response or output message. In this way, operators are constantly informed of the results of their inquiries.

1.2. INTERACTING WITH IMS

Action programs process messages

Application programs, called action programs, interact with IMS to process input messages from terminals, perform file retrieval or updating functions, and create output messages.

Languages used – BAL, COBOL, RPG II

You can write action programs in RPG II, COBOL, or basic assembly language (BAL). IMS also provides a set of action programs called the uniform inquiry update element (UNIQUE) that performs file retrieval and updating functions through the use of commands from the terminal.

Purpose of this manual

This manual tells you how to write action programs in COBOL and basic assembly language (BAL). Action programs are similar to standard COBOL and BAL programs, but must follow specific rules because they operate under the control of IMS.

Read IMS concepts and facilities manual first

Before reading further, be sure you understand IMS concepts. They are described in the IMS concepts and facilities user guide, UP-9205 (current version). You should also be able to code standard COBOL or BAL programs. For more information on programming in these two languages, see the current versions of:

Programming language documentation

- Extended COBOL programmer reference, UP-8059
- 1974 American National Standard COBOL programmer reference, UP-8613
- Assembler programmer reference, UP-8227

If your action programs access a DMS data base, consult the current versions of the following manuals:

Read these if IMS accesses data bases

- IMS/DMS interface user guide/programmer reference, UP-8748
- DMS data description language programmer reference, UP-8022
- DMS data manipulation language user guide/programmer reference, UP-8036

Prerequisites for using this manual

Throughout this manual, we assume you've read and understood UP-9205, and the appropriate language manual. However, as required, we briefly define terms and describe concepts that are directly related to RPG II action programming.

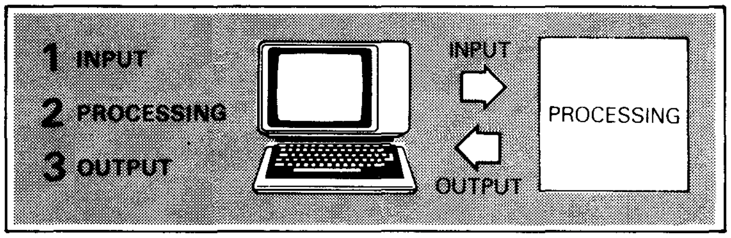
1.3. BASIC IMS TERMS

Action defined

What action programs do

The term **action programming** comes from the fact that the unit of work in IMS is the **action**. An action begins when an operator enters a message at a terminal and ends when a response to that message is returned. This is an important point to remember since the action programs you write are involved primarily with this activity – processing input messages, performing file retrieval or updating, and creating output messages.

An action always consists of three activities:

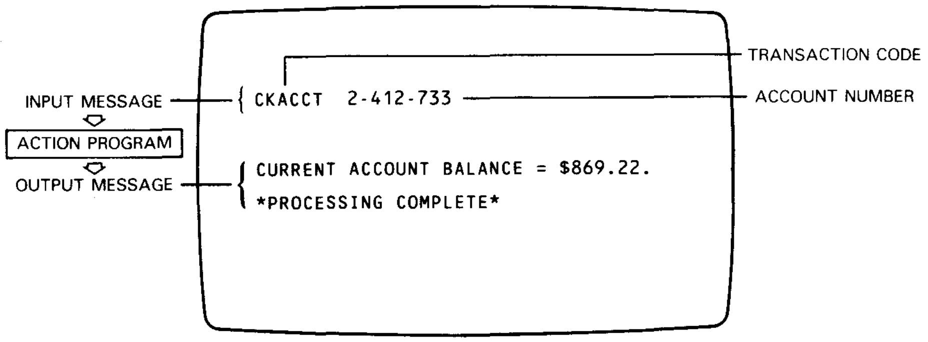


Transaction defined

A **transaction** is one action or a series of actions.

A **simple** transaction (Figure 1-1) consists of a single action.

Example – simple transaction



```
CKACCT  2-412-733

CURRENT ACCOUNT BALANCE = $869.22.
*PROCESSING COMPLETE*
```

Figure 1-1. A Simple Transaction. *In this example, one action program processes the input messsage and produces an output message – the checking account balance for the account specified and a* **processing complete** *notice.*

A **dialog** transaction (Figure 1-2) consists of two or more related actions.

Example – dialog transaction

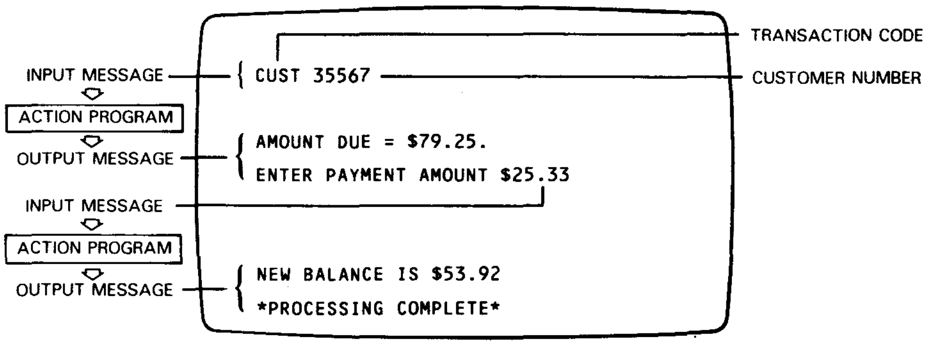


Figure 1-2. A Dialog Transaction. *In this example, two action programs are sequenced to produce amount due information, allow data entry, and compute a new balance for a specific customer account.*

Transaction codes initiate transactions

To begin a transaction, the operator enters a 1- to 8-character transaction code. (In single-thread IMS, the transaction code is 1 to 5 characters.) This code tells IMS the name of the action program that will process the input message.

Transaction code defined

Transaction codes are either the entire input message or a part of it. Transaction codes are defined to IMS at configuration time.

1.4. STRUCTURING TRANSACTIONS

Series of action programs processes transaction

Sometimes a single action program can process the function required. But more often, a series of action programs is needed. In either case, we create what we call a transaction structure.

Types of transaction termination

Transaction structure depends on how you terminate action programs. There are four major types of termination:

TYPES OF TERMINATION

- Normal
- External succession
- Delayed internal succession
- Immediate internal succession

From here on, we'll call the termination types normal termination, external, delayed, and immediate succession.

Distinction between termination and succession

Using the words **termination** and **succession** in the same context can be somewhat confusing. In IMS, termination means that an action program is finished processing. Whether you specify normal termination, external, delayed, or immediate succession, you are telling IMS that the current action program is finished processing and is now terminating.

Succession means that although the action program is terminating, the transaction is not complete. A successor action program will continue processing the transaction.

Normal termination

Normal termination means that the transaction itself is complete. No more processing occurs.

However, external, delayed, or immediate succession means that another action program follows and to resume processing.

Figures 1-3 through 1-6 illustrate these concepts.

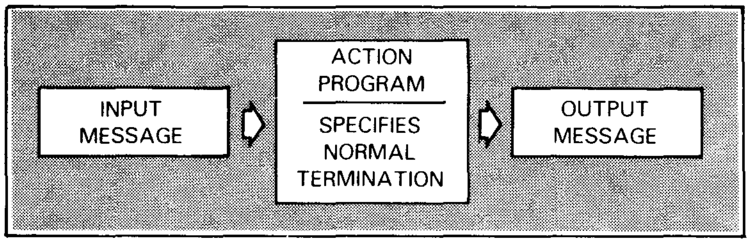


Figure 1–3. Normal Termination

Normal termination

Use normal termination to tell IMS that once your program creates an output message, the transaction is complete. When you don't specify the type of termination, IMS terminates normally. The last action program in a transaction always ends with normal termination.

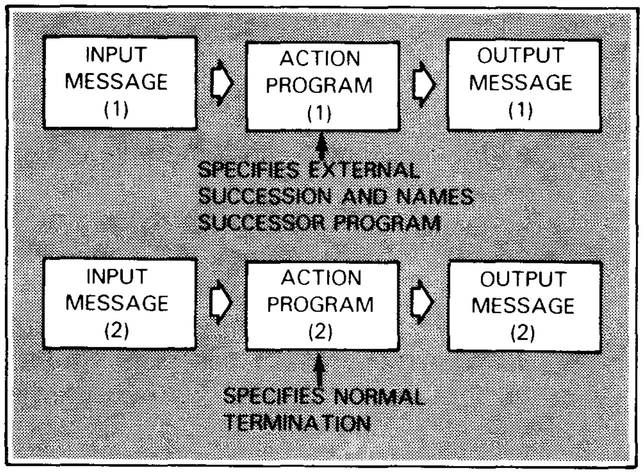


Figure 1–4. External Succession

External succession

Use external succession to tell IMS that the current action program is sending an output message and terminating; however, the transaction is not complete. When the terminal operator enters a second input message, the action program you named as external successor processes the second action, produces an output message, and terminates.

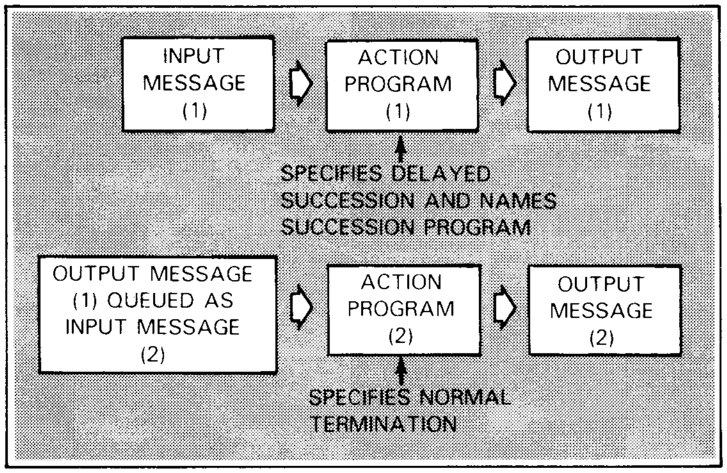

INPUT MESSAGE (1)
ACTION PROGRAM (1)
OUTPUT MESSAGE (1)
SPECIFIES DELAYED SUCCESSION AND NAMES SUCCESSION PROGRAM
OUTPUT MESSAGE (1) QUEUED AS INPUT MESSAGE (2)
ACTION PROGRAM (2)
OUTPUT MESSAGE (2)
SPECIFIES NORMAL TERMINATION

Figure 1-5. Delayed Internal Succession

Delayed succession

Use delayed succession to tell IMS that the current action program has processed an input message and produced an output message; however, that message isn't going to the terminal. Instead, it becomes the input message to the action program you named as successor. The successor program produces an output message that does go to the terminal and terminates. With delayed succession, the second action program uses the output message of the predecessor as its input message. Even though only one input message and one output message are seen at the terminal, internally there are two separate actions, each with an input and output message.

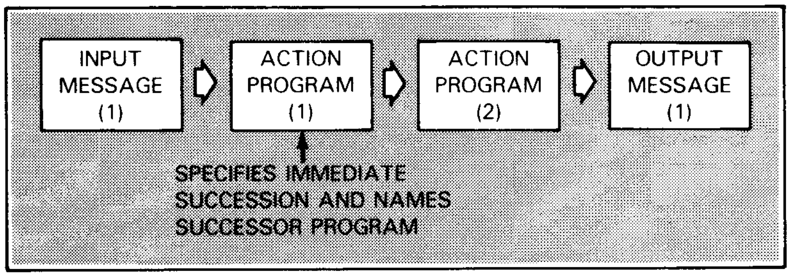


Figure 1-6. Immediate Internal Succession

Immediate succession

Use immediate succession to tell IMS that the current action program processed an input message but is not producing an output message. When it terminates, its successor action program immediately takes up where processing left off, produces an output message and terminates. In immediate succession, there is only one input and one output message. Thus, two action programs are processing a single action.

Combining transaction structures

With these four types of termination or transaction structures there is a good deal of flexibility in structuring transactions. There are basically no limitations as to how you can combine them. For example, you can specify immediate succession, delayed succession, external succession, and finally normal termination, all in turn (Figure 1-7).

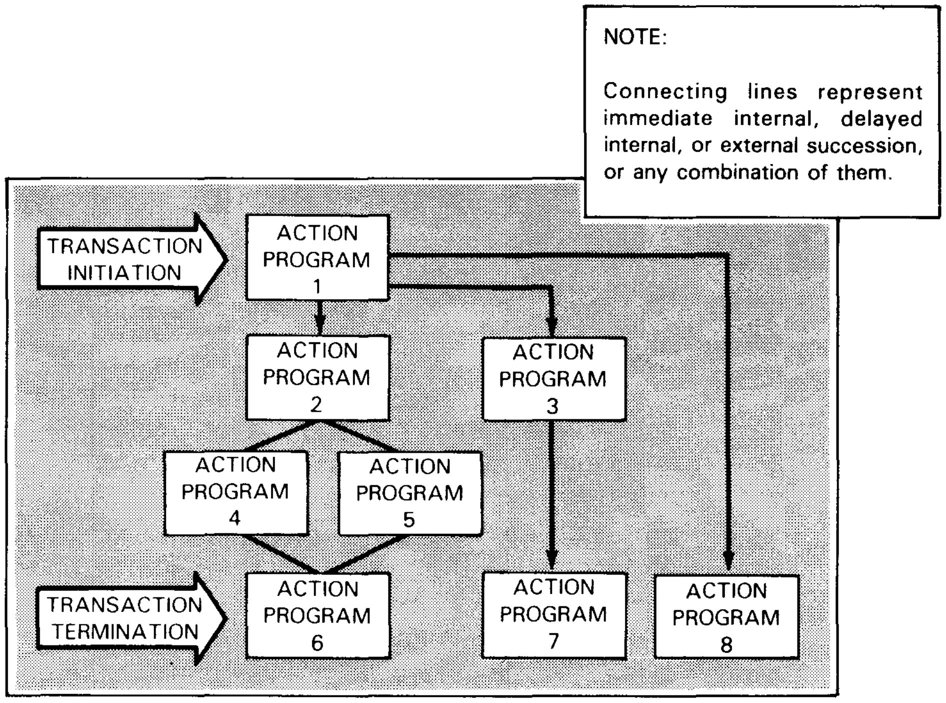


Figure 1-7. Dynamic Transaction Structure

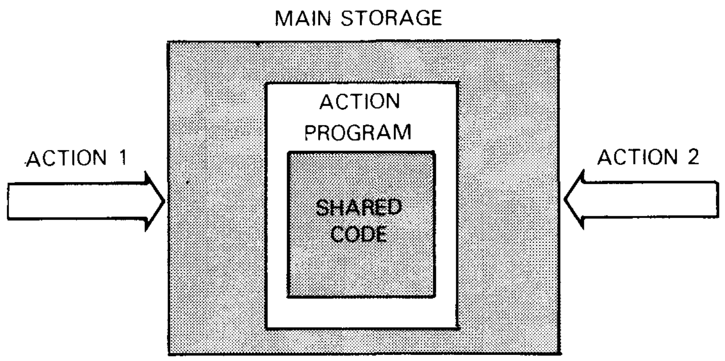
1.5. WRITING EFFICIENT ACTION PROGRAMS

Reentrant or sharable code most efficient

In part, the coding you use in your action program determines the efficiency of your message processing. The most efficient way to code an action program is to make the code reentrant or sharable. Action programs can be shared only in a multithread IMS environment. However, even in a single-thread environment you should write reentrant or sharable code, because you may later wish to use multithread IMS.

Reentrant code

A reentrant program is completely sharable, and none of the code is self-modifying. BAL action programs can be reentrant. This can mean great performance improvement because it avoids waiting when several actions require the same action program.

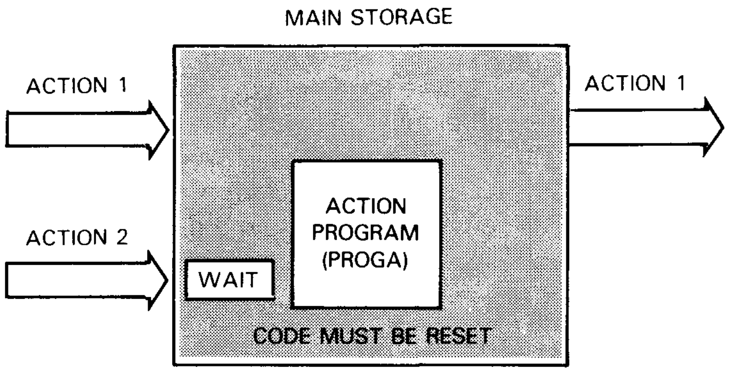


Shared code

Shared code is a means of executing a COBOL program as if it were reentrant. COBOL programs are sharable in the Procedure Division and Working Storage Section but not in IMS control regions.

Serially reusable code

A third type of coding that we use for action programs is serially reusable code. Serially reusable action programs can process only one action at a time. You can modify the action program code but you must reset or restore it, because the same copy of the program sometimes remains in storage to process the next action.



Programming clear messages

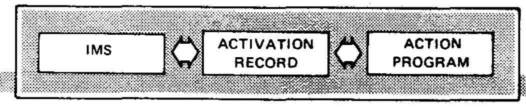
Remember that your action programs should serve the best interests of terminal operators who request information from your file. For this reason, messages you receive or create should be simple and understandable with a minimum of operator-entered codes or other data required at the terminal.

1.6. HOW IMS ACTION PROGRAMS INTERFACE WITH IMS

Activation record links action program to IMS

To communicate with IMS, an action program must link itself to IMS. This link is the activation record. The activation record handles the control and communication of data between IMS and your action program. The activation record can contain up to six interface areas:

Interface area names



- Input message area (IMA)
- Output message area (OMA)
- Program information block (PIB)
- Continuity data area (CDA)
- Work area (WA)
- Defined record area (DRA)

Interface area usage

Whether or not you use all six interface areas depends on the needs of your action program. All the interface areas are optional except the input message area and program information block.

Even if you don't access the program information block IMS automatically returns values there to the status code fields after each I/O request.

Layout of the activation record in main storage

Figure 1–8 shows how main storage looks when the action program PROG01 is loaded in a multithread IMS system. The layout of the activation record is slightly different in single-thread IMS.

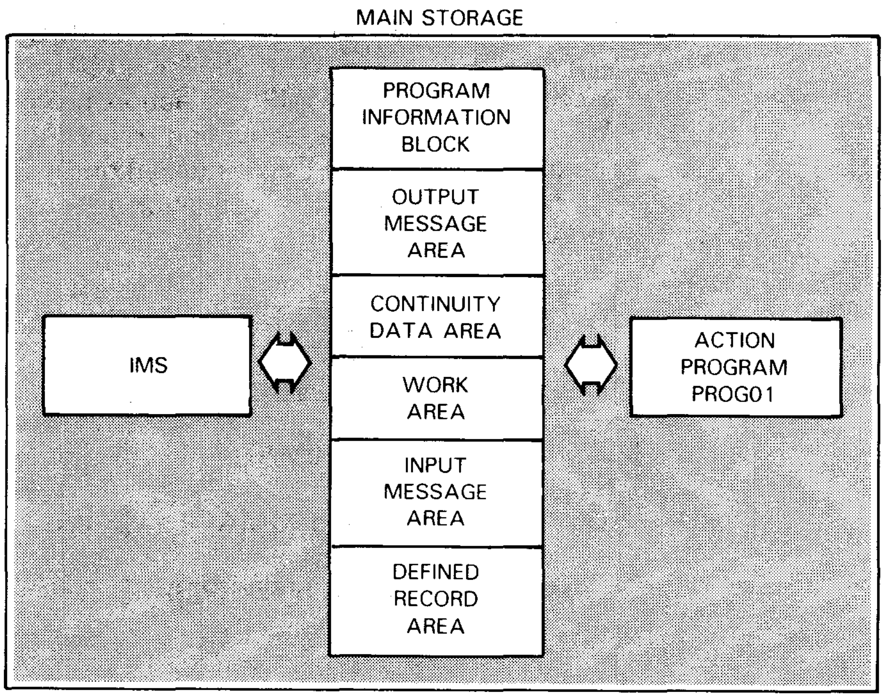
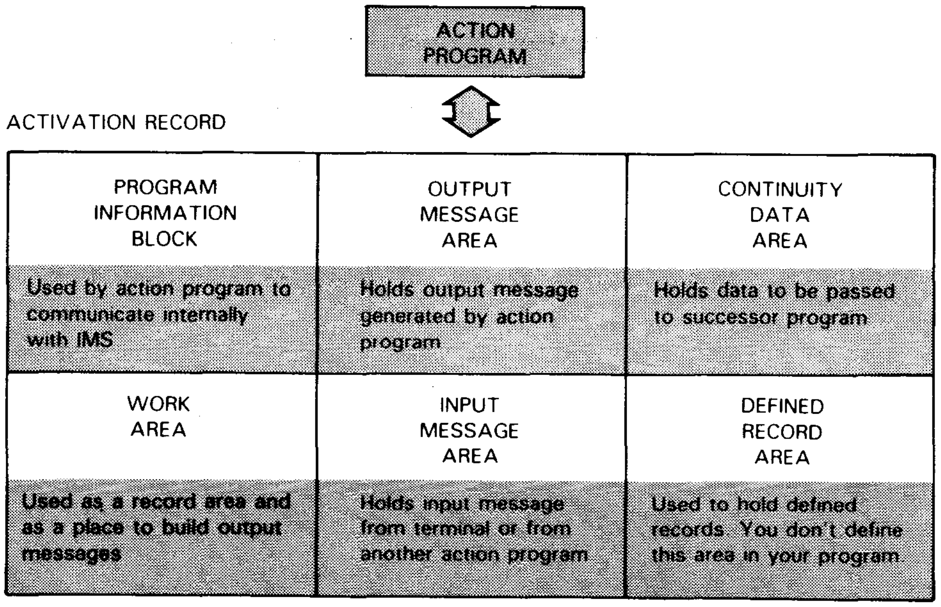


Figure 1–8. Activation Record in Main Storage

Action program and interface area relationship

Figure 1-9 shows the relationship between an action program and its interface areas.



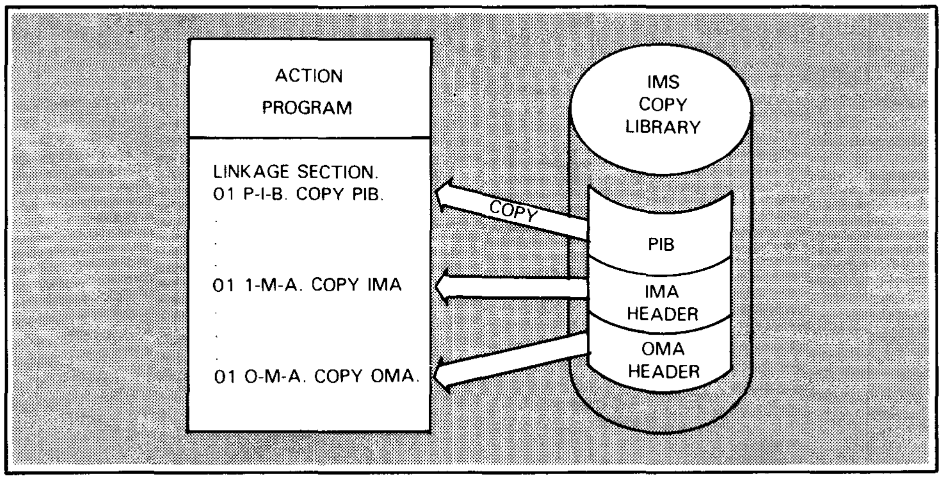
PROGRAM INFORMATION BLOCK	OUTPUT MESSAGE AREA	CONTINUITY DATA AREA
Used by action program to communicate internally with IMS	Holds output message generated by action program	Holds data to be passed to successor program
WORK AREA	INPUT MESSAGE AREA	DEFINED RECORD AREA
Used as a record area and as a place to build output messages	Holds input message from terminal or from another action program	Used to hold defined records. You don't define this area in your program.

Figure 1–9. The Action Program and Its Interface Areas

Formats of PIB, IMA, and OMA headers in IMS COPY library

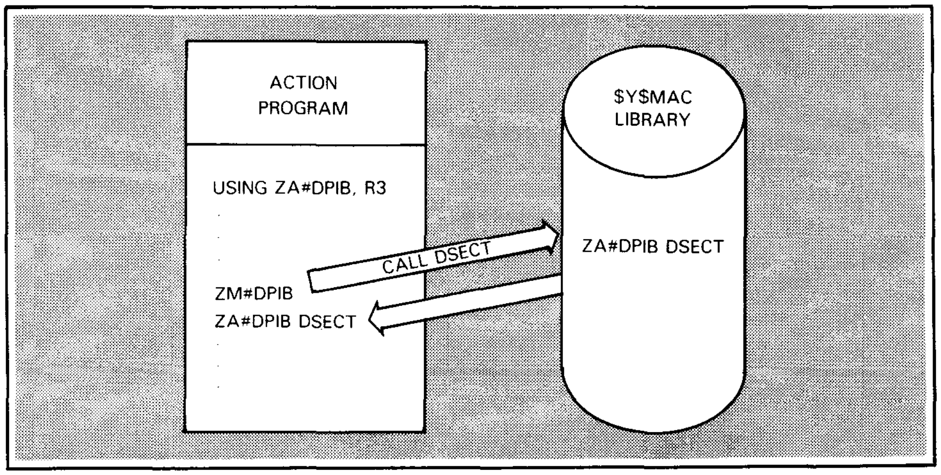
Your action program must define the formats of the interface areas that make up the activation record.

For COBOL action programs, you use COPY statements to copy the program information block, and the input and output message area headers into the linkage section of your action program. You have to code the descriptions of the continuity data area and work area according to the action program application.



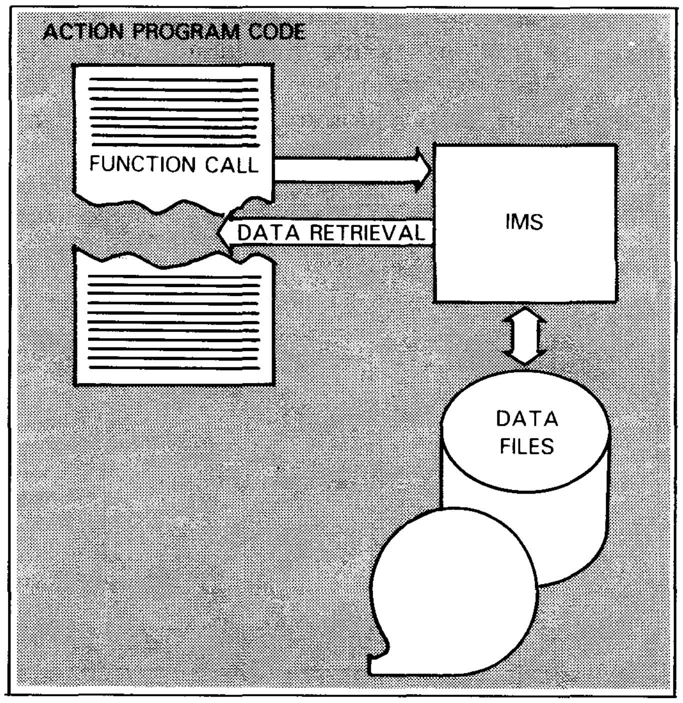
Receiving interface areas in a BAL action program

In BAL action programs, you assign registers to receive the addresses of interface areas. The formats for the program information block and the input and output message area headers are in the form of DSECTS in the system macro library, $Y$MAC. You issue macroinstructions to copy these formats into your program.



CALL function interface

Action programs also interface with IMS through the COBOL CALL statement or the BAL CALL or ZG#CALL macroinstruction. You use these CALL functions to issue requests to IMS for file access and other operations.



PART 2

BASIC IMS ACTION PROGRAMMING

2. General Rules for Coding Action Programs

2.1. COBOL ACTION PROGRAM STRUCTURE

Though COBOL action programs are similar to conventional COBOL programs, certain differences characterize them.

Identification Division

No differences

The identification division is the same as any COBOL identification division.

Environment Division

The first important difference is in the environment division.

Omitting input-output section

You must omit the input-output section in the environment division. It is not needed because you supply a file description in the file section of the IMS configuration. You also name your files, give file types, and any additional information concerning file processing as part of IMS configuration.

Data Division

Omitting the file section

Instead of using an FD statement to name the file you are accessing, omit the file section and place the file name in the working-storage section.

Files described in IMS configuration

When you use a function CALL statement for a particular file later in your program, IMS associates the file name you specified at configuration time with the file you name in working-storage.

Working-storage contents

In a sharable COBOL action program, the working-storage section in an action program may contain constants only. Describe each elementary item in the working-storage section with a VALUE clause.

Figure 2-1 shows an example of correct and incorrect working-storage section coding for an action program.

Working-storage example

INCORRECT	CORRECT
DATA DIVISION.	DATA DIVISION.
WORKING-STORAGE SECTION.	WORKING-STORAGE SECTION.
77 ERR-INDICATOR PIC X(19).	77 DMOALT PIC X(6) VALUE 'DMOALT'.
Ø1 ERR-MSG-LITS.	Ø1 ERR-MSG-LITS.
Ø2 ERR-1 PIC X(19).	Ø2 ERR-1 PIC X(19)
	VALUE '**INVALID KEY**'.
Ø2 ERR-2 PIC X(19).	Ø2 ERR-2 PIC X(19)
	VALUE '**END OF FILE**'.
Ø2 ERR-3 PIC X(19).	Ø2 ERR-3 PIC X(19)
	VALUE '**INVALID REQUEST**'.
Ø ERR-4 PIC X(19).	Ø2 ERR-4 PIC X(19)
	VALUE '**I/O ERROR'.
NO VALUE CLAUSES	

Figure 2-1. **Describing Working-Storage Items in a Sharable COBOL Action Program**

Linkage section required

Every COBOL action program requires a linkage section. This section is optional in a conventional COBOL program.

Your action program's linkage section defines the areas your program uses to interface with IMS. The names of these areas must correspond with the interface areas in the activation record and also with the names in the USING clause parameter list in the procedure division. (See Figure 2-2.)

COBOL coding for interface areas

```
DATA DIVISION.
.
.
.
LINKAGE SECTION.
Ø1  P-I-B.  COPY PIB74.
Ø1  I-M-A.  COPY IMA74.
Ø1  W-A.
Ø1  O-M-A.  COPY OMA74.
Ø1  C-D-A.
PROCEDURE DIVISION USING P-I-B I-M-A W-A
    O-M-A C-D-A.
```

Figure 2-2. Describing Interface Areas in a COBOL Action Program

Procedure Division

USING clause names interface areas

An action program always contains a USING clause in the procedure division statement. This is for naming the interface areas your program uses in processing messages.

Sequence of USING list parameters

Because parameters in the USING list are positional, you must code them in the prescribed order shown in Figure 2-3.

Dummy parameters indicate omissions

If, for example, your COBOL action program does not need the work area and continuity data area, you must still code a dummy parameter to indicate their omission from the USING list as follows:

```
PROCEDURE DIVISION USING PROGRAM-INFORMATION-BLOCK
  INPUT-MESSAGE-AREA D OUTPUT-MESSAGE-AREA.
```

In this case, you are choosing the letter D as a dummy parameter name. Because continuity data area is the last parameter of the list, you can omit the dummy parameter.

CALL functions replace COBOL verbs

Action programs do not use standard I/O COBOL verbs in the procedure division. Instead, they issue CALL function statements to IMS. (See Section 5.)

Example

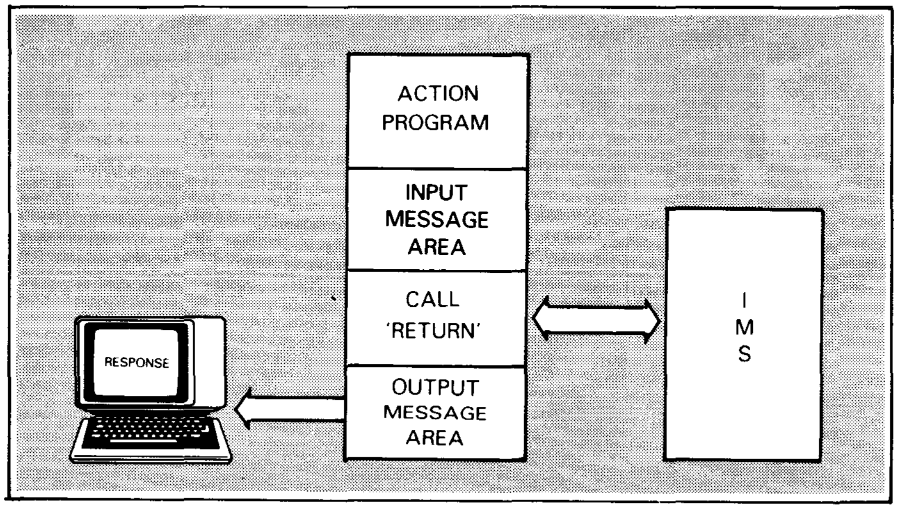
Figure 2-3 shows the correct and incorrect way to access data files from a COBOL action program.

INCORRECT	CORRECT
PROCEDURE DIVISION USING	PROCEDURE DIVISION USING
PROGRAM-INFORMATION-BLOCK	PROGRAM-INFORMATION-BLOCK
INPUT-MESSAGE-AREA D	INPUT-MESSAGE-AREA D
OUTPUT-MESSAGE-AREA.	OUTPUT-MESSAGE-AREA.
BEGIN-ROUT.	BEGIN-ROUT.
OPEN MYFIL.	CALL 'GET' USING MYFIL MYREC
READ MYFIL.	MYKEY.
MUST BE CALL FUNCTION, NOT COBOL VERB	

Figure 2-3. Accessing a Data File

Ending an action program

When you want to end an action program, use the CALL 'RETURN' function. It returns control to IMS, and if you've built an output message in the output message area, the CALL 'RETURN' sends the output message to the destination terminal.



2.2. COBOL PROGRAM STRUCTURE COMPARISON

Identifying a COBOL action program

COBOL action programs are distinguished from conventional COBOL programs by the

COBOL action program characteristics

- absence of an input-output Section;
- absence of a file section;
- linkage section containing a 77- or 01-level data description corresponding to each parameter on the procedure division USING clause;
- CALL functions to access and manipulate files; and by the
- CALL 'RETURN' function that ends the action program.

Figure 2-4 shows the similarities and differences between conventional COBOL action programs.

COBOL ACTION PROGRAM CODING STRUCTURE

Action program and conventional COBOL program compared

CONVENTIONAL PROGRAM STRUCTURE	ACTION PROGRAM STRUCTURE
IDENTIFICATION DIVISION.	IDENTIFICATION DIVISION.
PROGRAM-ID. program-name.	PROGRAM-ID. program-name.
(Any optional entry)	(Any optional entry)
ENVIRONMENT DIVISION.	ENVIRONMENT DIVISION.
CONFIGURATION SECTION.	CONFIGURATION SECTION.
SOURCE-COMPUTER. UNIVAC OS3.	SOURCE-COMPUTER. UNIVAC OS/3.
OBJECT-COMPUTER. UNIVAC OS3.	OBJECT-COMPUTER. UNIVAC OS/3.
SPECIAL-NAMES.	SPECIAL-NAMES.
(Any OS/3 implementor-names)	(No special names)
INPUT-OUTPUT SECTION.	(No input-output section)
FILE-CONTROL.	
SELECT filename	
ASSIGN TO DISK-lfdname-V	
ORGANIZATION file-type.	
DATA DIVISION.	DATA DIVISION.
FILE SECTION.	(No file section)
FD filename	
LABEL RECORD STANDARD.	
Ø1 data-name-2	
Ø2 data-name-2	
Ø2 data-name-3	
WORKING-STORAGE SECTION.	WORKING-STORAGE SECTION.
77 data-name.	77 data-name.
Ø1 record-name.	.
	.
	.
[LINKAGE SECTION.]	LINKAGE SECTION.
	Ø1 PROGRAM-INFORMATION-BLOCK
	.
	.
	.
(No control area description)	Ø1 INPUT-MESSAGE-AREA
	.
	.
	.
	[Ø1 WORK-AREA]
	.
	.
	.
	[Ø1 OUTPUT-MESSAGE-AREA]
	.
	.
	.
	[Ø1 CONTINUITY-DATA-AREA]
PROCEDURE DIVISION.	PROCEDURE DIVISION USING program-information-block input-message-area [work-area][output-message-area] [continuity-data-area].
	Para-1.
	.
	.
	.
	Para-2.
	.
	.
	.
	CALL'RETURN'.

Figure 2-4. Conventional COBOL Versus COBOL Action Program Structures

2.3. COBOL LANGUAGE RESTRICTIONS

In addition to omitting input-output and file sections, there are several restrictions to observe when you write a COBOL action program.

Identifying action programs with function keys

How to use

Some programmers like to use a function key to identify the action program load module. If you do this, don't use a function key (F#nn) as the PROGRAM-ID name because the COBOL compiler treats the # symbol as invalid. Instead, supply a valid PROGRAM-ID name in the identification division and then include a LOADM statement with F#nn as the load module name at link-edit time.

Example

For example, you identify your action program as follows:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CREDIT.
```

CREDIT is your program name. You then associate your program-id with a function key at link-edit time in the following job control stream:

```
// EXEC LNKEDT
/$
   LOADM F#01
   INCLUDE CREDIT
/*
```

Illegal syntax

Some COBOL verbs, clauses, and sections are illegal in action programs. If you compile them with the shared code parameter, PARAM IMSCOD=YES, the compiler locates and deletes them from your program. (See Section 11.)

The following reserved words are illegal in 1974 COBOL action programs. For language restrictions on extended COBOL programs, refer to G.6.

Reserved words

ACCEPT MESSAGE COUNT	SEGMENT-LIMIT
ALTER	SEND
CALL identifier	SORT
CANCEL	START
CLOSE	STOP
COMMUNICATION SECTION	SYSCHAN-n
DECLARATIVES	SYSCONSOLE
DELETE	SYSFORMAT
DISABLE	SYSIN
ENABLE	SYSIPT
EXHIBIT	SYSLOG
FILE SECTION	SYSLST
INPUT-OUTPUT SECTION	SYSOPT
MERGE	SYSOUT
OPEN	SYSSCOPE
READ	SYSTERMINAL
RECEIVE	SYSWORK
RELEASE	TRACE
RETURN	WRITE
REWRITE	

Illegal verbs with working-storage items

Other COBOL verbs must not have working-storage items as receiving operands. These verbs are:

ACCEPT	PERFORM (varying)
ADD	SEARCH (varying)
COMPUTE	SET
DIVIDE	STRING
INSPECT	SUBTRACT
MOVE	TRANSFORM
MULTIPLY	UNSTRING

Precautionary diagnostics

If you compile your action program with the shared code parameter, the compiler flags the erroneous statement and issues a precautionary diagnostic.

Extended COBOL language restrictions

For extended COBOL language-restrictions on action programs, refer to G.6.

2.4. BAL ACTION PROGRAM STRUCTURE

Activation record definition

Similar to COBOL action programs, BAL action programs must provide a receiving area for the IMS activation record interface areas. You handle this by assigning registers to receive the addresses of the interface areas.

DSECTs generate interface area descriptions

There are macroinstruction calls for the program information block and input and output message header formats. When you issue one of these macroinstructions, it calls a corresponding DSECT that generates the interface area format into your action program.

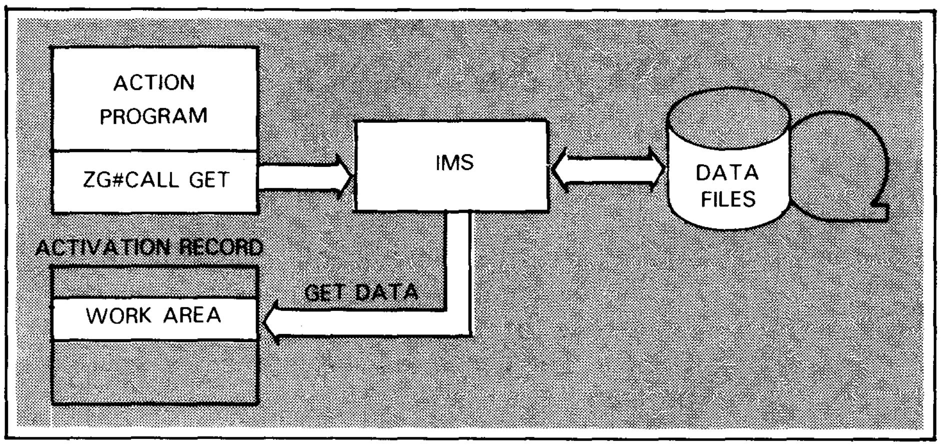
BAL coding for interface areas

```
USING ZA#DPIB,R9
ZM#DPIB
.
.
.
USING ZA#IMH,R12
ZM#DIMH
.
.
.
USING WA,R6
.
.
.
USING ZA#DOMH
.
.
.
USING CDA,R4
```

Figure 2–5. Describing Interface Areas in a BAL Action Program

Function call macroinstructions

A BAL action program, like COBOL, uses function calls to access files. There are two forms of function calls, the CALL or the ZG#ALL macroinstruction.



Register 1 parameter list

When you enter a message at the terminal and IMS transfers control to your BAL action program entry point, register 1 always points to a parameter list containing, in order:

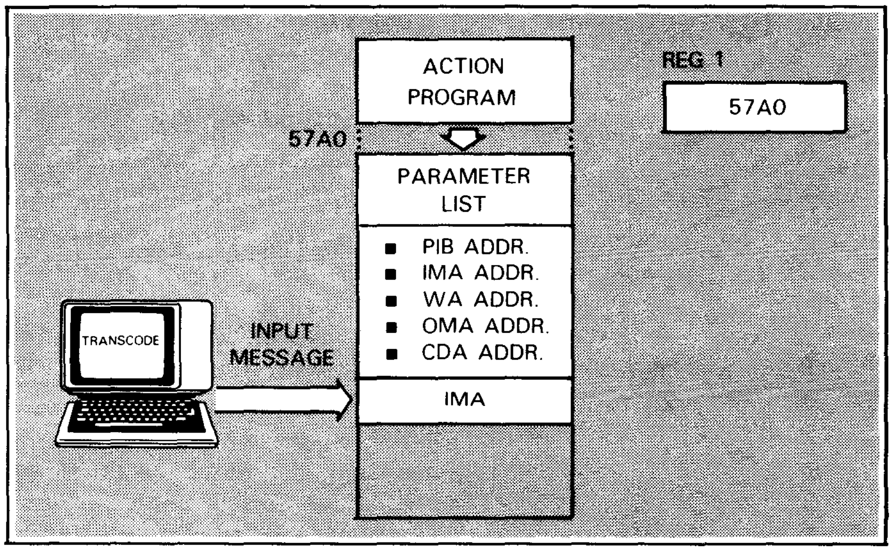
Program information block address

Input message area address

Work area address

Output message area address

Continuity data area address



Parameter list entries for unused areas

The work area, output message area, and continuity data area are optional. If you don't need them in your program, IMS assigns a binary 0 to their place in the parameter list.

Other registers contain save area and action program entry point addresses. (See 6.5 for more detail about BAL action programming.)

Characteristics of a BAL action program

Several ways you can distinguish a BAL action program from other BAL programs are:

- Registers assigned to the addresses of interface area DSECTs
- Use of CALL or ZG#CALL macroinstructions to access and manipulate files
- Use of ZM#DPIB, ZM#DOMH, ZM#DIMH macroinstructions to transfer the program information block and the control header formats from the IMS activation record to the BAL program.
- Use of ZG#CALL RETURN function to end the action program.

2.5. THE ACTIVATION RECORD

Defining and constructing activation record

Each time IMS initiates an action, it constructs an activation record in main storage.

Activation record structure

Each activation record has a program information block and an input message area. It may also have an output message area, work area, continuity data area, and a defined record area.

How IMS uses the program information block

The program information block contains information that IMS uses to communicate with your action program. By testing fields in the program information block for the status of IMS functions, your program can control the processing of files and the succession of action programs.

How IMS uses the input message area

IMS uses the input message area to exchange input message processing information with your program. Fields in the IMA hold control information that identifies input terminals, and gives message text length as well as message text.

How action programs use the work area

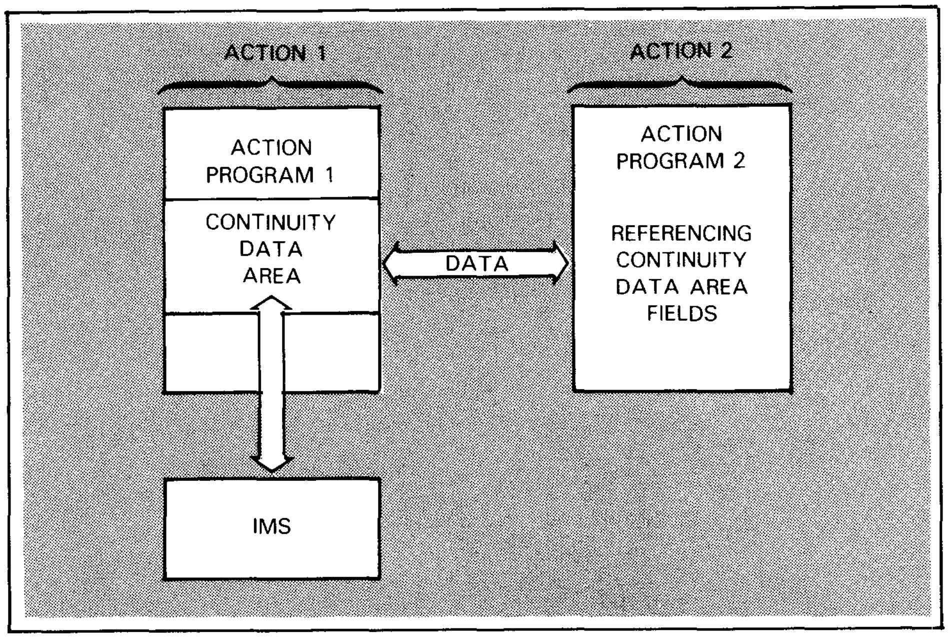
The work area is an interface area that you often use when your action programs are sharable or reentrant. It is modifiable working storage that your action program uses to build output messages (see 6.1) or as a record area for file input/output.

How IMS uses the output message area

Output message area fields notify IMS of output message control information such as output terminal identification, special output options, and output message text length. It also provides a place where IMS can interface with output message text.

How IMS uses the continuity data area

When used, the continuity data area provides the interface area where your action program passes data from action to action in a dialog transaction. IMS uses the continuity data area to interface with your action program's transfer of data from one action to another.



How IMS uses the defined record area

IMS uses the defined record area to reference defined records. Your action program can't access a defined record area (DRA) or write into the defined record area. You do not define this area in your program.

IMS/action program conversation

When you enter a message at a terminal, IMS:

Activation record allocation

- dynamically allocates the activation record interface areas that your program needs to converse with IMS; and

Action program scheduling

- schedules and loads the action program needed to process the action.

COBOL action program receiving area

When IMS schedules a COBOL action program, that program must contain a linkage section where it can exchange data with IMS. Part of the linkage section must be formatted in a certain way. The IMS copy library provides this formatted source code.

COPY statement

You use a COPY statement to transfer the formats of the program information block, input message area header, and output message area header from the IMS copy library areas to the linkage section of your COBOL action program.

Extended COBOL copy library names

When you compile your COBOL action program using the extended COBOL compiler, the IMS copy library makes the program information block format and the output message area and input message area control headers available under the names PIB, OMA, and IMA.

1974 COBOL copy library names

When you use the 1974 American National Standard COBOL compiler, your COPY statement must use the names PIB74, OMA74, and IMA74 to transfer the interface area formats needed by your program.

Figure 2-6 shows how a COBOL action program converses with IMS via the activation record. IMS sets up space in the activation record for each interface area your action program uses.

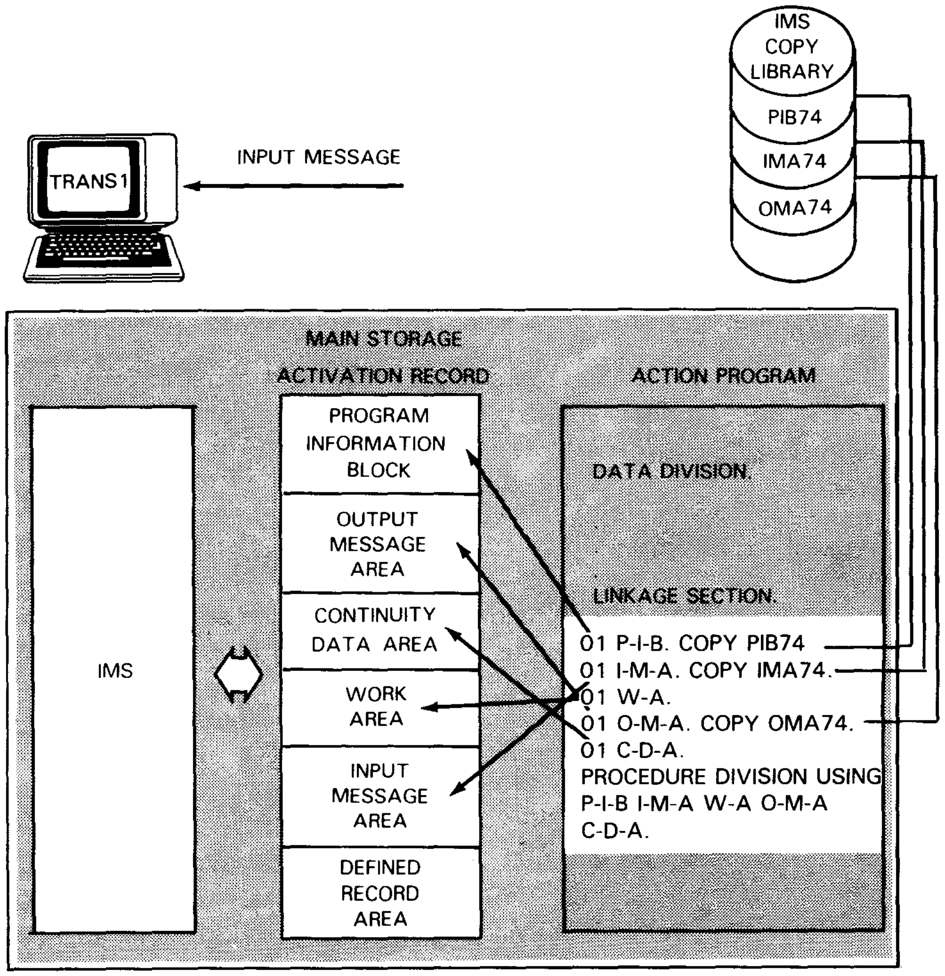


Figure 2-6. IMS/COBOL Action Program Interface. *The COPY verb moves interface area formats from the IMS copy library to your action program's Linkage Section and your program converses with the IMS interface areas in the activation record. Note, your action program cannot access or write into the defined record area.*

BAL DSECT names for interface areas

A BAL action program accesses the activation record interface areas via macroinstructions that call DSECTs from the $Y$MAC system macro library or a user macro library. The ZM#DPIB macroinstruction calls the ZA#DPIB DSECT, the ZM#DOMH macroinstruction calls the ZA#OMH, and the ZM#DIMH macroinstruction calls the ZA#IMH DSECT.

Example

Figure 2-7 shows IMS communicating with a BAL action program via the activation record. Again, IMS sets up an interface area in the activation record for each interface area used by your BAL action program.

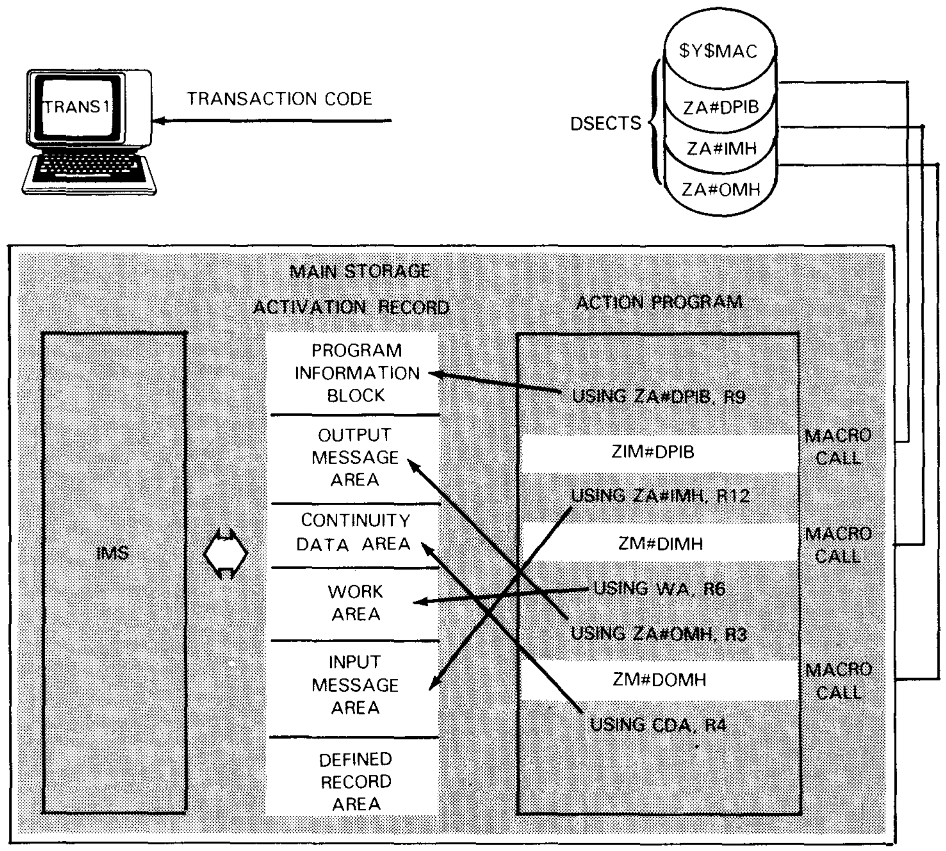


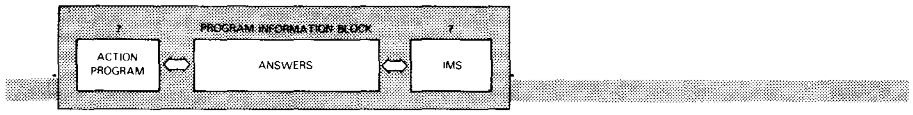
Figure 2-7. **IMS/BAL Action Program Interface.** *The ZM#DPIB, ZM#DOMH, and ZM#DIMH macroinstructions call the format headers from the $Y$MAC system macro library. If you use a work area or continuity data area, you must define and cover them in your action program. Note, your action program cannot access or write into the defined record area.*

3. Communicating with IMS

3.1. IMS ANSWERS ACTION PROGRAM MESSAGE PROCESSING QUESTIONS

The program information block (PIB) is where IMS and action programs exchange data. IMS sets some program information block fields and your action program sets others.

By testing the contents of the program information block fields, you can find the results of file input/output operations, obtain values of indicators, and construct your action program logic to handle error or other processing conditions accordingly. Figure 3-1 shows some of the message processing questions answered by the PIB.



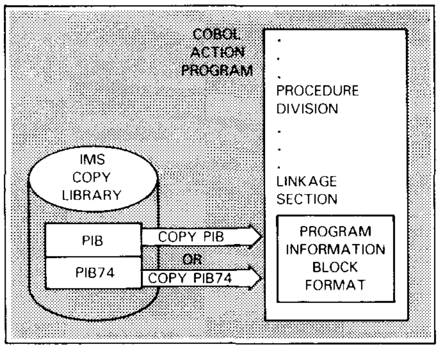
1. WAS MY READING, CHANGING, OR WRITING A RECORD SUCCESSFUL?
2. WHAT ELSE CAN YOU TELL ME ABOUT MY I/O OPERATION?
3. WHAT TYPE IS MY DEFINED RECORD?
4. WHAT'S MY NEXT ACTION PROGRAM'S NAME?
5. HOW AM I GOING TO TERMINATE THIS ACTION?
6. WILL I SET A NEW ROLLBACK POINT AT ACTION TERMINATION?
7. AM I HOLDING RECORD LOCKS?
8. WHEN DID THIS TRANSACTION OCCUR?
9. WHAT IS MY DEFINED FILE NAME?
10. WHAT IS MY DEFINED RECORD NAME?
11. HOW LARGE IS MY WORK AREA AND HOW MUCH LARGER CAN IT BECOME?
12. HOW LARGE IS MY INPUT CONTINUITY AREA TO RECEIVE THE CONTINUITY DATA?
13. HOW LARGE IS THE CONTINUITY DATA AREA IN MY CURRENT ACTION PROGRAM?
14. DO I NEED TO INCREMENT MY RECEIVING CONTINUITY AREA?
15. WHEN DOES EACH ACTION START?
16. WHAT TYPE TERMINAL SENT THE INQUIRY AND WHAT ARE ITS ATTRIBUTES?

Figure 3-1. The Program Information Block Answers Action Program Processing Questions

3.2. COBOL PROGRAM INFORMATION BLOCK FORMAT

Copying PIB format into Linkage Section

When you write COBOL action programs, you must copy the predefined COBOL program information block format into the linkage section of your action program from the IMS copy library.



Extended COBOL PIB

Use the name PIB74 to copy the program information block format for 1974 American National Standard COBOL into your linkage section. If you write your action program in extended COBOL, use the name PIB.

COBOL header format for PIB

Figure 3-2 shows the COBOL program information block format for 1974 American National Standard COBOL. Note that the data names for TODAY and HR-MIN-SEC are different in extended COBOL.

```
02   STATUS-CODE                   PIC 9(4) COMP-4.
02   DETAILED-STATUS-CODE          PIC 9(4) COMP-4.
02   RECORD-TYPE REDEFINES DETAILED-STATUS-CODE.
     03 PREDICTED-RECORD-TYPE      PIC X.
     03 DELIVERED-RECORD-TYPE      PIC X.
02   SUCCESSOR-ID                  PIC X(6).
02   TERMINATION-INDICATOR         PIC X.
02   LOCK-ROLLBACK-INDICATOR       PIC X.
02   TRANSACTION-ID.
     03  YEAR                      PIC 9(4) COMP-4.
     03  TODAY                     PIC 9(4) COMP-4. ①
     03  HR-MIN-SEC                PIC 9(9) COMP-4. ②
02   DATA-DEF-REC-NAME             PIC X(7).
02   DEFINED-FILE-NAME             PIC X(7).
02   STANDARD-MSG-LINE-LENGTH      PIC 9(4) COMP-4.
02   STANDARD-MSG-NUMBER-LINES     PIC 9(4) COMP-4.
02   WORK-AREA-LENGTH              PIC 9(4) COMP-4.
02   CONTINUITY-DATA-INPUT-LENGTH   PIC 9(4) COMP-4.
02   CONTINUITY-DATA-OUTPUT-LENGTH   PIC 9(4) COMP-4.
02   WORK-AREA-INC                 PIC 9(4) COMP-4.
```

Figure 3-2. 1974 American National Standard COBOL Format for Program Information Block (Part 1 of 2)

```
02  CONTINUITY-DATA-AREA-INC   PIC 9(4) COMP-4.
02  SUCCESS-UNIT-ID.
    03 TRANSACTION-DATE.
       04 YEAR                 PIC 99.
       04 MONTH                PIC 99.
       04 TODAY                PIC 99.
    03 TIME-OF-DAY.
       04 HOUR                 PIC 99.
       04 MINUTE               PIC 99.
       04 SECOND               PIC 99.
    03 FILLER                  PIC XXX.
02  SOURCE-TERMINAL-CHARS.
    03 SOURCE-TERMINAL-TYPE  PIC X.
    03 SOURCE-TERM-MSG-LINE-LENGTH    PIC 9(4) COMP-4.
    03 SOURCE-TERM-MSG-NUMBER-LINES   PIC 9(4) COMP-4.
    03 SOURCE-TERM-ATTRIBUTES PIC X.
02  EDF-MODE                   PIC X.
```

NOTES:

① The name of this field in extended COBOL is DAY.

② The name for this field in extended COBOL is TIME.

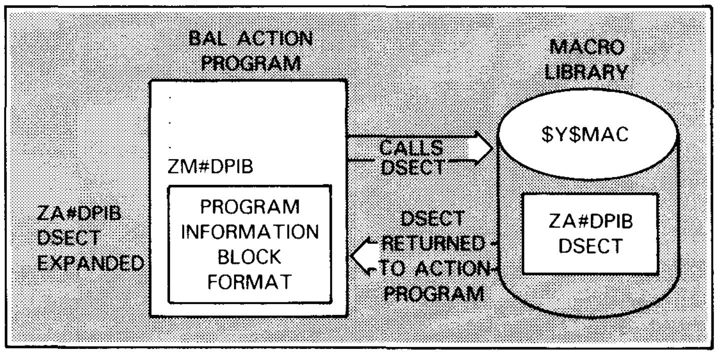
Figure 3-2. 1974 American National Standard COBOL Format for Program Information Block (Part 2 of 2)

Subsections 3.5 through 3.18 describe the contents and meaning of each field in the COBOL program information block.

3.3. BASIC ASSEMBLY LANGUAGE PROGRAM INFORMATION BLOCK FORMAT

Generating PIB DSECT

When you write action programs in BAL, issue the ZM#DPIB macroinstruction to generate the BAL program information block DSECT (ZA#DPIB). The macroinstruction expands inline in your BAL action program coding and you can test program information block fields.



BAL format for PIB

Figure 3-3 shows the format of the ZA#DPIB DSECT that the ZM#DPIB macroinstruction calls into your action program from the $Y$MAC system library.

```
         PROC
ZM#DPIB  NAME
ZA#DPIB  DSECT PROGRAM-INFORMATION-BLOCK
ZA#PSC   DS     H                        STATUS-CODE
*
* * *    EQUATES FOR ZA#PSC
*
ZA#PTSUC EQU    0                        SUCCESSFUL REQUEST
ZA#PTKEY EQU    1                        INVALID KEY
ZA#PTEOF EQU    2                        END OF FILE
ZA#PTREQ EQU    3                        INVALID REQUEST
ZA#PTIOE EQU    4                        I/O ERROR
ZA#PTIUP EQU    5                        INVALID UPDATE (DRM ONLY)
ZA#PTIMC EQU    6                        IMC ERROR
ZA#PDSC  DS     H                        DETAILED-STATUS-CODE
*
* * *    EQUATES FOR ZA#PDSC (WHEN STATUS IS INVALID KEY ONLY)
*
ZA#PDNOI EQU    X'E1'                    NO IDENTIFIER SUPPLIED
ZA#PD2BI EQU    X'E2'                    IDENTIFIER IS TOO LONG
*
ZA#PDRAN EQU    X'E4'                    IDENTIFIER IS OUT OF RANGE
*
* * *    EQUATES FOR ZA#PDSC (WHEN STATUS IS INVALID REQUEST ONLY)
*
```

Figure 3-3. BAL Format for Program Information Block (ZA#DPIB DSECT) (Part 1 of 3)

```
ZA#PDPAR  EQU   1                   INCORRECT NUMBER OF PARAMETERS
ZA#PDCOD  EQU   2                   FUNCTION CODE OUT OF LEGAL RANGE
ZA#PDVAL  EQU   3                   INCORRECT PARAMETER VALUE
ZA#PDSHR  EQU   4                   SHARED RECORD NOT IN USE BY TRANS.
*                                   DUPLICATE KEY ON INSERT (VS/9 ONLY)
ZA#PDDEF  EQU   5                   FILE NOT DEFINED
ZA#PDOPN  EQU   6                   FILE NOT OPEN
ZA#PDTYP  EQU   7                   FUNCTION INVALID FOR TYPE OF FILE
ZA#PDLOC  EQU   8                   RECORDS NOT LOCKED
*                                   UPDATE SUPPRESSED FOR FILES (VS/9)
ZA#PDTUP  EQU   9                   PUT OR DELETE NOT PRECEDED BY GETUP
ZA#PDILL  EQU   10                  ILLEGAL FUNCTION REQUESTED
ZA#PDASS  EQU   11                  FILE NOT ASSIGNED TO THIS ACTION
ZA#PDMOD  EQU   12                  REQUIRED MODULE NOT CONFIGURED
*                                   NOT USED ON VS/9
ZA#PDCAP  EQU   13                  FILE CAPACITY EXCEEDED ON INSERT
*                                   NOT USED ON VS/9
ZA#PDSPA  EQU   14                  INSUFFICIENT SPACE IN MAIN STORAGE
*                                   NOT USED ON VS/9
ZA#PDUPD  EQU   15                  UPDATE NOT PERMITTED IN CONF.
ZA#PDSUP  EQU   17                  UPDATE SUPPRESSED FOR FILES
*                                   NOT USED ON VS/9
ZA#PDREC  EQU   18                  RECORD ALREADY LOCKED (S/T ONLY)
*                                   NOT USED ON VS/9
*
* * *     EQUATES FOR ZA#PDSC (WHEN STATUS IS IMC ERROR ONLY)
*
ZA#PDOIQ  EQU   2                   OUTPUT-TO-INPUT QUEUING ERROR
ZA#PDDES  EQU   4                   MISSING OR INVALID DESTINATION
ZA#PDNBA  EQU   5                   NO ICAM NETWORK BUFFER AVAILABLE
ZA#PDDER  EQU   6                   ICAM DISK ERROR
ZA#PDOFL  EQU   7                   INVALID OUTPUT MESSAGE LENGTH
ZA#PSID   DS    CL6                 SUCCESSOR-ID
ZA#PSIND  DS    CL1                 TERMINATION-INDICATOR
* * *     EQUATES FOR ZA#PSIND
ZA#PSNN   EQU   C'N'                NORMAL TERM
ZA#PSNA   EQU   C'A'                ABNORMAL TERM
ZA#PSNS   EQU   C'S'                ABNORMAL TERM WITH SNAP
ZA#PSNI   EQU   C'I'                IMMEDIATE INTERNAL SUCCESSION
ZA#PSND   EQU   C'D'                DELAYED INTERNAL SUCCESSION
ZA#PSNE   EQU   C'E'                EXTERNAL SUCCESSION
ZA#PLRI   DS    CL1                 LOCK-ROLLBACK-INDICATOR
* * *     EQUATES FOR ZA#PLRI
ZA#PLRIN  EQU   C'N'                WRITE ROLLBACK POINT,RELEASE LOCKS
ZA#PLRIO  EQU   C'O'                ROLLBACK UPDATES
ZA#PLRIH  EQU   C'H'                HOLD LOCKS IND
ZA#PLRIR  EQU   C'R'                RELEASE PENDING LOCKS INDICATOR
ZA#PTID   DS    CL8                 TRANSACTION-ID
ZA#PDDRN  DS    CL7                 DATA-DEF-REC-NAME
ZA#PDFN   DS    CL7                 DEFINED-FILE-NAME
ZA#PMLL   DS    H                   STANDARD-MSG-LINE-LENGTH
ZA#PMNL   DS    H                   STANDARD-MSG-NUMBER-LINES
ZA#PWA    DS    H                   WORK-AREA-LENGTH
ZA#PCDIN  DS    H                   CONTINUITY-DATA-INPUT-LENGTH
ZA#PCDL   EQU   *                   CDA LEN : OS/3 BASIC
ZA#PCDO   DS    H                   CONTINUITY-DATA-OUTPUT-LENGTH
ZA#PWAI   DS    H                   WORK-AREA-INC
```

Figure 3–3. BAL Format for Program Information Block (ZA#DPIB DSECT) (Part 2 of 3)

```
ZA#PCDI  DS    H                        CONTINUITY-DATA-AREA-INC
ZA#DTE   DS    CL6                      CURRENT DATE - YYMMDD
ZA#TME   DS    CL6                      SUCCESS-UNIT TIME - HHMMSS
ZA#UNID  DS    CL3                      SUCCESS-UNIT-UNIQUE-ID
ZA#TTTYP DS    CL1                      TERMINAL TYPE
*
* * *    EQUATES FOR ZA#TTTYP
*
ZA#TFCC  EQU   C'F'                     UTS400 CP (U4 MODE) OR UTS400
ZA#TNON  EQU   C'V'                     U100 OR U200
ZA#TWS   EQU   C'W'                     UTS20 OR WORKSTATION
ZA#TNOV  EQU   C'N'                     UTS10, TTY, DCT500, OR DCT1000
ZA#T327  EQU   C'3'                     3271
ZA#TCNS  EQU   C'C'                     CONSOLE
ZA#T4CP  EQU   C'U'                     UTS400 CP (U2 MODE)
ZA#T4PR  EQU   C'P'                     UTS400 PR
ZA#T40   EQU   C'4'                     UTS40
ZA#TEDT  EQU   C'T'                     UTS400 TEXT EDITOR
ZA#TMLL  DS    H                        TERMINAL-MSG-LINE-LENGTH
ZA#TMNL  DS    H                        TERMINAL-MSG-NUMBER-LINES
ZA#TATTR DS    CL1                      TERMINAL ATTRIBUTES
*
* * *    EQUATES FOR ZA#TATTR
*
ZA#TANOV EQU   C'N'                     NONVIDEO
ZA#TASB  EQU   C'S'                     SCREEN BYPASS
ZA#TAKAT EQU   C'K'                     KATAKANA
ZA#TASBK EQU   C'A'                     SCREEN BYPASS AND KATAKANA
ZA#TAZER EQU   C'Z'                     NO ATTRIBUTES
*
* * *    EQUATES FOR ZA#DDPMD
*
ZA#DDPMD DS    X                        DDP MODE
ZA#DTR   EQU   C'R'                     DIRECTORY TRANS ROUTING
ZA#PTRA  EQU   C'A'                     PROGRAM TRANS ROUT'G - ACTIVATE
ZA#PTRC  EQU   C'C'                     PROGRAM TRANS ROUT'G - ABORT/CANCEL
ZA#PTRE  EQU   C'E'                     PROGRAM TRANS ROUT'G - END
         DS    0F
ZT#HSAAP EQU   *                        ACTION PROGRAM SAVE AREA
ZA#PSAAP EQU   *                        ACTION PROGRAM SAVE AREA
ZT#HSAIW EQU   *+28                     CONT ACT AND INTERNAL
ZA#PSAVE DS    0CL72                    SAVE AREA
         DS    5A
ZA#GINT  DS    A
         DS    12A
*                                       R15=EM ENTRY PT USED BY LNK MOD
         CNOP  0,8
ZA#PLGTH EQU   *-ZA#DPIB
ZA#PLEN  EQU   *-ZA#DPIB                PIB LENGTH
&SYSECT  CSECT
         END
```

Figure 3-3. BAL Format for Program Information Block (ZA#DPIB DSECT)
(Part 3 of 3)

3.4. CONTENTS OF THE PROGRAM INFORMATION BLOCK

The program information block is always present in the activation record. Each field in the program information block contains data that aids IMS and your action program in processing messages.

COBOL data names correspond to BAL labels

The data names given for each COBOL field correspond to the labels of the DS statements in the BAL program information block.

3.5. OBTAINING COMPLETION STATUS (STATUS-CODE)

IMS sets value after CALL functions

Each time you issue a CALL function, IMS sets a half-word binary value in the STATUS-CODE field (ZA#PSC) of the program information block to indicate the results of your file operation or other requests.

Testing PIB fields

You should test this value after performing a file operation. IMS can return the following status codes:

Status code meanings

Code (Hex)	Meaning
00	Successful
01	Invalid key or record number
02	End of file or unallocated optional file
03	Invalid request
04	I/O error
05	Violation of data definition
06	Internal message control error
07	Screen formatting error

Testing Status Codes in a COBOL Action Program

Testing method

One way to test the status code is to compare the contents of the STATUS-CODE field with the possible status code values. If the status code is zero, the function request was successful and processing continues. If the status code is greater than zero, an error has occurred and the program goes to the error routine. Figure 3-4 illustrates coding to test the STATUS-CODE field for invalid record type (status code 1) after a GET function.

```
GET-STATE-RECORD.
    CALL 'GET' USING STATE WORK-AREA STATE-NAME-IN.
    IF STATUS-CODE EQUAL 1 GO TO PROCESS-ERROR.
    .
    .
    .
PROCESS-ERROR.
       (error routine)
```

Figure 3-4. Testing the Status Code in a COBOL Action Program

Testing Status Codes in a BAL Action Program

Testing method

After issuing a CALL macroinstruction, you test the ZA#PSC location in the program information block using a compare logical immediate (CLI) instruction and branch to the appropriate error routine that handles the specific error returned by IMS. If the status code is not zero, it is an error; if it's 1, it's an invalid key; and 4 indicates it's an I/O error. Figure 3-5 illustrates this coding. For status code values related to specific function calls, see Appendix D.

```
1        10     16
------------------------------------------
         ZG#CALL GET,(STATE,RECORD,SNKEY)
         CLI    ZA#PSC+1,0
         BNE    ERROR
          .
          .
          .
ERROR    MVC    OUTTEXT(4),NEWLINE
         CLI    ZA#PSC+1,1
         BNE    IOERROR
         MVC    OUTTEXT+4(L'MSGCON2),MSGCON2
         B      TERM
IOERROR  MVC    OUTTEXT+4(L'MSGCON3),MSGCON3
MSGCON2  DC     C'INVALID STATE NAME'
MSGCON3  DC     C'I/O ERROR'
```

Figure 3-5. Testing the Status Code in a BAL Action Program

Receiving Error Returns

Invalid request I/O errors

When IMS detects an error before it performs the CALL function, it returns the invalid request code 3. Errors detected after IMS passes control to data management, the control system, or ICAM, are considered unrecoverable and IMS returns the I/O error code of 4.

Accepting all error returns

You can accept all error returns or only status codes 1 and 2. If you want your action program to receive control after all error code returns, specify ERET=YES in the PROGRAM section of your configuration. Then, each time a CALL function is completed, you must test for all possible error status codes. (See Figures 3-4 and 3-5.)

Accepting limited error returns

When you want to receive only status codes 1 and 2 in your action program, take the ERET=NO default at configuration time. If IMS returns any other error status codes and you've taken the ERET default, IMS cancels the transaction, terminates your program, and sends an error message to the terminal.

3.6. OBTAINING ADDITIONAL STATUS INFORMATION (DETAILED-STATUS-CODE)

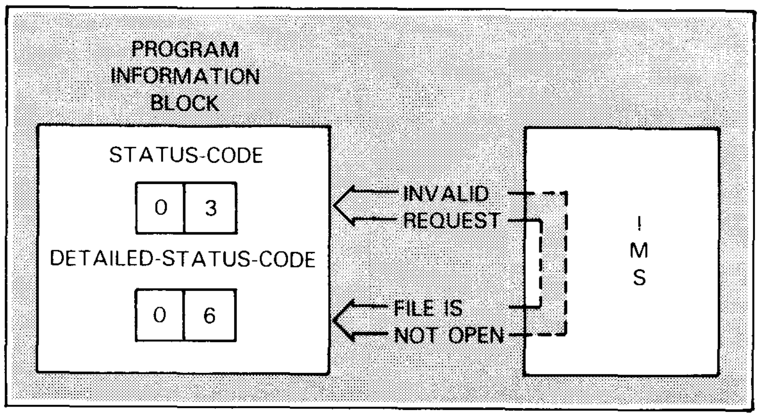
IMS sets value after CALL functions

When IMS returns status codes 3, 4, 6, or 7, it also returns a detailed status code. The DETAILED-STATUS-CODE field (ZA#PDSC) of the program information block provides additional data about the error.

Detailed status codes in Table D–3

Detailed Status Codes for Invalid Request (Status Code 3)

For example, you might receive a status code of 3 indicating invalid request. Invalid requests can occur for a number of reasons, so IMS returns a detailed status code to further explain the invalid request error. Table D–3 describes the detailed status codes that IMS can return with status code 3.



Detailed Status Codes for I/O Error (Status Code 4)

Codes for MIRAM files

Suppose IMS returns a status code of 4 (I/O error). If your action program uses MIRAM files, IMS returns a detailed status code composed of a data management error code (DMnn) and a subcode. For example, if you received a binary value equal to hexadecimal 16 in the first byte of the detailed status code, and a binary value equal to hexadecimal 01 in the second byte, you interpret this as a DM16 error code with a subcode of 01.

Interpreting DM error codes

The DM16 error message tells you that a partition table was not associated with the DTF at OPEN time and that there was a wrong key location. To determine the reason for the error, refer to the system messages programmer/operator reference, UP-8076 (current version); look up the error code under alphabetically prefixed messages and the error subcode under data management error message subcodes.

Codes for other file types

When your files are not MIRAM files and IMS returns an I/O error (status code 4), it returns a detailed status code from filenameC+2. By referring to the data management user guide, UP-8068 (current version) under the bit significance of filenameC byte 2, you'll get more detail on what caused the I/O error according to the bits set. See Table D-4 for these detailed status codes.

Detailed Status Codes for Internal Message Control Error (Status Code 6)

Detailed status codes in Table D-5

These detailed status codes pertain to messages sent via the SEND function call. See Table D-5 for these detailed status codes.

Detailed Status Codes for Screen Formatting Errors (Status Code 7)

Detailed status codes in Table D-6

When you use screen format services and errors occur, you receive a status code of 7 and IMS returns the detailed status codes that we show in Table D-6.

3.7. OBTAINING DEFINED RECORD STATUS (RECORD-TYPE)

When accessing defined records, the detailed status code has a different meaning.

Detailed status code redefinition (COBOL programs)

The COBOL program information block redefines the DETAILED-STATUS-CODE field as RECORD-TYPE, naming the first byte the PREDICTED-RECORD-TYPE and the second byte the DELIVERED-RECORD-TYPE.

Predicted record type

The predicted record type is a 1-byte alphanumeric indicator that tells defined record management the type of defined record to expect after a GET, GETUP, or INSERT function call. It also tells the type of the next sequential record expected after the SETL and GET function calls. You assign the value to the record type in the TYPE statement of your defined file definition. (See the information management system data definition and UNIQUE user guide, UP-9209 (current version).)

Delivered record type

The delivered record type is also a 1-byte alphanumeric indicator that tells the record type actually returned by defined record management to your action program.

BAL use of detailed status code

When your action program is in BAL and you access defined records, the values returned in the half-word detailed status code field, ZA#PDSC, represent a 1-byte alphanumeric value for the predicted record type followed by a 1-byte alphanumeric value for the delivered record type.

Additional information

Subsection 5.5 explains in greater detail the use and interpretation of the detailed status codes returned for defined record management.

3.8. IDENTIFYING SUCCEEDING ACTION PROGRAMS SUCCESSOR-ID)

Function

The SUCCESSOR-ID (ZA#PSID) field identifies the action program you want activated after the current action program terminates. It is a 6-byte field left-justified and zero-filled (i.e., X'F0').

Normal termination

Other termination

When your action programs terminate normally, you need not place a value in SUCCESSOR-ID. For programs ending in immediate internal, external delayed, or immediate internal succession, you must give the name of the next or succeeding action program to which control passes.

Subprogram succession

If your program calls a subprogram, you place the name of the subprogram you're calling into the SUCCESSOR-ID field. (For more details, see Section 8.)

3.9. USING SUCCESSOR-ID TO DISPLAY ERROR CODES

The SUCCESSOR-ID field can also be used to indicate specific errors to the terminal operator.

Interpreting error codes

When you issue a function call, IMS returns a status code and detailed status code. To find the cause of an error, you can look at your program information block and then check the status and detailed status code values documented in this manual. (See Appendix D.)

Displaying error code

Method

You have a programming alternative that gives you immediate error data at the originating terminal or console after the error occurs. First, determine the possible causes of errors and associate a successor-id with each possible error condition. Then, assign a termination code to each error type. When an error occurs, move your error termination code to the SUCCESSOR-ID field and terminate your action program abnormally by moving A or S to TERMINATION-INDICATOR (see 3.10).

Where displayed

IMS sends the termination error code from the SUCCESSOR-ID field to the originating terminal or console after abnormal termination occurs. By looking at the error code at the terminal, you can quickly find out the cause of error.

Conditions generating invalid request errors

Suppose you want to know quickly on which function call an invalid request occurred. If you retrieve records (GET), retrieve them for update (GETUP), switch from random to sequential mode (SETL), or from sequential back to random mode (ESETL); you have at least four possibilities for an invalid request error.

Example – setting up error status display

Figure 3–6 shows how you describe the error termination code for each function call your program uses and how to test for invalid request errors and move the appropriate indicators.

```
LINKAGE SECTION.
Ø1  PROGRAM-INFORMATION-BLOCK. COPY PIB74.
Ø1  INPUT-MESSAGE-AREA. COPY IMA74.
    .
    .
    .
PROCEDURE DIVISION             USING PROGRAM-INFORMATION-BLOCK
                                     INPUT-MESSAGE-AREA
                                     WORK-AREA
                                     OUTPUT-MESSAGE-AREA
                                     CONTINUITY-DATA-AREA.
    .
    .
    .
IMS-CALLS SECTION.
5ØØ-SETL.
    CALL 'SETL'   USING IMS-FILENAME
                        IMS-FILE-POSITION.
    IF STATUS-CODE IS 3 MOVE 'S' TO TERMINATION-INDICATOR
                        MOVE 'SETL  ' TO SUCCESSOR-ID.
5ØØ-EXIT.
    EXIT.
5Ø1-ESETL.
    CALL 'ESETL'  USING IMS-FILENAME.
    IF STATUS-CODE IS 3 MOVE 'S' TO TERMINATION-INDICATOR
                        MOVE 'ESETL ' TO SUCCESSOR-ID.
5Ø1-EXIT.
    EXIT.
5Ø2-GET.
    CALL 'GET'    USING IMS-FILENAME
                        IMS-RECORD-AREA
                        IMS-KEY.
    IF STATUS-CODE IS 3 MOVE 'S' TO TERMINATION-INDICATOR
                        MOVE 'GET   ' TO SUCCESSOR-ID.
5Ø2-EXIT.
    EXIT.
5Ø3-GETUP.
    CALL 'GETUP'  USING IMS-FILENAME
                        IMS-RECORD-AREA
                        IMS-KEY.
    IF STATUS-CODE IS 3 MOVE 'S' TO TERMINATION-INDICATOR
                        MOVE 'GETUP ' TO SUCCESSOR-ID.
5Ø3-EXIT.
    EXIT.
```

Figure 3–6. Testing Error Termination Codes and Moving them to Sucessor-id

Example – test and display

First, set up an item in your work area using a VALUE clause to associate it with the error code values that can be returned.

Then, in your procedure division after you perform a function call, test the status code for a code 3 (invalid request). If IMS returns an invalid request status code, move the appropriate error termination code to the SUCCESSOR-ID field and move an S to TERMINATION-INDICATOR to terminate the action program with a snap dump.

3.10. TERMINATING ACTION PROGRAMS (TERMINATION-INDICATOR)

Determines how action program terminates

IMS needs to know how your action program terminates. You choose the type of termination by moving one of six different values to the TERMINATION-INDICATOR (ZA#PSIND) field of the program information block. Termination actually occurs with the execution of the CALL 'RETURN' statement in COBOL programs or the ZG#CALL RETURN macroinstruction in BAL programs.

CALL RETURN ends program

Normal Termination (N Indicator)

Use N for last action program

In normal termination, the output message is sent to the terminal and all resources are released including the current action program. When you use several successive action programs to process messages, terminate the last action program normally by moving 'N' to the termination indicator.

Default value

Move N to indicator after immediate succession

IMS places a default value of N in the termination indicator. However, when more than one action program processes an action, as in immediate internal succession, you may have moved another value to the termination indicator before the final action program executes. Any value you moved there remains until changed by the successor action program. To be sure of obtaining a normal termination when needed in a series of action programs, move the normal (N) indicator to the termination indicator.

External Succession (E Indicator)

Function

The value E in the termination indicator tells your IMS that the current action program terminates in external succession. IMS sends the output message to the terminal, releases all resources including the current action program, and saves continuity data for use by the successor program. When IMS receives the next input message from the originating terminal, it schedules the succeeding action program as indicated in the SUCCESSOR-ID field.

When external succession is used

Sometimes you need to process more than one message to perform a transaction. The input of a second message depends upon the response a terminal operator gives to a previous output message. Using external succession in your action program allows the terminal operator to enter data required by the succeeding action program.

Example of use

Suppose your action programs are moving file data to the terminal screen. One action program might move menu data to the screen and succeed externally to a second action program that requires the terminal operator to enter a specific customer account number and choose one of the menu items (Figure 3-7).

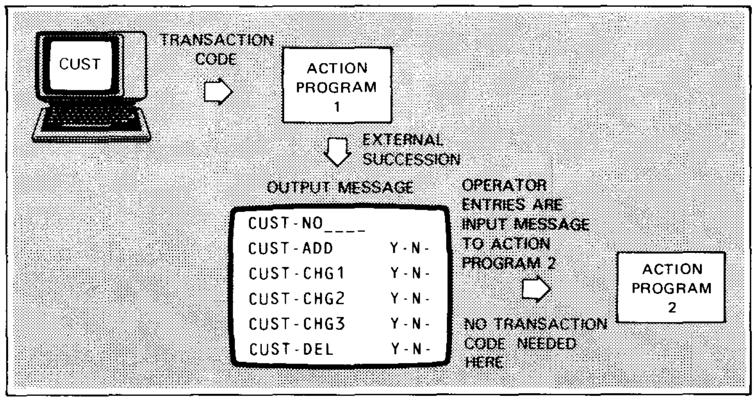


Figure 3-7. Using External Succession

Immediate Internal Succession (I Indicator)

Function

File availability

When you move the value I to the termination indicator, it tells IMS that you are terminating the current action program in immediate internal succession. This is characterized by the execution of two or more action programs during one action. In other words, several action programs execute without operator intervention to produce one output message. Because immediate internal succession involves only one action, all files accessed by the successor program must be available at the beginning of the initial program execution. To make files available, specify them in the configurator ACTION section.

Immediate internal succession process

When your current action program terminates, IMS:

- releases it;
- initiates the succeeding action program; and,
- passes all areas referenced by your current action program to the successor action program, without sending an output message to the terminal.

Successor program

The successor action program receives control of all interface areas used by the previous action program. Because IMS passes the contents of these areas on to the successor program, no deallocation or reallocation of resources is needed.

Example of use

In Figure 3-8, action program 1 outputs a menu and terminates in external succession, as in Figure 3-7. The terminal operator enters a customer number and chooses from the menu the operation he wants to perform. Action program 2 receives the input entries, determines which successor program is needed to process the particular menu selection, and terminates in immediate internal succession. Action program 3 performs the requested operation and sends a response to the terminal.

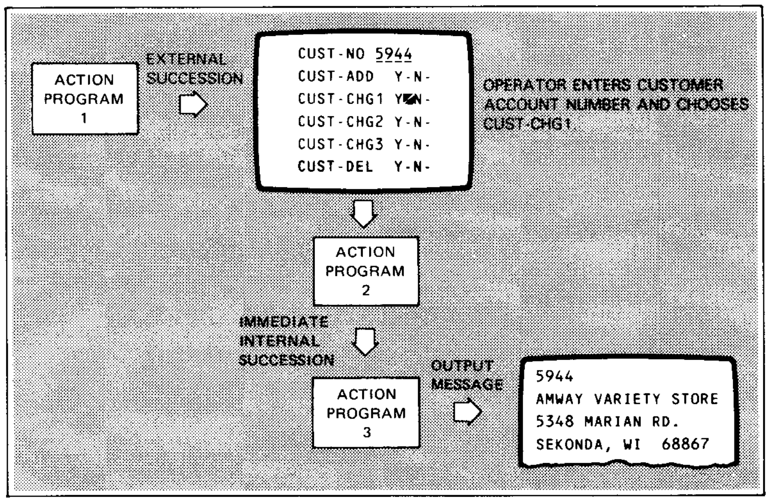


Figure 3-8. Using Immediate Internal Succession

Response time increased in multithread IMS

Avoid immediate internal succession in a multithread IMS environment. Because IMS holds storage areas from one action program to another and multithread IMS queues transactions, response time can be slowed.

Files allocated to first program

Another disadvantage of immediate internal succession is that the first action program must have all files allocated to it even if they are only being used by the succeeding program. This could waste main storage.

Delayed Internal Succession (D Indicator)

Function

You can terminate an action program in delayed internal succession by moving the value D to your termination indicator. When you terminate this way, it holds the output message of the current action program and queues it as the input message to the successor action program.

PIB FIELD: TERMINATION-INDICATOR

When used

Delayed internal succession process

In some situations during message processing, your main storage areas must be changed or different files must be accessed. At the same time, it may not be necessary for the terminal operator to receive an output message between action programs. In delayed internal succession, your first action program passes an output message internally to the successor action program that, in turn, uses the output message as its input. To complete the delayed internal succession transaction, your internal messages must be transferred as well as any data contained in the continuity data area.

Output messages are queued

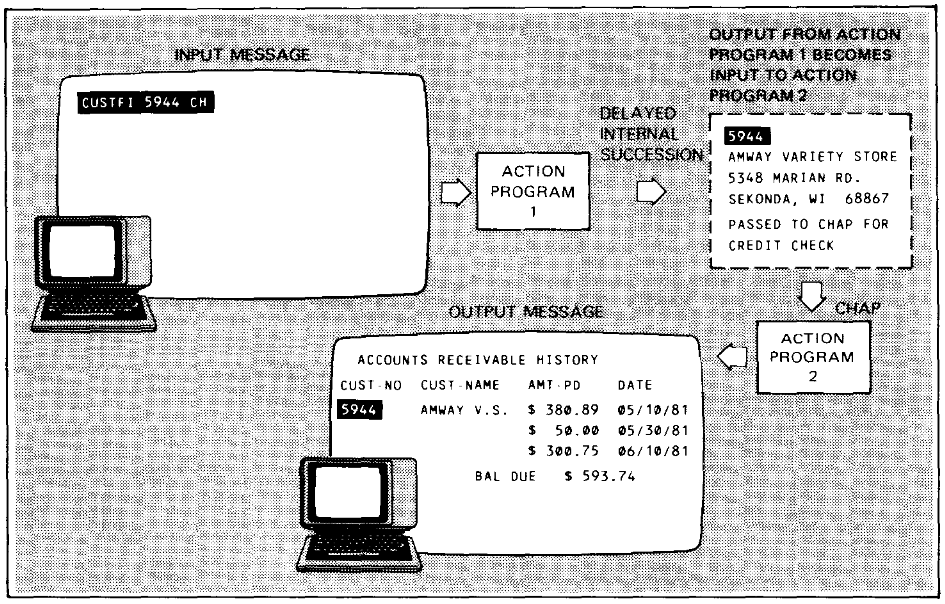
Instead of immediately sending the output message to a successor action program, IMS queues the message as input to the successor program you name in the SUCCESSOR-ID field.

Advantages

During action scheduling, IMS dynamically allocates I/O areas for all files referenced in the action. You can reduce I/O area requirements for actions by using delayed internal succession and then specifying frequently accessed files for one action, and less frequently accessed files for another action, in the ACTION section of your configuration.

Example of use

Suppose, for example, a terminal operator generally enters a transaction code and customer number to obtain data about a customer account. Occasionally he needs a credit history for a customer. This data is located on a less frequently accessed file and the input message containing the special code, CH, requires credit history data supplied by a different action program through delayed internal succession; Figure 3-9 illustrates this.


INPUT MESSAGE
ACTION PROGRAM 1
DELAYED INTERNAL SUCCESSION
OUTPUT FROM ACTION PROGRAM 1 BECOMES INPUT TO ACTION PROGRAM 2
CHAP
ACTION PROGRAM 2
OUTPUT MESSAGE

```
CUSTFI 5944 CH
```

```
5944
AMWAY VARIETY STORE
5348 MARIAN RD.
SEKONDA, WI  68867
PASSED TO CHAP FOR
CREDIT CHECK
```

```
ACCOUNTS RECEIVABLE HISTORY
CUST-NO  CUST-NAME   AMT-PD    DATE
5944     AMWAY V.S.  $ 380.89  05/10/81
                     $  50.00  05/30/81
                     $ 300.75  06/10/81
             BAL DUE   $ 593.74
```

Figure 3-9. Using Delayed Internal Succession

Abnormal Termination (A and S Indicators)

When an action program abnormally terminates, IMS:

Abnormal termination

- creates and sends an error message to the originating terminal;
- releases all resources; and
- rolls back files where applicable.

Abnormal termination messages

After abnormal termination, single-thread IMS sends an error message to the originating terminal in this format:

Single-thread error message format

```
TRANSACTION CANCELLED, TERM ID:id TRANS ID:id
TRANSCODE:code ACTION:name PROGRAM:name
error-description
```

Multithread error message format

Multithread IMS sends an error message in this format:

```
TRANSACTION ABORTED.TRANS ID:id. TERM ID:id.
TRANS CODE:code.CURR ACTION:name.CURR PROG:name.
REASON:error-description
```

You can find explanations of abnormal termination errors in the system messages programmer/operator reference, UP-8076 (current version).

Voluntary abnormal termination

In some cases, you may not want an action program to continue executing if certain requirements, such as file availability or input/output function error status codes, are not met. You can voluntarily cause an abnormal termination by moving A to the TERMINATION-INDICATOR field after testing these or other conditions.

Abnormal termination with snap dump

When you place the value S in the TERMINATION-INDICATOR field, IMS performs the same operations except, in addition, it provides a snap dump that can be very helpful in debugging action program errors. For a more detailed explanation of the snap dump, see Section 12.

IMS rolls back data files

Voluntary termination with either the A or S indicator causes IMS to roll back your data files to the previous rollback point (or the beginning of the transaction).

Using SUCCESSOR-ID with abnormal termination

When you use either A or S termination indicators to voluntarily terminate your action program, do not name an action program in the SUCCESSOR-ID field. Instead, move a termination code (often containing the status and detailed status codes) to the SUCCESSOR-ID field.

Displaying error codes at terminal

When an action program terminates, IMS clears the STATUS-CODE and DETAILED-STATUS-CODE fields to zeros, so you cannot determine the cause of errors resulting from CALL functions by examining a dump. However, you can obtain these codes at the terminal after abnormal termination by moving them to the SUCCESSOR-ID field. Be sure to convert the status and detailed status codes from binary to display format before moving them to SUCCESSOR-ID. When IMS receives the A or S termination indicators, it automatically moves the contents of the SUCCESSOR-ID field to the originating terminal or console. Thus, you send the status and detailed status codes to the terminal.

See Figure 3-6 for an example of error termination code descriptions and how to move them to the SUCCESSOR-ID field.

Involuntary Termination

Causes

Sometimes action programs terminate abnormally without your action program moving a value to the termination indicator. This type of termination is involuntary and occurs when IMS encounters an abnormal condition in processing action program requests. Two other causes of involuntary termination are the program-check and the timer-check error conditions.

Program-check interrupt (COBOL)

A program-check interrupt can occur, for example, when a COBOL action program describes a field as numeric and the data is not numeric. This is a data exception program check (error code, 07).

Program-check interrupt (BAL)

In a BAL action program, the program-check interrupt can occur if the action program uses the wrong registers to cover an area. This is an address exception program check (error code, 05).

Program check results

When a program check occurs, IMS terminates the current transaction, sends a transaction termination message to the terminal with the reason for abnormal termination, and provides a snap dump of the action program and its activation record. See the description of A and S termination indicators for the message formats and the OS/3 system messages programmer/operator reference, UP-8076 (current version) for their explanation. Also, see Section 12 for more about snap dumps.

Use snap dump to find cause

By looking at the contents of the snap dump, you can determine the error code and consequently, the type of error exception caused by the program check.

Timer-check interrupt

The timer-check interrupt occurs when an execution loop in an action program continues beyond a specified time limit. In single-thread IMS, a timer-check interrupt also occurs when an action program executes for longer than a specified time. The same operations result for timer check as for program check; i.e., IMS cancels the transaction, sends the error message to the terminal, and provides a snap dump.

Summary

Table 3-1 lists the termination indicator values, the type of termination each value selects, and the IMS operations performed.

Table 3-1. Termination Indicators

Termination types and IMS operations

To Terminate Current Action Program With:	Move To Termination-Indicator	IMS Operations
Normal Termination	N	Output message is sent to terminal. All resources, including current action program, are released. When you don't move a value to this field, normal termination is assumed.
External Succession	E	Output message is sent to terminal. Any data saved by this program is stored in the continuity data file. All resources, including current action program, are released. Successor action program is scheduled when another input message is received from originating terminal.
Delayed Succession	D	No output message goes to the terminal. Output message is queued as input message to successor action program. Any data saved by the program is stored in the continuity data file. All resources, including current action program are released. Successor action program is initiated by normal scheduling process.
Immediate Succession	I	No output message goes to the terminal. Current action program only is released. Successor action program is immediately initiated and IMS passes to it (intact) the interface areas of the predecessor program.
Abnormally without Snap Dump	A	Sends error message to originating terminal (includes value moved to successor-id). All resources are released. All files are rolled back.
Abnormally with Snap Dump	S	Same as A except a snap dump of current action program and its activation record is also provided. To get a snap dump, specify // OPTION DUMP, JOBDUMP, or SYSDUMP in your IMS job control stream.

3.11. HOLDING RECORD LOCKS (LOCK-ROLLBACK-INDICATOR)

Automatic record locking

While your action program is updating records, you don't want other action programs to access them. To protect records, IMS automatically locks them while your program is updating them. Normally, IMS releases these record locks at the end of each action.

Recovery requirements

What happens to the record your program is updating when abnormal termination occurs? To recover record images before the abnormal error occurred, IMS needs:

- the previous image of the record you were updating; and
- the rollback point.

Automatic rollback points

Normally, IMS establishes a rollback point at the end of each action.

Controlling locks and rollback

You can control the release or holding of record locks and the establishment of rollback points by moving values to the LOCK-ROLLBACK-INDICATOR field (ZA#PLRI). Two of the values indicate that you want record locks held or released (H or R) from action to action. The other two values indicate that you want to establish a new rollback point and release all locks (N) or reestablish a previous rollback point and release all locks (O).

Single-thread restriction

In single-thread IMS, you can use the H and R indicators only when you specify RECLOCK=YES in the OPTIONS section of the configuration.

Before-images saved

Position of rollback points

IMS saves before-images of records your program intends to update in the audit file. The audit file contains only the before-images of updates between established rollback points. Rollback points can occur at the end of an action or transaction depending on the termination indicator used jointly with the lock rollback indicator.

Table 3-2 summarizes the lock rollback indicators, their meanings, and applicable termination indicators.

Table 3-2. Summary of Record Locks and Rollback

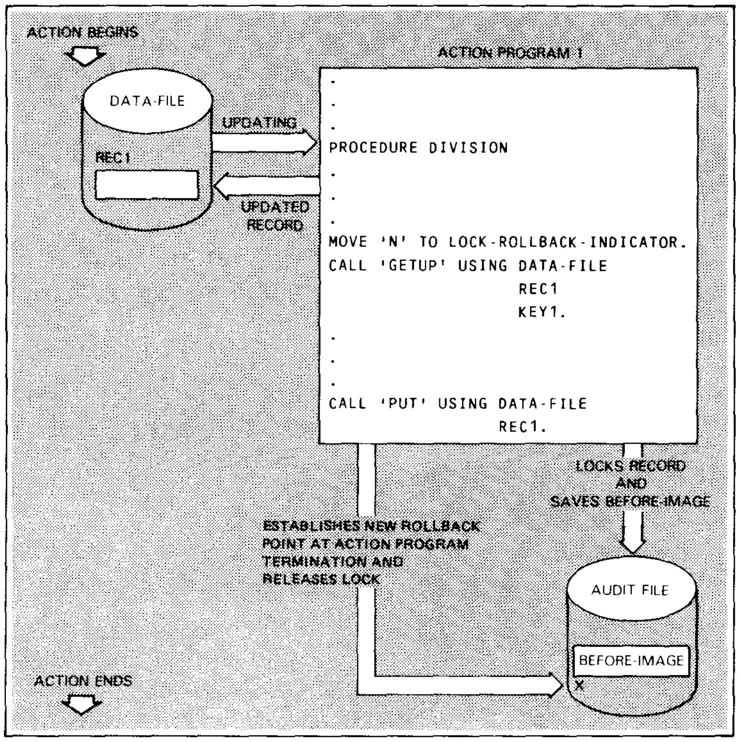
Lock rollback indicators and meanings

Lock-Rollback-Indicator	Termination-Indicator	Description
H	E, D	Holds all locks imposed by the current action program into the successor program.
R	E, D	Releases all pending locks set by the current action program. Update locks are held into the successor program.
N	E, D, N	Releases all locks for the transaction. Establishes a new rollback point in the audit file. This is the default value.
O	E, D, N	Releases all locks for the action or transaction. Rolls back all updates for this action or transaction. Establishes new rollback point in the audit file.

Establishing a New Rollback Point (N Indicator)

Default value

When you don't move a value into the LOCK-ROLLBACK-INDICATOR, IMS defaults to the value N. This value establishes a new rollback point in the audit file and releases all record locks. Figure 3-10 shows what happens to your data file and audit file when you use the N indicator.


ACTION BEGINS
ACTION PROGRAM 1
DATA-FILE
REC1
UPDATING
UPDATED RECORD
LOCKS RECORD AND SAVES BEFORE-IMAGE
ESTABLISHES NEW ROLLBACK POINT AT ACTION PROGRAM TERMINATION AND RELEASES LOCK
AUDIT FILE
BEFORE-IMAGE
X
ACTION ENDS

```
.
.
.
PROCEDURE DIVISION
.
.
.
MOVE 'N' TO LOCK-ROLLBACK-INDICATOR.
CALL 'GETUP' USING DATA-FILE
                   REC1
                   KEY1.
.
.
.
CALL 'PUT' USING DATA-FILE
                 REC1.
```

Figure 3-10. Using the N Lock Rollback Indicator

Purpose

Saving disk space

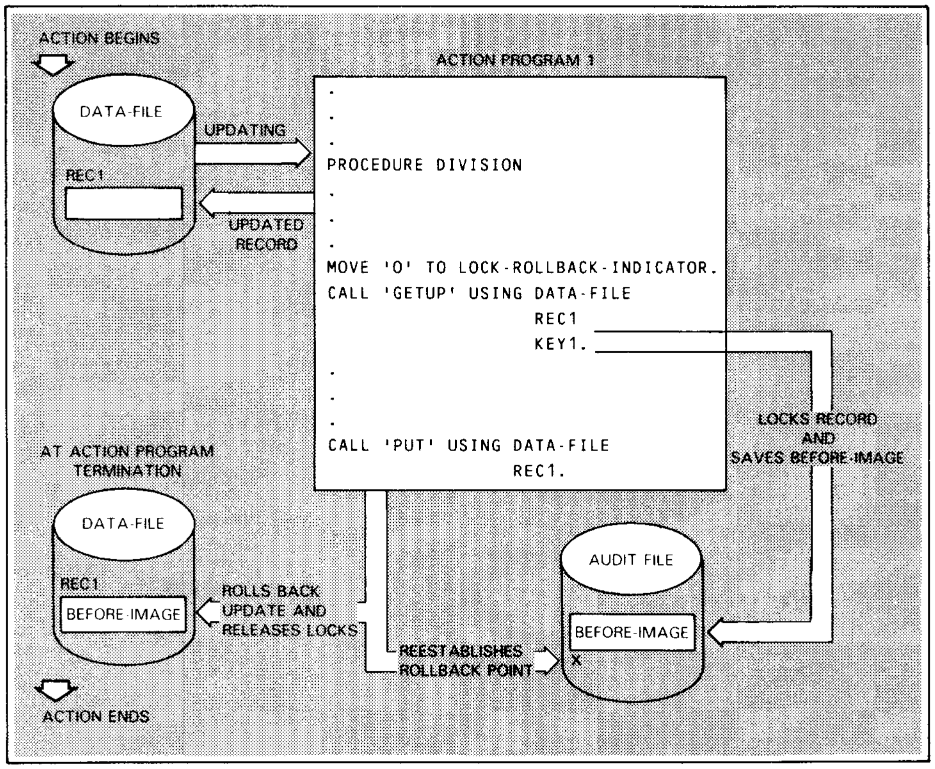
The N indicator is useful when your program involves long-running update transactions and terminates in external or delayed internal succession. By releasing locks, it frees records so that other action programs can access them. Also, establishing additional rollback points with more limited range can reduce the size of the audit file and save disk space.

Reestablishing the Old Rollback Point (O)

Function

Example

When you move the value O to the LOCK-ROLLBACK-INDICATOR field, you reestablish the old rollback point. In other words, this indicator tells IMS to roll back all updates to the previous rollback point and reestablish the rollback point. The O indicator also releases all record locks. Figure 3-11 shows what happens to your data file and audit file when you use the O lock rollback indicator.


ACTION BEGINS
DATA-FILE
REC1
UPDATING
UPDATED RECORD
ACTION PROGRAM 1
LOCKS RECORD AND SAVES BEFORE-IMAGE
AT ACTION PROGRAM TERMINATION
DATA-FILE
REC1
BEFORE-IMAGE
ROLLS BACK UPDATE AND RELEASES LOCKS
REESTABLISHES ROLLBACK POINT
AUDIT FILE
BEFORE-IMAGE
X
ACTION ENDS

```
.
.
.
PROCEDURE DIVISION
.
.
.
MOVE 'O' TO LOCK-ROLLBACK-INDICATOR.
CALL 'GETUP' USING DATA-FILE
                   REC1
                   KEY1.
.
.
.
CALL 'PUT' USING DATA-FILE
                 REC1.
```

Figure 3-11. Using the O Lock Rollback Indicator

Allowable termination indicators

The O lock rollback indicator is effective when you use it with the normal (N), external (E), or delayed (D) termination indicator.

When used

The O rollback indicator is useful when you want to actually roll back the data file to its contents before the current action's changes were made. This is helpful, for example, when the program updates a record invalidly and you want to assure validity by rolling back to a point before the invalid update occurred.

Holding Record Locks Across Actions (H Indicator)

Function

There are situations in which you may want to hold record locks until you make further changes in a succeeding action. To do this, you move the value H to the LOCK-ROLLBACK-INDICATOR field during the first action. IMS does not establish a rollback point when you use this indicator. It simply holds locks between actions. Figure 3-12 illustrates the use of H in the lock rollback indicator.

Allowable termination indicators

The H lock rollback indicator is effective only when you use it with the external (E) or delayed internal (D) termination indicators.

When used

Use the hold indicator, for example, when you want to prevent other action programs from accessing a record until the entire transaction finishes processing. You should avoid using the hold indicator when your transactions are long and when your programs are executing in a multithread environment. Holding locks across many actions in a multithread environment can cause deadlocks.

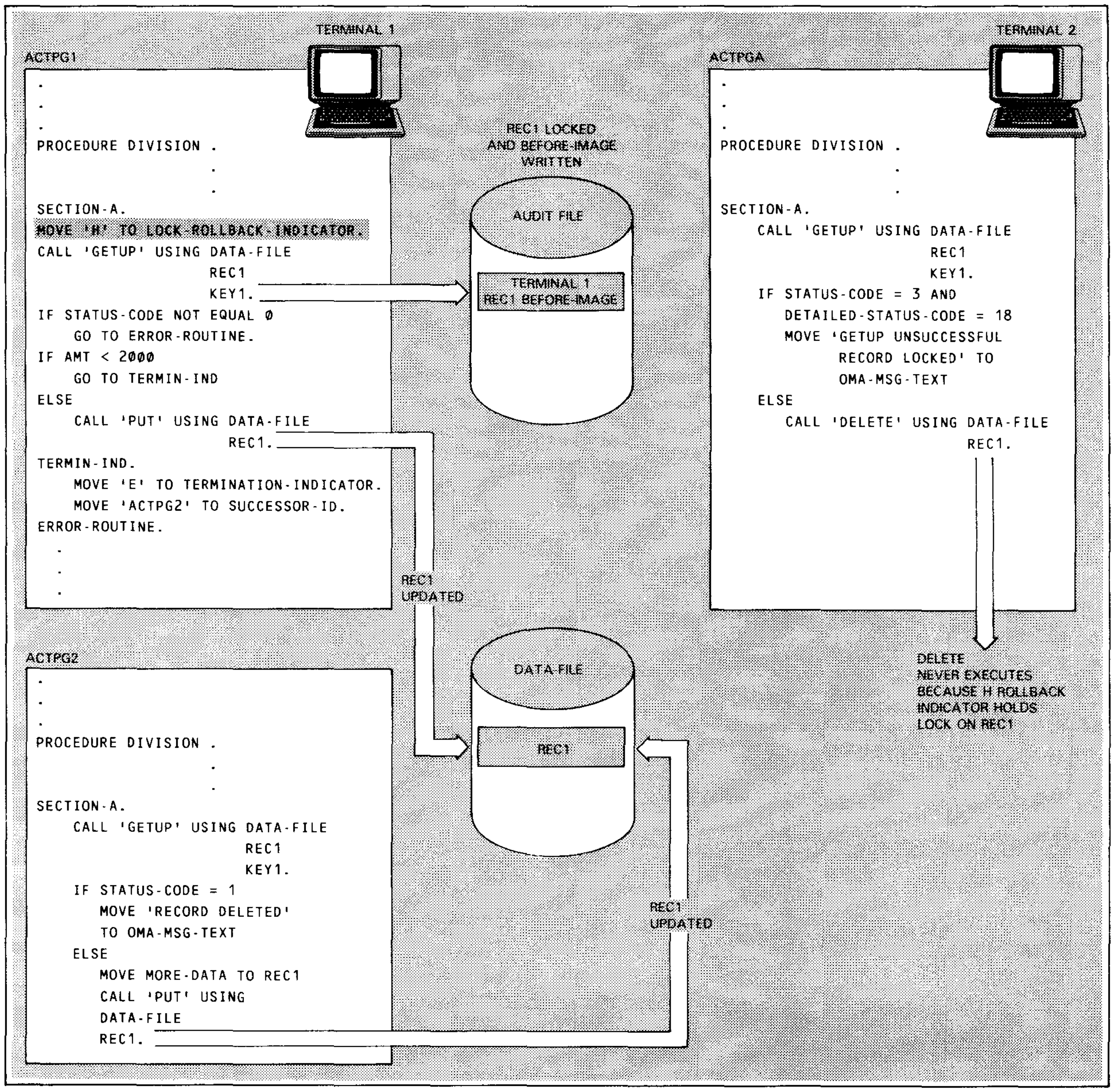
Releasing Record Locks for Pending Updates (R)

Function

Moving the value R to the LOCK-ROLLBACK-INDICATOR field allows the release of locks imposed on records that are pending update. Only records that were updated remain locked. IMS does not establish a rollback point or roll back updates when you use this indicator. Figure 3-13 shows the use of the R lock rollback indicator.

Allowable termination indicators

The R, like the H indicator, is effective only when you use it with the external (E) or delayed internal (D) termination indicators.


TERMINAL 1
ACTPG1
REC1 LOCKED AND BEFORE-IMAGE WRITTEN
AUDIT FILE
TERMINAL 1 REC1 BEFORE-IMAGE
TERMINAL 2
ACTPGA
REC1 UPDATED
ACTPG2
DATA FILE
REC1
REC1 UPDATED
DELETE NEVER EXECUTES BECAUSE H ROLLBACK INDICATOR HOLDS LOCK ON REC1

ACTPG1

```
.
.
.
PROCEDURE DIVISION .
                    .
                    .
SECTION-A.
MOVE 'H' TO LOCK-ROLLBACK-INDICATOR.
CALL 'GETUP' USING DATA-FILE
                   REC1
                   KEY1.
IF STATUS-CODE NOT EQUAL 0
    GO TO ERROR-ROUTINE.
IF AMT < 2000
    GO TO TERMIN-IND
ELSE
    CALL 'PUT' USING DATA-FILE
                      REC1.
TERMIN-IND.
    MOVE 'E' TO TERMINATION-INDICATOR.
    MOVE 'ACTPG2' TO SUCCESSOR-ID.
ERROR-ROUTINE.
  .
  .
  .
```

ACTPGA

```
.
.
.
PROCEDURE DIVISION .
                   .
                   .
SECTION-A.
    CALL 'GETUP' USING DATA-FILE
                       REC1
                       KEY1.
    IF STATUS-CODE = 3 AND
       DETAILED-STATUS-CODE = 18
       MOVE 'GETUP UNSUCCESSFUL
             RECORD LOCKED' TO
             OMA-MSG-TEXT
    ELSE
       CALL 'DELETE' USING DATA-FILE
                           REC1.
```

ACTPG2

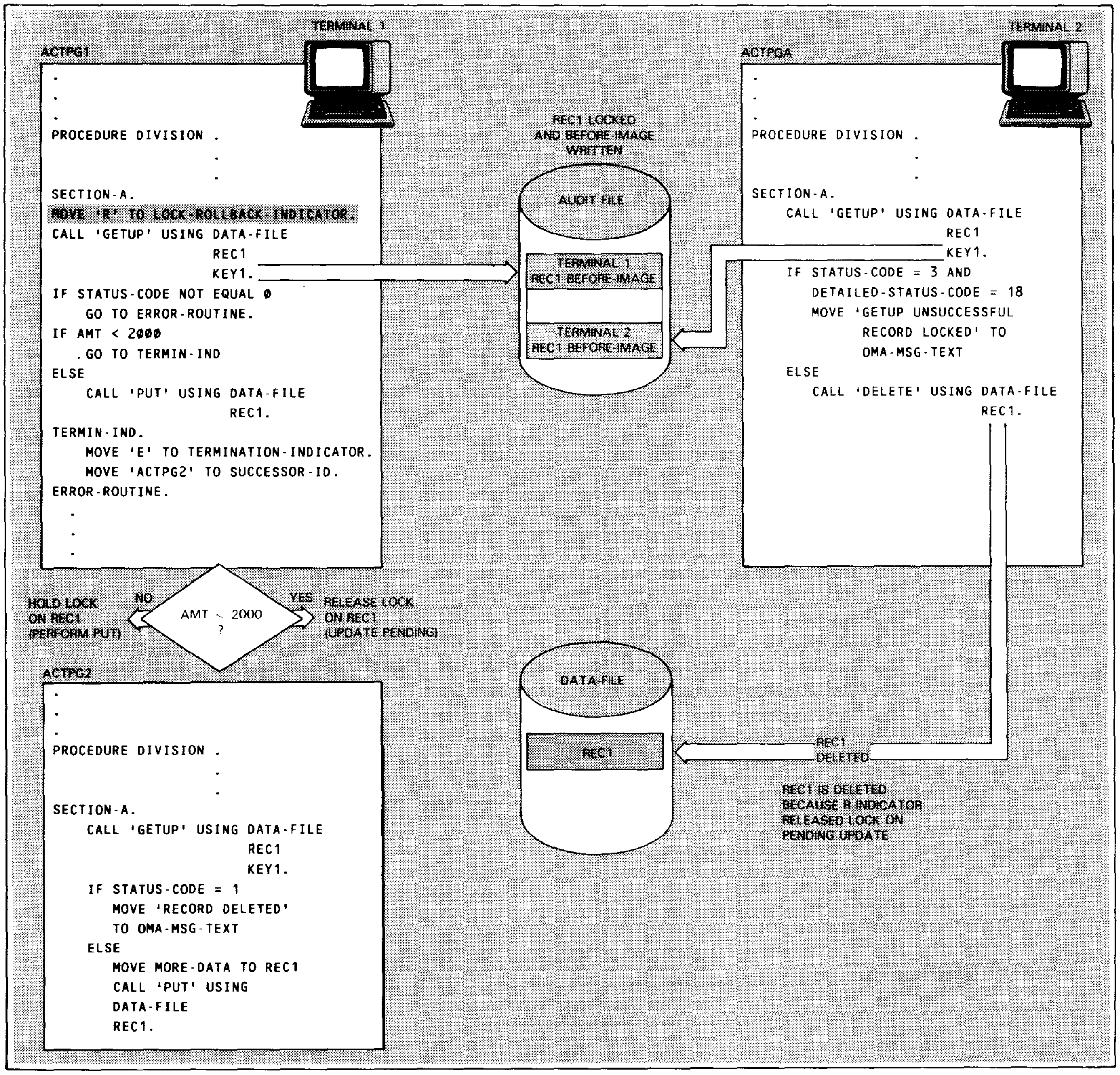
```
.
.
.
PROCEDURE DIVISION .
                   .
                   .
SECTION-A.
    CALL 'GETUP' USING DATA-FILE
                       REC1
                       KEY1.
    IF STATUS-CODE = 1
       MOVE 'RECORD DELETED'
       TO OMA-MSG-TEXT
    ELSE
       MOVE MORE-DATA TO REC1
       CALL 'PUT' USING
       DATA-FILE
       REC1.
```

Figure 3–12. Using the H Lock Indicator. *ACTPG1 on terminal 1 executes, terminates externally, and sets the H lock rollback indicator. ACTPGA on terminal 2 executes and attempts to obtain REC1. ACTPG1 holds the lock for REC1 and ACTPGA receives a status code of 3 and detailed status code of 18 (12_{16}) on single thread. For multithread IMS, the request for REC1 is queued. When ACTPG2 gets control, the delete operation has not been executed in ACTPGA. Thus, ACTPG2 updates REC1.*

When used

You generally use the release indicator when you've read a record for update and your program tests the record to determine whether or not it needs updating. If it doesn't need updating, you want to release the lock so other actions can access it. At the same time, you want to hold locks on records that you have updated, so they can be rolled back if an error occurs during the following action.

PIB FIELD: LOCK-ROLLBACK-INDICATOR



```
.
.
.
PROCEDURE DIVISION .
                   .
                   .
SECTION-A.
MOVE 'R' TO LOCK-ROLLBACK-INDICATOR.
CALL 'GETUP' USING DATA-FILE
                   REC1
                   KEY1.
IF STATUS-CODE NOT EQUAL 0
    GO TO ERROR-ROUTINE.
IF AMT < 2000
   .GO TO TERMIN-IND
ELSE
    CALL 'PUT' USING DATA-FILE
                     REC1.
TERMIN-IND.
    MOVE 'E' TO TERMINATION-INDICATOR.
    MOVE 'ACTPG2' TO SUCCESSOR-ID.
ERROR-ROUTINE.
    .
    .
    .
```

```
.
.
.
PROCEDURE DIVISION .
                   .
                   .
SECTION-A.
    CALL 'GETUP' USING DATA-FILE
                       REC1
                       KEY1.
    IF STATUS-CODE = 3 AND
       DETAILED-STATUS-CODE = 18
       MOVE 'GETUP UNSUCCESSFUL
             RECORD LOCKED' TO
             OMA-MSG-TEXT
    ELSE
       CALL 'DELETE' USING DATA-FILE
                           REC1.
```

```
.
.
.
PROCEDURE DIVISION .
                   .
                   .
SECTION-A.
    CALL 'GETUP' USING DATA-FILE
                       REC1
                       KEY1.
    IF STATUS-CODE = 1
       MOVE 'RECORD DELETED'
       TO OMA-MSG-TEXT
    ELSE
       MOVE MORE-DATA TO REC1
       CALL 'PUT' USING
       DATA-FILE
       REC1.
```

Figure 3–13. Using the R Lock Indicator. *ACTPG1 on terminal 1 executes, terminates externally, and sets the R lock rollback indicator. ACTPGA on terminal 2 executes and attempts to obtain REC1. If ACTPG1 holds the lock for REC1, ACTPGA receives a status code of 3 and detailed status code of 18 (12_{16}) on single thread. For multithread IMS, the request for REC1 is queued. When ACTPG2 gets control, if ACTPG1 released the lock on REC1, REC1 was deleted by ACTPGA and ACTPG2 issues a record deleted message. Otherwise, ACTPG2 updates REC1.*

Lock for Update Feature

Release locks at end of update

If you specify lock for update (LOCK=UP) for a particular file in the FILE section at configuration time, IMS releases record locks when updates are completed rather than at the end of an action. When you use this option, IMS does not save before-images in the audit file and does not roll back updates at abnormal termination. You can use the H indicator to hold locks on uncompleted updates into the next action.

3.12. TRANSACTION IDENTIFICATION (TRANSACTION-ID)

Function

When the terminal operator enters the first input message of a transaction, IMS places a unique message identifier in the TRANSACTION-ID (ZA#PTID) field of the program information block. IMS sets this value for all action programs that are part of the same transaction.

3.13. IDENTIFYING A DEFINED FILE (DATA-DEF-REC-NAME, DEFINED-FILE-NAME)

Function

When your action programs access defined files, the DATA-DEF-REC-NAME field (ZA#PDDRN) names the data definition record and the DEFINED-FILE-NAME (ZA#PDFN) field names the defined file or subfile. Both are 7-byte items, left-justified, and blank filled. The description of the defined file is contained in the data definition record in the named record file.

How IMS uses these fields

Assuming your current action program is not succeeding another, when IMS schedules an action it also:

- Moves the data definition record name specified by the DDRECORD configurator parameter into the DATA-DEF-REC-NAME field

- Moves the defined file name specified by the DFILE configurator parameter into the DEFINED-FILE-NAME field.

Accessing defined files in successive actions

When your action program terminates in external (E) or delayed internal (D) succession and the successor action program accesses a different defined file, you can pass the new data definition record name and defined file name to the succeeding action program by:

- moving the new names to DATA-DEF-REC-NAME field and DEFINED-FILE-NAME field in your action program (see Figure 3-14); or,
- moving binary zeros (LOW-VALUES) to both fields and allowing IMS to insert the data definition record name and defined file name specified in the configurator for the successor action (see Figure 3-15).

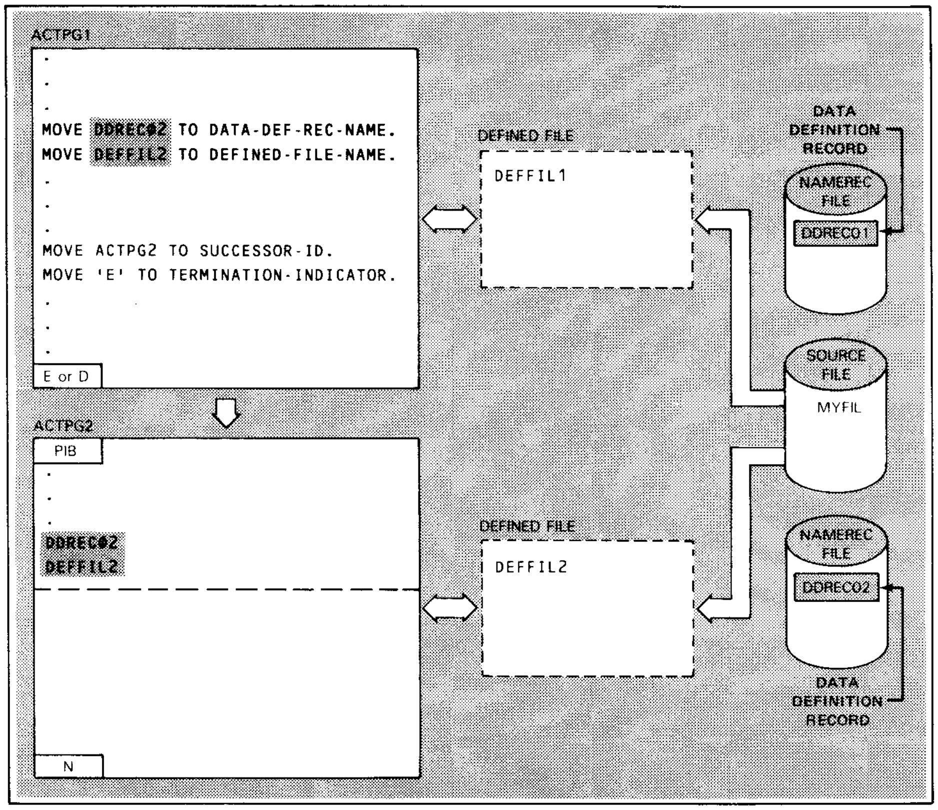


Figure 3-14. Action Program Passing Data Definition Record Name and Defined File Name to Successor Action Program

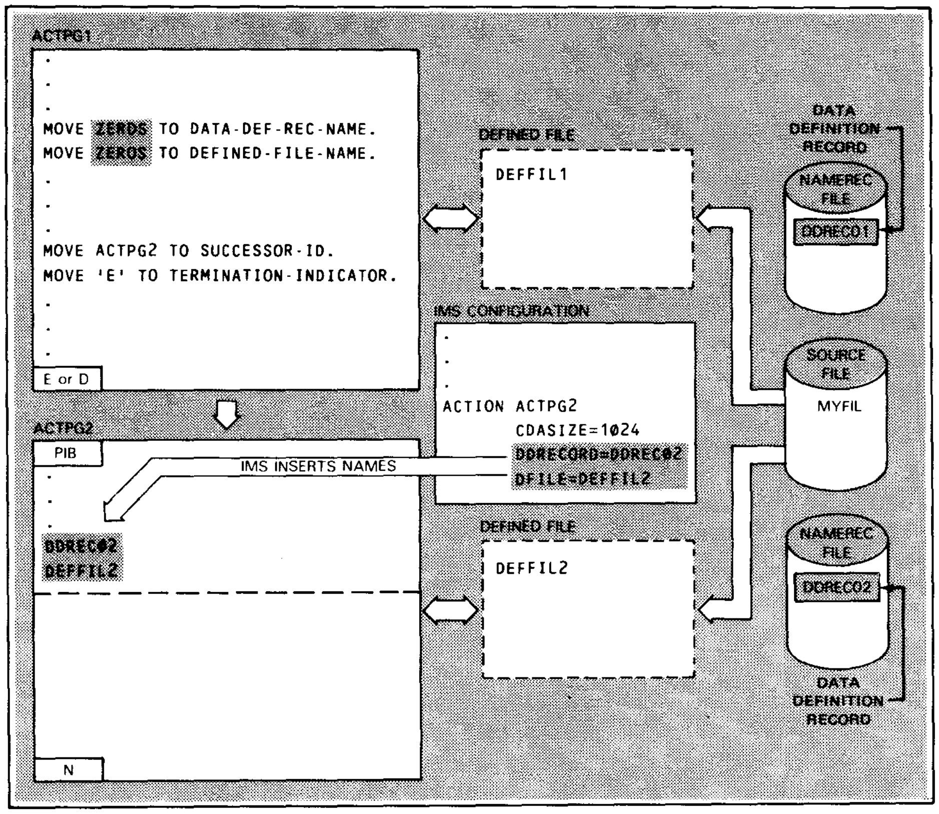


Figure 3-15. IMS Passing Data Definition Record Name and Defined File Name to Successor Action Program

Accessing conventional files in successive actions

When a succeeding action program accesses only conventional files, your action program should move zeros to the DATA-DEF-REC-NAME and DEFINED-FILE-NAME fields of the program information block. This allows the successor action to access records that have contributed to the defined file used by the previous action. Figure 3-16 shows you how clearing the DATA-DEF-REC-NAME and DEFINED-FILE-NAME fields frees the source file for use by the successor action program.

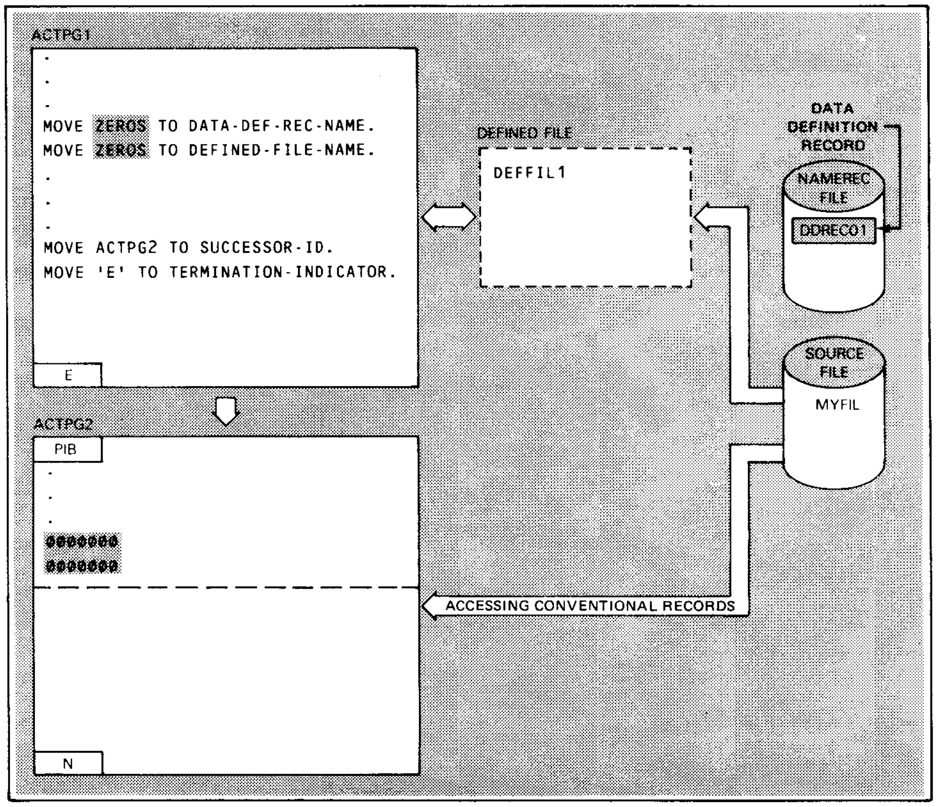

ACTPG1
.
.
.
MOVE ZEROS TO DATA-DEF-REC-NAME.
MOVE ZEROS TO DEFINED-FILE-NAME.
.
.
.
MOVE ACTPG2 TO SUCCESSOR-ID.
MOVE 'E' TO TERMINATION-INDICATOR.
E
ACTPG2
PIB
.
.
.
0000000
0000000
N
DEFINED FILE
DEFFIL1
DATA DEFINITION RECORD
NAMEREC FILE
DDREC01
SOURCE FILE
MYFIL
ACCESSING CONVENTIONAL RECORDS

Figure 3-16. Freeing Source File for Use by Successor Action

3.14. OBTAINING STANDARD MESSAGE SIZE (STANDARD-MSG-LINE-LENGTH, STANDARD-MSG-NUMBER-LINES)

Message line length

IMS places the configured values for standard message line length into the STANDARD-MSG-LINE-LENGTH field of the COBOL program information block or into the ZA#PMLL location of the BAL program information block. This field is a half-word binary value obtained from the CHRS/LIN configuration parameter.

Lines per message

Another value IMS inserts along with the standard message line length is the standard number of lines per message. This value is a half-word binary integer. In the COBOL program information block, this field is the STANDARD-MSG-NUMBER-LINES and the location in the BAL program information block is ZA#PMNL. IMS obtains the standard number of lines value from the LNS/MSG configuration parameter.

Use of fields

Your action program does not use these values. IMS uses them to determine the output message area size when OUTSIZE=STAN is configured.

3.15. SETTING WORK AREA VALUES (WORK-AREA-LENGTH, WORK-AREA-INC)

Work area length

IMS sets the WORK-AREA-LENGTH field (ZA#PWA), which is a half-word binary value indicating the length of the work area allocated to an action. IMS obtains this value from your configuration WORKSIZE parameter.

Adding work-area space

When your action program succeeds to another action program, additional work area space may be needed. Under multithread IMS only, your action program can set a half-word binary value in the WORK-AREA-INC field (ZA#PWAI) to increment the number of bytes for the work area. This value is additional to the value you specified in the WORKSIZE parameter.

3.16. SETTING CONTINUITY DATA VALUES (CONTINUITY-DATA-INPUT-LENGTH, CONTINUITY-DATA-OUTPUT-LENGTH, CONTINUITY-DATA-AREA-INC)

Passing continuity data

When you use delayed internal or external succession, you use the continuity data area. IMS passes data to a successor program via the continuity data record. The continuity data record contents begin with the first byte of the continuity data area.

Receiving continuity data

Input length field

A successor action program must define in the linkage section a continuity data area to access the contents of the continuity data record saved from its predecessor action. When your successor action program receives the data passed from the previous action, IMS places a half-word binary value into the CONTINUITY-DATA-INPUT-LENGTH field (ZA#PCDIN) to specify the length of the continuity data record passed to the current action by its predecessor action.

Output length field

IMS sets a half-word binary value in the CONTINUITY-DATA-OUTPUT-LENGTH field (ZA#PCDO) to specify the size of the continuity data area allocated to the current action. The value in this field at the end of the action indicates the number of bytes of data to be saved when the current action terminates in external or delayed internal succession.

Continuity data area increment

Before the current action terminates, IMS checks the CONTINUITY-DATA-AREA-INC field (ZA#PCDI) to determine if the continuity data area should be incremented for the next action. The half-word binary value set by the current action indicates the number of bytes needed to save additional data for a successor action. IMS adds this increment value to the length of the saved continuity data record and compares it to the length specified in the CDASIZE configurator parameter. The larger value then becomes the continuity data area size (CONTINUITY-DATA-OUTPUT-LENGTH field) for the succeeding action program. Note that continuity data area size should not exceed the track size of the disk used for the continuity data file.

Figure 3-17 illustrates how IMS establishes continuity data area input and output lengths and increment values.

3.17. SUCCESS-UNIT IDENTIFICATION (SUCCESS-UNIT-ID)

Obtaining date and time

Each time IMS schedules a new action, it identifies the beginning of the action or success-unit by sending a date and time stamp to the SUCCESS-UNIT-ID field (ZA#DTE and ZA#TME) of the program information block. When your action program requires an accurate date/time value, it should reference the TRANSACTION-DATE and TIME-OF-DAY fields in the SUCCESS-UNIT-ID of the COBOL program information block, or the ZA#DTE and ZA#TME locations in the BAL program information block.

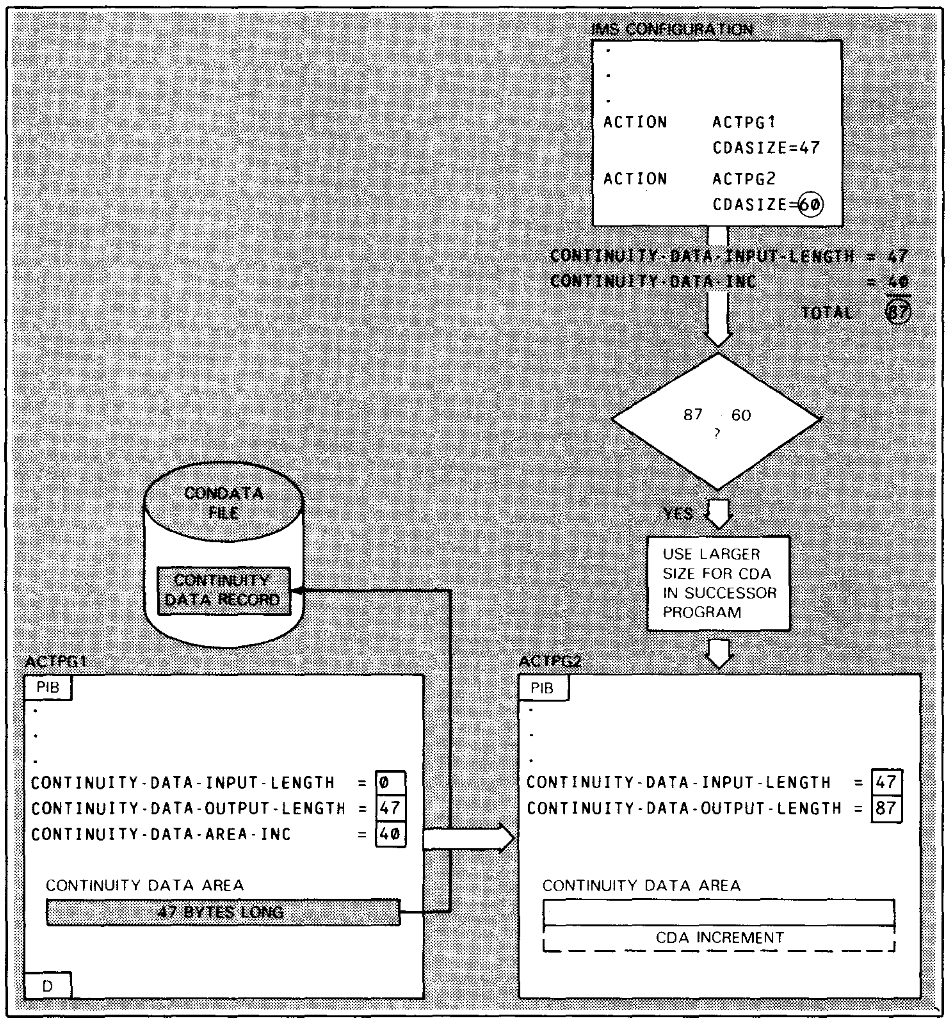

IMS CONFIGURATION
ACTION ACTPG1 CDASIZE=47
ACTION ACTPG2 CDASIZE=60
CONTINUITY-DATA-INPUT-LENGTH = 47
CONTINUITY-DATA-INC = 40
TOTAL 87
87 60 ?
YES
USE LARGER SIZE FOR CDA IN SUCCESSOR PROGRAM
CONDATA FILE
CONTINUITY DATA RECORD
ACTPG1
PIB
CONTINUITY-DATA-INPUT-LENGTH = Ø
CONTINUITY-DATA-OUTPUT-LENGTH = 47
CONTINUITY-DATA-AREA-INC = 4Ø
CONTINUITY DATA AREA
47 BYTES LONG
D
ACTPG2
PIB
CONTINUITY-DATA-INPUT-LENGTH = 47
CONTINUITY-DATA-OUTPUT-LENGTH = 87
CONTINUITY DATA AREA
CDA INCREMENT

Figure 3-17. Establishing Continuity Data Area Sizes

3.18. DETERMINING SOURCE TERMINAL CHARACTERISTICS (SOURCE-TERMINAL-CHARS)

Terminal type identification

When the terminal operator issues an input message, IMS sets an indicator in the SOURCE-TERMINAL-TYPE (ZA#TTTYP) field to identify the type of terminal sending the input message. Each 1-byte character value it sends identifies a device type as follows:

Value	Description
C	System console
F	UTS 400 in native mode (with or without character protect feature)
N	UTS 10, DCT 500, DCT 1000, or teletypewriter
P	UTS 400 in UNISCOPE mode with FCC protect feature
T	UTS 400 text editor
U	UTS 400 in UNISCOPE mode with character protect feature
V	UNISCOPE 100 or UNISCOPE 200
W	Workstation or UTS 20
3	IBM 3270
4	UTS 40

Message line length

After IMS identifies the terminal type that sent the input message, it places the message line length for the source terminal in the SOURCE-TERM-MSG-LINE-LENGTH (ZA#TMLL) field as a half-word binary value.

Lines per message

IMS also sets the number of lines per message for the source terminal type in the SOURCE-TERM-MSG-NUMBER-LINES field (ZA#TMNL).

Testing terminal type and message length

If you are going to send a message back to the source terminal, your action program can interrogate these fields to determine whether the terminal receiving your output message is capable of accommodating your message length. If your destination terminal is not the same as your source terminal, your program should use the STANDARD-MSG-LINE-LENGTH and STANDARD-MSG-NUMBER-LINES (see 3.14) when constructing the output message.

Example of use

Suppose you know that all terminals in your installation at Denver are UTS 400 devices and those in Pittsburgh are UNISCOPE 100 devices. Your COBOL action program could issue an IF statement as follows to determine from which city you are receiving input.

```
TERM-TEST.
     IF SOURCE-TERMINAL-TYPE EQUAL 'F'
               GO TO DENVER-ROUT
     ELSE IF SOURCE-TERMINAL-TYPE
               EQUAL 'V'
               GO TO PITTS-ROUT.
     GO TO ERR-ROUT.
NEXT-ROUT.
```

After your action program determines the source terminal type, the first statements in each city routine would compare the length of the output message you want to send back to the source terminal with the values in the SOURCE-TERM-MSG-LINE-LENGTH and SOURCE-TERM-MSG-NUMBER-LINES fields. For example:

```
DENVER-ROUT.
     IF SOURCE-TERM-MSG-LINE-LENGTH
     EQUAL 8Ø AND
     SOURCE-TERM-MSG-NUMBER-LINES < 24
     MOVE MSG-2 TO OMA-TEXT
     GO TO NEXT-ROUT.
PITTS-ROUT.
     IF SOURCE-TERM-MSG-LINE-LENGTH
     EQUAL 8Ø AND
     SOURCE-TERM-MSG-NUMBER-LINES < 12
     MOVE MSG-1 TO OMA-TEXT
     GO TO NEXT-ROUT.
ERR-ROUT.
```

Terminal attributes

IMS returns one of the following 1-byte character values in the SOURCE-TERM-ATTRIBUTES field of the COBOL program information block field or in the ZA#TATTR field of the BAL program information block:

Value	Description
A	Screen bypass and Katakana
K	Katakana character set
N	Nonvideo device
S	Screen bypass feature
Z	None of these attributes

3.19. DETERMINING REMOTE TRANSACTION STATUS (DDP-MODE)

Initiating remote transactions

You initiate remote transactions either from a terminal or from an action program. (See Section 9.) IMS supplies values in the DDP-MODE (ZA#DDPMD) field.

Remote transaction processing results

Two of the values (R and A) indicate how the remote transaction was initiated. The other two values (C and E) indicate the successful or unsuccessful completion of the remote transaction.

DDP-mode values

The 1-byte character values returned by IMS to describe remote transaction status are:

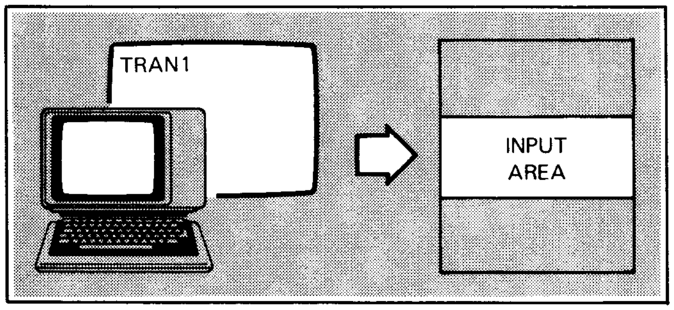
Value	Description
R	Operator-initiated remote transaction with operator or directory routing (Received by action programs processing remote transactions at the secondary IMS.)
A	Action-program initiated transaction (Received by action programs processing remote transactions at the secondary IMS.)
C	Unsuccessful remote transaction (Received by action programs that issued a CALL ACTIVATE function at the primary IMS.)
E	Successful completion of remote transaction (Received by action programs that issued a CALL ACTIVATE function at the primary IMS.)

4. Receiving Input Messages

4.1. NEED FOR INPUT MESSAGE AREA

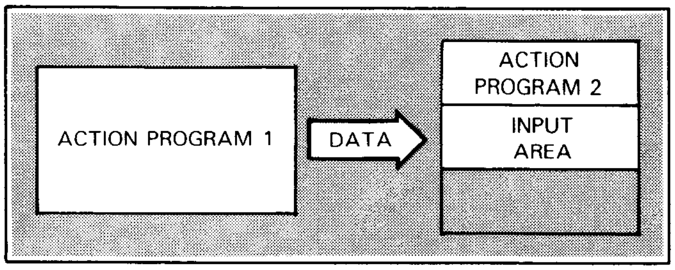
Input message area required

When a terminal operator enters a transaction code, your action program must define an input area to receive it. The same is true when the terminal operator enters an input message in response to an output message.



Receiving input message from previous program

When you use internal succession and pass data as input to the next action program, you must define an input area in the successor program to receive the data.



An input message area is always required in your action program because each action program must receive an input message, either via the terminal or action program succession, to produce an output response. Without an input message, no message processing is possible.

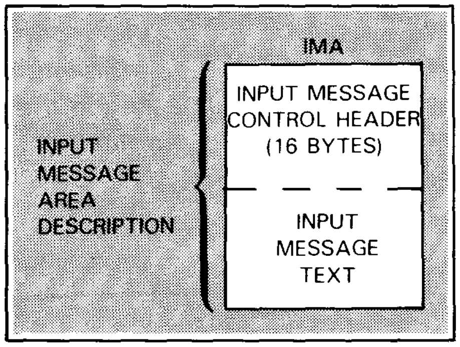
4.2. INPUT MESSAGE AREA CONTENTS

Control header

The first part of any input message area description is the 16-byte control header. Your program obtains the appropriate COBOL or BAL input message control header format from the copy library or macro library.

Input message text

The second part of the input message area description is the text of the message itself. The input message text consists of the input fields your program expects to receive either from the terminal operator or by succession from a previous action program.

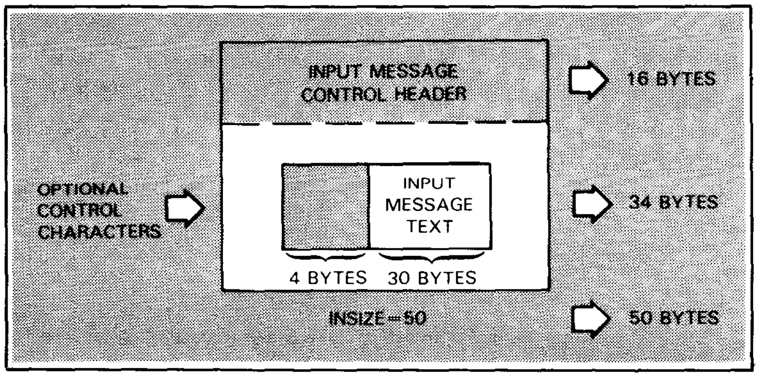

IMA
INPUT MESSAGE CONTROL HEADER (16 BYTES)
INPUT MESSAGE AREA DESCRIPTION
INPUT MESSAGE TEXT

4.3. SIZE OF INPUT MESSAGE AREA

Configuring input message area size

Receiving control characters in input message area

You tell IMS the size of your input message area at configuration time when you specify the INSIZE parameter in the ACTION section. The value given for the INSIZE parameter is the number of bytes in the input message header plus the message text length, including any control characters you expect to receive in your program. You receive control characters in your action program only when you specify EDIT=NONE in the configurator ACTION section.

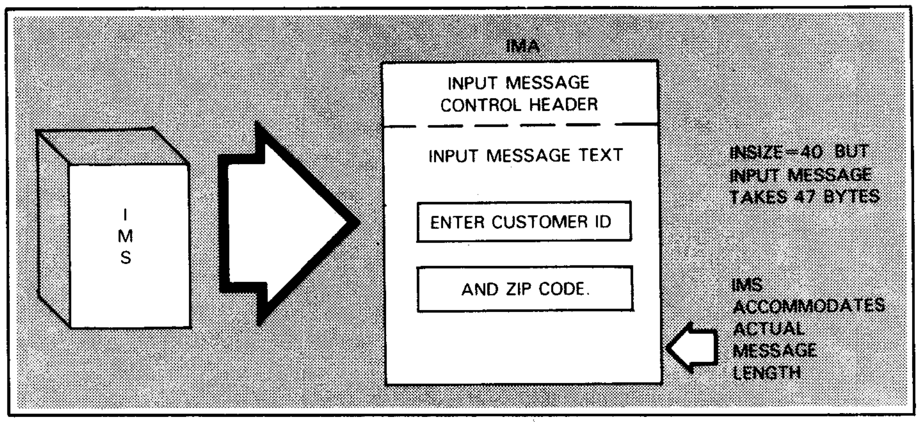


Specifying standard message size

Instead of specifying an input message area length on the INSIZE parameter, you can specify a standard message size (INSIZE=STAN); IMS allocates an area based on your CHRS/LIN and LNS/MSG parameter values in the GENERAL section.

Automatic space allocation

When you omit the INSIZE parameter or specify an inadequate amount of space for the input message area, IMS automatically allocates an area large enough to contain the actual input message.

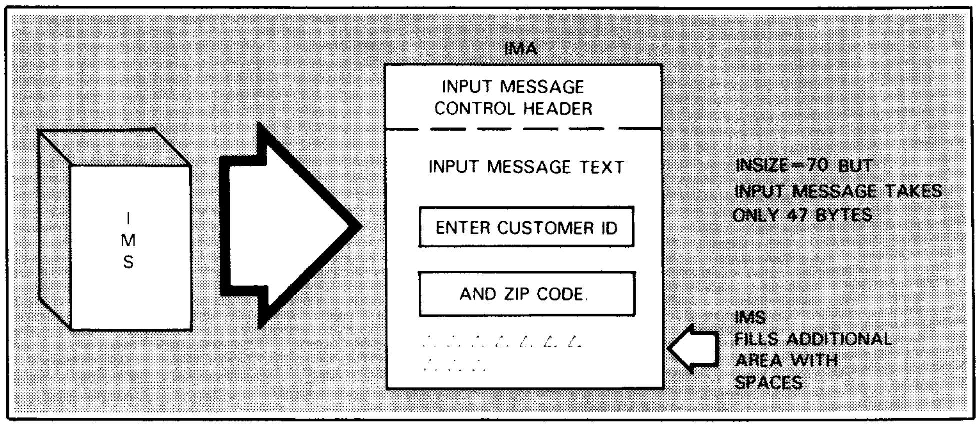

IMA
INPUT MESSAGE CONTROL HEADER
INPUT MESSAGE TEXT
ENTER CUSTOMER ID
AND ZIP CODE.
I M S
INSIZE=40 BUT INPUT MESSAGE TAKES 47 BYTES
IMS ACCOMMODATES ACTUAL MESSAGE LENGTH

Edit table consideration

Automatic space allocation doesn't occur if you use an edit table (EDIT=tablename), so you must specify the number of bytes for the input message area on the INSIZE parameter.

Overestimating IMA space

On the other hand, if you specify more space than is needed, IMS fills the balance of the area with blanks.


IMA
INPUT MESSAGE CONTROL HEADER
INPUT MESSAGE TEXT
ENTER CUSTOMER ID
AND ZIP CODE.
I M S
INSIZE=70 BUT INPUT MESSAGE TAKES ONLY 47 BYTES
IMS FILLS ADDITIONAL AREA WITH SPACES

Overestimating wastes storage

Note that you're wasting storage when you overestimate input message area size. If you're not using the edit table generator and you aren't sure of the input message area size, omit the INSIZE parameter and let IMS determine the input message area length.

4.4. COBOL ACTION PROGRAM INPUT MESSAGE AREA

Input Message Header Format

Format names for 1974 COBOL and extended COBOL

IMS supplies input message control header formats for extended COBOL and 1974 American National Standard COBOL. There is only a slight difference in their content. The COBOL input message header format is available in the IMS copy library under the name IMA for extended COBOL, or under the name IMA74 for 1974 American National Standard COBOL. Figure 4-1 shows the format of the 1974 COBOL input message area control header. Note the different data names of TODAY and HR-MIN-SEC fields for extended COBOL.

```
Ø1  INPUT-MESSAGE-AREA.
    Ø2  SOURCE-TERMINAL-ID         PIC X(4).
    Ø2  DATE-TIME-STAMP.
        Ø3  YEAR                   PIC 9(4)  COMP-4.
        Ø3  TODAY                  PIC 9(4)  COMP-4.  ①
        Ø3  HR-MIN-SEC             PIC 9(9)  COMP-4.  ②
    Ø2  TEXT-LENGTH                PIC 9(4)  COMP-4.
    Ø2  AUXILIARY-DEVICE-ID.
        Ø3  FILLER                 PIC X.
        Ø3  AUX-DEVICE-NO          PIC X.
```

NOTES:

① The name of this field in extended COBOL is DAY.

② The name of this field in extended COBOL is TIME.

Figure 4-1. 1974 COBOL Format for Input Message Area Control Header

Copying input message header

When you code your COBOL action program's linkage section, copy the input message area control header format into your action program from the copy library by using a COPY verb.

Input Message Text Description

Describing input message fields

The input message text description immediately follows the input message control header format. You describe the input message text expected by your program from the terminal or previous action program. In COBOL, describe the input message text as data items subordinate to the 01-level input message area description. The shaded area in Figure 4-2 shows the input message area control header formats generated by the COPY verb. Fields immediately following the shaded area represent the input text expected by the program.

Refer to the CSCAN action program example, PAYMT-3, in Appendix B for an example of this input text. When you copy the input message control header format from the copy library, all its fields are accessible to the CSCAN action program and can be referenced in the procedure division.

```
LINKAGE SECTION.
Ø1  P-I-B                   COPY PIB74.
Ø1  I-M-A.                                                  )
    Ø2  SOURCE-TERMINAL-ID  PIC X(4).                       |
    Ø2  DATE-TIME-STAMP.                                    |
        Ø3  YEAR            PIC 9(4)    COMP-4.             |      IMA
        Ø3  TODAY           PIC 9(4)    COMP-4.             |      CONTROL
        Ø3  HR-MIN-SEC      PIC 9(9)    COMP-4.             }      HEADER
    Ø2  TEXT-LENGTH         PIC 9(4)    COMP-4.             |      DESCRIPTION
    Ø2  AUXILIARY-DEVICE-ID.                                |
        Ø3  FILLER          PIC X.                          |
        Ø3  AUX-DEVICE-NO   PIC X.                          )
    Ø2  FILLER              PIC X(6).                               )
    Ø2  CUSTID              PIC X(6).                               |
    Ø2  FILLER              PIC X.                                  |  INPUT
    Ø2  MSG-PAY.                                                    }  MESSAGE
        Ø3  MSG-CHAR        PIC X   OCCURS 7   INDEXED BY 1.        |  TEXT
    Ø2  FILLER              PIC X.                                  )  DESCRIPTION
```

Figure 4-2. Sample COBOL Input Message Area Description

4.5. BAL ACTION PROGRAM INPUT MESSAGE AREA

Input Message Header Format

IMS supplies an input message area control header format for BAL action programs. It is in the form of a DSECT called by a macroinstruction in your action program. Figure 4-3 shows the format of the BAL input message area control header.

```
ZA#IMH    DSECT
*
*  INPUT MESSAGE HEADER
*
ZA#ISTID  DS    CL4                  SOURCE TERMINAL ID
ZA#IDTS   DS    XL8                  DATE/TIME STAMP
ZA#ITRID  EQU   ZA#IDTS,L'ZA#IDTS    UNIQUE TRANSACTION ID
ZA#IMHL   EQU   *-ZA#IMH             INPUT MESSAGE AREA HEADER LENGTH
ZA#ITL    DS    H                    TEXT LENGTH
          DS    CL1                        RESERVED FOR SYSTEM USE
ZA#IDEV   DS    CL1                        AUX DEVICE ID
*
*         EQUATES FOR ZA#IDEV
*
ZA#IDID1  EQU   C'1'                       DEVICE = AUX 1
ZA#IDID2  EQU   C'2'                       DEVICE = AUX 2
ZA#IDID3  EQU   C'3'                       DEVICE = AUX 3
ZA#IDID4  EQU   C'4'                       DEVICE = AUX 4
ZA#IDID5  EQU   C'5'                       DEVICE = AUX 5
ZA#IDID6  EQU   C'6'                       DEVICE = AUX 6
ZA#IDID7  EQU   C'7'                       DEVICE = AUX 7
ZA#IDID8  EQU   C'8'                       DEVICE = AUX 8
ZA#IDID9  EQU   C'9'                       DEVICE = AUX 9
&SYSECT   CSECT
          END
```

Figure 4-3. BAL Format for Input Message Area Control Header (ZA#IMH DSECT)

Calling input message header DSECT

You issue the ZM#DIMH macroinstruction in your BAL action program to generate inline the input message control header (ZA#IMH DSECT). If you don't want to see the ZM#DIMH macro expansion inline, use the PRINT NOGEN instruction before you issue the ZM#DIMH macroinstruction. Even though the input message control header fields are not seen in your program coding, they are still available and you can reference them in your program.

Describing input message fields

Immediately following the ZM#DIMH macroinstruction, you describe the input message text fields. Using define storage (DS) statements, you describe each field of your input message text. Figure 4-4 illustrates the macroinstruction to generate the input message control header format followed by the description of input message text expected from the terminal (transaction code and state name key). Refer to Appendix B for this example in the full context of the IMS state capital action program. Note that PRINT NOGEN is specified and the ZM#DIMH macroinstruction is not expanded inline. Nevertheless, this action program can still access any fields in the control header for values placed there by IMS.

```
1        10    16
         PRINT NOGEN    <-----------------  Suppresses inline
 .                                          macro expansion
 .
 .
*ACTIVATION RECORD DEFINITION
         ZM#DIMH    <---------------------  Makes IMA control header
                                            fields available.
TCODE    DS    X             TRANSACTION CODE\
         DS    X             SPACE            } Input Message Text
SNKEY    DS    XL4           STATE NAME KEY  /
```

Figure 4–4. Sample BAL Input Message Area Description

4.6. CONTENTS OF INPUT MESSAGE AREA CONTROL HEADER

The header format identifies the terminal that sent the input message, the date and time when the message was sent, the length of the input text, and whether or not an auxiliary device transmitted input to the action program. Figure 4–5 shows some of the questions about input messages that the input message control header answers when IMS sets values in the control header fields. Subsections 4.7 through 4.10 describe input message header fields.

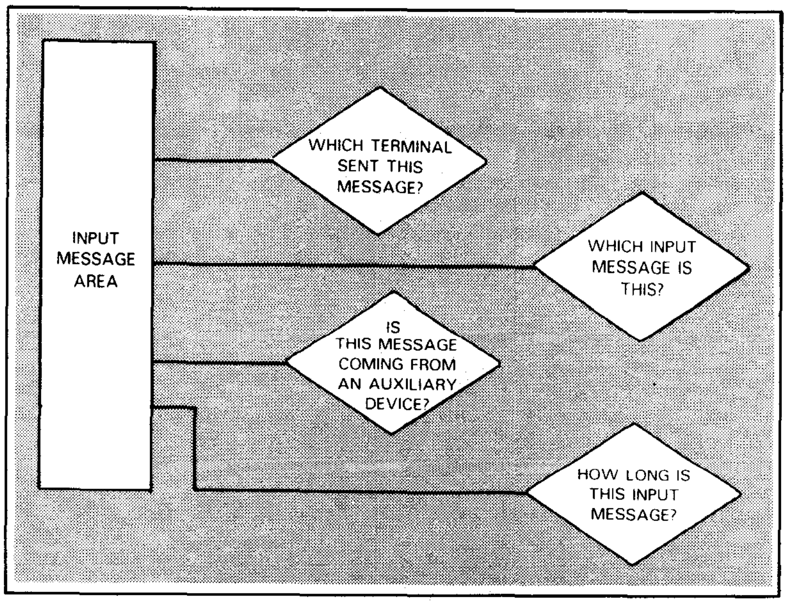

INPUT MESSAGE AREA
WHICH TERMINAL SENT THIS MESSAGE?
WHICH INPUT MESSAGE IS THIS?
IS THIS MESSAGE COMING FROM AN AUXILIARY DEVICE?
HOW LONG IS THIS INPUT MESSAGE?

Figure 4–5. Answers to Input Message Processing Questions

4.7. IDENTIFYING THE SOURCE TERMINAL (SOURCE-TERMINAL-ID)

Source terminal identification

The SOURCE-TERMINAL-ID (ZA#ISTID) field specifies a 1- to 4-byte name of the terminal that originated the input message. Your action program may need to check this field to determine which terminal sent a particular input message. This terminal name is the same name specified for the terminal in the ICAM network definition and in a TERMINAL section of the configuration (Figure 4–6).

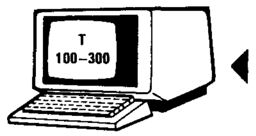
```
                     ICAM NETWORK DEFINITION
IMS1     CCA    TYPE=(GBL,,S),GAWAKE=YES,SAVE=YES,                      X
                FEATURES=(OPCOM,TRACEMAX,OUTDELV)
         BUFFERS 10,512,2,ARP=20
WOLO     LOCAP  TYPE=(TCI),LOW=MAIN,MEDIUM=MAIN,HIGH=MAIN
LNE1     LINE   DEVICE=(LWS)
WS1      TERM   ADDR=(312),FEATURES=(LWS),LOW=MAIN,INPUT=(YES),         X
                MEDIUM=MAIN,HIGH=MAIN,TCTUPD=YES
LNE2     LINE   DEVICE=(LWS)
WS2      TERM   ADDR=(313),FEATURES=(LWS),LOW=MAIN,INPUT=(YES),         X
                MEDIUM=MAIN,HIGH=MAIN,TCTUPD=YES
LNE3     LINE   DEVICE=(LWS)
WS3      TERM   ADDR=(314),FEATURES=(LWS),LOW=MAIN,INPUT=(YES),         X
                MEDIUM=MAIN,HIGH=MAIN,TCTUPD=YES
LNE4     LINE   DEVICE=(LWS)
WS4      TERM   ADDR=(315),FEATURES=(LWS),LOW=MAIN,INPUT=(YES),         X
                MEDIUM=MAIN,HIGH=MAIN,TCTUPD=YES
PRC1     PRCS   LOW=MAIN
         ENDCCA
                           IMS CONFIGURATION
NETWORK
  .
  .
  .
TERMINAL WS1    UNSOL=ACTION
TERMINAL WS2    UNSOL=ACTION
TERMINAL WS3    UNSOL=ACTION
TERMINAL WS4    UNSOL=ACTION
TRANSACT MENU   ACTION=JAMENU
TRANSACT SIGN   ACTION=JASIGN
ACTION   JAMENU CDASIZE=1024  EDIT=NONE   MAXSIZE=12000
                OUTSIZE=4096  WORKSIZE=1024
                FILES=SYSCTL,CUSTMST,XREF1,XREF2
ACTION   JASIGN CDASIZE=1024  EDIT=NONE   MAXSIZE=12000
```

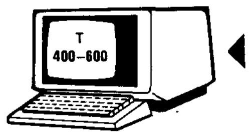
Figure 4–6. Identifying the Source Terminal to ICAM and the Configurator

Testing source terminal identification

Suppose your action program processes input messages differently, depending on which terminal sent the message. Before it can decide how to process the message, your program needs to check the name of the source terminal that sent the input message.

Let's say that if your program receives a message from source terminals T100 through T300, it performs routine A. On the other hand, if your program receives a message from source terminals T400 through T600, it performs routine B. Your program simply interrogates the SOURCE-TERMINAL-ID field of the input message header as shown in Figure 4-7 and processes the input message according to the values placed in the SOURCE-TERMINAL-ID field.





```
100-TERM-TEST.
    IF SOURCE-TERMINAL-ID GREATER THAN OR EQUAL 'T100'
              AND LESS THAN OR EQUAL TO 'T300'
          PERFORM ROUT-A
    ELSE IF SOURCE-TERMINAL-ID GREATER THAN OR EQUAL 'T400' AND
              LESS THAN OR EQUAL 'T600'
          PERFORM ROUT-B.
    GO TO ERR-ROUT.
ROUT-A.
    .
    .
    .
ROUT-B.
    .
    .
    .
ERR-ROUT.
```

Figure 4-7. Interrogating the SOURCE-TERMINAL-ID Field

4.8. IDENTIFYING THE ACTION (DATE/TIME STAMP)

When input message received

Identifying specific input message

When IMS receives an input message, it places the date and time as a binary value in the DATE-TIME-STAMP field (ZA#IDTS) of your input message header. The first half word of the field contains the year; the second half word of the field contains the Julian day. The second word contains a sequence number unique to this input message. The date/time stamp is used for recovery purposes and not for determining the time of day.

Using date/time stamp

IMS uses this field to distinguish actions. Each time IMS receives an input message, it identifies the action via this date/time stamp. If you need the accurate date or time in your action program, you should interrogate the TRANSACTION-DATE and TIME-OF-DAY under SUCCESS-UNIT-ID in the program information block.

Testing specific input messages

The last word of the DATE-TIME-STAMP field contains a unique sequence number represented as a binary value for each input message processed. This sequence number is useful at error recovery time. In the error routine, your action program may choose to process messages 1 to 100 in one manner, and messages 101 to 200 in another manner. Thus, Figure 4-8 shows coding that tests the HR-MIN-SEC (ZA#IDTS) field to determine the input message sequence number on which the error occurred and processes it accordingly.

Note that when testing the DATE-TIME-STAMP field, all comparisons must be made in binary. Be sure to compare DATE-TIME-STAMP with values you define in working storage as binary items.

```
WORKING-STORAGE SECTION.
77   ONE-HUNDRED             PIC 9(4)    COMP-4    VALUE 100.
LINKAGE SECTION.
    .
    .
    .
PROCEDURE DIVISION           .
                             .
                             .
MSG-SEQ-TEST.
       IF HR-MIN-SEC LESS THAN OR EQUAL ONE-HUNDRED
             PERFORM ERR-ROUT-1
       ELSE  IF HR-MIN-SEC GREATER THAN ONE-HUNDRED
             PERFORM ERR-ROUT-2.
ERR-ROUT.
    .
    .
    .
ERR-ROUT-1.
    .
    .
    .
ERR-ROUT-2.
```

Figure 4-8. Testing Input Message Sequence

4.9. OBTAINING INPUT MESSAGE TEXT LENGTH (TEXT-LENGTH)

Input message length

Once the terminal operator enters an input message, or a previous action program passes input data to a successor action program, IMS places a binary half-word value indicating the input message length plus four bytes for the TEXT-LENGTH (ZA#ITL) field itself into the TEXT-LENGTH field.

Using TEXT-LENGTH field

Your action program may want to print out all input messages for a day's transactions. Suppose the input messages received by your action program can vary in length and you plan to write them as variable-length unblocked records to a sequential file.

The value IMS places in the TEXT-LENGTH field contains the length of the input message text your action program receives plus four bytes for the TEXT-LENGTH field. Each time your program receives an input message, it must first subtract four bytes from the value in TEXT-LENGTH. Your program then compares the resulting value with the different input message lengths that the program expects. When the program determines which size message was received, it moves TEXT-LENGTH minus four bytes to the record length field of your record area description in the work area. Finally, it moves the appropriate input message to the work area and writes it to the sequential file. Figure 4–9 shows the coding to test the TEXT-LENGTH field in the input message area. Note that you must subtract a binary 4 from the COMP-4 TEXT-LENGTH field, and the record length field in the work area must also be a binary value.

Qualifying TEXT-LENGTH field

When you access the TEXT-LENGTH field in the input message area, your COBOL program must qualify the TEXT-LENGTH field by identifying it as a part of the input message area header; i.e., TEXT-LENGTH IN INPUT–MESSAGE-AREA.

```
WORKING-STORAGE SECTION.
77   FOUR                     PIC 9    COMP-4      VALUE 4.
77   FORTY                    PIC 99   COMP-4      VALUE 4Ø.
LINKAGE SECTION.
Ø1   INPUT-MESSAGE-AREA.       COPY IMA74.
     Ø5  MSG-IN-1.
         1Ø  TRANS-CODE-1       PIC X(5).
         1Ø  IN-MSG-TEXT-1      PIC X(35).
     Ø5  MSG-IN-2   REDEFINES MSG-IN-1.
         1Ø  IN-MSG-TEXT-2.
             2Ø  TRANS-CODE-2       PIC X(5).
             2Ø  TEXT-2             PIC X(2Ø).
         1Ø  FILLER                 PIC X(15).
Ø1   WORK-AREA.
     Ø5  IN-MSG-REC.
         1Ø  REC-LEN                PIC 9(4)    COMP-4.
         1Ø  MSG-TEXT.
             2Ø  MSG-1              PIC X(25).
             2Ø  FILLER             PIC X(15).
Ø1   OUTPUT-MESSAGE-AREA.      COPY OMA74.
        .
        .
        .
PROCEDURE DIVISION             USING PROGRAM-INFORMATION-BLOCK
                               INPUT-MESSAGE-AREA
                               WORK-AREA
                               OUTPUT-MESSAGE-AREA.
        .
        .
        .
IN-MSG-MOVE
     MOVE TEXT-LENGTH IN INPUT-MESSAGE-AREA TO REC-LEN.
     SUBTRACT FOUR FROM TEXT-LENGTH IN INPUT-MESSAGE-AREA.
     MOVE SPACES TO MSG-TEXT.
      IF TEXT-LENGTH IN INPUT-MESSAGE-AREA EQUAL FORTY
          MOVE MSG-IN-1 TO MSG-TEXT
     ELSE MOVE IN-MSG-TEXT-2 TO MSG-1.
     CALL 'PUT'  USING IN-MSG-FIL     IN-MSG-REC.
     IF STATUS-CODE  > Ø  GO TO ERR-ROUT.
        .
        .
        .
ERROR-ROUT.
```

Figure 4-9. Testing the TEXT-LENGTH Field

4.10. IDENTIFYING AUXILIARY DEVICES (AUXILIARY-DEVICE-ID)

Auxiliary device identification

When an input message is received from an auxiliary device, IMS places the number of the auxiliary device in the second byte of the AUXILIARY-DEVICE-ID (ZA#IDEV) field, AUX-DEVICE-NO. Auxiliary device values range from 1 to 9. The first byte is reserved for system use.

Obtaining auxiliary device number

Just as your action program can check the source terminal identification, it can also check auxiliary device identification. To determine which auxiliary device sent the input message, your action program interrogates the AUX-DEVICE-NO field.

Example of use

COBOL coding

Suppose your action program logic depends upon which auxiliary device transmitted a particular input message. If your input message came from auxiliary device 1, your program performs one routine. If device 2 transmitted the message, your program performs another routine. Figure 4–10 shows the procedure division coding used to check the number of the auxiliary device that sent the input message to your action program.

```
AUX-DEV-TEXT.
       IF AUX-DEVICE-NO EQUAL 1
              PERFORM ROUT-A
       ELSE IF AUX-DEVICE-NO EQUAL 2
              PERFORM ROUT-B.
       GO TO ERR-ROUT.
ROUT-A.
   .
   .
   .
ROUT-B.
   .
   .
   .
ERR-ROUT.
```

Figure 4–10. Testing the AUX-DEVICE-NO Field in a COBOL Action Program

BAL coding

The same test can be performed in a BAL action program by using the CLI instruction and branching to the appropriate routine to handle the processing of a message from either auxiliary device 1 or 2. Figure 4-11 shows this coding for a BAL action program.

```
1           10     16
-------------------------------------
            CLI    ZA#IDEV+1, C'1'
            BE     ROUTA
            CLI    ZA#IDEV+1, C'2'
            BE     ROUTB
ROUTA
            .
            .
            .
ROUTB
            .
            .
            .
```

Figure 4-11. Testing the AUX-DEVICE-NO Field in a BAL Action Program

4.11. INPUT MESSAGE TEXT

Though input message texts vary according to individual applications, you must consider three important options before defining your input message area in your action program:

- receiving control character sequences;
- use of the edit table generator to edit input messages; and
- use of screen format services to receive input on formatted screens

Control Character Sequences

Input message control sequences

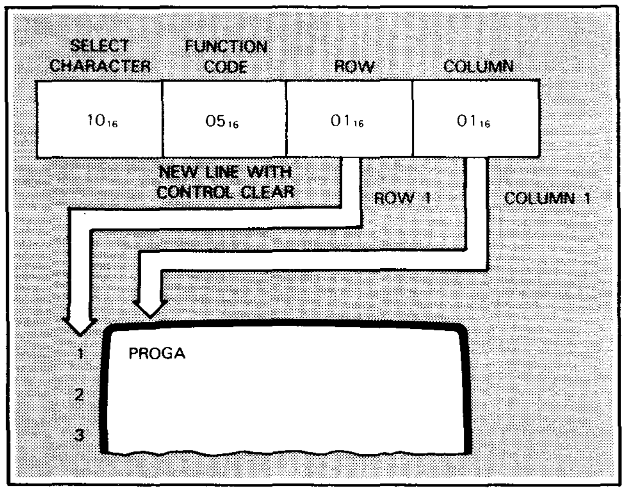
Two input message control character sequences are used on input messages: device independent control expressions (DICE) and field control character sequences (FCC). Field control characters apply only to universal terminal system devices and workstations.

Device Independent Control Expressions

Use of DICE sequences

DICE sequence contents

ICAM automatically inserts DICE sequences into input messages. DICE sequences show the format of input messages. A DICE sequence consists of the select character (10_{16}), a hexadecimal function code, and two hexadecimal coordinates: the first representing a row, and the second representing a column on the terminal. Function codes position the cursor, control carriage return, control forms, control line, feed line, and erase the screen. (See Table F-1 for further details.) The following diagram shows the relationship between the DICE sequences received in your program and their appearance on the screen.



EDIT configurator parameter

In most cases you configure the removal of DICE codes from input messages by specifying EDIT=tablename or EDIT=c in the configurator ACTION section, or by omitting the EDIT parameter.

Configuring receipt of DICE sequences

If you wish to receive DICE sequences on input messages, you configure EDIT=NONE, which indicates no input message editing. You may want to receive DICE sequences on input in order to:

- obtain cursor positioning control values for an input message and use this data in screen positioning output messages; or
- switch a message to another terminal via the SEND function.

Receiving blanks

Leading blanks removed from console input

Configuring EDIT=NONE also means that all blanks entered at the terminal, including leading blanks, are received in your input message area. However, in the case of an input message from the system console, leading blanks are removed.

Example of DICE sequence use

Suppose you receive an input message from a terminal and want to send that message to another terminal; you want that message to arrive at the destination terminal in the same screen position as when it was entered on input.

First, define an area in the first four bytes of your input message area to receive the DICE control sequence. In the procedure division, move the DICE sequence from the input message area to the output message area before moving the destination terminal identification and output message text to the output message area and issuing the SEND function (Figure 4–12).

```
WORKING-STORAGE SECTION.
77   ELEVEN              PIC 99        COMP-4      VALUE 11.
LINKAGE SECTION.
     .
     .
     .
Ø1   INPUT-MESSAGE-AREA.     COPY IMA.
     Ø5   DICE-SEQ       PIC X(4). <——————RECEIVE DICE CONTROL SEQUENCES
     Ø5   TRANS-CODE     PIC X(5).
     Ø5   FILLER         PIC X.
     Ø5   DEST-TERM      PIC X(4).
     Ø5   FILLER         PIC X.
     Ø5   IN-TEXT        PIC X(28).
Ø1   OUTPUT-MESSAGE-AREA.    COPY OMA.
     Ø5   CURSOR-POS     PIC X(4). <——————RECEIVE DICE CONTROL SEQUENCES
     Ø5   OUT-TEXT       PIC X(28).
PROCEDURE DIVISION       USING    PROGRAM-INFORMATION-BLOCK
                                  INPUT-MESSAGE-AREA
                                  OUTPUT-MESSAGE-AREA.
     .
     .
     .
MOVE-MESSAGE.
     MOVE DEST-TERM TO DESTINATION-TERMINAL-ID.
     SUBTRACT ELEVEN FROM TEXT-LENGTH IN INPUT-MESSAGE-AREA
              GIVING TEXT-LENGTH IN OUTPUT-MESSAGE-AREA.
     MOVE DICE-SEQ TO CURSOR-POS.
     MOVE IN-TEXT TO OUT-TEXT.
     CALL 'SEND' USING OUTPUT-MESSAGE-AREA.
     IF STATUS-CODE NOT EQUAL  Ø  GO TO ERROR-PROC.
     .
     .
     .
ERROR-PROC.
```

Figure 4–12. Receiving DICE Sequence on Input Message

Field Control Character Sequences

Use of FCC sequences in input messages

To receive FCC sequences in your input from a universal terminal system terminal or workstation, specify EDIT=NONE or FCCEDIT=NO in the configurator ACTION section. Leave five bytes in your input message text wherever you expect to receive the sequences. You describe the input message text including the FCC sequences much the same as you do for DICE sequences. Both FCC and DICE sequences can be interspersed in the message text instead of just at the beginning.

For more detailed information about the use of FCC sequences, see F.9, UTS 400 programmer reference, UP-8359 (current version), and OS/3 hardware and software summary, UP-8203 (current version).

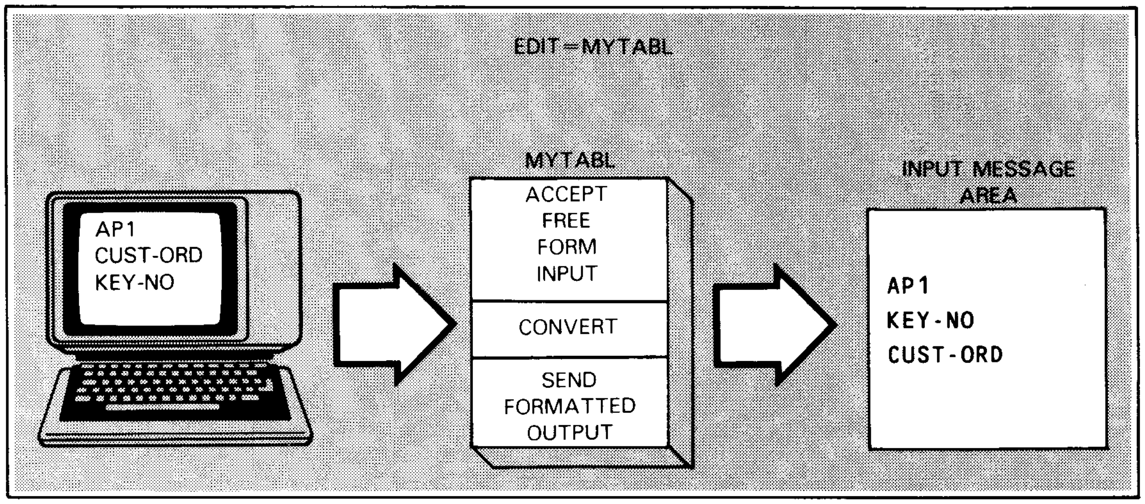
Receiving Freeform Input

Purpose of edit table generator

Let's now consider the use of an edit table (EDIT=tablename) to edit input messages. You create an edit table by executing an offline IMS utility, the edit table generator, and configuring EDIT=tablename. This allows the operator to enter input messages in free form at the terminal. IMS uses the edit table to convert the free-form input message into the format your program requires.

Describing input message text

You describe the input message text in your action program to reflect the formatted input message you want to receive. IMS receives free-form input from the terminal, formats and validates this input as you specify on edit table parameters, and sends it to your program's input message text in the format described there. For a description of how to use the edit table generator, and a sample program that uses an edit table, see Appendix E.



Receiving Screen Formatted Input

Defining input message text

Your action program can receive input entered on screen formats, using screen format services. Your action program displays the screen format by issuing a BUILD function. In your input message area, you describe all input or input/output fields entered by the operator. For more detail about receiving screen formatted input, see Section 7.

5. Processing Data Files

5.1. ACCESSING FILES

Action programs access files via function calls

Most IMS applications require access to data files. Your action programs exist to process messages that depend on data obtained from files. Though your action programs don't directly access data files, they do issue I/O function calls that tell IMS to retrieve, insert, update, or delete records.

IMS/data management interface

When IMS receives a function call from your action program, it makes records available for processing. Data management access methods, SAM, DAM, ISAM, or MIRAM, perform the functions your action program requests. To access IRAM files, you must configure them as MIRAM files.

File types supported

IMS supports sequential, relative, and indexed files as well as defined files that are in indexed organization. Table 5-1 summarizes the files supported by IMS.

Table 5-1. Summary of File Types Supported by IMS

File Organization	Access Mode	Data Management Access Method	Functions Available Through IMS File Management
Sequential	Sequential	SAM/dedicated MIRAM (tape and disk)	Retrieve, Append (write unblocked output)
Relative (nonindexed)	Random	DAM/MIRAM	Retrieve*, Update, Insert, Delete
	Sequential	MIRAM	Retrieve
Indexed	Random	ISAM/MIRAM	Retrieve*, Update, Insert, Delete
	Sequential	ISAM/MIRAM	Retrieve
Indexed (defined file)	Random	ISAM/DAM/ MIRAM	Retrieve*, Update, Insert, Delete
	Sequential	ISAM/DAM/ MIRAM	Retrieve

*Both retrieve and retrieve-with-the-intent-to-update can be requested.

Random and sequential functions

Your action programs may issue random and sequential I/O functions to indexed and relative files but only sequential I/O functions to sequential files. Table 5-2 lists the file I/O functions allowed with each file organization and the CALL function parameters.

Table 5-2. Summary of File I/O Function Calls

File Organization	Random Functions		Sequential Functions	
	CALL	Parameters	CALL	Parameters
Sequential			GET PUT	filename record-area filename record-area
Relative (nonindexed)	GET GETUP PUT INSERT DELETE	filename record-area record number[1] filename record-area record number filename record-area [record-number][2] filename record-area record-number filename record-area record-number	SETL GET ESETL SETK	filename position [record-number] filename record-area filename filename [key-of-ref]
Indexed	GET GETUP PUT INSERT DELETE	filename record-area key [key-of-ref[dup-key-ct]][3] filename record-area key filename record-area filename record-area filename record-area	SETL GET ESETL SETK	filename position [key[partial-key-count]] [3] filename record-area filename filename [key-of-ref][3]
Indexed (defined file)	GET GETUP PUT INSERT DELETE	filename record-area key filename record-area key filename record-area filename record-area key filename record-area	SETL GET ESETL	filename position [key] filename record-area filename

NOTES:

[1] Sequential functions available with MIRAM, not DAM

[2] Record-number required for DAM files

[3] Optional parameters available for MIRAM only

5.2. I/O FUNCTION CALLS

Function calls are your program's means of accessing data on files. You can issue an I/O function call in either COBOL or BAL action programs; their formats differ slightly.

The COBOL CALL function statement format is:

COBOL function call format

```
CALL 'function' USING filename, param-1,...param-n.
```

The BAL CALL function is in the format of a macroinstruction. BAL action programs use either the CALL or ZG#CALL macroinstruction:

BAL function call format

```
1        10     16
-------------------------------------------------------------
         {CALL   }function,(filename,param-1,...param-n)
         {ZG#CALL}
```

where:

Function call name

`function`
: Is the name of the I/O function requested by your action program.

Function call parameters

`filename`
: Is the name of the file on which the function is performed.

`param-1,...param-n`
: Indicates the record area, record number, key, partial-key-count, key-of-reference, duplicate-key-count, or position relative to the record being processed.

Status codes set after function calls

After processing an I/O function call, IMS sets a status code value in the STATUS-CODE field (COBOL action program) or ZA#PSC location (BAL action program) of the program information block. The status codes returned by IMS are explained in more detail in Table D-1.

Detailed status code returns

IMS returns detailed status codes after processing certain I/O functions. These detailed status codes give more description of the error that occurred. For detailed status codes and their descriptions, see Tables D-2 through D-6 and 3.6.

Function Call Positional Parameters

Parameters refer to data names (COBOL) or labels (BAL)

Both COBOL and BAL function CALL statements contain positional parameters that refer to data names in the data division of a COBOL action program or labels of storage locations in a BAL action program. Positional parameters include *filename, record area, record number, key, partial key count, key of reference, duplicate key count,* and *position.*

Filename

Filename is a field containing the 7-character name of the file on which the specified function is performed. This name is left-justified and blank-filled.

In a COBOL action program, the file name can be defined in working-storage:

```
WORKING-STORAGE SECTION.
77  CUST-FILENAME         PIC X(7)    VALUE 'CUSTMST'.
```

To call the file, issue a function call using the data name for the file:

```
CALL 'GET' USING CUST-FILENAME IMS-RECORD-AREA IMS-KEY.
```

In a BAL action program, the file name can be defined as a constant in storage:

```
1        10    16
STATE    DC    CL7'STATE'
```

and called in the macro:

```
1        10    16
         CALL  GET,(STATE,IMS-RECORD-AREA,IMS-KEY)
```

Record-area

Record-area is the area to or from which IMS moves a logical or defined record. You define the record area within an 01-level item of the linkage section, usually the work area.

```
01   WORK-AREA.
     05   PARAMETER-LIST.
          10   IMS-FILENAME       PIC X(7).
          10   IMS-RECORD-AREA    PIC X(256).
```

In a BAL action program, you define the record area in a defined storage statement:

```
1        10    16
--------------------------------------------------------------
WORK     DSECT               WORK AREA
RECORD   EQU   *
SNAME    DS    XL14          STATE NAME
SPOP     DS    XL8           STATE POPULATION
SCAPITAL DS    XL25          STATE CAPITAL
```

Record area size

The record area size must be equal to or greater than the largest logical record it will contain. If your records are ISAM variable length, your record description must begin with a 2-byte binary field describing the length of the record. Other file types need a 4-byte binary field describing length. In a COBOL action program, describing MIRAM or SAM variable-length records, the description might be:

```
02   DATA-RECORD.
     10   IMS-REC-LENGTH          PIC 99    COMP-4.
     10   FILLER                  PIC XX.
     10   FIXED-PORTION.
          20   MAIN-INFO          PIC X(25).
          20   NR-OF-TRAILERS     PIC 99    COMP-4.
     10   VARIABLE-PORTION    OCCURS 0 TO 10 TIMES
               DEPENDING ON NR-OF-TRAILERS.
          20   TRAILER             PIC X(15).
          20   TRAILER-2           PIC X(5).
```

ISAM and DAM considerations

The description for an ISAM variable-length record would not need the FILLER statement after the record length field. For DAM files, the record area should be a multiple of 256 bytes and larger than or equal to the record size.

In a BAL action program, the statement might be:

```
1        10    16
--------------------------------------------------------------
VARLN    DS    CL4
```

Record-number

Record-number is an 8-byte field containing a right-justified binary number that specifies the position of the record relative to the beginning of a relative file. The first number is 1. The COBOL description of this field might be:

```
10   IMS-REC-NUMBER   PIC 9(10)    USAGE    COMP-4.
```

A BAL action program might describe the record number as:

```
 1        10    16
------------------------------------------------
 RECNO    DS    XL8
```

Before issuing function calls containing the *record-number* parameter, move a record number value to this field.

Key

Key contains the value that identifies the record to be retrieved from or inserted into a file. You describe it in a COBOL action program's linkage section. A record key description in your COBOL action program might be:

```
10   IMS-KEY    PIC X(14).
```

In a BAL action program, the equivalent statement might be:

```
RECKEY   DS    CL14
```

Again, before issuing function calls containing the key parameter, you must place a key value in this field.

Partial-key-count

Partial-key-count is used in the SETL function call for indexed MIRAM files when the *position* parameter is G, K, or H. It is the symbolic address of a 4-byte field containing a right-justified binary number. This binary number indicates the number of leading bytes in the key used to locate the record.

The partial key count can be defined in the linkage section or the working-storage section of a COBOL action program. If defined in working storage, it must have a VALUE clause. For example,

```
WORKING-STORAGE SECTION.
77  STPT            PIC 9(4)     USAGE    COMP-4    VALUE 3.
```

defines your partial key count before you issue the SETL function call using STPT as your *partial-key-count* parameter.

The following data item has a binary value of 3 referring to the first three characters (279) of the specified key

```
CALL 'SETL' USING MYFIL  POS  IMS-KEY  STPT.
```

The partial key count should be defined in a BAL action program using a DC statement:

```
1       10   16
STPT    DC   X'00000003'
```

before being referenced in the macroinstruction:

```
1       10   16
        ZG#CALL SETL,(MYFIL,POS,IMS-KEY,STPT)
```

Key-of-reference

Key-of-reference is the symbolic address of a 4-byte field containing a right-justified binary number. This binary number indicates which key of multiple keys is used for retrieving the record. Use the same type working-storage (COBOL) or defined storage (BAL) statements as in the partial key count example to define the key of reference, and assign a value to it before issuing the SETK function call. The value of key-of-reference must be between 1 and 5.

Duplicate-key-count

Duplicate-key-count is the symbolic address of a 4-byte field containing a right-justified binary number. This binary number indicates the number of the record for retrieval within a duplicate key set. The duplicate key count value must be defined before you reference it in your I/O function call. See examples of how this is done in the previous description of *partial-key-count*.

Position

Position is a symbolic address of a storage location containing a 1-byte value. This value designates the position of the file at completion of the SETL function. Values are listed in the SETL function descriptions.

5.3. ACCESSING INDEXED FILES

ISAM and MIRAM access methods

Only primary key used for updating

The indexed sequential and multiple indexed random access methods (ISAM and MIRAM) process function calls issued by your action program to indexed files. With several exceptions, a key specification characterizes most file functions issued to indexed files. Although IMS supports multiple keyed MIRAM files, you must use only the primary key identified in the configurator FILE section (PKEY=n parameter) to insert or update records. Changes or duplicates of alternate keys are allowed, except for primary keys.

NOTE:

Configuring MIRAM files for random access

You must specify MODE=RAN in the FILE section of the configuration to access MIRAM files randomly. If a file is configured as MODE=SEQ, you can use only the sequential functions GET and PUT (5.9).

5.4. RANDOM FUNCTIONS FOR INDEXED FILES

Summary of random functions

The random function calls GET, GETUP, PUT, INSERT, and DELETE:

- retrieve records with or without updating;
- write records back to a file;
- logically or physically delete records; and
- overwrite an existing record or add a new record to a file.

For error status codes resulting from the execution of each of the random I/O function calls, see Table D-1.

Reading Records Randomly (GET)

Description

The random GET function retrieves the record designated by the key value from the named file and places it into the specified record area. IMS does not perform the GET function if the requested record is currently locked by a different transaction. You cannot update a record retrieved by the GET function; use GETUP to retrieve a record for updating.

The COBOL and BAL formats for the random GET function calls are:

▶ COBOL Format 1 (ISAM files)

COBOL format for ISAM files

```
CALL 'GET' USING filename record-area key.
```

▶ COBOL Format 2 (MIRAM files)

COBOL format for MIRAM files

```
CALL 'GET' USING filename record-area key
            [key-of-reference [duplicate-key-count]].
```

▶ BAL Format 1 (ISAM files)

BAL format for ISAM files

```
{CALL    } GET,(filename,record-area,key)
{ZG#CALL }
```

▶ BAL Format 2 (MIRAM files)

BAL format for MIRAM files

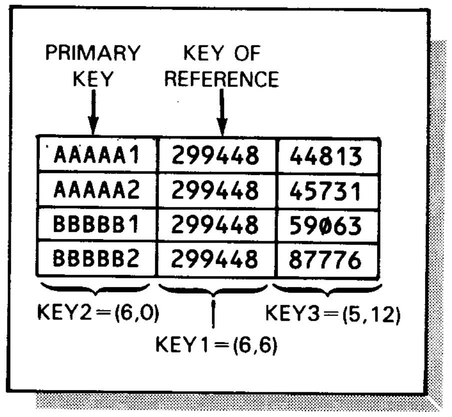
```
{CALL    } GET,(filename,record-area,key
{ZG#CALL }  [,key-of-reference[,duplicate-key-count]])
```

Key-of-reference use

For MIRAM files (Format 2), the *key-of-reference* value indicates which key of multiple keys is used for retrieving the record. This key level number must coincide with one of the data management KEYn specifications designated at configuration time.

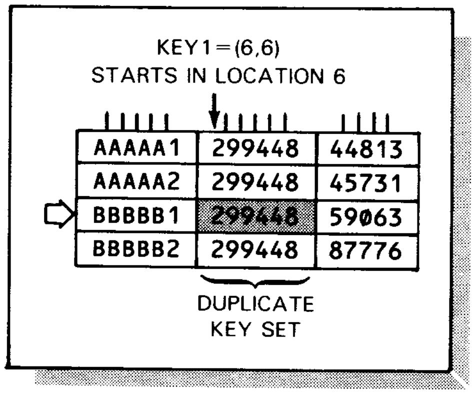
For example, your configurator FILE section might have KEYn designations of KEY1=(6,6), KEY2=(6,0), and KEY3=(5,12). (Key 1 starts in position 6 of the file, key 2 starts in position 0, and key 3 starts in position 12.) Key 2 is configured as the primary key (PKEY=2 specifiction), so key 1 and key 3 are alternate keys. You want to access the file using key 1, so you use the key-of-reference value 1. When the key of reference is omitted, IMS uses the primary key, in this case, key 2.

```
WORKING-STORAGE SECTION.
77  ONE      PIC 9    COMP-4   VALUE '1'
77  TWO      PIC 9    COMP-4   VALUE '2'
77  THREE    PIC 9    COMP-4   VALUE '3'
     .
     .
     .
PROCEDURE DIVISION.
     .
     .
     .
CALL 'GET' USING FIL-A  REC-A  KEY-A  ONE.
```



Duplicate-key-count use Also, on function calls to MIRAM files you can specify a duplicate-key-count value to indicate which record within a duplicate key set to retrieve.

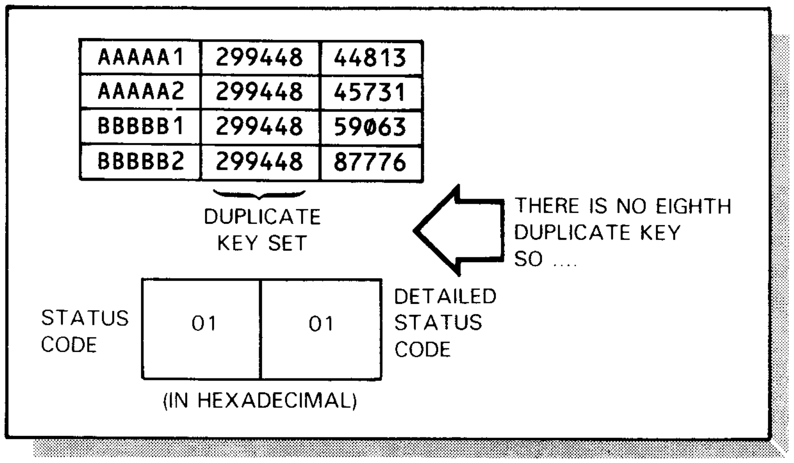
```
WORKING-STORAGE SECTION.
77  DUP-KEY-CT       PIC 9  COMP-4  VALUE '3'.
PROCEDURE DIVISION.
     .
     .
     .
CALL 'GET' USING FIL-A  REC-A  KEY-A  ONE  DUP-KEY-CT.
```



Duplicate-key-count default

If you omit this parameter or if it equals 1, IMS retrieves the first record within the duplicate key set. If the value is zero or exceeds the number of records within the duplicate key set, IMS sets status code and detailed status code to 1.

```
WORKING-STORAGE SECTION.
77   DUP-KEY-CT     PIC 9  USAGE COMP-4  VALUE '8'.
```



Sequence changes when record is deleted

Note that the sequence of records in a duplicate key set changes when one of the records in the set is deleted. If the deleted record is later restored by online or offline recovery, it is placed at the end of the duplicate key set instead of its original position.

Retrieving logically deleted records

If you configure physical deletion of records (DELETP=YES in the FILE section, you can retrieve any logically deleted records on MIRAM files as normal data. You must configure physical deletion of records when files are multikeyed.

Reading Records for Update (GETUP)

Description

The GETUP function retrieves the record for updating and temporarily locks the requested record from access by other transactions. IMS does not perform the GETUP function if the requested record is currently locked by a different transaction. As with the GET function, IMS uses the key you specify on the GETUP function to locate the required record. Unlike the GET function, you can access a record for update only by the primary key.

The COBOL and BAL formats for the GETUP function call to all indexed files are:

▷ COBOL Format

COBOL format

```
CALL 'GETUP' USING filename record-area key.
```

▷ BAL Format

BAL format

```
{CALL    } GETUP,(filename,record-area,key)
{ZG#CALL }
```

Updating and deleting records

To update or delete the record requested, issue a PUT or DELETE function call following the GETUP function. Other function calls to the same file may not intervene. Otherwise, the record must be retrieved again with a GETUP function before a PUT or DELETE can be performed. You may, however, issue other instructions and function calls to other files between the GETUP and PUT or DELETE functions.

Function call to same file may not intervene

Incorrect	Correct
CALL 'GETUP' USING MYFIL IMS-REC-AREA MYKEY. CALL 'GET' USING MYFIL IMS-REC-AREA MYKEY. MOVE CUST-NAME TO NAME-FIELD. CALL 'PUT' USING MYFIL IMS-REC-AREA.	CALL 'GETUP' USING MYFIL IMS-REC-AREA MYKEY. MOVE CUST-NAME TO NAME-FIELD. CALL 'PUT' USING MYFIL IMS-REC-AREA.

Key value must not be changed for ISAM

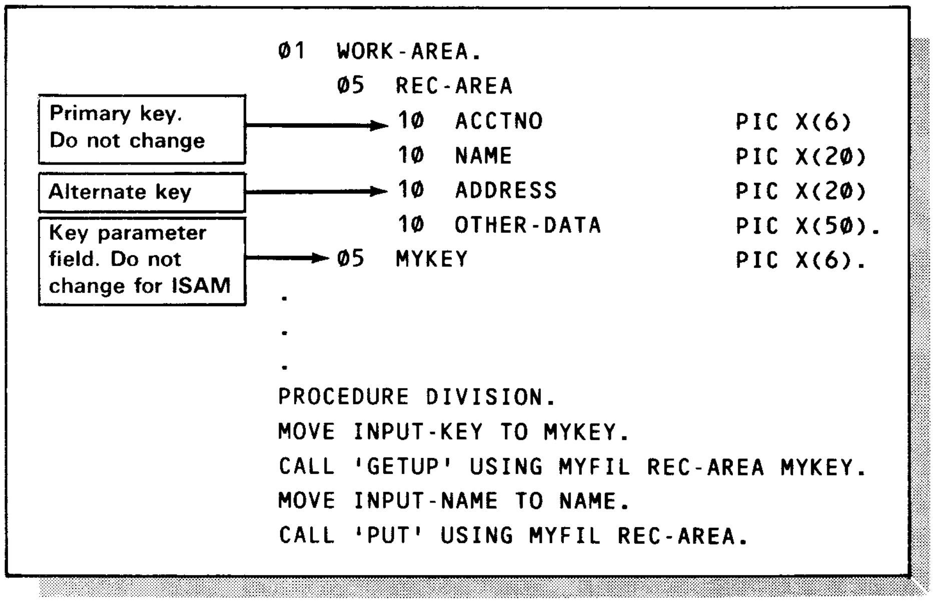
For ISAM files, you must not change the key value in the record area between the GETUP and succeeding PUT or DELETE function calls. IMS does not return an error, but you may damage your data file.

Primary key must not be changed for MIRAM

For MIRAM files, do not change the value of the primary key in the record area between the GETUP and succeeding PUT or DELETE function calls. You may, however, change the value of alternate keys.

Key parameter field for ISAM and MIRAM

For ISAM files, do not change the value of the key field used for the key parameter between the GETUP and succeeding PUT or DELETE function calls. This value may be changed when you use MIRAM files.


Primary key. Do not change
Alternate key
Key parameter field. Do not change for ISAM

```
Ø1  WORK-AREA.
    Ø5  REC-AREA
        1Ø  ACCTNO                  PIC X(6)
        1Ø  NAME                    PIC X(2Ø)
        1Ø  ADDRESS                 PIC X(2Ø)
        1Ø  OTHER-DATA              PIC X(5Ø).
    Ø5  MYKEY                       PIC X(6).
 .
 .
 .
PROCEDURE DIVISION.
MOVE INPUT-KEY TO MYKEY.
CALL 'GETUP' USING MYFIL REC-AREA MYKEY.
MOVE INPUT-NAME TO NAME.
CALL 'PUT' USING MYFIL REC-AREA.
```

Retrieving logically deleted records

If you configure physical deletion of records, you can retrieve any logically deleted records on MIRAM files as normal data.

Writing Updated Records (PUT)

Description

The random PUT function writes an updated record back to the file. It must be preceded by a GETUP function that retrieves the record for update. The first byte of nonkey data must not contain X'FF', unless you have configured physical deletion for MIRAM files (DELETP=YES).

Keys not needed

No key is required on a PUT function because the key is in the specified key location in the record area. If you specify a key parameter, IMS returns a status code of 3 and a detailed status code of 1.

The COBOL and BAL formats for the PUT function call are:

▶ COBOL Format

COBOL format

```
CALL 'PUT' USING filename record-area.
```

▶ BAL Format

BAL format

```
{CALL    } PUT,(filename,record-area)
{ZG#CALL }
```

Deleting Records (DELETE)

Description

The DELETE function deletes a record that was retrieved for updating. The DELETE function must be preceded by a GETUP function. If other function calls to the same file intervene, you must reissue the GETUP function before the record can be deleted.

The COBOL and BAL formats for the DELETE function call are:

▷ COBOL Format

COBOL format

```
CALL 'DELETE' USING filename record-area.
```

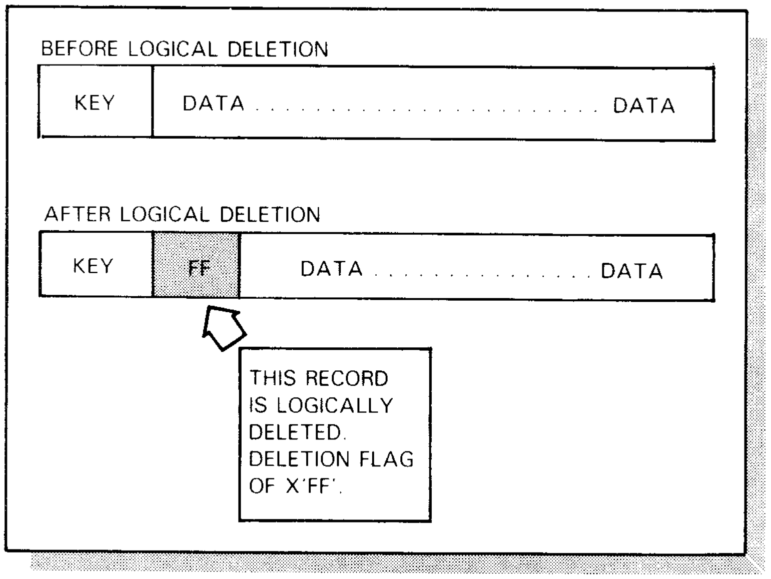
▷ BAL Format

BAL format

```
{CALL   } DELETE,(filename,record-area)
{ZG#CALL}
```

ISAM file logical deletion

The DELETE function for ISAM files is a logical deletion. A logical record deletion changes the first byte of nonkey data to X'FF' before the record is written back to the file.


BEFORE LOGICAL DELETION
KEY
DATA . . . . DATA
AFTER LOGICAL DELETION
KEY
FF
DATA . . . . DATA
THIS RECORD IS LOGICALLY DELETED. DELETION FLAG OF X'FF'.

Single-keyed MIRAM files

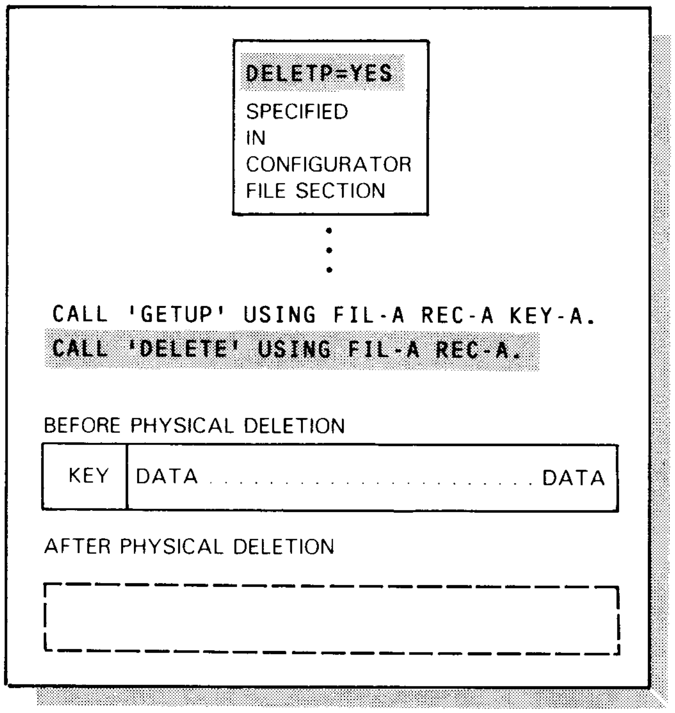
The DELETE function for single-keyed MIRAM files can be a logical or a physical deletion. A physical deletion is always performed for multikeyed MIRAM files.

Logical deletion

To logically delete single-keyed MIRAM records, configure DELETP=NO or default to this value. The results of this logical deletion are the same as for ISAM records on logical deletion (e.g., X'FF' in first byte of nonkey data).

Physical deletion

To physically delete a single-keyed MIRAM record, create the file with the data management keyword RCB=YES and configure IMS with the DELETP=YES parameter. (DELETP=YES is assumed for multikeyed MIRAM.) The DELETE function then physically deletes the record from the file.



```
CALL 'GETUP' USING FIL-A REC-A KEY-A.
CALL 'DELETE' USING FIL-A REC-A.
```

Results when record is flagged for logical deletion

Suppose the record you call for deletion is previously flagged as logically deleted. If you configure physical deletion, the GETUP function retrieves the requested record. If you configure logical deletion, the GETUP function returns a *record not found* status.

NOTE:

Consideration when accessing file from non-IMS program

When IMS logically deletes a record (X'FF' in the first byte of nonkey data) and you later access the file from a non-IMS program, the record will not be recognized as deleted. You must check for HIGH-VALUES or X'FF' in the first byte of nonkey data.

Adding Records (INSERT)

Description

The INSERT function places a new record into the file or overwrites a previously deleted record. This function is not preceded by a GETUP function. The first byte of nonkey data in the record being inserted must not contain a deleted record value of X'FF', unless you have configured physical deletion for MIRAM files. The COBOL and BAL formats for the random INSERT function calls are:

▷ COBOL Format

COBOL format

```
CALL 'INSERT' USING filename record-area.
```

▷ BAL Format

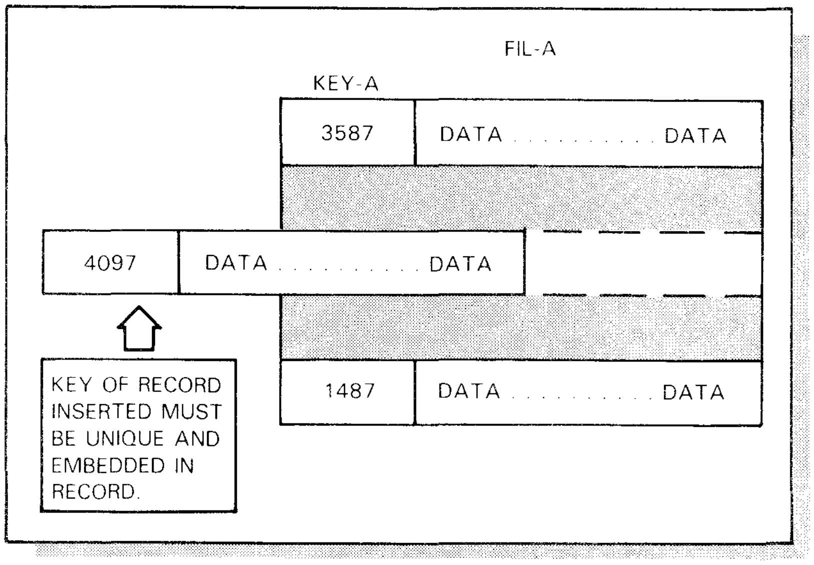
BAL format

```
{CALL   } INSERT,(filename,record-area)
{ZG#CALL}
```

Unique keys

Indexed files do not require a key parameter in the INSERT function. Their keys must be embedded in the record. The key of the new record must have a value that is different from any already existing in the file.

```
CALL 'INSERT' USING FIL-A REC-A.
```



Removes delete control character

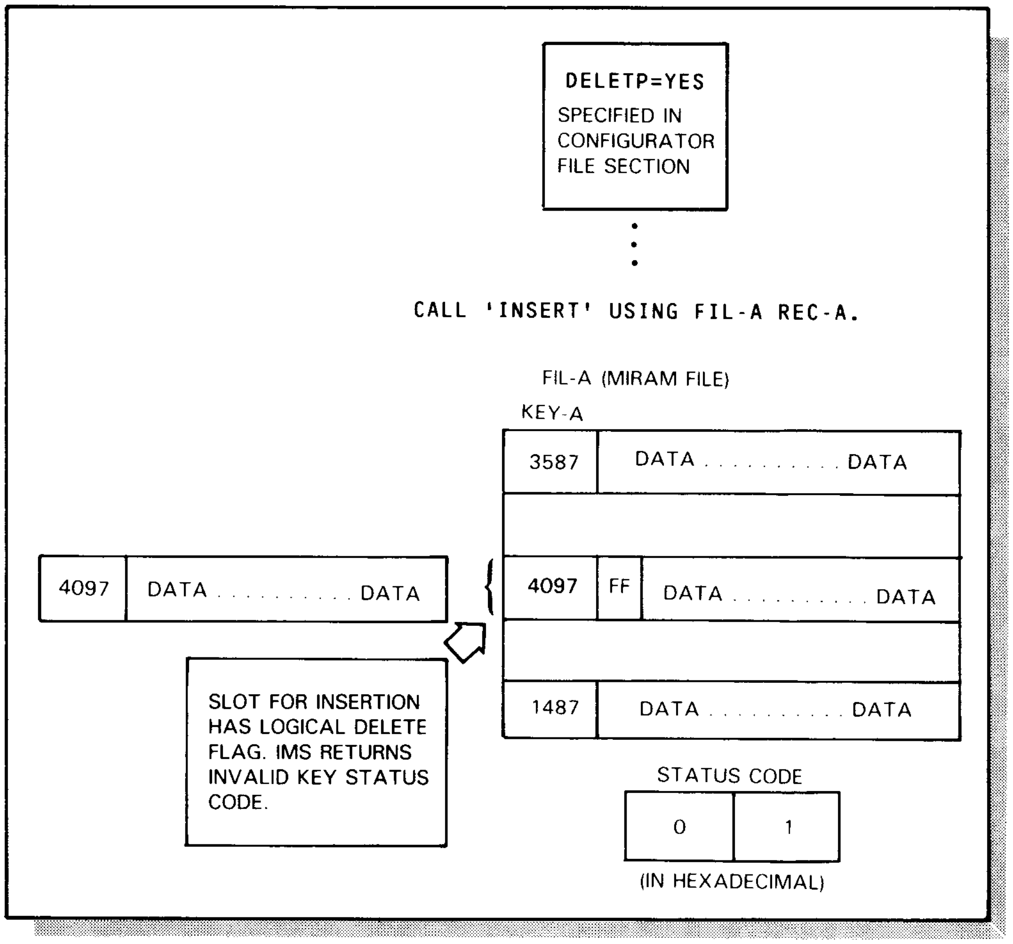
Changing length field

An INSERT function using a previously deleted record slot removes the delete control character. You can change the length field for variable-length records in MIRAM files, but not in ISAM files.

INDEXED FILES: INSERT FUNCTION

Physical delete consideration

For MIRAM files, you cannot overwrite a logically deleted record, when physical deletion is configured. An attempt to do this results in a status code of 1, invalid key.


DELETP=YES
SPECIFIED IN CONFIGURATOR FILE SECTION
CALL 'INSERT' USING FIL-A REC-A.
FIL-A (MIRAM FILE)
KEY-A
3587 DATA . . . . DATA
4097 FF DATA . . . . DATA
1487 DATA . . . . DATA
4097 DATA . . . . DATA
SLOT FOR INSERTION HAS LOGICAL DELETE FLAG. IMS RETURNS INVALID KEY STATUS CODE.
STATUS CODE
0 1
(IN HEXADECIMAL)

5.5. SEQUENTIAL FUNCTIONS FOR INDEXED FILES

Summary of sequential functions

Sequential function calls SETK, SETL, GET, and ESETL

- set a key of reference for sequential processing;
- set an indexed file into sequential mode and position it to a selected location in the file;
- retrieve records sequentially; and,
- reset the indexed file from sequential mode to random mode.

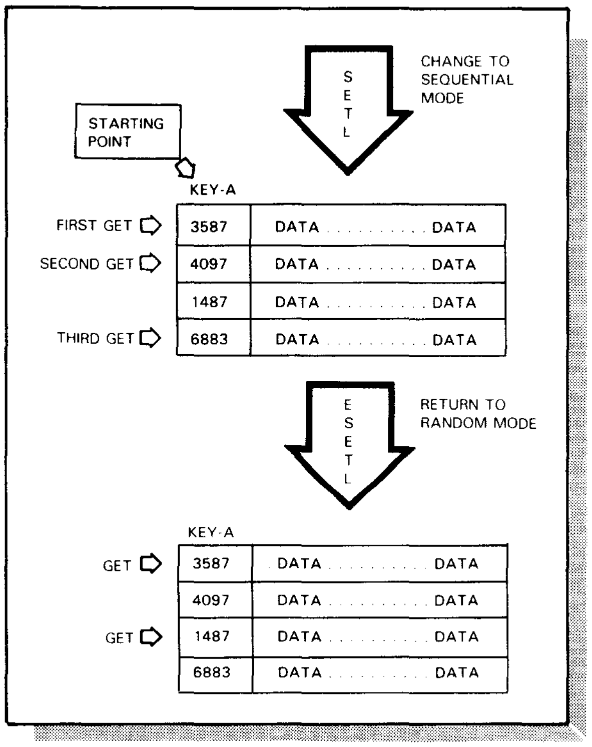
For error status codes resulting from the execution of each of the sequential I/O function calls, see Table D-1.

Changing access mode to sequential

When accessing an indexed file sequentially, your action program must first set the file into sequential mode via the SETL function. During this time, the file is accessed exclusively by the transaction that sets the mode. Requests by other transactions for sequential or random mode functions are queued for later processing.

Returning to random mode

Sequential mode exists until your program requests an ESETL function or until the current action terminates. In either case, the indexed file returns to random mode. The file also returns to random mode if an error occurs on a SETK or SETL function or an invalid request (status code 3) occurs on a GET function.



SETL
CHANGE TO SEQUENTIAL MODE
STARTING POINT
KEY-A
FIRST GET
SECOND GET
THIRD GET
3587 DATA .... DATA
4097 DATA .... DATA
1487 DATA .... DATA
6883 DATA .... DATA
ESETL
RETURN TO RANDOM MODE
KEY-A
GET
GET
3587 DATA .... DATA
4097 DATA .... DATA
1487 DATA .... DATA
6883 DATA .... DATA

NOTE:

Shared file access

Shared file access among transactions is done only in the random mode. The use of sequential mode by one transaction can significantly degrade the response time for other transactions accessing the same file.

Setting the Key of Reference for Sequential Processing (SETK)

Description

The SETK function establishes the key of reference for subsequent indexed file positioning and retrieval. This function is used exclusively with multikeyed MIRAM files.

The COBOL and BAL function call formats for the SETK function are:

▶ COBOL Format

COBOL format

```
CALL 'SETK' USING filename [key-of-reference].
```

▶ BAL Format

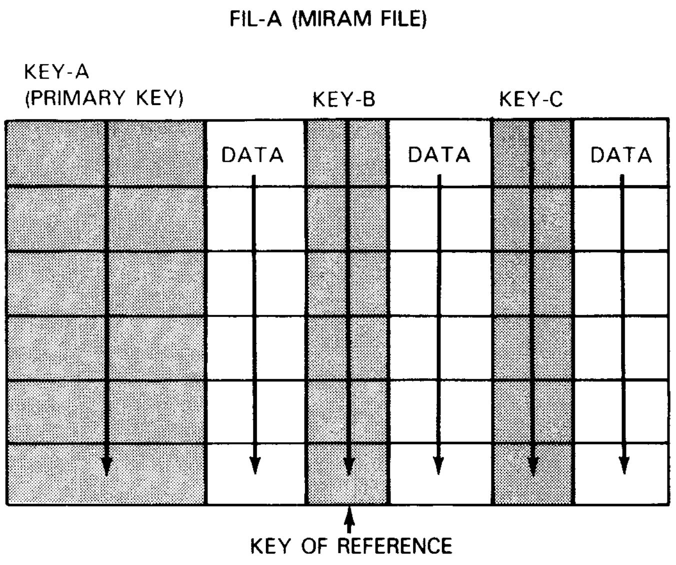
BAL format

```
{CALL   }SETK,(filename[key-of-reference])
{ZG#CALL}
```

Key-of-reference use

Omitting key of reference

The *key-of-reference* is the symbolic address of a 4-byte field containing a right-justified binary number. This value indicates which of the multiple keys to use on the succeeding SETL and GET functions. If the key-of-reference parameter is omitted, IMS uses the primary key for the search.



```
CONFIGURE:     FILE  FIL-A     FILETYPE=DMRAM
                               PKEY=1
                               KEY1=(6,Ø)
                               KEY2=(1,5Ø)
                               KEY3=(2,8Ø)
 WORKING-STORAGE SECTION.
 77   KEY-A        PIC 9(5)    COMP-4    VALUE 1.
 77   KEY-B        PIC 9(5)    COMP-4    VALUE 2.
 77   KEY-C        PIC 9(5)    COMP-4    VALUE 3.
 .
 .
 .
 PROCEDURE DIVISION.
 PARA-1.
   CALL 'SETK' USING FIL-A KEY-B.
   .
   .
   .
   CALL 'ESETL' USING FIL-A.
```

Changing key-of-reference

A GET function cannot directly follow a SETK function; you must position the file with the SETL function before retrieving records. It can be issued many times to change the key of reference. Once established, however, the specified key of reference remains in effect until another SETK, ESETL, or action termination.

Errors on SETK

When any error occurs on a SETK function, the file is reset to random mode and any file locks in effect are released. For further sequential processing, you must issue another SETL and SETK function to reestablish the sequential mode and the key of reference.

Setting Indexed Files from Random to Sequential Mode (SETL)

Description

The SETL function sets an indexed file into sequential mode and logically positions the file as follows:

Position values

Value	Meaning
B	Beginning of file
G	Greater than or equal to the key supplied
K	Equal to key supplied
H	Greater than key supplied

Start position

Changing access position

The value of the position parameter determines the logical position of the file at completion of the SETL function. Indexed files start at position 0. You can reissue the SETL function any time to change the sequential position of the file. For ISAM files, however, you must issue an ESETL function before reissuing another SETL function.

The COBOL and BAL formats for the SETL function call are:

▷ COBOL Format

COBOL format

```
CALL 'SETL' USING filename position [key[partial-key-count]].
```

▷ BAL Format

BAL format

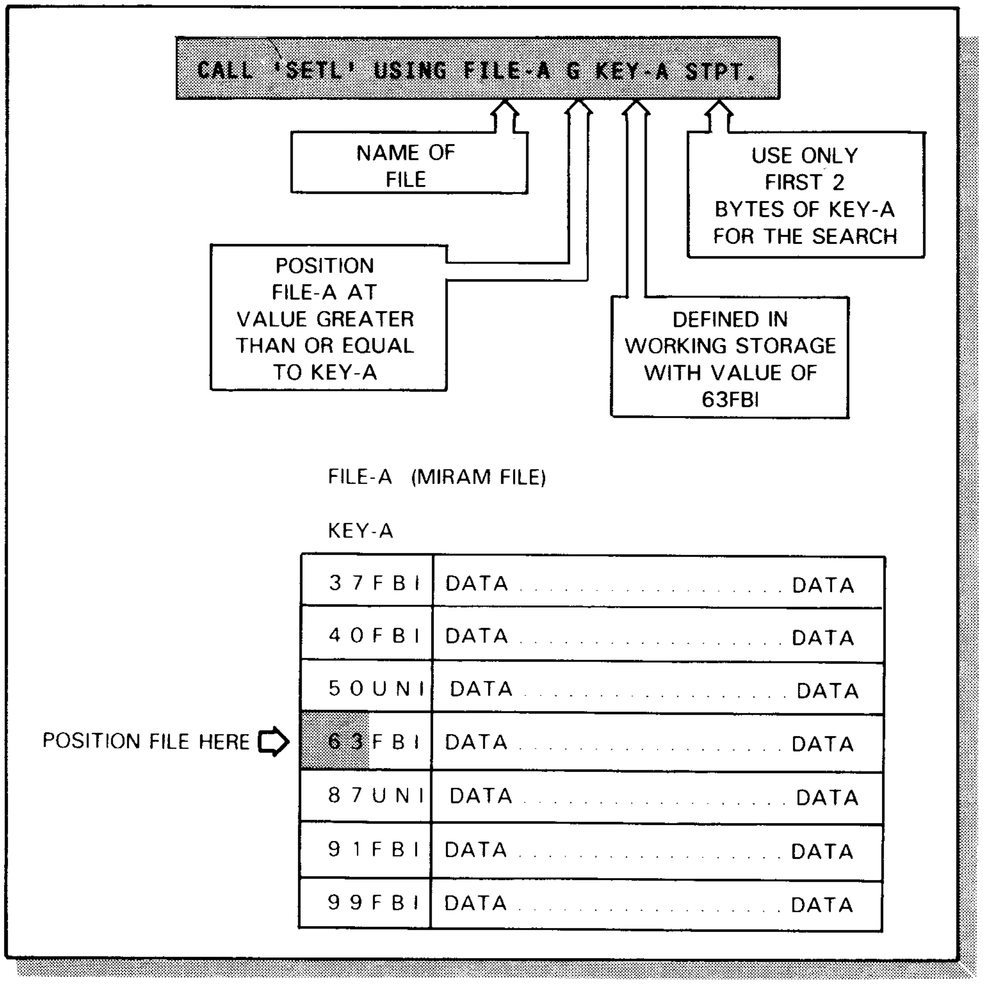
```
{CALL   } SETL,(filename,position[,key[,partial-key-count]])
{ZG#CALL}
```

Required and optional parameters

You must supply a file name and choose a position value. Depending upon the position chosen, you also supply a *key* parameter.

Partial key search for MIRAM files

In addition, the SETL function allows for partial key search of indexed MIRAM files. To do this, use the optional *partial-key-count* parameter. It is the symbolic address of a 4-byte field containing a right-justified binary number. This binary number indicates the number of leading bytes used from the key to locate the record. If you omit the *partial-key-count* parameter, data management uses the entire key to locate the record.


CALL 'SETL' USING FILE-A G KEY-A STPT.
NAME OF FILE
USE ONLY FIRST 2 BYTES OF KEY-A FOR THE SEARCH
POSITION FILE-A AT VALUE GREATER THAN OR EQUAL TO KEY-A
DEFINED IN WORKING STORAGE WITH VALUE OF 63FBI
FILE-A (MIRAM FILE)
KEY-A
3 7 F B I DATA .... DATA
4 0 F B I DATA .... DATA
5 0 U N I DATA .... DATA
POSITION FILE HERE
6 3 F B I DATA .... DATA
8 7 U N I DATA .... DATA
9 1 F B I DATA .... DATA
9 9 F B I DATA .... DATA

Errors on SETL

When any error occurs on a SETL function, the file is reset to random mode and any file locks in effect are released. For further sequential processing, you must issue another SETL function call.

Table 5-3 lists the SETL parameter choices for ISAM and MIRAM files.

Table 5-3. SETL Parameter Choices for Indexed Files

File Type	Parameters						
	Filename	Position					
		B	G	K	H	Key	Partial Key
ISAM	X	X	X	X		X	
Indexed MIRAM	X	X	X	X	X	X	X

Reading Records Sequentially (GET)

Description

The sequential GET function retrieves the next logical record in sequential order unless the record is marked logically deleted (i.e., X'FF' in the first byte). If the record is marked logically deleted, the GET function retrieves the following record. For MIRAM files, if DELETP=YES is configured or assumed, data management retrieves logically deleted records as normal data.

Required parameters

Filename and *record-area* parameters are required on sequential GET functions for indexed files.

The COBOL and BAL formats for the sequential GET function call are:

- COBOL Format

COBOL format

```
CALL 'GET' USING filename record-area.
```

- BAL Format

BAL format

```
{CALL    } GET,(filename,record-area)
{ZG#CALL }
```

Errors on sequential GET

When an invalid request error occurs on a sequential GET function, after a SETL function, the file is reset to random mode and any file locks in effect are released.

Setting Indexed Files from Sequential to Random Mode (ESETL)

Description

The ESETL function changes the mode of indexed files from sequential to random. If a file is in the sequential mode for a transaction and you do not issue an ESETL function before termination of the current action, IMS resets the file to random mode. The ESETL function always requires a filename parameter.

The COBOL and BAL formats for the ESETL function call are:

- COBOL Format

COBOL format

```
CALL 'ESETL' USING filename.
```

- BAL Format

BAL format

```
{CALL    } ESETL,(filename)
{ZG#CALL }
```

5.6. ACCESSING RELATIVE FILES

The direct and multiple indexed random access methods (DAM and MIRAM) process function calls issued by your action program to relative files. A record-number parameter characterizes most file functions to relative files although record numbers are not required on sequential functions. Random and sequential functions are supported for MIRAM files but only random functions for DAM files.

NOTE:

Configuring MIRAM files for random access

You must specify MODE=RAN in the FILE section of the configuration to access MIRAM files randomly. If a file is configured as MODE=SEQ, you can use only the sequential functions GET and PUT (5.9).

5.7. RANDOM FUNCTIONS FOR RELATIVE FILES

Summary of random functions

The random function calls GET, GETUP, PUT, INSERT, and DELETE:

- retrieve records with or without updating;
- write records back to a file;
- logically or physically delete records; and,
- overwrite an existing record or add a new record to a file.

For error status codes resulting from the execution of each of the random I/O functions, see Table D-1.

Preformatting DAM files

You must preformat DAM files offline before their initial use and they must contain the maximum number of physical records to be referenced online under IMS.

Reading Records Randomly (GET)

Description

The random GET function retrieves the record you request by record number and places it into the specified record area. All record number fields must be 8 bytes long and binary. You cannot update a record retrieved by the GET function; use GETUP to retrieve a record for updating.

If the requested record is currently locked by a different transaction, IMS does not perform the GET function.

The COBOL and BAL formats for the random GET function call are:

COBOL Format

COBOL format

```
CALL 'GET' USING filename record-area record-number.
```

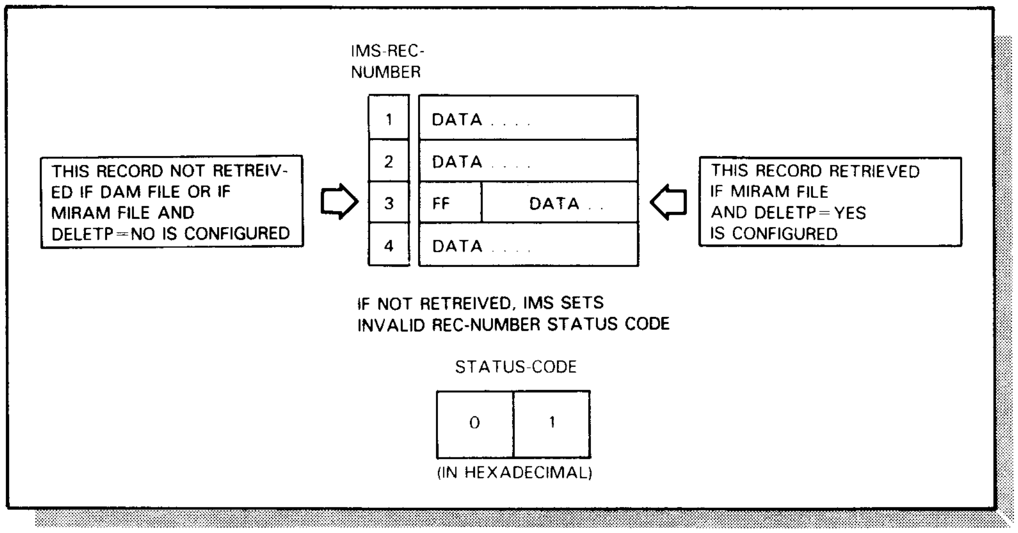
BAL Format

BAL format

```
{CALL    } GET,(filename,record-area,record-number)
{ZG#CALL }
```

Retrieving logically deleted records

If a transaction requests a logically deleted record (X'FF' in the first byte), IMS returns an invalid record number status code of 1. However, if DELETP=YES is configured for a MIRAM file, logially deleted records are retrieved, as normal data.


IMS-REC-NUMBER
1 DATA . . . .
2 DATA . . . .
3 FF DATA . .
4 DATA . . . .
THIS RECORD NOT RETREIV-ED IF DAM FILE OR IF MIRAM FILE AND DELETP=NO IS CONFIGURED
THIS RECORD RETRIEVED IF MIRAM FILE AND DELETP=YES IS CONFIGURED
IF NOT RETREIVED, IMS SETS INVALID REC-NUMBER STATUS CODE
STATUS-CODE
0 1
(IN HEXADECIMAL)

Reading Records for Update (GETUP)

Description

The random GETUP function uses a record number to retrieve a requested record for updating and temporarily locks that record from access by other transactions. IMS does not perform a random GETUP function if the requested record is currently locked by a different transaction. All record number fields must be 8 bytes long and binary.

The COBOL and BAL formats for the random GETUP function call are:

▶ COBOL Format

COBOL format

```
CALL 'GETUP' USING filename record-area record-number.
```

▶ BAL Format

BAL format

```
{CALL   } GET,(filename,record-area,record-number)
{ZG#CALL}
```

Updating and deleting records

A GETUP function can be followed by a PUT function to update the record, or a DELETE function to mark the record as logically deleted or to physically delete it.

Record number omission

If the record-number parameter is omitted from the PUT or DELETE function that follows a GETUP function (MIRAM files only), the record field in your program must remain unaltered until IMS completes the PUT or DELETE function.

Requesting logically deleted records

If the DELETP=YES parameter is configured and you issue a GETUP function call for a logically deleted record, IMS returns the logically deleted record as normal data. For DAM files, and for MIRAM files with DELETP=NO configured, IMS returns an invalid record number status of 1.

Writing Updated Records (PUT)

Description

The random PUT function is used with the GETUP function to write an updated record back to the file. A PUT function must be preceded by a GETUP function that retrieves the requested record for update. The first byte of data in a record must not contain an X'FF' unless you have configured physical deletion for MIRAM files.

The COBOL and BAL formats for the PUT function call are:

▶ COBOL Format

COBOL format

```
CALL 'PUT' USING filename record-area [record-number].
```

▶ BAL Format

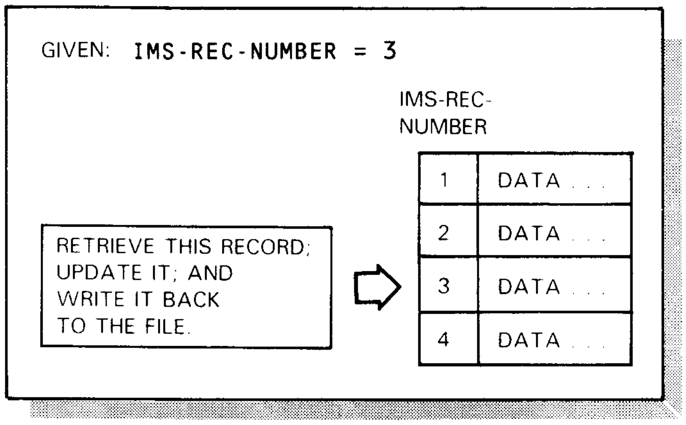
BAL format

```
{CALL   } PUT,(filename,record-area[,record-number])
{ZG#CALL}
```

Placement of PUT function

A record-number parameter is required on the PUT function for DAM files, but is optional for MIRAM relative files. When you omit record-number for MIRAM files, no function call for the same file may be between the GETUP and PUT function.

```
CALL 'GETUP' USING FIL-A REC-AREA IMS-REC-NUMBER.
MOVE NEW-AMT TO AMT-A.
CALL 'PUT' USING FIL-A REC-AREA.
```


GIVEN: IMS-REC-NUMBER = 3
IMS-REC-NUMBER
1 DATA . . .
2 DATA . . .
3 DATA . . .
4 DATA . . .
RETRIEVE THIS RECORD; UPDATE IT; AND WRITE IT BACK TO THE FILE.

Deleting Records (DELETE)

DAM files

The DELETE function for DAM files logically deletes a record that was retrieved for updating.

MIRAM files

For MIRAM files, this function physically deletes a record if the file was created with the data management keyword RCB=YES and configured with the DELETP=YES parameter. For MIRAM files configured with DELETP=NO, the deletion is logical.

Placement of DELETE function

For an effective logical or physical deletion, this function must be immediately preceded by a GETUP function. If other functions intervene, the GETUP function must be reissued before the record can be deleted.

The COBOL and BAL formats for the DELETE function call are:

COBOL Format

COBOL format

```
CALL 'DELETE' USING filename record-area [record-number].
```

BAL Format

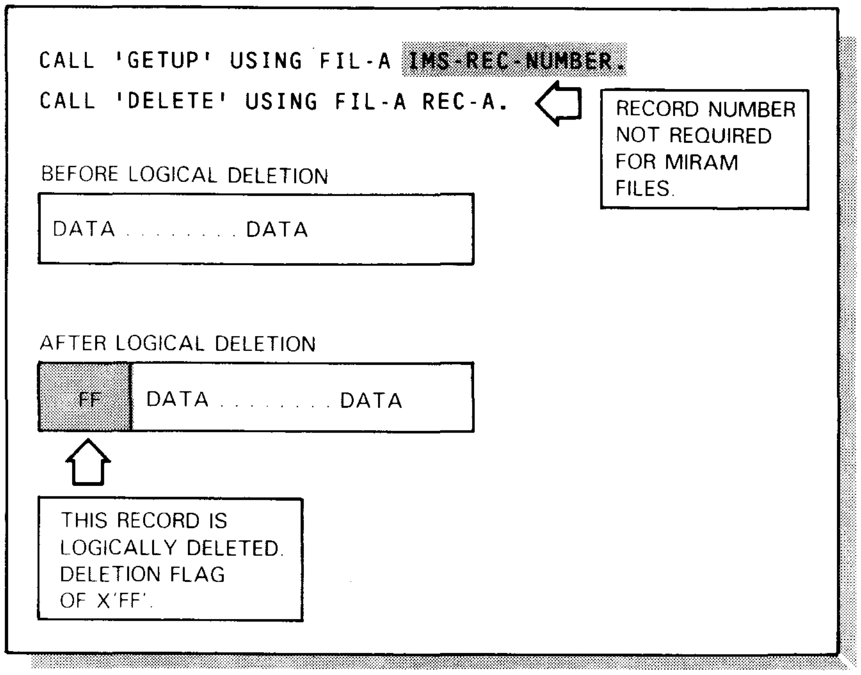
BAL format

```
{CALL    } PUT,(filename,record-area[,record-number])
{ZG#CALL }
```

You must supply a record-number parameter on the DELETE function for DAM files; it is optional for MIRAM files.

Logical deletion

The logical DELETE function changes the first byte of data in a record retrieved for update to X'FF' before the record is written to the file.

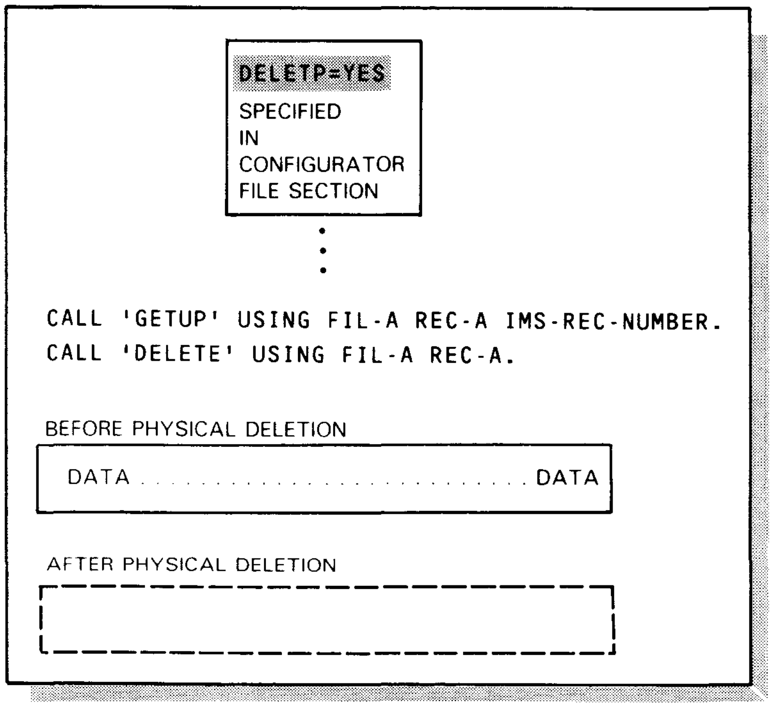


```
CALL 'GETUP' USING FIL-A IMS-REC-NUMBER.
CALL 'DELETE' USING FIL-A REC-A.
```

RECORD NUMBER NOT REQUIRED FOR MIRAM FILES.
BEFORE LOGICAL DELETION
DATA . . . . DATA
AFTER LOGICAL DELETION
FF
DATA . . . . DATA
THIS RECORD IS LOGICALLY DELETED. DELETION FLAG OF X'FF'.

Physical deletion

On the other hand, a physical DELETE actually removes the record from the file.



DELETP=YES SPECIFIED IN CONFIGURATOR FILE SECTION

```
CALL 'GETUP' USING FIL-A REC-A IMS-REC-NUMBER.
CALL 'DELETE' USING FIL-A REC-A.
```

BEFORE PHYSICAL DELETION
DATA . . . . DATA
AFTER PHYSICAL DELETION

NOTE:

Consideration when accessing file from non-IMS program

When IMS logically deletes a record (X'FF' in the first byte) and you later access the file from a non-IMS program, the record will not be recognized as deleted. You must check for HIGH-VALUES or X'FF' in the first byte.

Adding Records (INSERT)

Description

The INSERT function places a new record into the file or overwrites a previously deleted record. This function is not preceded by a GETUP function. The first byte of data in the record being inserted must not contain a deleted record value of X'FF'.

Previously deleted record slots

An INSERT function using a previously deleted record slot removes the delete control character. You can change the record length field for variable-length records in MIRAM files only. The INSERT function for MIRAM files can also overwrite nondeleted records.

The COBOL and BAL formats for the INSERT function call are:

▶ COBOL Format

COBOL format

```
CALL 'INSERT' USING filename record-area record-number.
```

▶ BAL Format

BAL format

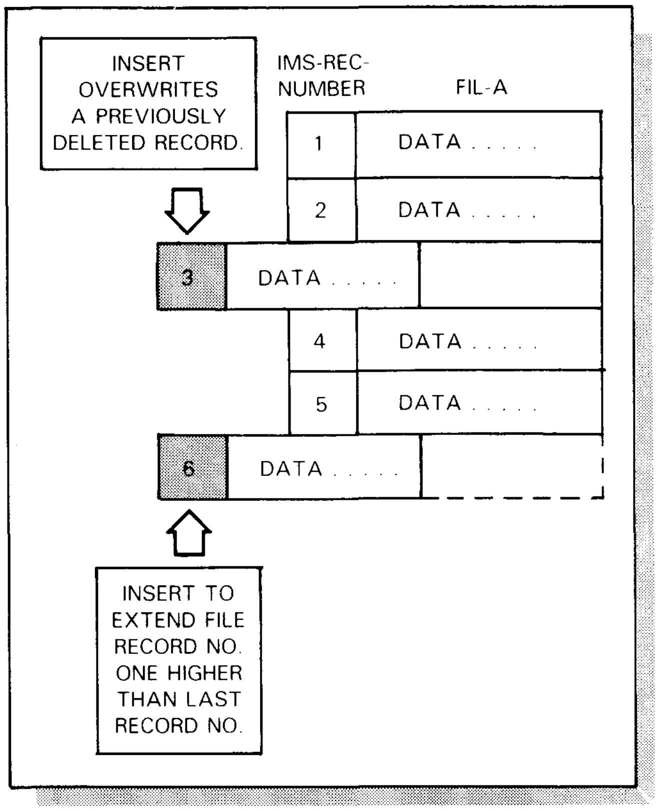
```
{CALL   } DELETE,(filename,record-area[,record-number])
{ZG#CALL}
```

Assigning relative record numbers

MIRAM file extension

INSERT functions issued to a relative file must supply a record-number parameter. If you configure MIRAM files with RCB=NO, any record you add to a relative file must be assigned a relative record number *one higher than the last record in the file.* This prevents the occurrence of erroneous data between the last record and the new inserted record. You may insert records within or beyond the limits of nonindexed MIRAM files; file extension is permitted.

```
        CALL 'INSERT' USING FIL-A REC-A REC-NO.
GIVEN:  REC-NO = 3
        CALL 'INSERT' USING FIL-A REC-A REC-NO.
GIVEN:  REC-NO = 6
```



5.8. SEQUENTIAL FUNCTIONS FOR RELATIVE FILES

Summary of sequential functions

Sequential function calls SETL, GET, and ESETL:

- set a nonindexed MIRAM file into sequential mode and position it to a selected location in the file;
- retrieve records sequentially; and
- reset the file from sequential mode to random mode.

Sequential functions cannot be processed by the direct access method (DAM).

For error status codes resulting from the execution of each of the sequential I/O functions, see Table D-1.

Setting access mode

When accessing a relative file sequentially, action programs must first set the file into sequential mode via the SETL function. During this time, files are accessed exclusively by the transaction that set the mode. Requests by other transactions for sequential or random mode functions are queued for later processing.

Returning to random mode

Sequential mode exists until your program requests an ESETL function or until the current action terminates. In either case, the indexed file returns to random mode.

NOTE:

Shared file access

Shared file access among transactions is done only in the random mode. The use of sequential mode by one transaction can significantly degrade the response time for other transactions accessing the same file.

Setting Relative Files from Random to Sequential Mode (SETL)

Description

The SETL function sets a relative file into sequential mode and logically positions the file as follows:

Position values

Value	Meaning
B	Beginning of file
G	Greater than or equal to the record number supplied
K	Equal to record number supplied
H	Greater than record number supplied

Starting position

The value of the *position* parameter determines the logical position of the file at completion of the SETL function. Relative files start at position 1. You can reissue the SETL function any time you wish to change the sequential position of the file.

The COBOL and BAL formats for the SETL function call are:

▷ COBOL Format

COBOL format

```
CALL 'SETL'  USING filename position[record-number].
```

▷ BAL Format:

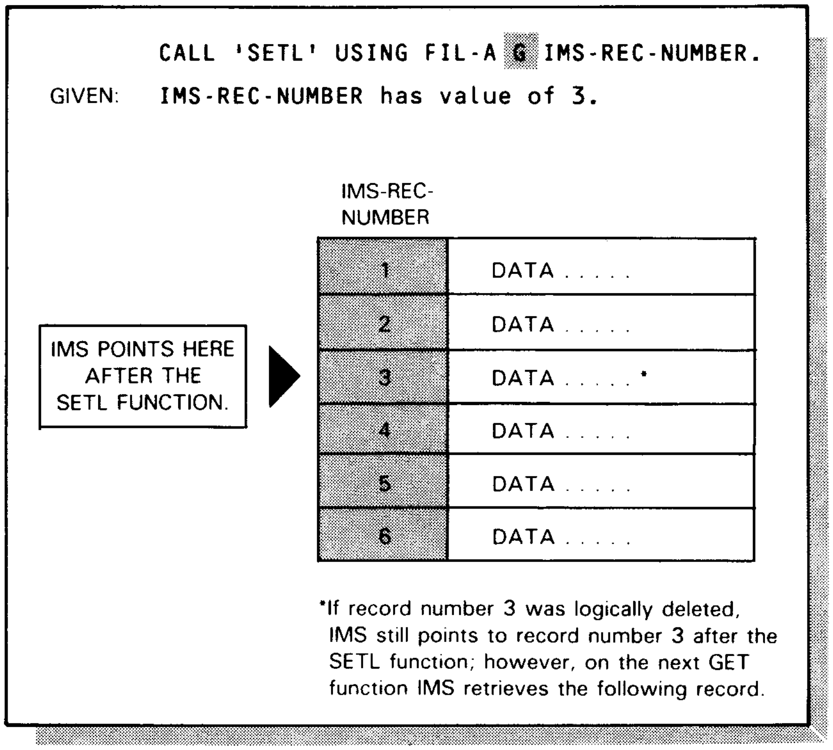
BAL format

```
{CALL    } INSERT,(filename,record-area,record-number)
{ZG#CALL }
```

Required and optional parameters

Record-number required for G, K, and H

You must supply a file name and choose a position value on the SETL function for relative files. The *record-number* parameter is not used with the B position value. When G, K, or H is specified for position, *record-number* must be specified.


CALL 'SETL' USING FIL-A G IMS-REC-NUMBER.
GIVEN: IMS-REC-NUMBER has value of 3.
IMS-REC-NUMBER
1 DATA . . . . .
2 DATA . . . . .
3 DATA . . . . . *
4 DATA . . . . .
5 DATA . . . . .
6 DATA . . . . .
IMS POINTS HERE AFTER THE SETL FUNCTION.

*If record number 3 was logically deleted, IMS still points to record number 3 after the SETL function; however, on the next GET function IMS retrieves the following record.

Errors on SETL

When any error occurs on a SETL function, the file is reset to random mode and any file locks in effect are released. For further sequential processing, you must issue another SETL function call.

Reading Records Sequentially (GET)

Description

The sequential GET function retrieves the next logical record in sequential order unless the record is marked logically deleted (i.e., X'FF' in the first byte). If the record is marked logically deleted, the GET function retrieves the following record. If DELETP=YES is configured, IMS retrieves logically deleted records as normal data.

The COBOL and BAL formats for the sequential GET function call are:

▶ COBOL Format

COBOL format

```
CALL 'GET' USING filename record-area.
```

▶ BAL Format

BAL format

```
{CALL    } SETL,(filename,position [,record-number])
{ZG#CALL }
```

Required parameters

Filename and *record-area* parameters are required.

Errors on sequential GET

When an invalid request error occurs on a sequential GET function, the file is reset to random mode and any file locks in effect are released.

Setting Files from Sequential to Random Mode (ESETL)

Description

The ESETL function changes the mode of relative files from sequential to random. If a file is in the sequential mode for a transaction and you do not issue an ESETL function before termination of the current action, IMS resets the file to random mode. The ESETL function always requires a *filename* parameter.

The COBOL and BAL formats for the ESETL function call are:

▶ COBOL Format

COBOL format

```
CALL 'ESETL' USING filename.
```

▶ BAL Format

BAL format

```
{CALL   } ESETL,(filename)
{ZG#CALL}
```

5.9. ACCESSING SEQUENTIAL DISK AND TAPE FILES

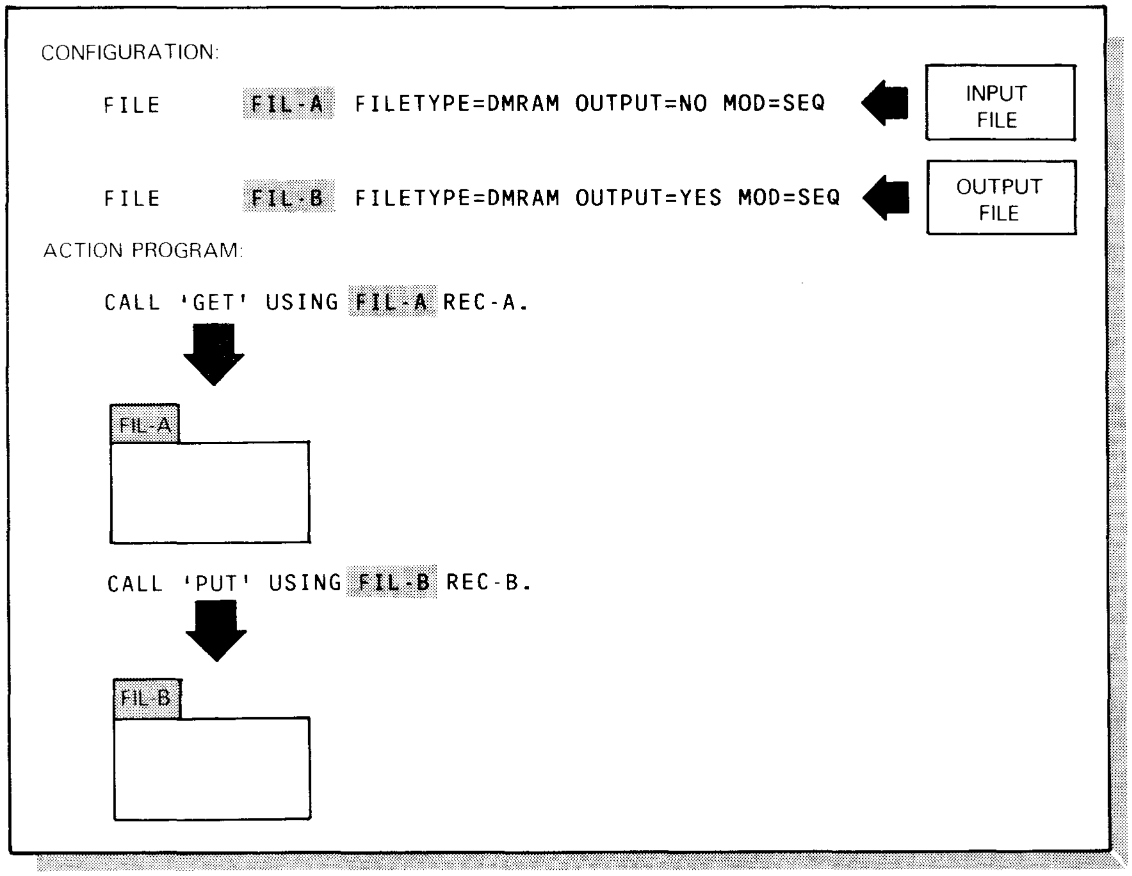
Sequential files are SAM or MIRAM

Sequential MIRAM files configured as MODE=SEQ

The sequential and multiple indexed random access methods (SAM and MIRAM) process function calls issued by your action program to sequential disk or magnetic tape files. A sequential MIRAM disk file is defined in the configurator FILE section as MODE=SEQ.

Same file can't be used for input and output

Only two functions, GET and PUT, are issued to sequential files. You can't use the same SAM or the same sequential MIRAM file for both input and output. (These files are defined individually in the configurator FILE section as input files or output files.) Input files may only be accessed by the sequential GET function. For output files, only the sequential PUT function is used.



CONFIGURATION:

```
FILE    FIL-A  FILETYPE=DMRAM OUTPUT=NO MOD=SEQ
FILE    FIL-B  FILETYPE=DMRAM OUTPUT=YES MOD=SEQ
```

ACTION PROGRAM:

```
CALL 'GET' USING FIL-A REC-A.
CALL 'PUT' USING FIL-B REC-B.
```

For error status codes resulting from the execution of each of the following sequential I/O functions, see Table D-1.

Reading Records (GET)

Description

The sequential GET function retrieves the next logical record in sequential order. Every record in the file is accessible regardless of contents. The first record of a sequential file retrieved in an IMS session is always the first record of the file.

The COBOL and BAL formats for the sequential GET function call are:

▶ COBOL Format

COBOL format

```
CALL 'GET' USING filename, record-area.
```

▶ BAL Format

BAL format

```
{CALL    } GET,(filename,record-area)
{ZG#CALL }
```

Required parameters

Filename and *record-area* parameters are required on the GET function.

Writing Records (PUT)

Description

Required parameters

The sequential PUT function writes fixed- or variable-length logical records to sequential files on tape or disk. *Filename* and *record-area* parameters are always required on this function.

MIRAM file extension

When writing to a MIRAM sequential file the records are appended to the end of the file, thus extending it. If you plan to write a new file, use the INIT parameter on the LFD statement for this file.

The COBOL and BAL formats for the sequential PUT function call are:

▶ COBOL Format

COBOL format

```
CALL 'PUT' USING filename record-area.
```

▶ BAL Format

BAL format

$$\left\{\begin{matrix}\texttt{CALL}\\ \texttt{ZG\#CALL}\end{matrix}\right\}\ \texttt{PUT,(filename,record-area)}$$

5.10. ACCESSING DEFINED FILES

Defined record management accesses defined files

Random and sequential functions

Defined record management services requests from action programs to retrieve and update the records of defined files. An action program can call upon the random access functions GET, GETUP, PUT, DELETE, and INSERT and also the sequential access functions SETL, GET, and ESETL. In response, IMS places defined records into (and takes them from) the record area named in the I/O function call.

Action can access one defined file

A transaction can access only one defined file during a given action – the file that was allocated before the beginning of the action. One action of a transaction can select a defined file not allocated to it and designate that the selected file be allocated to the succeeding action. (See the description of the DEFINED-FILE-NAME field in 3.13.)

Accessing defined and conventional files

During a given action, a transaction can access only one defined file but can also access ISAM, SAM, DAM, or MIRAM conventional files if they are not referenced by the defined file. Access standard files by using the I/O function call formats pertaining to them.

5.11. CONSTRUCTING FUNCTION CALLS TO DEFINED FILES

Certain rules apply to defined files and to the parameters accompanying the function calls for them.

Function Call Positional Parameters

I/O function calls to IMS defined record management use *filename, position, key,* and *record-area* parameters.

Filename

Filename is a data name (COBOL) or storage location (BAL) that contains the 7-byte defined file name or subfile name assigned to this action.

Position

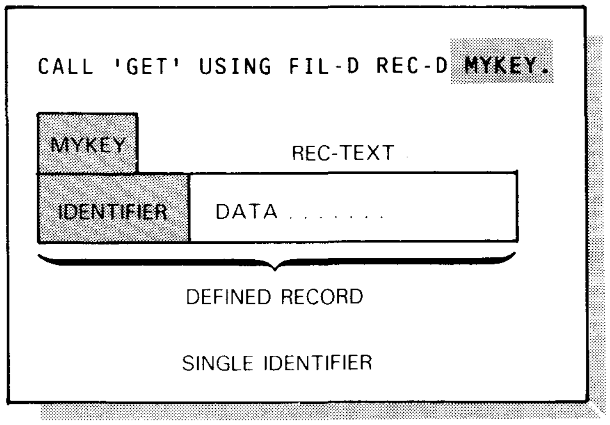
Position is a data name or storage location containing the value B, G, or H that determines which defined record is returned by the first execution of the GET call following the SETL function call.

Key

Key is a data name or storage location that contains the identifier of a defined record. An identifier consists of one or more segments.

Single identifier

Generally, action programs access a defined record via a single identifier.



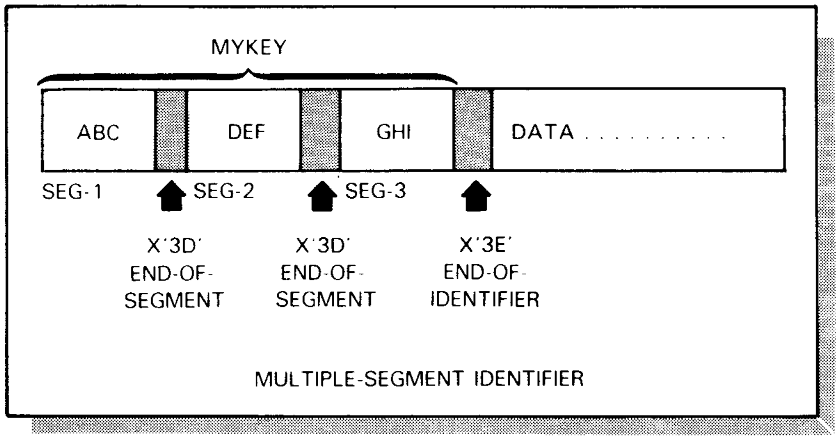
```
CALL 'GET' USING FIL-D REC-D MYKEY.
```

Multiple-segment identifier

There are instances when your program needs to access a defined record that contains an identifier with multiple segments.

A segment must be delimited by an end-of-segment character ($3D_{16}$), unless the segment contains the maximum number of characters defined for it, in which case this character is optional. Every segment must contain at least one character.

The entire identifier must be delimited by an end-of-identifier character ($3E_{16}$). The ignore character ($3F_{16}$) can appear any number of times within the identifier and is always ignored. It is used for editing input messages that contain characters not needed by your action program.



COBOL example of multiple-segment identifier

When this happens, define the identifier with all its segments and separators in your action program linkage section. Define your key (identifier) as a group item in COBOL followed by the segments and separators as follows:

```
Ø1  MYKEY.
    Ø5  SEG-1      PIC XXX.
    Ø5  SEP-1      PIC X.
    Ø5  SEG-2      PIC XXX.
    Ø5  SEP-2      PIC X.
    Ø5  SEG-3      PIC XXX.
    Ø5  SEP-3      PIC X.
```

Before issuing a function call using the *key* value, move the identifier segment values to SEG-1, SEG-2, and SEG-3, and the values '3D', '3D', and '3E' to SEP-1, SEP-2, and SEP-3.

BAL example of multiple-segment identifier

To define an identifier with multiple segments in a BAL action program, use define storage and define constant statements.

```
1        10    16
------------------------------
MYKEY    DS    CL12
         ORG
SEG-1    DS    CL3
SEP-1    DS    XL1
SEG-2    DS    CL3
SEP-2    DS    XL1
SEG-3    DS    CL3
SEP-3    DS    XL1
```

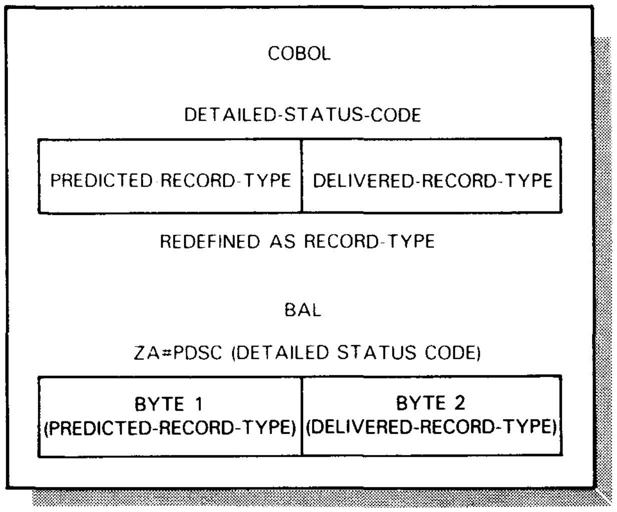
Record-area

Record-area is a data name or storage location that designates the area into which a defined record is moved by IMS on an input function, or from which a defined record is passed to IMS on an output function call. This area must be big enough to contain the entire defined record, including item status bytes.

5.12. PROCESSING DEFINED RECORDS

Determining record type

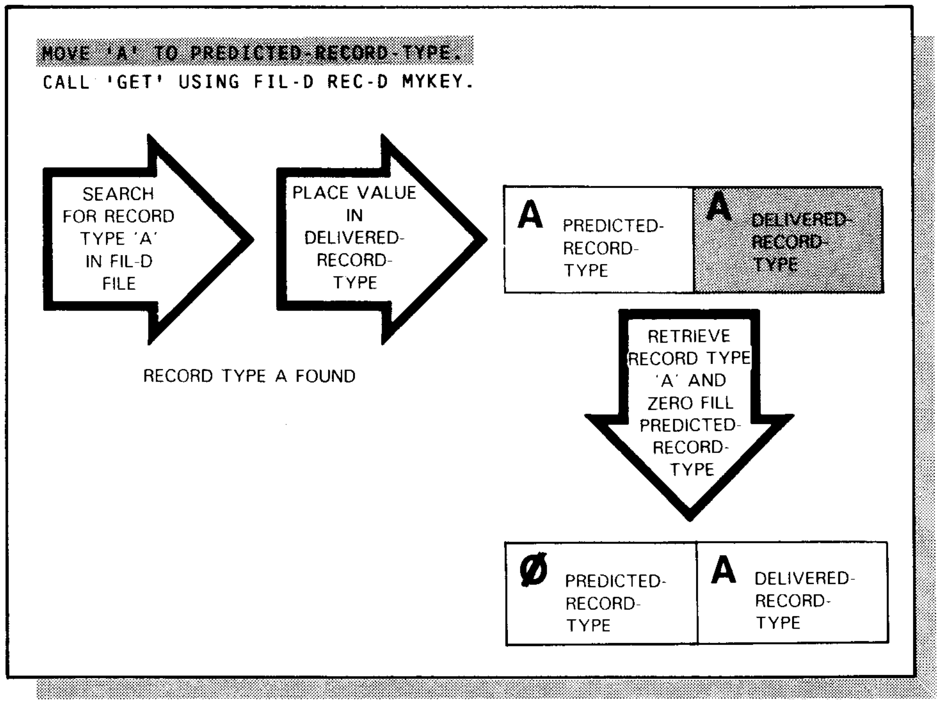
In response to a function call, IMS uses the TYPE statement of the data definition to determine the type of defined record involved in the call. IMS returns the record type to the action program in the program information block's DETAILED-STATUS-CODE field (ZG#PDSC) redefined in COBOL as the RECORD-TYPE field. IMS returns the requested record type in the DELIVERED-RECORD-TYPE portion of the RECORD-TYPE field (byte 2 of the ZA#PDSC in the BAL program information block).



Handling Record Types

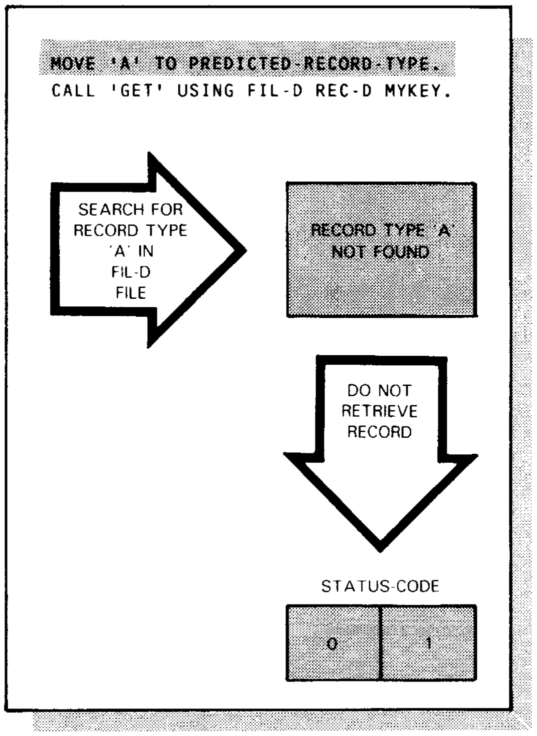
Choosing record type

Before issuing any random GET, GETUP, or INSERT function call, the action program can indicate to IMS the record type it expects to receive by placing the desired record type in the PREDICTED-RECORD-TYPE byte of the RECORD-TYPE field (byte 1 of the ZA#PDSC). If IMS finds a value other than zero, it verifies the prediction before carrying out the retrieval or insertion.


MOVE 'A' TO PREDICTED-RECORD-TYPE.
CALL 'GET' USING FIL-D REC-D MYKEY.
SEARCH FOR RECORD TYPE 'A' IN FIL-D FILE
PLACE VALUE IN DELIVERED-RECORD-TYPE
A PREDICTED-RECORD-TYPE
A DELIVERED-RECORD-TYPE
RECORD TYPE A FOUND
RETRIEVE RECORD TYPE 'A' AND ZERO FILL PREDICTED-RECORD-TYPE
Ø PREDICTED-RECORD-TYPE
A DELIVERED-RECORD-TYPE

Predicted record not found

If the predicted type is not correct, IMS does not move the requested record; instead, it returns a status code of 1 to the calling program.

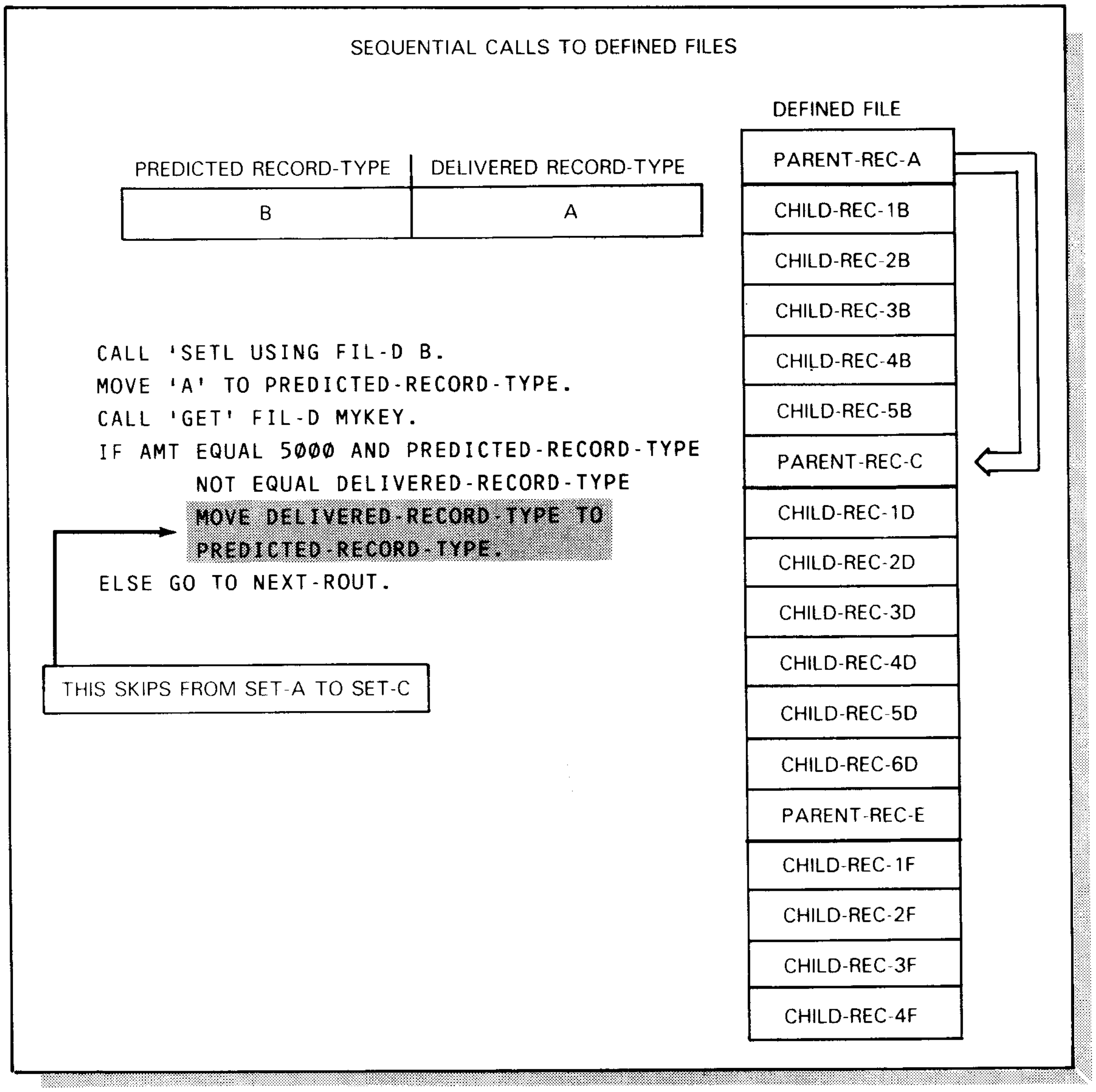

MOVE 'A' TO PREDICTED-RECORD-TYPE.
CALL 'GET' USING FIL-D REC-D MYKEY.
SEARCH FOR RECORD TYPE 'A' IN FIL-D FILE
RECORD TYPE 'A' NOT FOUND
DO NOT RETRIEVE RECORD
STATUS-CODE
0
1

Predicted record found

If the predicted type is correct, IMS performs the function and the PREDICTED-RECORD-TYPE byte reverts to zero. The action program, therefore, can use the PREDICTED- RECORD-TYPE byte before the request to prevent an unexpected type of defined record from being moved to (or from) the record area. If the defined file contains more than one type of defined record, you are strongly advised to use this feature. This assures that further processing applies the correct defined record definition.

Skipping to another record type

When you issue the sequential function calls SETL and GET, IMS returns the record type of the next sequential record to the PREDICTED-RECORD-TYPE byte in the program information block. If the delivered record type is the parent of the predicted record type and you wish to skip over the current record type to the next record type, you can change the contents of the predicted record byte in your action program to equal the DELIVERED-RECORD-TYPE byte. The result is that IMS skips all sets subordinate to the current delivered record type. When one or more records in a set have already been delivered, you cannot change the PREDICTED-RECORD-TYPE byte to skip over the remaining records of that set.



```
CALL 'SETL USING FIL-D B.
MOVE 'A' TO PREDICTED-RECORD-TYPE.
CALL 'GET' FIL-D MYKEY.
IF AMT EQUAL 5000 AND PREDICTED-RECORD-TYPE
       NOT EQUAL DELIVERED-RECORD-TYPE
       MOVE DELIVERED-RECORD-TYPE TO
       PREDICTED-RECORD-TYPE.
ELSE GO TO NEXT-ROUT.
```

Interpreting Status Byte Returns

Testing validity of defined record fields

When IMS responds to a GET, GETUP, PUT, or INSERT function request, it also places a value in the status byte associated with each item of the defined record. (Status bytes are allocated by the data definition processor and have data names in the format *S-item-name*. For sample data definition processor output listings showing status bytes, see the IMS data definition and UNIQUE user guide, UP-9209 (current version).) You can test these values (in COBOL programs for fixed-length records but not variable-length records) to check the validity of individual items in the defined record.

Successful delivery

IMS returns the value X'80' in the status byte for all functions to indicate that the item was successfully delivered.

When item does not exist

For GET and GETUP functions, IMS returns a value of X'40' to indicate that the item cannot be retrieved because it is null (nonexistent). Null items contain blanks if alphanumeric, zeros if numeric. If IMS returns X'40' for one or more items along with a value of zero in the status code, it means a supplement cannot be found via the value in the pointer item. If returned along with a value of 1 in the status code, it means the key parameter points to a nonexistent primary part. See Table D-2 for detailed status codes when the status code is 1.

Item inconsistent with data definition

For PUT and INSERT functions, IMS returns a value of X'20' in the item status byte, along with a value of 5 in the status code to indicate that the item being changed or added does not conform to conditions specified in the data definition. This error can be caused by any of the following:

- ▷ The new item value does not meet VALUE statement conditions
- ▷ The new item value is inconsistent with the PICTURE clause in the data division
- ▷ A change was not permitted for this item (PUT only)
- ▷ No new value was entered for a MUST ADD item (INSERT only)

Updates rolled back

If an error occurs while IMS is accessing a file, before returning control to your action program, IMS changes the LOCK-ROLLBACK-INDICATOR in the program information block to 'O'. This causes a rollback of any updates since the last rollback point.

Table 5-4 shows status byte returns and status codes for the GET, GETUP, PUT, and INSERT function calls to defined files.

Table 5-4. Status Byte Returns for Defined File Functions

Functions	Status Byte Values	Status Codes	Meaning
All	X'80'	X'0000'	Item successfully delivered
GET or GETUP	X'40'	X'0000'	Supplement can't be found using specified pointer
		X'0001'	Key points to nonexistent primary part
PUT or INSERT	X'20'	X'0005'	■ Incorrect VALUE statement ■ Inconsistent PIC clause ■ Change not permitted ■ Value missing for a MUST ADD item.

5.13. RANDOM FUNCTIONS FOR DEFINED FILES

Summary of random functions

I/O function calls to access defined files randomly are the GET, GETUP, PUT, DELETE, and INSERT. During random access to defined files, IMS locks logical records involved in the GETUP and INSERT functions. For error status codes resulting from the execution of each of the following random I/O function calls, see Table D-1.

Reading Defined Records Randomly (GET)

Description

Using a *key* parameter, the GET function retrieves a record from the named file and places the record into the record area of your action program. You cannot update or delete a record retrieved by a GET function.

The COBOL and BAL formats for the GET function call are:

▶ COBOL Format

COBOL format

```
CALL 'GET' USING filename record-area key.
```

▶ BAL Format

BAL format

```
{CALL    } GET,(filename,record-area,key)
{ZG#CALL }
```

Reading Defined Records for Update (GETUP)

Description

Using a *key* parameter, the GETUP function retrieves a record for update from the named file and places the record into the record area of your action program. A GETUP is followed by a PUT or DELETE function. No other function calls to the defined file can intervene.

The COBOL and BAL formats for the GETUP function call are:

▶ COBOL Format

COBOL format

```
CALL 'GETUP' USING filename record-area key.
```

▶ BAL Format

BAL format

```
{CALL    } GETUP,(filename,record-area,key)
{ZG#CALL }
```

Writing Defined Records (PUT)

Description

The PUT function writes a record that was retrieved for update back to the file. For the record to be effectively updated, the PUT function must immediately follow the GETUP function. The COBOL and BAL formats for the PUT function call are:

▶ COBOL Format

COBOL format

```
CALL 'PUT' USING filename record-area.
```

▶ BAL Format

BAL format

```
{CALL   } PUT,(filename,record-area)
{ZG#CALL}
```

Deleting Defined Records (DELETE)

Description

The DELETE function logically deletes a record that was retrieved for update. The DELETE function must immediately follow the GETUP function to effectively delete the record. COBOL and BAL formats for the DELETE function call follow.

▶ COBOL Format

COBOL format

```
CALL 'DELETE' USING filename record-area.
```

▶ BAL Format

BAL format

```
{CALL   } DELETE,(filename,record-area)
{ZG#CALL}
```

Adding Defined Records (INSERT)

Description

The INSERT function enters a new record into a file. The identifier value in the key parameter must not already exist in the file. COBOL and BAL formats for the INSERT function call follow.

▶ COBOL Format

COBOL format

```
CALL 'INSERT' USING filename record-area key.
```

▶ BAL Format

BAL format

```
{CALL   } INSERT,(filename,record-area,key)
{ZG#CALL}
```

5.14. SEQUENTIAL FUNCTIONS FOR DEFINED FILES

Summary of sequential functions

I/O function calls to access defined files sequentially include the SETL, sequential GET, and ESETL function calls. For error status codes resulting from the execution of each of the following sequential function calls, see Table D-1.

Setting Defined Files from Random to Sequential Mode (SETL)

Description

The SETL function sets a defined file into the sequential mode and logically positions the file. The position parameter is a data name or storage location that contains one of the following values:

Position values

Value	Meaning
B	Beginning of file
G	Greater than or equal to key
H	Greater than key

The COBOL and BAL formats for the SETL function call are:

▶ COBOL Format

COBOL format

```
CALL 'SETL' USING filename position [key].
```

▶ BAL Format

BAL format

```
{CALL    } SETL,(filename,position[,key])
{ZG#CALL }
```

When the value of the position parameter is B, the *key* parameter is omitted. The SETL function always returns successful completion (status code of 0).

Reading Defined Files Sequentially (GET)

Description

The GET function retrieves the next defined record in the file in sequential order.

The COBOL and BAL formats for the sequential GET function are:

▶ COBOL Format

COBOL format

```
CALL 'GET' USING filename record-area.
```

▶ BAL Format

BAL format

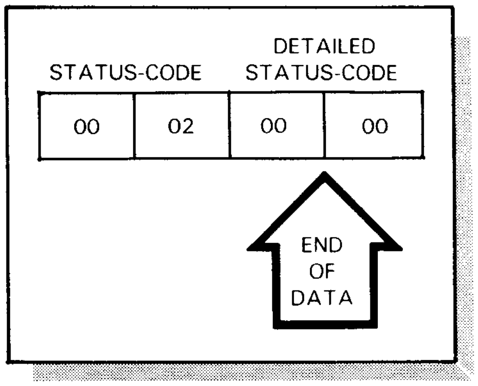
```
{CALL    } GET,(filename,record-area)
{ZG#CALL }
```

Status code 0

If IMS returns a status code of 0 (detail cycle), IMS returns a new defined record to your action program. The DELIVERED-RECORD-TYPE identifies the record type.

Status code 2

A status code of 2 (total cycle) means that there are no more records in the current set. IMS returns no new defined record. The detailed status code (RECORD-TYPE) indicates the record type of the completed set. A status code of 2 with a detailed status code of 0 indicates end of all data; there are no more sets in this defined file.

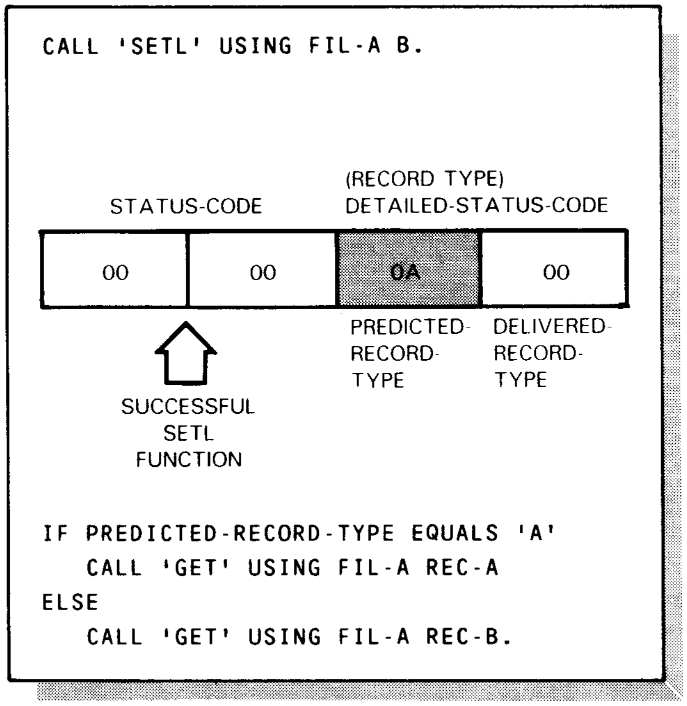


Empty record set

After IMS delivers a detail record, it also delivers all subordinate records in response to subsequent GET function calls. When a set of subordinate records is empty, the response to the GET function that requests the first record of the set is a status code of 2 and a detailed status code (DELIVERED-RECORD-TYPE) equal to the record type of the empty set.

Selecting record areas

Your action program selects the appropriate record area by interrogating the value in the first byte of the DETAILED-STATUS-CODE (PREDICTED-RECORD-TYPE byte) returned by the preceding GET or SETL function.


```
CALL 'SETL' USING FIL-A B.
```

```
IF PREDICTED-RECORD-TYPE EQUALS 'A'
   CALL 'GET' USING FIL-A REC-A
ELSE
   CALL 'GET' USING FIL-A REC-B.
```

Setting Defined Files from Sequential to Random Mode (ESETL)

Description

The ESETL function changes the mode of a defined file from sequential to random. If a file is in the sequential mode and an ESETL function is not performed before termination of the current action, IMS changes the file to random mode at action termination. COBOL and BAL formats for the ESETL function call follow.

COBOL format

COBOL Format

```
CALL 'ESETL' USING filename.
```

BAL format

BAL Format

```
{CALL    } ESETL,(filename)
{ZG#CALL }
```

5.15. UNLOCKING RECORDS (UNLOCK)

Description

The UNLOCK function releases record locks not released as a result of normal transaction termination or file updating. It also makes available for processing, ISAM and MIRAM files held for a transaction pending an update.

The COBOL and BAL formats for the UNLOCK function are:

▶ COBOL Format

COBOL format

```
CALL 'UNLOCK' USING filename.
```

▶ BAL Format

BAL format

```
{CALL    }UNLOCK,(filename)
{ZG#CALL }
```

Applies to lock-for-update and lock-for-transaction

The UNLOCK function applies to both the lock-for-update and lock-for-transaction imposed on DAM, MIRAM, or ISAM files. When you configure either type lock for these files and an update of a record is currently pending for a transaction, the UNLOCK function aborts the update by releasing the record lock. The following lines of COBOL code demonstrate:

Lock release example

```
CALL 'GETUP' USING MYFIL IMS-REC-AREA MYKEY.
MOVE CUST-NAME TO NAME-FIELD.
***********************************************
*    UPDATE PENDING / AWAITING PUT OR DELETE *
***********************************************
CALL 'UNLOCK' USING MYFIL.
```

Releases Lock on MYFIL

UNLOCK for ISAM files

UNLOCK for DAM files

For ISAM files, the UNLOCK function makes the file, as well as the individual record, accessible for processing requests from other transactions. For DAM files, the UNLOCK function unlocks only the individual record. The rest of the file remains accessible to other transactions.

5.16. FILE PROCESSING CONSIDERATIONS

Opening and Closing Files

Files opened at start-up

Closing and reopening files from master terminal

At start-up time, IMS opens all the files you configure and at shutdown time, IMS closes them. You must assign each file in the job control stream at start-up. You can close and reopen files from the master terminal using the master terminal commands, ZZCLS and ZZOPN. When IMS receives these commands, it issues calls to data management to perform close and reopen functions. You cannot open and close files from your action program. For a description of ZZCLS and ZZOPN, see the information management system (IMS) terminal users guide, UP-9208 (current version).

Identifying Files to IMS

Files defined in configurator FILE sections

Describe each of your data files in a FILE section of the IMS configuration. Each file you configure has a single file descriptor entry in the file control table. IMS uses this table to reference files that you access and to queue requests to each file while servicing each request.

Dynamic Allocation of I/O Areas

IMS allocates I/O areas

In a normal programming environment, you would allocate I/O areas to receive data from files and to contain changes sent back to files. In multithread IMS these I/O areas are preallocated. And, in single-thread IMS they are allocated when required. No more than one I/O area is allocated to a file at a given time. Once allocated, an I/O area can be used to support multiple file functions for a number of different transactions. When no function calls to a file are outstanding, IMS releases the I/O area to main storage management.

File Sharing

More than one transaction can share access to a file. Locking procedures for ISAM and MIRAM file updates make it more efficient to program more than one function call in one action (e.g., GETUP and its corresponding function call, PUT or DELETE, in the same action).

Single action updates

The lock on a record being updated can be held from one action to another. However, another GETUP must be issued. It is, therefore, more efficient to update ISAM or MIRAM files in a single action.

Work and Record Area Considerations

If your DAM file resides on a fixed-sector disk (for example, a SPERRY UNIVAC 8416 or 8418 Disk Subsystem), OS/3 data management requires that the length of the I/O area be some multiple of 256 bytes and half-word aligned. To achieve device independence across disk subsysyems, so that your program can access a DAM file on any disk used under OS/3, the same is true – I/O areas should be multiples of 256 bytes in length.

Record area size

To ensure device independence in a BAL or COBOL action program that accesses DAM files, you should ensure that the record-area parameter of any IMS function call (GET, GETUP, PUT, DELETE, or INSERT) refers to an area whose reserved length is some multiple of 256 bytes on a half-word boundary.

Other sizing considerations

There are other considerations (such as record or block length, and the track capacity of the disk subsystem in use) to keep in mind in establishing work-area and record-area lengths for your action programs. For further details, refer to the current versions of the OS/3 data management user guide, UP-8068, or consolidated data management macroinstructions user guide/programmer reference, UP-8826.

Test Mode Effects on File I/O

Simulating file changes

When you enter a ZZTMD terminal command to place that terminal in the test mode, any request to IMS to change the contents of a file are only simulated. No update, delete, or insert functions are performed. Control returns to the requesting transaction with a successful completion status code.

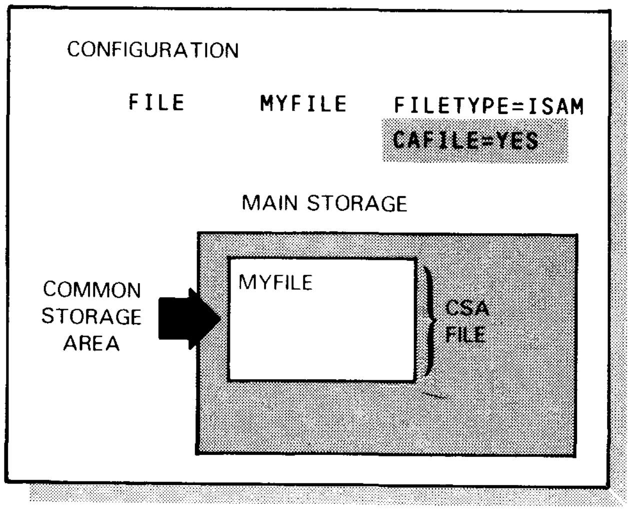
Using test mode and normal mode

You can put a terminal in the test mode after completing a transaction; i.e., when not in an interactive mode. To revert to normal mode, use the ZZNRM terminal command. Test mode is used to train new terminal operators to handle update transactions. All terminal entries made by the operator are the same in test mode as in the normal mode except that no file modifications actually occur. Test mode also is useful in testing newly written or modified action programs that perform file modifications. For more details about the ZZTMD and ZZNRM terminal commands, see the information management system (IMS) terminal users guide, UP-9208 (current version).

Common Storage Area Files

Using common storage area files

You can increase file processing efficiency by making frequently accessed ISAM or MIRAM files resident in a special common storage area (CSA). This feature is especially useful for maintaining vital information used by many action programs. You must have adequate main storage to use this feature.



Accessing CSA files

Invalid function calls to CSA files

You can access a common storage area file only in random mode. You use GET, GETUP, and PUT function calls the same way as for any ISAM or MIRAM file, but INSERT and DELETE functions are not valid.

Updating CSA files

If you specify CUPDATE=YES to the configurator, IMS updates the disk as well as the resident file. This saves disk accesses on reads but not on writes. However, if you omit CUPDATE or specify CUPDATE=NO, IMS does not update the disk file until shutdown, when the entire common storage area file is written to disk. File locking and recovery functions are the same for the common storage area file as for a disk file.

6. Sending Output Messages

6.1. PURPOSE OF OUTPUT MESSAGE AREA

Description

When an action program issues an output message, the message is normally sent from the output message area (OMA).

According to application requirements, action programs can issue output messages:

RETURN function

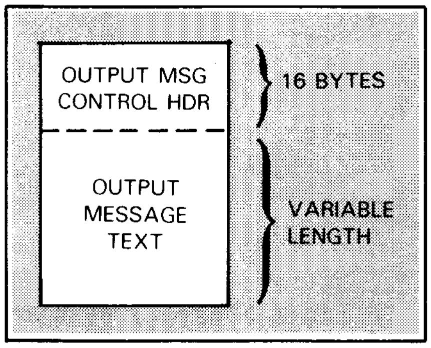
▷ to the source terminal, auxiliary device, or successor action program at the end of an action via the CALL RETURN function; or

SEND function

▷ to the source or other terminal or auxiliary device via the CALL SEND function.

6.2. YOUR ACTION PROGRAM'S OUTPUT MESSAGE AREA CONTENTS

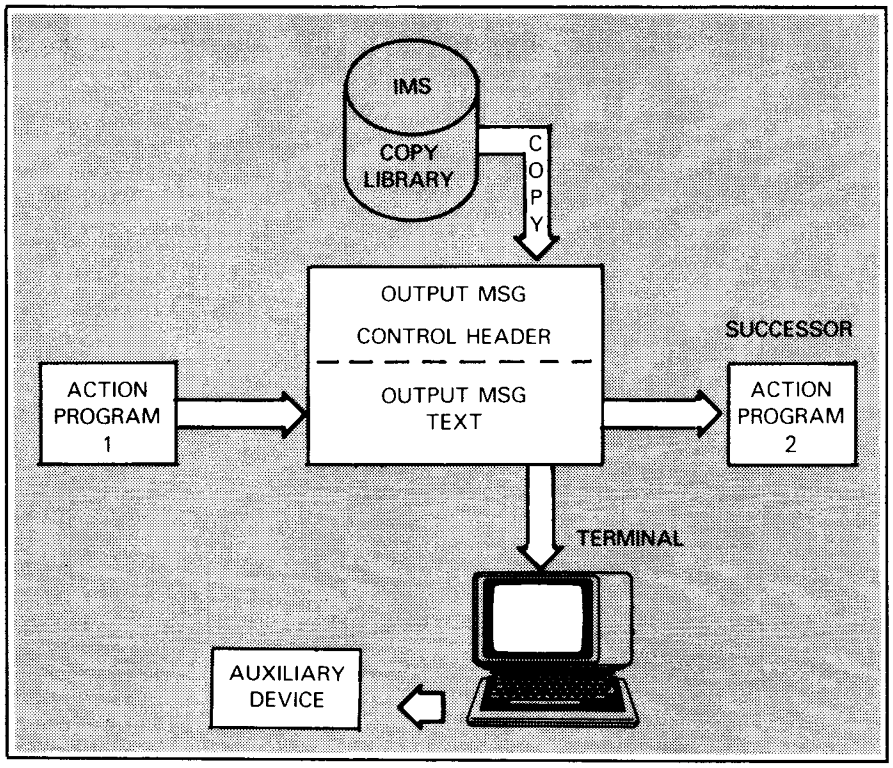
The output message area you describe has two parts: a 16-byte control header and a variable-length message text.



Control header format (COBOL)

Output message text

Your program copies the appropriate COBOL or BAL message control header format from the IMS copy library. The second part of the output message area contains the output message text your program sends to a terminal, auxiliary device, or successor action program.



Action initiation

At action initiation, IMS sets the message text portion of the output message area to blanks.

Action termination

When an action terminates normally, IMS sends the output message to the source terminal unless otherwise specified.

6.3. SIZE OF OUTPUT MESSAGE AREA

OUTSIZE parameter

Factors affecting size

The OUTSIZE parameter in the ACTION section of the configurator specifies the length of the output message area. The size you specify depends on whether you use screen format services for the action and whether you build your screen format in the output message area or in dynamic main storage.

OMA size for screen formatting

If you build a screen format in the output message area, the OUTSIZE value must be large enough to accommodate the screen format buffer contents including variable output data buffer contents, display constants, and device control characters.

Standard OMA size

Instead of specifying an output message area length on the OUTSIZE parameter, you can specify a standard output message size (OUTSIZE=STAN). IMS allocates an output area based on your CHRS/LIN and LNS/MSG parameter values in the GENERAL section of the configuration.

For formulas to calculate output message area length, see the IMS system support functions user guide, UP-8364 (current version).

6.4. COBOL ACTION PROGRAM OUTPUT MESSAGE AREA

Output Message Header Format

COBOL format name

The COBOL output message header format is available in the IMS copy library under the name OMA for extended COBOL or under the name OMA74 for 1974 American National Standard COBOL. Figure 6–1 shows the output message area control header format.

```
01  OUTPUT-MESSAGE-AREA.
    02 DESTINATION-TERMINAL-ID     PIC X(4).
    02 SFS-OPTIONS
       03 SFS-TYPE                 PIC X.
       03 SFS-LOCATION             PIC X.
    02 FILLER                      PIC X(2).
    02 CONTINUOUS-OUTPUT-CODE      PIC X(4).
    02 TEXT-LENGTH                 PIC 9(4) COMP-4.
    02 AUXILIARY-DEVICE-ID.
       03 AUX-FUNCTION             PIC X.
       03 AUX-DEVICE-NO            PIC X.
```

Figure 6–1. COBOL Format for Output Message Area Control Header

Copying the OMA

When you code your COBOL action program's linkage section, copy the output message area control header format into your action program from the IMS copy library using a COPY verb. Once you copy the output message control header from the IMS copy library, your program can access any of these control fields by referencing them in the procedure division.

Output Message Text Description

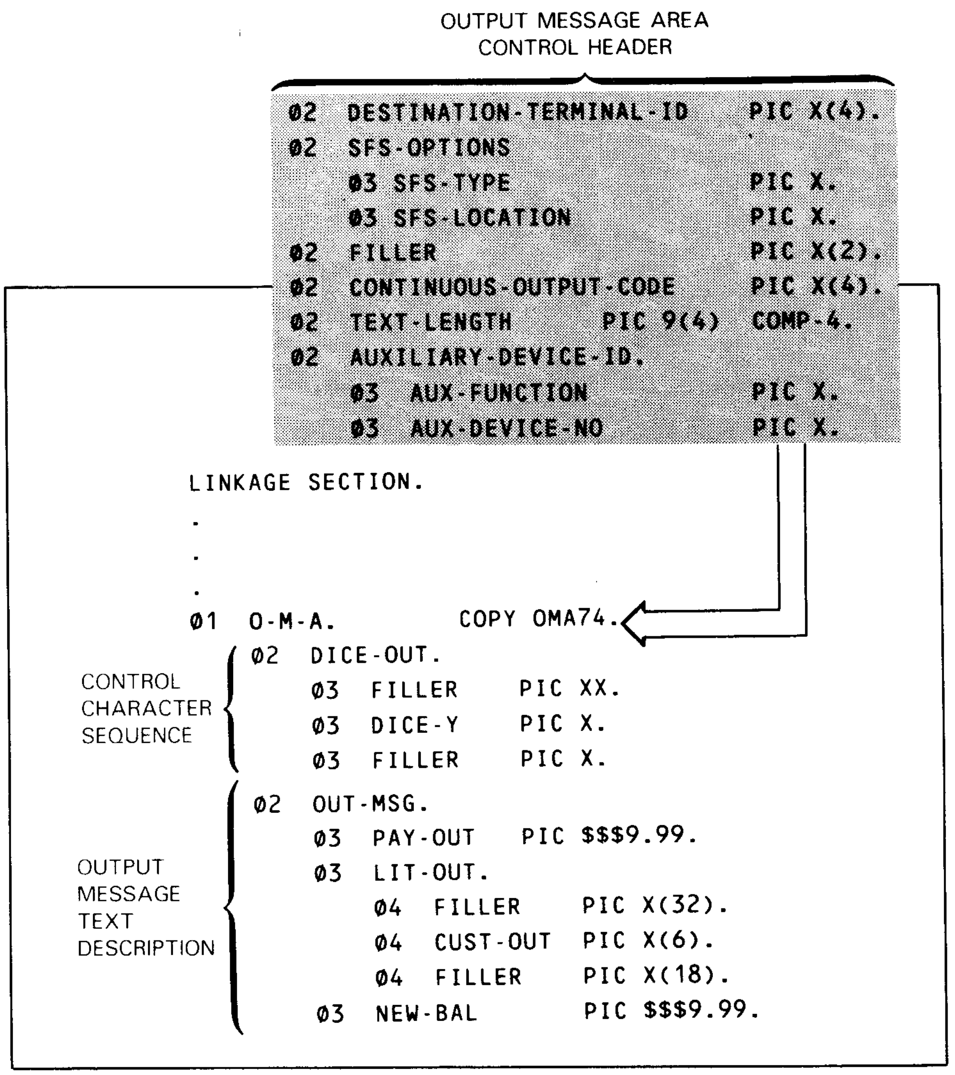
Output message text placement

Output message text description

The output message text description immediately follows the output message control header format copied from the IMS copy library. Describe the output message text fields your program issues to a terminal, auxiliary device, or succeeding action program. Define the output message text as those data items subordinate to the 01-level output message area description. The shaded area in Figure 6–2 shows the output message area control header fields generated by the COPY verb. Fields immediately following the control header represent output text sent by your program.

DICE sequences

Note that the first 02-level item describes the device independent control expression (DICE sequence) that formats the output message. (Appendix F explains this use in detail.) DICE control sequences are needed to position output messages unless you use screen format services (see Section 7).



```
02  DESTINATION-TERMINAL-ID     PIC X(4).
02  SFS-OPTIONS
    03 SFS-TYPE                 PIC X.
    03 SFS-LOCATION             PIC X.
02  FILLER                      PIC X(2).
02  CONTINUOUS-OUTPUT-CODE      PIC X(4).
02  TEXT-LENGTH         PIC 9(4) COMP-4.
02  AUXILIARY-DEVICE-ID.
    03  AUX-FUNCTION            PIC X.
    03  AUX-DEVICE-NO           PIC X.
```

```
LINKAGE SECTION.
.
.
.
Ø1  O-M-A.          COPY OMA74.
    Ø2  DICE-OUT.
        Ø3  FILLER      PIC XX.
        Ø3  DICE-Y      PIC X.
        Ø3  FILLER      PIC X.
    Ø2  OUT-MSG.
        Ø3  PAY-OUT     PIC $$$9.99.
        Ø3  LIT-OUT.
            Ø4  FILLER      PIC X(32).
            Ø4  CUST-OUT    PIC X(6).
            Ø4  FILLER      PIC X(18).
        Ø3  NEW-BAL         PIC $$$9.99.
```

Figure 6-2. Sample COBOL Output Message Area Description

6.5. BAL ACTION PROGRAM OUTPUT MESSAGE AREA

Output Message Header Format

Macroinstruction calls OMA DSECT

IMS also supplies an output message area control header format for BAL action programs. It is in the form of a DSECT called by a macroinstruction in your action program. Figure 6-3 shows the format of the BAL output message area control header.

```
ZA#OMH    DSECT
*
*  OUTPUT MESSAGE HEADER
*
ZA#ODTID DS    CL4               DESTINATION TERMINAL ID
ZA#OSFSO DS    ØCL2              SFS OPTIONS
*
ZA#SFTYP DS    CL1               FORMAT TYPE
ZA#SFLOC DS    CL1               FORMAT LOCATION
*  EQUATES FOR ZA#SFTYP & ZA#SFLOC
ZA#OSFSI EQU   C'I'              INPUT FORMAT
ZA#SFDYN EQU   C'D'              DYNAMIC MEMORY
         DS    CL2               RESERVED FOR SYSTEM USE
ZA#CONT  DS    XL4               CONTINUOUS OUTPUT CODE
ZA#OMHL  EQU   *-ZA#OMH          OUTPUT MSG AREA HEADER LENGTH
ZA#OTL   DS    H                 MESSAGE LENGTH
ZA#OAUX  DS    CL2               AUXILIARY-DEVICE-ID
*
*  EQUATES FOR ZA#OAUX
*
ZA#ONCOP EQU   X'ØØ'             NO COP SUPPORT REQUESTED
ZA#OCO   EQU   X'C3'             CONTINUOUS OUTPUT REQ
ZA#OOIQ  EQU   X'C9'             QUEUE AS INPUT FOR DEST: TCT
ZA#OHANG EQU   X'DØ'             RESERVED FOR IMS/9Ø SYSTEM USE
ZA#OCOP  EQU   X'FØ'             COP OUTPUT REQUESTED
ZA#OCOCP EQU   X'F3'             CONTINUOUS OUTPUT TO COP
ZA#OPTCP EQU   X'F4'             PRINT TRANSPARENT TO COP
ZA#OPCOC EQU   X'F7'             CONTINUOUS OUTPUT TO COP WITH
*                                PRINT TRANSPARENT
*        SS: SPACE SUPRESSION      ISS:     INHIBIT SPACE SUPPRESSION
*        C:     CONTINUOUS OUTPUT     NC:  NOT CONTINUOUS OUTPUT
*
ZA#OCSPM EQU   X'F3'                3: C,SS,PRINT MODE
ZA#ONSPM EQU   X'FØ'                Ø: NC,SS,PRINT MODE
ZA#OCSPT EQU   X'F7'                7: C,SS,PRINT TRANSPARENT
ZA#ONSPT EQU   X'F4'                4: NC,SS,PRINT TRANSPARENT
ZA#OCIPM EQU   X'F5'                5: C,ISS,PRINT MODE
ZA#ONIPM EQU   X'F2'                2: NC,ISS,PRINT MODE
ZA#OCIPT EQU   X'F9'                9: C,ISS,PRINT TRANSPARENT
ZA#ONIPT EQU   X'F6'                6: NC,ISS,PRINT TRANSPARENT
ZA#OCSPF EQU   X'C1'                A: C,SS,PRINT FORM (ESC H)
```

Figure 6-3. BAL Format for Output Message Area Control Header (ZA#OMH DSECT) (Part 1 of 2)

```
ZA#ONSPF EQU   X'D1'               J: NC,SS,PRINT FORM (ESC H)
ZA#OCSTA EQU   X'C2'               B: C,SS,TRANSFER ALL (ESC G)
ZA#ONSTA EQU   X'D2'               K: NC,SS,TRANSFER ALL (ESC G)
ZA#OCSTV EQU   X'C4'               D: C,SS,TRANSFER VARIABLE (ESC F)
ZA#ONSTV EQU   X'D4'               M: NC,SS,TRANSFER VARIABLE (ESC F)
ZA#OCSTC EQU   X'C5'               E: C,SS,TRANSFER CHANGED (ESC E)
ZA#ONSTC EQU   X'D5'               N: NC,SS,TRANSFER CHANGED (ESC E)
ZA#OCIPF EQU   X'C6'               F: C,ISS,PRINT FORM (ESC H)
ZA#ONIPF EQU   X'D6'               O: NC,ISS,PRINT FORM (ESC H)
ZA#OCITA EQU   X'C7'               G: C,ISS,TRANSFER ALL (ESC G)
ZA#ONITA EQU   X'D7'               P: NC,ISS,TRANSFER ALL (ESC G)
ZA#OCITV EQU   X'C8'               H: C,ISS,TRANSFER VARIABLE (ESC F)
ZA#ONITV EQU   X'D8'               Q: NC,ISS,TRANSFER VARIABLE (ESC F)
ZA#OCTIC EQU   X'E8'               Y: C,ISS,TRANSFER CHANGED (ESC E)
ZA#ONITC EQU   X'F8'               8: NC,ISS,TRANSFER CHANGED (ESC E)
ZA#ONTRM EQU   X'D9'               R: C,READ MODE
ZA#ONTRT EQU   X'E2'               S: C,READ TRANSPARENT
ZA#ONTSR EQU   X'E3'               T: C,SEARCH AND READ MODE
ZA#ONTST EQU   X'E5'               V: C,SEARCH AND READ TRANSPARENT
ZA#ONTRA EQU   X'E6'               W: C,REPORT ADDRESS
ZA#OCTBB EQU   X'D3'               L: C,BACK ONE BLOCK
ZA#ONTBB EQU   X'E7'               X: NC,BACK ONE BLOCK
ZA#OCTSP EQU   X'E9'               Z: C,SEARCH AND POSITION
ZA#ONTSP EQU   X'E4'               U: NC,SEARCH AND POSITION
*
*   EQUATES FOR ZA#OAUX+1
*
ZA#ODID1 EQU   C'1'                DEVICE = AUX1
ZA#ODID2 EQU   C'2'                DEVICE = AUX2
ZA#ODID3 EQU   C'3'                DEVICE = AUX3
ZA#ODID4 EQU   C'4'                DEVICE = AUX4
ZA#ODID5 EQU   C'5'                DEVICE = AUX5
ZA#ODID6 EQU   C'6'                DEVICE = AUX6
ZA#ODID7 EQU   C'7'                DEVICE = AUX7
ZA#ODID8 EQU   C'8'                DEVICE = AUX8
ZA#ODID9 EQU   C'9'                DEVICE = AUX9
```

Figure 6–3. BAL Format for Output Message Area Control Header (ZA#OMH DSECT) (Part 2 of 2)

Issuing ZM#DOMH macroinstruction

To generate inline the output message control header (the macro expansion of the ZA#OMH DSECT), you issue the ZM#DOMH macroinstruction in your BAL action program. If you don't want to see the ZM#DOMH macro expansion inline, use the PRINT NOGEN instruction before you issue the ZM#DOMH macroinstruction. Though the output message control header fields are not seen in your program coding, they are still available and you can reference them.

Output Message Text Description

Output message text

Immediately following the ZM#DOMH macroinstruction, you describe the output message text fields your program wants to send to the terminal, auxiliary device, or successor action program. Using defined constant (DC) statements, you describe each field of your output message text.

Figure 6-4 illustrates the macroinstruction that generates the output message control header followed by the description of output text being sent to a terminal (in this case, a 42-byte area containing a 4-byte control character field, the word CAPITAL, and space to enter the name of a state capital). Refer to Appendix B for this example in the full context of the IMS state capital action program. Note that PRINT NOGEN is specified and the ZM#DOMH macro is not expanded inline. Nevertheless, this action program can still access any field in the control header.

DICE sequences

Note that the first four bytes of OUTTEXT contain the device independent control expression (DICE sequence) that clears the line and positions the output message on the new line. (Appendix F explains their use in detail.) DICE control sequences are needed to format output messages unless you use screen format services. (See Section 7.)

```
1        10    16                                                      72
         PRINT NOGEN
.
.
.
*BUILD OUTPUT MESSAGE
         MVC   OUTTEXT(4),NEWLINE           PUT DEVICE INDEPENDENT CONTROL
*                                           CHARACTERS INTO MESSAGE TO CLEAR
*                                           TO END OF LINE AND POSITION TO
*                                           BEGINNING OF NEXT LINE
         MVC   OUTTEXT+4(L'MSGCON1),MSGCON1 PUT TEXT CONSTANT INTO MESSAGE
         MVC   OUTTEXT+4+L'MSGCON1(L'SCAPITAL),SCAPITAL  PUT CAPITAL NAME INTO
*                                                       MESSAGE
.
.
.
*CONSTANTS
STATE    DC    CL7'STATE'                   ISAM FILENAME
MSGCON1  DC    C'CAPITAL'
NEWLINE ZOPOSC 0,0                         {ICAM PROCEDURE TO GENERATE
*                                          {DICE SEQUENCE FOR NEW LINE
*                                          {CONTROL WITH CLEAR
SCAPITAL DS    XL25                         STATE CAPITAL
.
.
.
         ZM#DOMH                            COPY OMA CONTROL HEADER
OUTTEXT  DS    XL42                         OUTPUT MESSAGE TEXT AREA
.
.
.
```

Figure 6-4. Sample BAL Output Message Area Description

6.6. CONTENTS OF OUTPUT MESSAGE AREA CONTROL HEADER

The header format identifies the terminal that is to receive the output message, screen formatting options (if used), continuous output code (if used), the length of the output message text, auxiliary function code (if used), and auxiliary device number (if used). Figure 6-5 shows some of the questions about output messages that the output message control header answers when the action program sets values in the control header fields. Subsections 6.7 through 6.13 describe output message header fields.

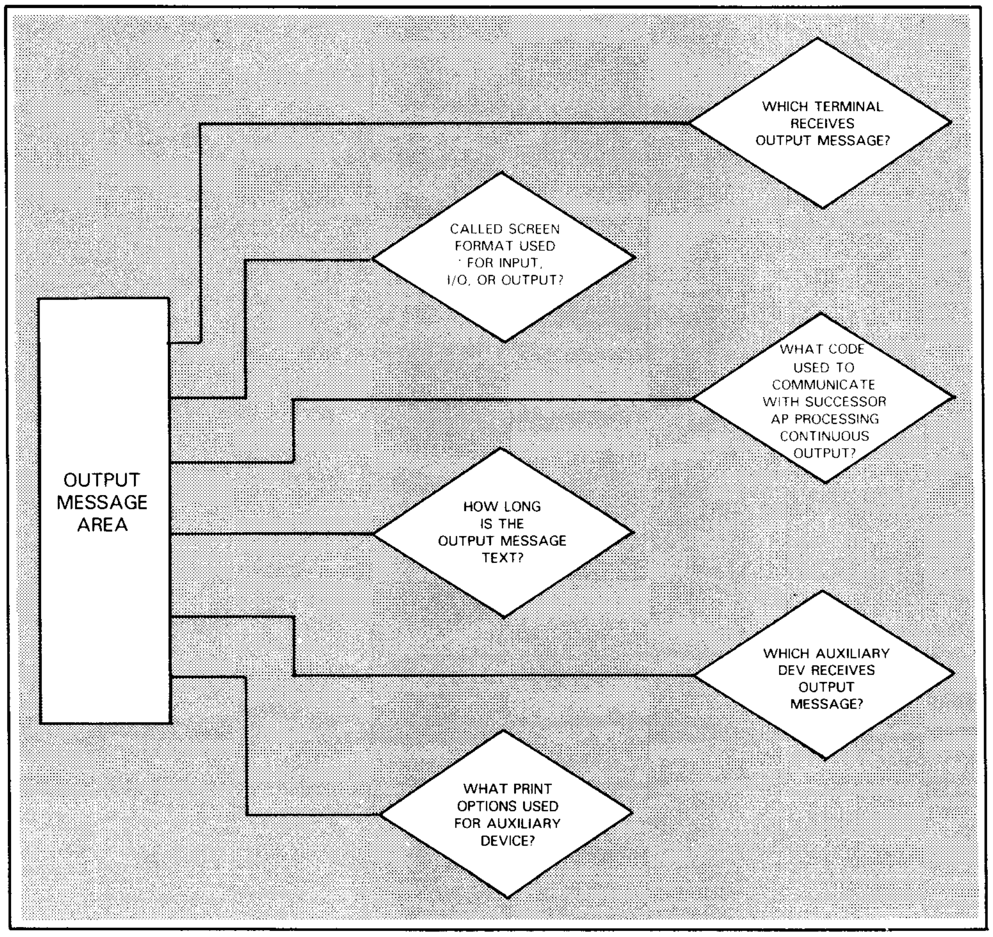

OUTPUT MESSAGE AREA
WHICH TERMINAL RECEIVES OUTPUT MESSAGE?
CALLED SCREEN FORMAT USED FOR INPUT, I/O, OR OUTPUT?
WHAT CODE USED TO COMMUNICATE WITH SUCCESSOR AP PROCESSING CONTINUOUS OUTPUT?
HOW LONG IS THE OUTPUT MESSAGE TEXT?
WHICH AUXILIARY DEV RECEIVES OUTPUT MESSAGE?
WHAT PRINT OPTIONS USED FOR AUXILIARY DEVICE?

Figure 6-5. Answers to Output Message Processing Questions

6.7. IDENTIFYING THE DESTINATION TERMINAL (DESTINATION-TERMINAL-ID)

IMS needs to know the terminal to which it sends the output message your action program builds. The 1- to 4-byte value in the DESTINATION-TERMINAL-ID field (ZA#ODTID) identifies the terminal to which IMS sends the output message.

Destination terminal default

If you don't move a value to this item before issuing a CALL RETURN or CALL SEND, IMS assumes the source terminal to be the destination terminal.

Matching terminal identifications

The destination terminal name must be left-justified and blank filled. Also, you must identify this terminal in your ICAM network definition and optionally in a TERMINAL section of the configuration (Figure 6-6).

ICAM NETWORK DEFINITION

```
.
.
.
LNE1     LINE  DEVICE=(LWS)
WS1      TERM  ADDR=(312),FEATURES=(LWS),LOW=MAIN,INPUT=(YES),          X
               MEDIUM=MAIN,HIGH=MAIN
LNE2     LINE  DEVICE=(LWS)
WS2      TERM  ADDR=(313),FEATURES=(LWS),LOW=MAIN,INPUT=(YES),          X
               MEDIUM=MAIN,HIGH=MAIN
LNE3     LINE  DEVICE=(LWS)
WS3      TERM  ADDR=(314),FEATURES=(LWS),LOW=MAIN,INPUT=(YES)           X
               MEDIUM=MAIN,HIGH=MAIN
LNE4     LINE  DEVICE=(LWS)
WS4      TERM  ADDR=(315),FEATURES=(LWS),LOW=MAIN,INPUT=(YES),          X
               MEDIUM=MAIN,HIGH=MAIN
.
.
.
```

IMS CONFIGURATION

```
.
.
.
TERMINAL WS1   UNSOL=ACTION
TERMINAL WS2   UNSOL=ACTION
TERMINAL WS3   UNSOL=ACTION
TERMINAL WS4   UNSOL=ACTION
TRANSACT MENU  ACTION=JAMENU
TRANSACT SIGN  ACTION=JASIGN
ACTION   JAMENU CDASIZE=1024  EDIT=NONE  MAXSIZE=12000
                OUTSIZE=4096  WORKSIZE=1024
                FILES  SYSCTL,CUSTMST,XREF1,XREF2
ACTION   JASIGN CDASIZE=1024  EDIT=NONE   MAXSIZE=12000
```

Figure 6-6. Identifying the Destination Terminal to ICAM and the Configurator

Using DESTINATION-TERMINAL-ID field

The most common use of the DESTINATION-TERMINAL-ID field is to send an output message to a terminal other than the source. Place a value in the DESTINATION-TERMINAL-ID field before issuing the SEND function to transmit the message.

The following COBOL statement moves a terminal identification other than the source terminal to the output message area DESTINATION-TERMINAL-ID field.

```
MOVE DEST-TERM TO DESTINATION-TERMINAL-ID.
```

The terminal operator enters the value of the desired destination terminal from the source terminal. This value is received in the input message area and described as a text field (DEST-TERM) in the input message area of the program's linkage section. For more detail, see the sample COBOL action program, BEGIN1 in Appendix B, Figure B-24.

6.8. SPECIFYING SCREEN FORMAT SERVICES FOR OUTPUT (SFS-OPTIONS)

SFS-TYPE field values

When you use screen format services for output messages and issue a CALL BUILD for an input or input/output screen format, IMS places a value of I in the SFS-TYPE field (ZA#SFTYP). This means that IMS is to use the screen format you name on your BUILD function call for the following input. When the screen format is for output only, this field contains hexadecimal zeros.

Each time you issue a BUILD function, IMS resets the SFS-TYPE field. To override an input/output format, set this field to hexadecimal zero before issuing a CALL RETURN. This tells IMS to use the screen format you name on the BUILD function call for output only. (For more information describing input only, input/output, and output only screen formats, refer to Section 7.)

SFS-LOCATION field values

To build a formatted output message in dynamic main storage instead of in your output message area, move a character D (C'D') to the SFS-LOCATION field (ZA#SFLOC), the second byte of the SFS-OPTIONS field (ZA#OSFSO). Once you've built the screen format in dynamic main storage, if you want to send a message from the output message area, first clear SFS-LOCATION by filling it with hexadecimal zeros before issuing the SEND or RETURN function. In a COBOL action program, you can do this by coding the statement:

```
MOVE LOW-VALUES TO SFS-LOCATION.
```

In a BAL action program, the statement

```
1            10      16
-------------------------------------
             MVI     ZA#SFLOC,X'ØØ'
```

does the same thing.

For a complete description of screen format services, see Section 7.

6.9. IDENTIFYING A CONTINUOUS OUTPUT MESSAGE (CONTINUOUS-OUTPUT-CODE)

When you issue a continuous output message, an action program can succeed to itself or to an other action program to continue sending output. The CONTINUOUS-OUTPUT-CODE can be used to communicate between the action program that originated the continuous output and its successor.

If you do not move a value into this field, IMS sets the field to zeros and when the program passes control to its successor, the first four bytes of input message received by the successor action program are zeros. Though the CONTINUOUS-OUTPUT-CODE field can be used, this field is not mandatory in generating continuous output. It can, however, be helpful to indicate the last output message sent. Set this field only when the AUX-FUNCTION field indicates that continuous output is desired. For a complete description of continuous output, see 6.17 through 6.23.

6.10. SUPPLYING OUTPUT MESSAGE TEXT LENGTH (TEXT-LENGTH)

Description

The TEXT-LENGTH field (ZA#OTL) is a binary half-word integer that specifies the length of the output message text. IMS sets this value to a predefined output message text length at action initiation and the action program may reduce the value to reflect the true output message text length. This output message length control is necessary when your action program issues multiple output messages. If the value is set to zero and no output message is sent by the action program, IMS sends a default termination message to the source terminal.

OUTSIZE parameter value

The predefined output message text length is specified at configuration via the OUTSIZE parameter in the ACTION section. In your action program, the value you place in TEXT-LENGTH

Additional bytes required

must include the length of the actual text plus four bytes for the TEXT-LENGTH field itself. Be sure to move this value to the TEXT-LENGTH field before your program sends an output message to a terminal. Figure 6-7 shows the logic involved in moving a message text length to the TEXT-LENGTH field in the output message area.

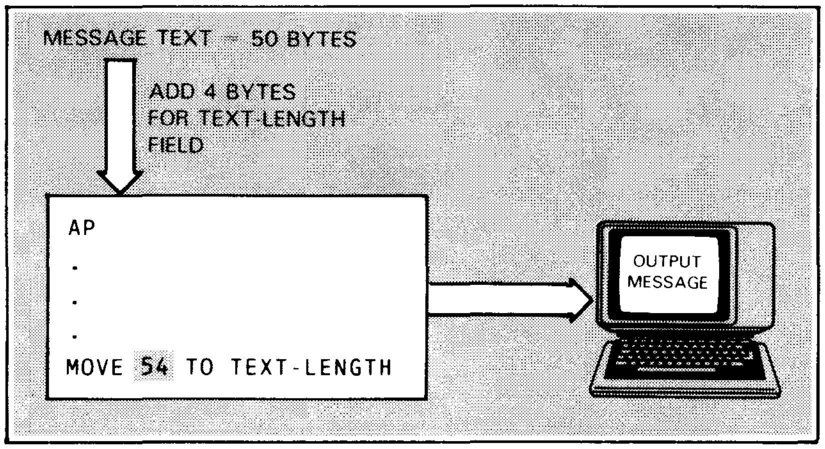


Figure 6-7. Setting Message Text Length for Output Messages

6.11. IDENTIFYING AUXILIARY DEVICES (AUXILIARY-DEVICE-ID)

AUXILIARY-DEVICE-ID field

The AUXILIARY-DEVICE-ID field (ZA#OAUX) is a 2-byte field that indicates whether the output message should be sent to an auxiliary device and, if so, it identifies the device. You also use this field to specify printing options.

Listing message on auxiliary device

To list the output message on an auxiliary device attached to the destination terminal, use each byte of the AUXILIARY-DEVICE-ID field – the AUX-FUNCTION byte (ZA#OAUX) and the AUX-DEVICE-NO byte (ZA#OAUX+1).

AUX-FUNCTION field

AUX-DEVICE-NO field

The AUX-FUNCTION byte describes the print options used for continuous output and when sending the output message to an auxiliary device. For AUX-FUNCTION byte settings refer to Table 6-2. The AUX-DEVICE-NO specifies the number of the auxiliary device receiving the output message (1 through 9), as defined in the ICAM network definition.

Displaying message on primary device

If you don't send the output message to an auxiliary device or want continuous output, set the entire field to binary zeros. This is the original value of the field set by IMS when it generates the output message area control header. Zeroing out this field displays or lists the output message on the primary device – the destination terminal with no special options. The following COBOL coding zeros out the AUXILIARY-DEVICE-ID field in the output message area control header:

```
MOVE LOW-VALUES TO AUXILIARY-DEVICE-ID.
```

6.12. SPECIFYING SPECIAL PRINT OPTIONS FOR AUXILIARY DEVICES (AUX-FUNCTION)

Using AUX-FUNCTION byte

You can choose numerous print options to send output messages to auxiliary devices. For example, to list the output message on the communications output printer (COP) or terminal printer (TP) in print mode, set the AUX-FUNCTION byte to X'F0'; to list it in print-transparent mode, set the AUX-FUNCTION byte to X'F4'.

The AUX-FUNCTION field has another use when you send continuous output to a terminal rather than an auxiliary device. For more detail, see 6.19.

Figure 6-8 shows the coding statements that specify continuous output to an auxiliary device at the primary destination terminal, or continuous output in print-transparent mode at a communications output printer attached to the first auxiliary device configured at that terminal.

```
CREATE-CONTINUOUS-OUTPUT.
    IF COP-OUTPUT NOT EQUAL TO 'COP'
           MOVE 'C' TO AUX-FUNCTION
    ELSE MOVE '7' TO AUX-FUNCTION
           MOVE 1 TO AUX-DEVICE-NO.
    MOVE CURRENT-CONT-CODE TO CONTINUOUS-OUTPUT-CODE.
```

Figure 6–8. Specifying Output to an Auxiliary Device

For an explanation of print mode, print-transparent mode, space suppression, and other print options, see 6.19; also, refer to Table 6–1 for a summary of the AUX-FUNCTION byte settings.

6.13. NAMING AUXILIARY DEVICES (AUX-DEVICE-NO)

Using AUX-DEVICE-NO byte

When you send an output message to an auxiliary device, you must identify its number in the AUX-DEVICE-NO byte of the AUXILIARY-DEVICE-ID field. The value you place in this byte must be a character 1–9. This number identifies the auxiliary device number appended to the AUX operand of the TERM macroinstruction in your ICAM network definition. (See the IMS system support functions user guide, UP-8364, current version.)

AUX operand appendage

If you send an output message to an auxiliary device attached to the destination terminal as shown in Figure 6–8, the network definition must contain a TERM macroinstruction with an AUX operand appended with the same value placed in the AUX-DEVICE-NO. The following portion of a network definition shows the AUX operand with the appended number:

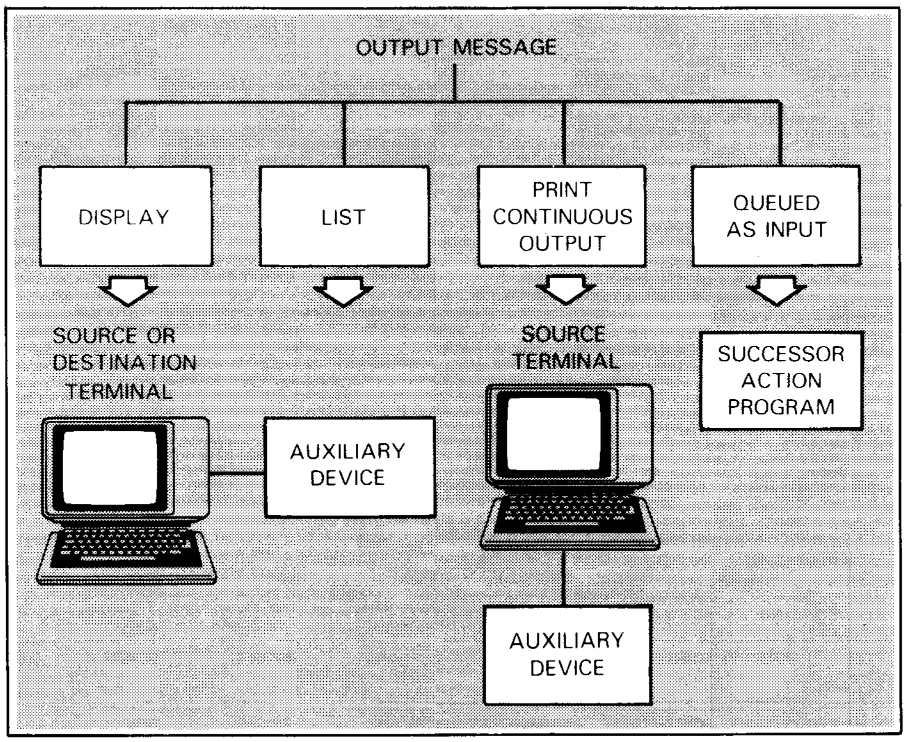
```
MOVE 1 TO AUX-DEVICE-NO.
1        10     16                                        72
TRM1     TERM   ADDR=(29,52),                              X
                FEATURES=(U4ØØ,192Ø),                      X
                AUX1=(TP,77),                              X
                HIGH=MAIN,                                 X
                MEDIUM=MAIN,                               X
                LOW=DQFILE1
```

6.14. SENDING A MESSAGE AT THE END OF AN ACTION

Forms of output

Normally, action programs send messages from the output message area to the designated terminal when you issue the RETURN function at action termination. This output can be:

- displayed on the source terminal or the terminal indicated by the DESTINATION-TERMINAL-ID field;
- listed on an auxiliary device attached to the source terminal or destination terminal;
- printed as continuous output at the source terminal or on an auxiliary device attached to the source terminal (see 6.11); or
- queued as input to a successor action program terminating in delayed internal succession.



6.15. SENDING ADDITIONAL MESSAGES (SEND FUNCTION)

Sending multiple and switched output messages

Sometimes you may want to issue more than one message during an action, or you may want to send a message to a terminal other than a source terminal. This is called switched output. To issue multiple and switched output use the SEND function call.

Transmitting Messages via the SEND Function

Description

The SEND function transmits messages to a terminal other than the source terminal or multiple messages to the source terminal. It can also initiate a transaction at another terminal via output-for-input queueing (described in 6.25). In addition, the SEND function can designate the master terminal as the destination for messages without naming the master terminal in the program. This is useful for sending error messages to the master terminal when the source terminal can't handle the error.

The COBOL and BAL source formats for the SEND function call are:

- COBOL Format:

COBOL format

```
CALL 'SEND' USING output-buffer [master].
```

- BAL Format:

BAL format

$$\left\{\begin{array}{l}\texttt{CALL}\\\texttt{ZG\#CALL}\end{array}\right\}\texttt{SEND,(output-buffer [,master])}$$

Output-buffer parameter

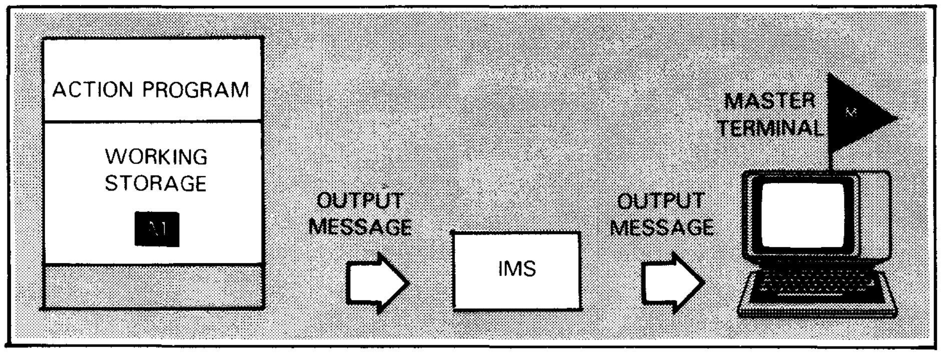
Output-buffer parameter refers to a data-name (COBOL) or storage area (BAL) where the output message is built. This area must contain an output message header and text. The output buffer doesn't have to be the output message area described in the linkage section. You can send an output message from the work area or other interface area. This area, however, must be aligned on a full-word boundary. Subsection 6.16 discusses the use of a work area to build output messages and explains how to send output messages from a work area.

Sending message from work area

Using master parameter

The *master* parameter refers to a data-name or storage location that contains the value 'M' indicating that this message is sent to the master terminal.

Figure 6-9 illustrates COBOL coding to send an output message to the master terminal.



```
WORKING-STORAGE-SECTION.
77 MAST-TERM        PIC X      VALUE 'M'.
.
.
.
PROCEDURE DIVISION.
.
.
.
CALL 'SEND' USING OUTPUT-MESSAGE-AREA MAST-TERM.
```

Figure 6–9. Sending an Output Message to the Master Terminal

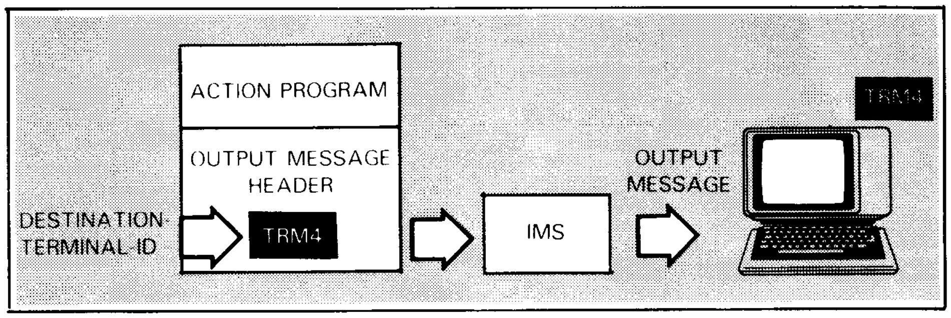
Reference without 'M' value

When the data name referenced does not contain the value 'M', IMS returns a status code of 3 (invalid request) and a detailed status code of 3 (incorrect parameter value) to the program information block of your action program.

Omitting master parameter

When you omit this parameter, IMS sends the message to the terminal specified in the DESTINATION-TERMINAL-ID field of the output message area, or to the source terminal when DESTINATION-TERMINAL-ID is not specified.

Figure 6–10 illustrates the COBOL coding to send an output message to a destination terminal.



```
PROCEDURE DIVISION.
 .
 .
 .
MOVE 'TRM4' TO DESTINATION-TERMINAL-ID.
CALL 'SEND' USING OUTPUT-MESSAGE-AREA.
```

Figure 6-10. Sending an Output Message to a Destination Terminal

Sending messages to the console

Message size restriction

You can send a message to the system console or master workstation if console support is configured. To send a message to the console or master workstation, enter the name 1CNS in the DESTINATION-TERMINAL-ID field. When you send a message to the console, your message may not exceed 120 characters. For more information about the system console and master workstation, see 6.28.

Queued messages to master terminal

IMS does not send an output message to the designated terminal until the successful termination of the current action. After IMS moves the output message from the output message area and writes it to the output message queue, control returns to the statement following the CALL SEND statement.

If the transaction terminates abnormally or is canceled in the current action, IMS deletes from the queue all output messages generated in the action and does not deliver any messages to the terminal. Instead, it sends a message to the source terminal indicating the reason for termination.

SEND function considerations

To use the SEND function, you must specify the UNSOL=YES parameter in the OPTIONS section of the configurator. In your ICAM network definition, you must:

1. Specify FEATURES=(OUTDELV) on the CCA macroinstruction.

2. Create three queues for each terminal (LOW, MEDIUM, and HIGH operands on the TERM macroinstruction).

3. Create at least one process file (PRCS macroinstruction).

4. If a global network, create a static session for each process file in the SESSION macroinstruction.

If you use the SEND function frequently, you should specify disk queueing. Refer to the IMS system support functions user guide, UP-8364 (current version).

Returns from the SEND Function

After executing a SEND function, IMS notifies the action program whether the request succeeded or failed by placing binary values in the STATUS-CODE and DETAILED-STATUS-CODE fields of the program information block. Table 6-1 shows status and detailed status codes IMS can return after unsuccessful completion of the SEND function:

Table 6-1. Status Codes and Detailed Status Codes Returned after the SEND Function

Status-Code (Decimal)	Detailed-Status-Code (Decimal)	Description
0		Successful
3	3	Parameter error
3	12	UNSOL=YES or CONTOUT=YES was not configured, or no process files were created in ICAM network definition.
6	2	Returned when output-for-input queueing is requested and: 1. Destination terminal is in interactive mode 2. Destination terminal has an input message on queue 3. ZZHLD or ZZDWN command was entered for destination terminal 4. Destination terminal is marked physically down to ICAM 5. IMS cannot allocate a main storage buffer (multithread only); INBUFSIZ specification inadequate.
6	3	Destination terminal physically or logically down; message queued
6	4	Invalid destination terminal, auxiliary device, or auxiliary function specified
6	5	No ICAM network buffer available
6	6	Disk error or recoverable system error on output message to console
6	7	Invalid length specification

Detailed-status-code=2

IMS returns a status code of 6 and a detailed status code of 2 only when you use the SEND function to initiate a transaction at another terminal (output-for-input queueing). The conditions causing this error are not permanent. The output message header is valid, and you may be able to retransmit the same message successfully at a later time.

Detailed-status-code=3

Some of the conditions causing a detailed status code of 3 (with status code 6) are the same as those for a detailed status code of 2. However, this error is returned when you use the SEND function for message switching, not output-for-input queueing. In this case, the message sent is queued for the destination terminal and is automatically transmitted when the terminal is operational.

Regaining program control

Abnormal termination

If you configure ERET=YES, the action program regains control at the instruction after the SEND function call and must interrogate these status bytes. If you don't configure ERET=YES, the program does not regain control if the SEND function is unsuccessful and IMS abnormally terminates the program. At this time, IMS also sends a 3-line transaction termination message to the system console. Transaction termination messages are documented in the OS/3 system messages programmer/operator reference, UP-8076 (current version).

6.16. USING A WORK AREA TO BUILD OUTPUT MESSAGES

Configuration

When you use the SEND function you can use the work area or other interface area in the activation record to build your output message. If you decide to use the work area, you must configure the work area size via the WORKSIZE parameter in the configuration ACTION section. IMS does not generate a work area without this parameter. You describe the work area in your action program's linkage section.

Work area size

The length of the work area in multithread IMS equals the WORKSIZE length configured, plus the work area increment (WORK-AREA-INC) length specified by the preceding action. In single-thread IMS, the work area length equals the WORKSIZE length configured. The WORK-AREA-INC value is not supported in single-thread IMS.

Where to build output messages

You can build output messages in four areas in your action program. The output message area is most commonly used. In addition, you have the convenience of building output messages in the work area or continuity data area. If you don't need to save the previous contents of the input message area, you can even build an output message there.

Using RETURN and SEND functions

The important difference is that when you build your output message in the output message area, you may use the CALL RETURN function to transmit the message. On the other hand, you must use the SEND function to transmit messages built in any area other than the output message area.

Directing IMS to output message

When you issue a SEND function to transmit an output message from the output message area or any other area, you must be sure to use the same name you use for the *output-buffer* parameter in your SEND function call as you use for the output message description in your work area or continuity data area. This tells IMS where to go to find the output message you are sending.

Need to code output message header

When sending an output message from any area other than the output message area, you must code your own output message header. You can't use the IMS copy library when creating the OMA header in a section other than the output message area.

Coding example

Figure 6-11 shows the COBOL coding to send a message to the master terminal from the work area.

```
WORKING-STORAGE SECTION.
77 MAST-TERM   PIC X   VALUE 'M'.
.
.
.
LINKAGE SECTION.
.
.
.
Ø1 WORK-AREA.
   Ø2 OUTPUT-MSG.
      Ø3  DESTINATION-TERMINAL-ID   PIC X(4).
      Ø3  SFS-OPTIONS               PIC X(2).
           .
           .
           .
      Ø3 OUTPUT-TEXT-1              PIC X(5Ø).
.
.
.
PROCEDURE DIVISION
.
.
.
PARA-X.
    CALL 'SEND' USING OUTPUT-MSG MAST-TERM.
```

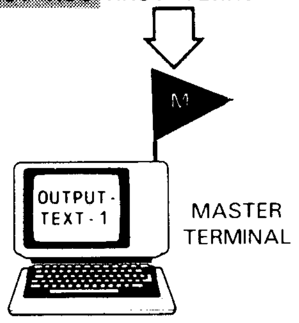


Figure 6–11. Sending an Output Message from the Work Area

6.17. GENERATING CONTINUOUS OUTPUT

When to use continuous output

When you want to print lengthy reports at a terminal or auxiliary device attached to a terminal, the continuous output feature is very useful.

By generating continuous output you can transmit a series of output messages to a terminal, or more commonly to an auxiliary device attached to a terminal, without operator intervention.

Configuring continuous output

To use this feature, you must specify CONTOUT=YES in the OPTIONS section of your configuration.

You also must define an ICAM network that supports unsolicited output. (ICAM requirements are discussed in 6.15.)

Can be used in batch mode

Continuous output can be used in batch processing mode – online for production or offline for listing – as well as in interactive mode.

6.18. DEVICES THAT CAN RECEIVE CONTINUOUS OUTPUT

Action programs can direct continuous output to hard copy terminals or to auxiliary devices (printer, tape cassette, or diskette) at display terminals. For a complete list of terminals and auxiliary devices supported by IMS, see the IMS system support functions user guide, UP-8364 (current version).

6.19. CODING FOR CONTINUOUS OUTPUT

Identifying continuous output message

To distinguish continuous output messages from other output messages, an action program must move a specific value to the AUX-FUNCTION field (ZA#OAUX) of the output message area header. When the program terminates, IMS checks this field and recognizes that the program generated a continuous output message.

Identifying auxiliary devic

If that message goes to an auxiliary device rather than a terminal, the program must also move a value to the AUX-DEVICE-NO field (ZA#OAUX+1) of the output message header. This value tells IMS which auxiliary device (1 through 9) receives the continuous output message. Remember to assign a unique number to each auxiliary device when you define your communications network.

AUX-FUNCTION values

Table 6-2 summarizes the settings for the AUX-FUNCTION field when your action program transmits continuous output to a terminal or to an auxiliary device. Note that you can use these print and transfer options to transmit messages to auxiliary devices for normal output as well as continuous output.

Table 6-4. UNISCOPE and UTS 400 Auxiliary Device Condition Codes

Auxiliary Device Condition	Label①	Hexadecimal Value Equated to Label	Hexadecimal Value when ORed with TM#TDNAX②	UNISCOPE or UTS 400 Auxiliary Status
Ready (good) status but COP/TP write function inoperative	TM#TDDS1	01	41	1
Device out of paper, inoperative, or in test mode	TM#TDDS2	02	42	2
Data error on TCS	TM#TDDS3	03	43	3
Device is not responding; it may be disconnected, or a read of unwritten tape may have occurred.	TM#TDDS4	04	44	4

NOTES:

① Your action program should access the labels in the DSECT instead of testing the value directly, because the equate (EQU) value for each label in the DSECT can vary in future releases. The labels will always remain the same.

② The label TM#TDNAX represents the auxiliary-device-down condition. (Refer to Table 6-3.)

Polled and unpolled devices

The DCT 1000, UNISCOPE 100 and 200, UTS 10, 20, 40, and 400 terminals, and workstations are polled devices and transmit an acknowledgment to ICAM after receiving a continuous output message. The nonpolled devices, Teletype and DCT 500 terminals, do not. For nonpolled devices, a delivery notice is automatically generated; it always indicates successful delivery regardless of whether or not the output message was successfully delivered. Only a line-down condition returns an unsuccessful delivery notice.

Problem caused by nonpolled devices

Consequently, IMS almost always receives a successful completion status from ICAM when a message is delivered to a nonpolled device. IMS sends this delivery code to the successor action program which, in turn, generates more continuous output. As you can see, this is a situation to be avoided. So, in critical parts of continuous output applications, avoid using nonpolled devices.

Queueing and delivery time errors

You can use delivery codes to recover continuous output messages when output message errors are detected at queueing time as well as at delivery time. Errors with hexadecimal values 84 through 87 (Table 6-3) are discovered at output queueing time. All others are detected at the time output is delivered to the terminal.

Reasons for output message errors are:

Error causes

- A missing or invalid destination in the output message header
- An invalid output buffer length in the output message header
- No ICAM network buffer available
- A disk error occurred

If the no-ICAM-network-buffer-available status exists, your action program can try to resend the last continuous output message.

Testing the Delivery Code in a COBOL Action Program

Values returned in delivery code byte

When IMS returns the delivery code in the fifth byte of your action program's input message text, your program must test this byte to see if the continuous output message was delivered successfully. IMS places a hexadecimal 81 or the letter A into this fifth byte when a successful completion occurs. It returns the letter A (hexadecimal C1) when you configure the lowercase-to-uppercase translate option for messages input to a successor action. Otherwise, it returns the hexadecimal value 81. Tables 6-3 and 6-4 list the hexadecimal values for delivery codes returned by IMS.

Coding for delivery code test (COBOL)

To test for a successful delivery code, you can set up a 77-level item in working storage to contain the hexadecimal value 81 or the value A (depending on the translate option configured) and compare the value with the value IMS returns in the fifth byte of the input message text. You can also compare the first 5 bytes of input message text with a 5-byte literal containing the value A or 81 (e.g., ='   A' or ='   81'). Figure 6-12 shows the specific statements needed to test for a successful output delivery code of A. For a complete continuous output program example in COBOL, see the PRINT program in Appendix B.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
77   DEL-NOTICE                       PIC X VALUE 'A'.
LINKAGE SECTION.
Ø1   PIB.COPY-PIB74.
Ø1   IMA.COPY IMA74.
     Ø2   TRANS-IN.
          Ø4   CODE                   PIC X(5).
          Ø4   DEL-NOTICE-MSG REDEFINES CODE.
               Ø8   DEL-NOTICE-CODE       PIC X(4).
               Ø8   DEL-NOTICE-STATUS     PIC X.
          Ø4   FILLER                 PIC X.
          Ø4   TST-NUM                PIC X.
          Ø4   INPUT-TEXT             PIC X(1ØØ).
          Ø4   FILLER                 PIC X(1813).
Ø1   OMA.COPY OMA74.
      Ø2  PRNT-LINE.
          Ø4   DI-1                   PIC 9(4) COMP.
          Ø4   DI-2                   PIC 9(4) COMP.
          Ø4   OUTPUT-TEXT            PIC X(1916).
PROCEDURE DIVISION       USING PIB IMA OMA.
START-HERE.
    IF CODE EQUAL 'PRTPO' GO TO START-IT.
    IF CODE EQUAL 'PPPP' or EQUAL 'TTTT' GO TO TEXT-RETURN.
    IF CODE EQUAL 'CCCC' GO TO CONT-CONTINUE.
    IF CODE EQUAL 'STOP' GO TO TERMINATION-EXIT.
START-IT.
    .
    .
    .
CONT-PRINT.
    .
    .
    .
TEST-RETURN.
    IF DEL-NOTICE-STATUS NOT EQUAL DEL-NOTICE GO TO TERMINATION-EXIT.
CONT-CONTINUE.
    MOVE 'E' TO TERMINATION-INDICATOR.
    MOVE 'BUSØ2Ø' TO SUCCESSOR-ID.
    GO TO ALL-EXITS.
TERMINATION-EXIT.
    MOVE 'N' TO TERMINATION-INDICATOR.
    .
    .
    .
ALL-EXITS.
    CALL 'RETURN'.
```

Figure 6-12. Testing for Successful Delivery Code in a COBOL Action Program

After the PRINT action program determines from a terminal input value that it will process a continuous output message, it processes this message by succeeding to itself (external succession) and testing for a successful delivery code of A in the fifth byte of the input message text after each screenful of output message. If the delivery code is successful, PRINT terminates in external succession. If it is unsuccessful, PRINT handles the error status code and terminates normally. When continuous output is completed, PRINT terminates normally.

Testing the Delivery Code in a BAL Action Program

Accessing TCS DSECT

BAL action programs processing continuous output should access the ICAM labels in the transaction control section (TCS) DSECT, TM#TCS. Tables 6-3 and 6-4 list these labels that correspond with the hexadecimal values equated to the delivery notice status codes.

Generating TCS DSECT

BAL action programs should generate the TCS DSECT inline and access the labels instead of testing the hexadecimal value directly in the input message. The reason for this is that these hexadecimal values are equated (EQU) for each DSECT label and can change in future releases; however, the ICAM DSECT labels always remain the same. If you access the labels, you only have to reassemble your BAL action program with each new release to be sure your DSECT is current; otherwise, you must change your code and reassemble.

To generate the TCS DSECT inline when your BAL program is assembled, call the ICAM procedure, TM#DSECT, using the operand, TCS. Figure 6-13 shows the TM#DSECT procedure and a portion of the ICAM TCS DSECT showing output delivery notice status codes and their labels. Also shown are the specific BAL statements that test for a successful delivery code in the fifth byte of the input message area. Note that the contents listed with each label in the DSECT indicate that the message is being held by ICAM; however, IMS deletes these messages from the queue.

TRANSLAT option considerations

Note also that if you configure TRANSLAT=YES for the action, you cannot use ICAM DSECTs to evaluate delivery status codes because the codes are changed by the translate routine.

PORTION OF ICAM DSECT SHOWING OUTPUT DELIVERY STATUS CODES

```
1            10     16
         TM#DSECT TCS
         .
         .
         .
TM#TDDNA EQU    X'12'     TERMINAL MARKED DOWN, MESSAGE HELD
TM#TDNAX EQU    X'4Ø'     AUXILIARY DEVICE DOWN, MESSAGE HELD
*                         OUTPUT CAN STILL BE SENT TO PRIMARY
TM#TDDS1 EQU    X'Ø1'     UNISCOPE AUXILIARY STATUS ONE
*                         MESSAGE HELD, GOOD STATUS BUT READ/WRITE
*                         FUNCTION INOPERATIVE
TM#TDDS2 EQU    X'Ø2'     UNISCOPE AUX STATUS TWO
*                         MESSAGE HELD, PRINTER OUT OF PAPER,
*                         INOPERATIVE OR IN TEST MODE
TM#TDDS3 EQU    X'Ø3'     UNISCOPE AUX STATUS THREE
*                         MESSAGE HELD, TAPE CASSETTE END-OF-TAPE
TM#TDDS4 EQU    X'Ø4'     UNISCOPE AUX STATUS FOUR
*                         MESSAGE HELD, NO RESPONSE FROM DEVICE WHEN
*                         ATTEMPTING TO READ BLOCK OF TAPE
TM#TDNEM EQU    X'81'     SUCCESSFUL OUTPUT COMPLETION
*
TM#TDLNO EQU    X'11'     LINE DOWN/DISCONNECTED, MESSAGE HELD
*
TM#TEDST EQU    X'84'     MISSING OR INVALID DESTINATION
TM#TENBA EQU    X'85'     NO ICAM NETWORK BUFFER AVAILABLE
TM#TEDER EQU    X'86'     DISK ERROR OCCURRED SERVICING SVC
TM#TEILG EGU    X'87'     INVALID OUTPUT BUFFER LENGTH
*
* TEST DELIVERY CODE
         CLI    CODE+4,TM#TDNEM
         BE     EXIT
*
*
*
EXIT
*
         ZM#IMH
CODE     DS     CL5
TEXT     DS     CL1ØØ
```

Figure 6-13. Testing for Successful Delivery Code in a BAL Action Program

6.23. CONTINUOUS OUTPUT AND CASSETTE/DISKETTE USE

Available options

You can read and write, search, or position data on cassette and diskette auxiliary devices by using the continuous output feature. To do this, you must move a value to the AUX-FUNCTION and AUX-DEVICE-NO fields of the output message area header just as you do when generating a continuous output message. Table 6-2 summarizes the settings for the AUX-FUNCTION field when reading from or writing data to cassettes or diskettes.

Notice in Table 6-2 that all the options beginning with the read option, except backward-one-block and search-and-position, must be used with the IMS continuous output feature. Backward-one-block and search-and-position can be used with continuous output and regular output by simply moving the appropriate value to the AUX-FUNCTION and AUX-DEVICE-NO fields.

Input Options

There are four input options used with cassette/diskette: read, read-transparent, search-and-read, and search-and-read-transparent. The continuous output feature must be used with any of these input options.

1. The **read option** reads a block of data from the cassette/diskette to the terminal screen. When you specify this option, do not put any message text in the output message area. Also, you must move the value 4 to the TEXT-LENGTH field of the output message area header.

2. The **read-transparent** option reads a block of data from the cassette/diskette and the remote device handler deletes the SOE cursor sequence, carriage return codes, and DICE codes.

3. The **search-and-read** option reads a block of data from the cassette/diskette only if a search argument specified in the message text of the output message area is satisfied. When the argument is satisfied, the block of data is moved to the terminal screen. Your search argument may be in one of three search and read modes. Table 6-5 shows the formats for these modes. When you use the search-and-read option, the only contents of the output message area message text should be the search argument in the mode you choose.

4. The **search-and-read-transparent** option performs the same function as the search-and-read option except the remote device handler removes all DICE sequences, SOE cursor sequence, and carriage return characters from the input message.

Table 6-5. User Message Text for Searching Cassette/Diskette

Search Argument Format	Search Type
Ataaaa or 1taaaa or ataaaa	Mode search to position the tape to a particular address and then read one block, where A, 1, or a is constant, and: t Is the track address (1 or 2). aaaa Is the address where the tape is to be positioned.
Btaaaa/c . . . c or 2taaaa/c . . . c or btaaaa/c . . . c	Mode search to position the tape to a particular address, search for a specific character string, and read one block, where B, 2, or b is constant, and: t Is the track address (1 or 2). aaaa Is the block address. c . . . c Is the character string. Up to 16 characters can be specified.
Ct/c . . . c or 3t/c . . . c or ct/c . . . c	Mode search to find the specified character string, where C, 3, or c is constant, and: t Is the track address (1 or 2). c . . . c Is the character string. Up to 16 characters can be specified. The search starts at the present tape position.

Report address option

The **report-address** option displays the address of the cassette/diskette device on the terminal screen. To use this option you must also use the continuous output feature and must specify the value 4 in the TEXT-LENGTH field of the output message area header.

Input options with/without continuous output

The two other options available for cassette/diskette are the search-and-position and backward-one-block options. Only these two input options can be used with either continuous or regular output messages.

- The **search-and-position** option positions the cassette/diskette to the block requested in the search argument that your action program suplies in the output mesage text. (See Table 6-6 for formats used in describing the search argument.) Your output message text cannot contain any other entries.

- The **backward-one-block** option repositions the cassette/diskette one block in reverse. The AUX-DEVICE-NO field must be set and the TEXT-LENGTH field in the output message area must be 4.

Table 6-6. User Message Text for Search and Positioning

Search Argument Format	Search Type
@taaaa or Otaaaa or 'taaaa	Mode search to position the tape, where: @, 0, or (grave accent mark) is constant, and: t Is the track address (1 or 2). ssss Is the address where the tape is to be positioned. If specified as 0000, the tape is rewound.

Identifying continuous output code

In addition to making the required settings in the AUX-FUNCTION and AUX-DEVICE-NO fields of the output message area header, you can also insert into the 4-character CONTINUOUS-OUTPUT-CODE field of the output message area header a code that identifies the continuous output message you generated. This code is returned to the successor program as part of a 5-character input message. If you do not specify a code, the first four characters of the input message generated by IMS for your external successor contains binary zeros.

Using continuous output code

The CONTINUOUS-OUTPUT-CODE field assumes special importance when you use any of the four input options or the report address option for cassettes and diskettes. When you specify one of these options, IMS returns a delivery code to the successor program only if the message wasn't delivered. Otherwise, there is no input to the successor program until a message is transmitted from the cassette/diskette via the terminal screen. For any terminals performing these input options, unless the terminal operator always presses the transmit key, no input is transmitted to the successor program until the AUTO-TRANSMIT feature is set on to allow data to be transmitted from the cassette/diskette.

Precautions for screen bypass terminals

When using a screen bypass terminal, set the control page for that terminal to take advantage of the autotransmit capability. If this is not done for any of these five input options and a successful delivery notice is returned by the cassette/diskette device, the screen bypass terminal stays in the interactive mode waiting for input it won't receive.

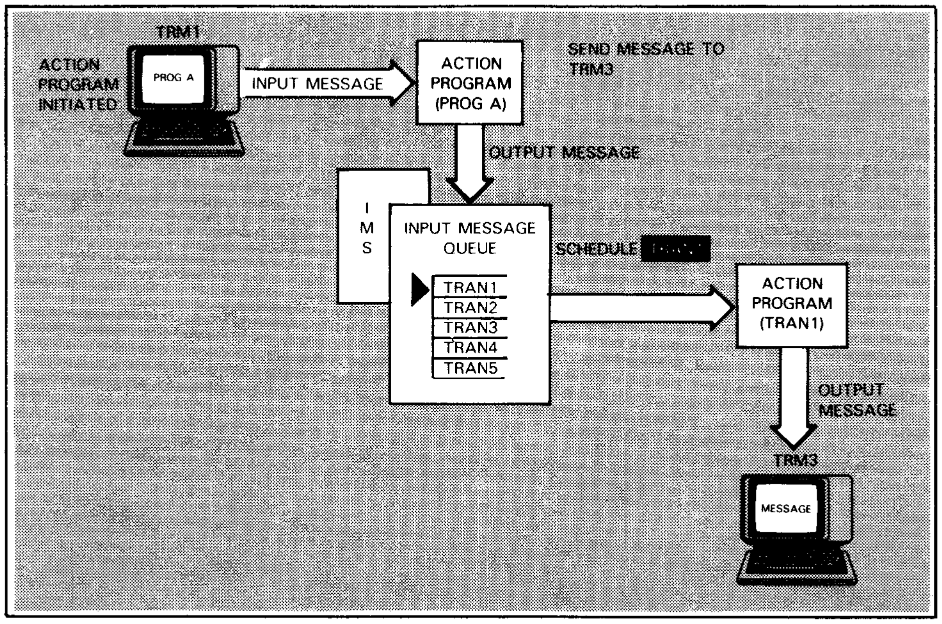
Handling input message or delivery code

Because a successor action program may receive as input either a delivery notice error or an input message from the cassette or diskette, the CONTINUOUS-OUTPUT-CODE specified by the predecessor action program should be distinguishable from the first four characters of any input message being read from the cassette or diskette. In this way, the successor program determines what type of input message it receives (i.e., delivery notice error or input message text) and processes it accordingly. In either case, the successor action program must be capable of handling both unsuccessful delivery notices and standard input messages.

6.24. INITIATING A TRANSACTION AT ANOTHER TERMINAL

Description

Another special capability of an output message generated by an action program is to initiate a transaction at another terminal. We call this output-for-input queueing. It means that when an action program issues a CALL SEND, the output message generated by that program is queued as input to IMS for the destination terminal in the form of a transaction code that initiates a transaction there.



Configuration requirement

To use the output-for-input queueing option, specify the CONTOUT=YES parameter in the OPTIONS section of the IMS configuration.

6.25. CODING FOR OUTPUT-FOR-INPUT QUEUEING

Transmitting output messages for input queueing

Identifying receiving terminal

Example

You must transmit any output message that initiates a transaction at another terminal using a SEND function. To do this, your action program moves the hexadecimal value C9 or the character I to the AUX-FUNCTION field of the output message area header. This value tells IMS to queue the generated output message as input to IMS from another terminal. You identify the receiving terminal by moving its configured value to the DESTINATION-TERMINAL-ID field. Figure 6-14 shows the coding required to accomplish these functions.

```
LINKAGE SECTION.
Ø1    PROGRAM-INFORMATION-BLOCK  COPY PIB74.
.
.
.
Ø1    INPUT-MESSAGE-AREA     COPY IMA.
      Ø1   TEXT              PIC X(1ØØ).
Ø1    OUTPUT-MESSAGE-AREA    COPY OMA.
      Ø2   DESTINATION-TERMINAL-ID     PIC X(4).
      Ø2   SFS-OPTIONS
           Ø3 SFS-TYPE                 PIC X.
           Ø3 SFS-LOCATION             PIC X.
      Ø2   FILLER                      PIC X(4).
      Ø2   CONTINUOUS-OUTPUT-CODE      PIC X(4).
      Ø2   TEXT-LENGTH                 PIC 9(4)  COMP-4.
      Ø2   AUXILIARY-DEVICE-ID.
           Ø3   AUX-FUNCTION           PIC X.
           Ø3   AUX-DEVICE-NO          PIC X.
      Ø2   OUTPUT-TEXT                 PIC X(1ØØ).
PROCEDURE DIVISION     USING PROGRAM-INFORMATION-BLOCK
                             INPUT-MESSAGE-AREA
                             OUTPUT-MESSAGE-AREA.
.
.
.
GO-CONT-OUTPUT.
      MOVE 'I' TO AUX-FUNCTION.
      MOVE 'TRM3' TO DESTINATION-TERMINAL-ID.
      MOVE TEXT TO OUTPUT-TEXT.
      CALL 'SEND' USING OUTPUT-MESSAGE-AREA.
```

Figure 6-14. Initiating a Transaction at Another Terminal

Output message transaction code

The only other requirement is that the output message must contain the transaction code that initiates the new transaction at the destination terminal. This code, and any other output generated along with it, is queued immediately as input to IMS for the destination terminal.

Abnormal termination results

If, after issuing the SEND function using output-for-input queueing, the action program terminates abnormally, then the new transaction is still initiated at the destination terminal.

Unsuccessful STATUS-CODE value

If the destination terminal is in interactive mode when the SEND function is executed (that is, an IMS transaction is already in progress) or if it already has an outstanding input message queued for it, the output message sent using output-for-input queueing cannot cause scheduling of a new transaction. In this case, the action program issuing the SEND function receives an unsuccessful status-code in the program information block. (See 6.27.)

Output-for-input queueing errors

When an action program generates an output message and requests that it be queued as input to another terminal, IMS validates the output message area header and the status of the destination terminal. Any errors are indicated to the originating action program by values returned to the STATUS-CODE and DETAILED-STATUS-CODE fields in the program information block.

Output message text errors

Any errors in the text of the output message (such as, invalid transaction code) are not reported to the originating action program but rather to the action program processing the new transaction at the destination terminal. As a result, this program must be prepared to handle such error conditions, and if necessary, to report these conditions to the originating terminal.

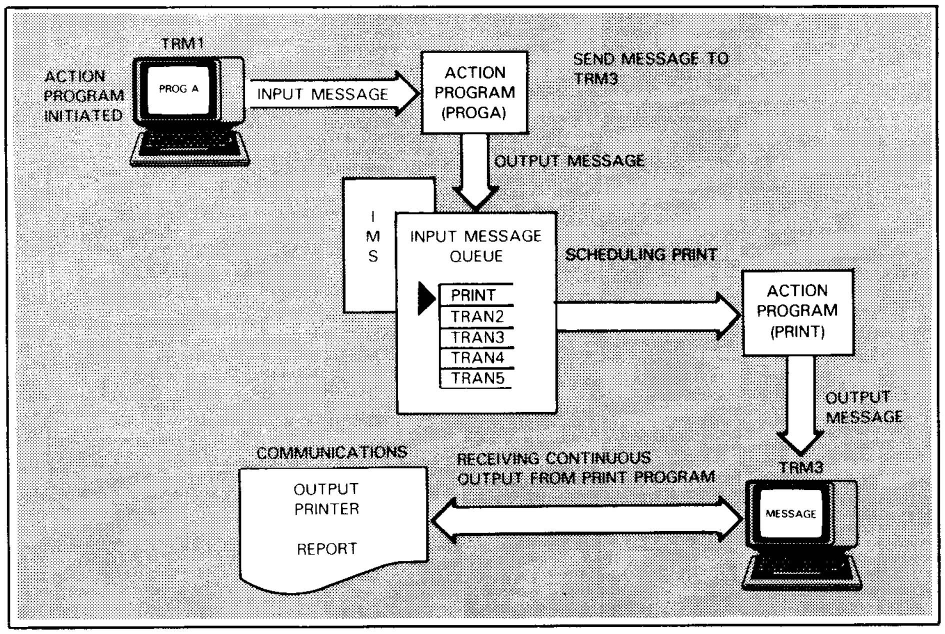
For a complete listing of error codes that IMS returns to the STATUS-CODE and DETAILED-STATUS-CODE fields of your action program following the SEND function, see Table D-2 .

Termination indicators

Generally, a program that generates output using the output-for-input queueing option terminates with normal termination; however, it can specify external succession. It cannot terminate with delayed internal succession.

6.26. OUTPUT-FOR-INPUT-QUEUEING WITH CONTINUOUS OUTPUT

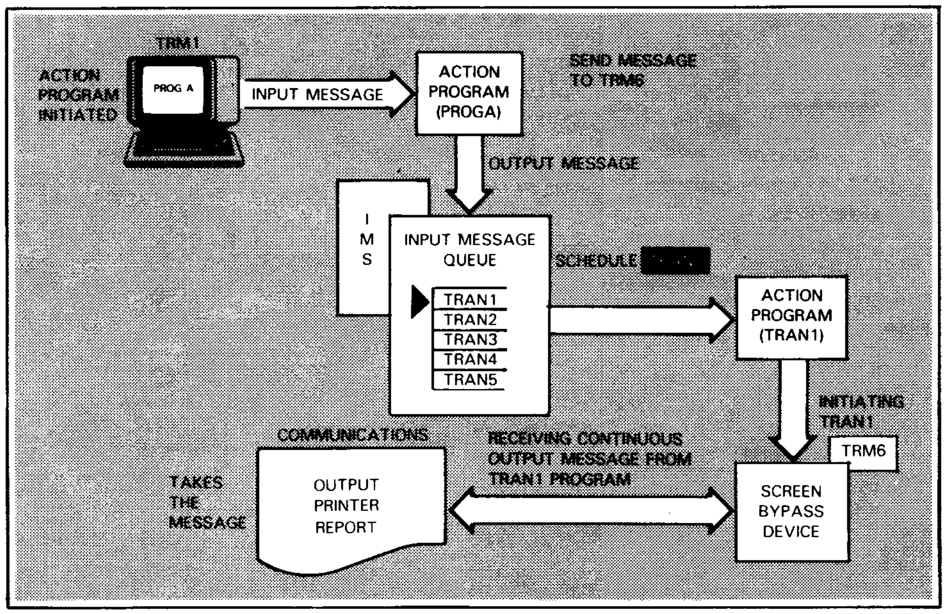
It is fairly common to use the output-for-input queueing and continuous output features together. For instance, one transaction could create the records you want printed and write them to a MIRAM file. The last stage of this transaction could then generate an output message using output-for-input queueing for a destination terminal where another transaction actually prints the records. The transaction initiated at the destination terminal reads the MIRAM file and prints the message as continuous output. See Figures B-24 and B-25 for sample COBOL action programs performing output-for-input queueing and continuous output.



6.27. OUTPUT-FOR-INPUT QUEUEING WITH A SCREEN BYPASS DEVICE

Initiating transactions at screen bypass terminal

Another situation where you can use output-for-input queueing is with the UTS 400 screen bypass device. This device is defined to the communications network as a logical terminal. Nevertheless, because it is physically a separate buffer that can have a telecommunications printer attached to it, it has no way of sending input. Thus, the only way to access a screen bypass device is to use output-for-input queueing. Another terminal in the IMS network calls an action program to generate an output message that initiates a transaction at the screen bypass device. This must be a continuous output transaction and a report could be generated as output on a printer attached to the screen bypass device.



6.28. SENDING MESSAGES TO THE SYSTEM CONSOLE

Configuring console support

Your action program can send output messages to the system console if console support is configured. You configure console support by specifying OPCOM=YES in the OPTIONS section of the IMS configuration or by not specifying a master terminal in any TERMINAL section.

Terminal-id is 1CNS

To send output to the system console, place the terminal-id ICNS in the DESTINATION-TERMINAL-ID field of the output message header:

```
MOVE '1CNS' TO DESTINATION-TERMINAL-ID.
```

When IMS session has a master workstation

Sometimes an IMS session has a master workstation associated with it. A master workstation is a workstation from which the IMS start-up job control stream is entered, or it may be defined in the job control stream. When there is a master workstation and you use the destination-terminal-id 1CNS, your output message goes to the master workstation instead of the console. When the master workstation logs off or is disabled, then the message goes to the console.

Types of output you can send

You can send normal output, multiple output, switched output, continuous output, and output-for-input queueing messages to the system console. However, there are certain restrictions on output to the console:

Auxiliary devices not supported

- You cannot send output to an auxiliary device at the system console. The only auxiliary function settings you can use are hexadecimal 00, C3 (continuous output), or C9 (output-for-input queueing).

Message length restriction

- The maximum length of the output message is 120 characters, not including the output message header. Additional characters are truncated.

No screen formats

- Because of the message length restriction, you cannot output a screen format to the console.

Messages not edited

- Output messages are not edited. DICE functions, FCCs, and other control characters appear as blanks, or in a few cases as printable characters.

No message waiting signal

- There is no message waiting signal. Switched and multiple output messages are sent out immediately.

Error Returns on Output to the Console

Auxiliary device error

IMS returns a status code of 6 and detailed status code of 4 when you attempt to send output to an auxiliary device at the system console. These are the same codes IMS returns when you have an invalid destination terminal, auxiliary device, or auxiliary function specification on output messages to regular terminals.

When console is down

When your output message can't be delivered because the console is physically or logically down, the action IMS takes depends on the type of output message.

Switched and continuous output messages

- With a switched message, IMS returns a status code of 6 and detailed status code of 6. With a continuous output message, IMS returns a delivery notice status of X'86'. These codes indicate recoverable system errors.

Other output messages

- With other types of output messages (such as normal output in response to input from the console), IMS returns a successful status code of 0. The reason IMS does this is that an error status would cause a ''TRANSACTION CANCELLED'' message to be sent to the console, and this could cause an abnormal termination of the IMS session.

PART 3

USING IMS SPECIAL FEATURES

7. Using Screen Format Services to Format Messages

7.1. REQUIREMENTS FOR USING SCREEN FORMAT SERVICES

Saves programming effort

The OS/3 screen format services facility lets you display predefined formatted screens at terminals without tedious programming of DICE codes and other control characters. In addition, screen format services does validation checking of input data. As you know, screen formats simplify the task of data entry and are an essential tool in a transaction processing environment.

BUILD and REBUILD functions

To display screen formats, issue the BUILD and REBUILD function calls in your action program. The BUILD function places the predefined screen format you request in the action program or in a dynamic main storage area; the REBUILD function replenishes input fields or builds an error formatted screen.

Terminals supporting screen formats

You can direct screen formats to any display terminal supported by IMS except the IBM 3270, and also to auxiliary devices attached to display terminals. You cannot output screen formats to hard copy terminals.

Terminal restrictions

UNISCOPE 100 and UNISCOPE 200 terminals must have the screen protection feature, and UTS 400 terminals operating in native mode must have the **PROTECT/FCC** switch set to **FCC** and the control page set to **XMIT VAR.** For local workstations, specify a line buffer length of at least 900 words on the LBL operand in the ICAM network definition.

Screen formats generated offline

Formats stored for later use

You predefine screen formats offline using the screen format generator. (See the screen format services concepts and facilities, UP-8802, current version.) The screen format generator stores the formats in the system screen format file $Y$FMT or other disk file in MIRAM format. The screen formats for an IMS session may reside in one or two screen format files.

Data management mode considerations

To use screen format services, you must generate a supervisor in consolidated data management (CDM) or mixed mode. However, you can configure IMS in either CDM or DTF mode.

Configurator requirements

To make screen format services available to action programs, include the SFS parameter in the OPTIONS section at IMS configuration, specifying the maximum number of terminals that may use screen formats at one time. With the RESFMT parameter, also in the OPTIONS section, specify the number of screen formats you want retained in main storage between function calls.

Start-up requirements

In the job control stream at IMS start-up, include a device assignment set for each screen format file, using the LFD name TC01FMTF for the primary file and TC02FMTF for the secondary file, if there is one.

The IMS system support functions user guide, UP-8364 (current version) describes the configuration and start-up requirements.

Figure 7-1 illustrates the steps you require to create and use screen formats with IMS.

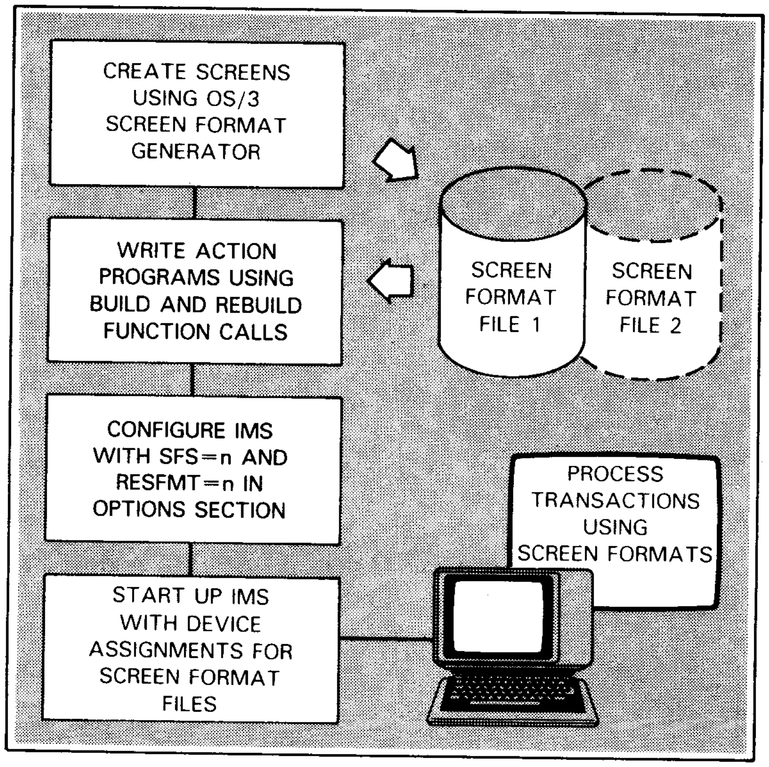

CREATE SCREENS USING OS/3 SCREEN FORMAT GENERATOR
WRITE ACTION PROGRAMS USING BUILD AND REBUILD FUNCTION CALLS
CONFIGURE IMS WITH SFS=n AND RESFMT=n IN OPTIONS SECTION
START UP IMS WITH DEVICE ASSIGNMENTS FOR SCREEN FORMAT FILES
SCREEN FORMAT FILE 1
SCREEN FORMAT FILE 2
PROCESS TRANSACTIONS USING SCREEN FORMATS

Figure 7-1. Creating and Using Screen Formats

7.2. HOW SCREEN FORMATTED MESSAGES ARE PROCESSED

Requesting a screen format

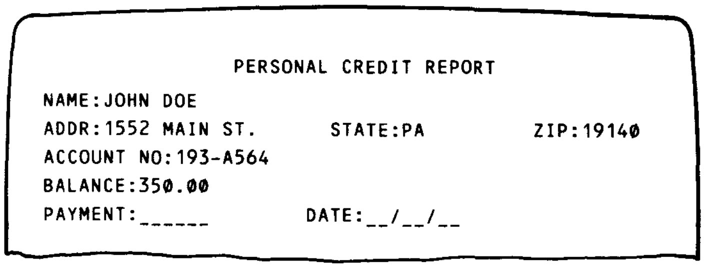
Your action program requests a screen format by issuing a BUILD function call. IMS retrieves the screen format from the screen format file. (When you assign two screen format files, IMS checks TC01FMTF first, then TC02FMTF.) IMS places the screen format in an output buffer area in your program or in dynamic main storage.

Output display constants

The screen format placed in the buffer area contains the output display constants defined at screen format generation. These constants are always protected; the terminal operator cannot change them.

Variable data inserted into screen buffer

IMS inserts into the screen buffer any variable data you supply in the action program. Figure 7-2 shows a screen format containing display constants and variable data. Underlines represent input fields.



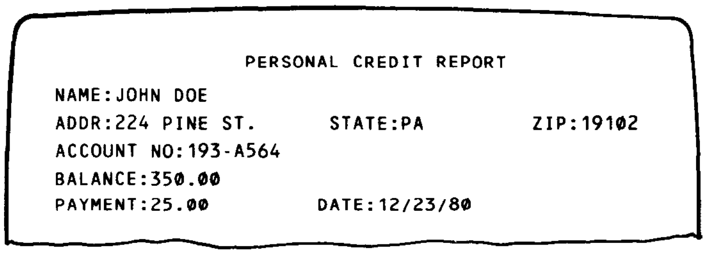
```
                    PERSONAL CREDIT REPORT
NAME:JOHN DOE
ADDR:1552 MAIN ST.         STATE:PA           ZIP:19140
ACCOUNT NO:193-A564
BALANCE:350.00
PAYMENT:______          DATE:__/__/__
```

Figure 7-2. Screen Format with Display Constants, Variable Data, and Input Fields

Input and input/output fields

Output-only fields

Variable fields defined at screen format generation as input or input and output are unprotected. The terminal operator can enter data in input fields and can make changes to input/output fields. Fields defined as output-only are protected. In Figure 7-3, the terminal operator has changed the address field and entered a payment amount and date.

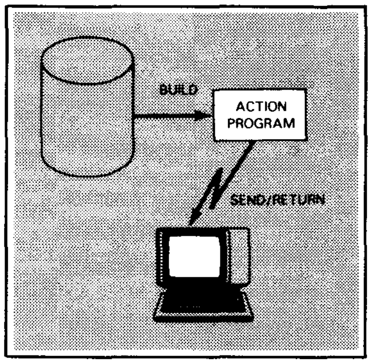


```
                    PERSONAL CREDIT REPORT
NAME:JOHN DOE
ADDR:224 PINE ST.        STATE:PA           ZIP:19102
ACCOUNT NO:193-A564
BALANCE:350.00
PAYMENT:25.00          DATE:12/23/80
```

Figure 7-3. Screen Format with Input Entries and Changed Address Field

RETURN and SEND functions

Like any other output message, screen formats are not actually sent to the terminal until a RETURN function call ends the action. You can also output a screen format by issuing a SEND function call. The CALL SEND lets you send a formatted message to a different terminal or multiple formatted messages to the originating terminal.



Output-only screens required for:
SEND function
continuous output
delayed internal succession

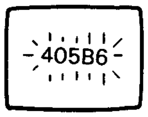
When you use the SEND function or continuous output to transmit a screen format, the format must be output-only, because the terminal operator does not have an opportunity to enter input. Also, when your action program ends in delayed internal succession, you can use only an output format. Instead of going out to the terminal, the screen format is queued as input to the successor action program.

Input/output screens used with:
external succession
normal termination

You can transmit an input/output screen format by terminating the action program with external succession or normal termination. The terminal operator enters input on the format, and IMS schedules a successor action program or a new transaction based on this input.

Transaction code required with normal termination

For normal termination, the first input or input/output field in the format must contain a transaction code. IMS verifies the transaction code and if it is invalid, resends the screen format and causes the transaction code to blink. The terminal operator can reenter the input message.



Input checked for terminal commands

IMS also checks the input for terminal commands. If the input contains a terminal command other than ZZRSD, IMS processes the command and cancels the screen format.

Results when ZZRSD is entered

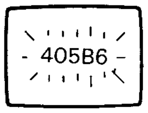
Normally, ZZRSD causes the last output message to be sent again, thus retaining the current screen format. However, if the screen format is built in dynamic main storage instead of the output message area, it can't be sent again and the screen format is canceled. The terminal operator receives a "NO MSG IN QUEUE" message and can't enter input on the formatted screen.

Input validation

When the input does not contain a terminal command or invalid transaction code, the screen format coordinator validates the data before IMS passes it to the successor action program. IMS does no additional input editing regardless of the type of editing configured for the action.

When input data is invalid

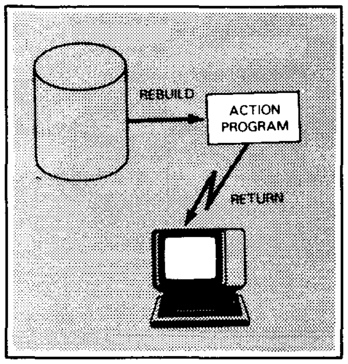
If the input contains errors, the screen format coordinator blinks the invalid fields. The terminal operator can correct the input until the retry count specified at screen format generation time is reached. Once the retry count is exhausted, the successor action program receives control.



Additional input validation

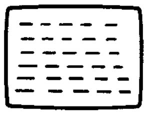
Building an error screen

Your action program can validate input data on a more detailed level than the screen format coordinator. When an action program determines that input data is invalid, you can issue the REBUILD function call to construct an error screen format. IMS replaces fields in which you place hexadecimal F's with blink characters. Then, when your program issues a RETURN function call, the error fields blink on the screen format at the terminal and all other fields remain unchanged.



Replenishing input fields

You can also use the REBUILD function call to replenish input and input/output fields instead of constructing a new screen format for each input record. After the terminal operator transmits an input screen, the input data is replaced by underlines (or other replenish values defined at screen format creation).



Use option indicators to make temporary changes to format

You can make temporary changes to a screen format by defining option indicators at screen format generation time and setting the indicators on before issuing a BUILD function call. Option indicators let you protect fields that are normally unprotected, highlight fields, blink error fields, and replenish input fields. For example, you can build an error screen or replenish screen by using option indicators and issuing a BUILD instead of a REBUILD function call. You cannot use the REBUILD function with a screen format that has option indicators defined.

REBUILD function restriction

7.3. DISPLAYING A SCREEN FORMAT

DO the following in your action program to display a screen format . . .

Defining output buffer

1. Define an output buffer (usually the output message area). This area must be full-word aligned and begin with a 16-byte output message header. When you use the dynamic main storage option, you still need the output message header.

Identifying destination terminal

2. Move the destination terminal-id into the first 4 bytes of the output message header. This step is optional when you want to display the screen format at the source terminal.

Setting text length

3. When you want the screen format built in the output buffer, move the output buffer length into the TEXT-LENGTH field of the output message header. (See the formula described on the OUTSIZE parameter in the configurator ACTION section in the IMS system support functions user guide, UP-8364 (current version).) On return from a successful BUILD function, IMS places the actual length required for the format in this field.

Requesting dynamic main storage

4. When you want the screen format built in dynamic main storage, move C'D' to SFS-LOCATION (COBOL) or set ZA#SFDYN in ZA#SFLOC (BAL).

Identifying screen format

5. Define an 8-byte field containing the name of the screen format. This area must be left-justified and space-filled.

Defining variable data area

Setting option indicator bytes

6. When your screen format uses output option indicators or variable data, define a variable data area and a 2-byte field containing the length of the variable data area. Define option indicator bytes, if any, as the first entries in the variable data area. To set option indicators on, move C'1' to the option indicator byte locations before issuing the BUILD function call.

Defining output status area

7. When you want the screen format coordinator to validate output data, define an output status area large enough to contain one status byte for each variable field.

Issuing BUILD call

8. Issue the BUILD function call.

Overriding input format

9. If you defined an input or input/output screen at screen format generation time and want to use the screen for output-only, move the value X'0' to the SFS-OPTIONS field (COBOL) or ZA#OSFSO field (BAL) of the output message header.

Issuing RETURN or SEND call

10. Issue the RETURN or SEND function call.

Restriction

Once an action program issues the BUILD function, do not change the contents of the buffer area. Modifying the area can cause unpredictable results in both the output screen and any input entered on the format.

Clearing SFS-LOCATION

If you want to send a message from the output message area after building a screen format in dynamic main storage, clear the SFS-LOCATION field to zeros in a COBOL program or move X'00' to the ZA#SFLOC field in a BAL program. This might be necessary, for example, when you output a screen format using the SEND function and then want to output a nonformatted message with the CALL RETURN.

7.4. BUILDING A SCREEN BUFFER (BUILD)

BUILD function constructs screen buffer

The BUILD function call constructs a screen buffer in the output buffer or in dynamic main storage. The screen buffer contains the display constants defined at screen format generation time and any variable data defined in the program.

The COBOL and BAL formats for the BUILD function call are:

- COBOL format

COBOL format

```
CALL 'BUILD' USING output-buffer format-name
  [variable-data data-size [output-status]].
```

- BAL format

BAL format

```
{CALL   } BUILD, (output-buffer,format-name[,variable-data,
{ZG#CALL}             data-size [,output-status]])
```

Output-buffer

Output-buffer identifies the output area where the screen format is built. This area is full-word aligned and begins with a 16-byte output message header. When you use the dynamic main storage option, this area contains only the output message header.

Format-name

Format-name identifies an 8-byte field containing the name of the desired screen format.

Variable-data

Variable-data identifies an area containing output option indicator bytes (if any) followed by a string of variable data (if any). Omit this parameter when your screen format does not use either option indicators or variable data.

Data-size

Data-size identifies a 2-byte field containing the length of the variable data area. This parameter is required when you specify a variable data area.

Output-status

Output-status identifies an area where the screen format coordinator places status errors found in the output validation of variable data. If omitted, no output validation is performed.

7.5. EXAMPLE CODING TO DISPLAY A SCREEN FORMAT

Description of sample coding

Figure 7-4 shows excerpts from a COBOL action program that builds a screen format in the output message area. The program provides two variable data fields (date and time) and a status area for output validation. The complete action program, JAMENU, is illustrated in Appendix B. Figure 7-5 shows the equivalent coding in a BAL action program.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
Ø1   SCREEN-FORMAT-IDS.
     Ø5   SF-MENU                         PIC X(8)  VALUE 'JA$MENU '.
     .
     .
     .
LINKAGE SECTION.
     .
     .
     .
Ø1   WORK-AREA.
     Ø5   IMS-PARAMETER-LIST.
          1Ø   IMS-SCREEN-ID              PIC X(8).
          1Ø   SCREEN-SIZE                PIC 9(4)  COMP SYNC.
     Ø5   SCREEN-RECORD.
          1Ø   SR-DATE                    PIC 9(6).
          1Ø   SR-TIME                    PIC 9(6).
     Ø5   REFORMAT-DATE.
          1Ø   P-MONTH                    PIC 99.
          1Ø   P-DATE                     PIC 99.
          1Ø   P-YEAR                     PIC 99.
     Ø5   SG-STAT                         PIC X(5).
     .
     .
     .
Ø1   OUTPUT-MESSAGE-AREA.    COPY OMA.
     Ø5   OMA-TEXT                        PIC X(3ØØØ).
     .
     .
     .
PROCEDURE DIVISION         USING PROGRAM-INFORMATION-BLOCK
                                 INPUT-MESSAGE-AREA
                                 WORK-AREA
                                 OUTPUT-MESSAGE-AREA
                                 CONTINUITY-DATA-AREA.
     .
     .
     .
```

Figure 7-4. Building a Screen Format in a COBOL Action Program (Part 1 of 2)

```
200-BUILD-SCREEN.
    MOVE SOURCE-TERMINAL-ID TO DESTINATION-TERM-ID.
    MOVE SF-MENU                TO IMS-SCREEN-ID.
    MOVE ALL'Ø'                 TO SCREEN-RECORD.
    MOVE REFORMAT-DATE          TO SR-DATE.
    MOVE TIME-OF-DAY            TO SR-TIME.
    MOVE 12                     TO SCREEN-SIZE.
    PERFORM 5Ø5-BUILD.
    .
    .
    .
5Ø5-BUILD.
    CALL 'BUILD'          USING OUTPUT-MESSAGE-AREA
                                IMS-SCREEN-ID
                                SCREEN-RECORD
                                SCREEN-SIZE
                                SG-STAT.
    IF STATUS-CODE IS GREATER THAN Ø
       MOVE '3' TO ERR-FLAG.
    .
    .
    .
5Ø7-RETURN.
    CALL 'RETURN'.
```

Figure 7–4. Building a Screen Format in a COBOL Action Program (Part 2 of 2)

```
1          10     16                                                    72
PROG1      START 0
*  ALLOCATE REGISTERS TO COVER ACTIVATION RECORD
           USING *,R2
           USING ZA#DPIB,R3
           USING ZA#IMH,R4
           USING WORK,R5
           USING ZA#OMH,R6
           USING CONT-DTA,R7
*  INITIALIZE REGISTERS
.
.
.
*  BUILD SCREEN
           MVC    ZA#ODTID,ZA#ISTID        MOVE SOURCE-TERMINAL-ID TO
*                                          DESTINATION-TERMINAL-ID
           MVC    SCRNID,SFMENU            MOVE SCREEN NAME TO SCREEN-ID
           MVC    SCRNREC(12),ZEROS        CLEAR DATE/TIME FIELD
           MVC    SRDATE(2),ZA#DTE+2       MOVE PIB DATE TO SCREEN RECORD
           MVC    SRDATE+2(2),ZA#DTE+4     AFTER REFORMATTING DATE
           MVC    SRDATE+4,ZA#DTE
           MVC    SRTIME,ZA#TME            MOVE PIB TIME TO SCREEN RECORD
           MVC    SCRNSIZ,TWELVE           SET SCREEN SIZE
           B      SCRNBLD
.
.
.
SCRNBLD    ZG#CALL BUILD,(OMAREA,SCRNID,SCRNREC,SCRNSIZ,SSGSTAT)
           CLI    ZA#PSC+1,X'00'          ERROR CHECKING
           BNE    BLDERR
           B      TERM
BLDERR
.
.
.
TERM     ZG#CALL RETURN
*  CONSTANTS
SFMENU            CL8'JAMENU '            SCREEN FORMAT NAME
ZEROS    DC       CL12'000000000000'
TWELVE   DC       XL2'OC'
*
*  ACTIVATION RECORD DEFINITION
           ZM#DPIB
           ZM#DIMH
.
.
.
WORK       DSECT                          WORK AREA
PRMLST     EQU    *
SCRNID     DS     CL8                     SCREEN IDENTIFICATION
SCRNSIZ    DS     XL2                     SCREEN SIZE
SCRNREC    EQU    *
SRDATE     DS     CL6
SRTIME     DS     CL6
SGSTAT     DS     CL5
OMAREA     ZM#DOMH
OMATEXT    DS    CL3000                 OUTPUT MESSAGE TEXT AREA
```

Figure 7-5. Building a Screen Format in a BAL Action Program

Restriction

Note that the COBOL action program moves zeros to the variable data area before entering values. Do not use the LOW-VALUES figurative because it translates to binary zeros.

Output buffer length

The example action programs do not move the output buffer length into the TEXT-LENGTH field, but we recommend that you do so when building a screen format in the output buffer. This is not necessary when you want to build a format in dynamic main storage.

Coding for dynamic main storage

To build a format in dynamic main storage, include the following statement in a COBOL action program:

```
MOVE 'D' to SFS-LOCATION.
```

In BAL, code the following instruction:

```
1        10      16
---------------------------------------------------
         MVI     ZA#SFLOC,ZA#SFDYN
```

Coding option indicator bytes

When your screen format uses both output option indicators and variable data, code the option indicator bytes as the first entries in the variable data area. For instance, if you defined option indicators that highlight certain fields on the screen format displayed by the COBOL action program in Figure 7-4, the variable data area might look like this:

```
Ø5  SCREEN-RECORD.
    1Ø  OPTION-INDICATOR-1      PIC X     VALUE 'Ø'
    1Ø  OPTION-INDICATOR-2      PIC X     VALUE 'Ø'
    1Ø  SR-DATE                 PIC 9(6)
    1Ø  SR-TIME                 PIC 9(6)
```

Setting option indicators

Then, to turn either option indicator on, move '1' to OPTION-INDICATOR-1 or OPTION-INDICATOR-2.

Remember to include the option indicator bytes in the length of the variable data area:

```
MOVE 14 to SCREEN-SIZE.
```

7.6. ERROR RETURNS FROM THE BUILD FUNCTION

Types of error returns

Action programs can receive two types of error returns:

1. Status codes and detailed status codes in the program information block when the BUILD function is unsuccessful.
2. Error codes in the variable data area when the screen format coordinator finds output validation errors.

Unsuccessful BUILD function

When the BUILD function call is unsuccessful, no screen buffer is constructed and IMS returns one of the following pairs of status and detailed status codes to the program information block:

Status and detailed status codes

Status Code (Decimal)	Detailed Status Code (Decimal)	Explanation
1		Named format cannot be found
3	1	Incorrect number of parameters
3	3	Invalid parameter value
3	12	Screen format services not configured
6	4	Invalid terminal name or type
7	0	Output validation error
7	1	Buffer area not large enough; IMS places the actual length required for the format in the TEXT-LENGTH field
7	2	Variable data area not large enough
7	3	Not enough terminals configured
7	4	Variable-data parameter specified when no variable data area exists
7	5	Format size larger than screen size
7	6	I/O error reading screen format file
7	10	Screen format incorrectly generated

Status Code (Decimal)	Detailed Status Code (Decimal)	Explanation
7	11	System error
7	16	Inadequate main storage available in system; or format contains protected fields and terminal does not have protect feature or is not in protect mode
7	17	Screen format services error
7	18	Action program processing DDP transaction attempted to send screen format to initiating action program.

See Appendix D for a complete listing of status and detailed status codes in hexadecimal.

Output validation errors

When you define variable data and an output status area in your program, the screen format coordinator validates the variable data. When validation errors occur, the screen format coordinator places X'FF' into each error field in variable data area and one of the following error codes into the status byte for each invalid field:

Output validation error codes

Output Validation Error Code	Explanation
1	Nonnumeric value defined for a numeric field
2	Nonalphabetic value defined for an alphabetic field
5	Range check failure
6	Numeric field not in packed decimal format

7.7. RECEIVING FORMATTED INPUT IN THE SUCCESSOR PROGRAM

Termination types allowed

As we already mentioned, you can display an input or input/output screen format only when the action program terminates with external succession or normal termination. The terminal operator enters input on the format, and IMS schedules a successor action program or a new transaction based on this input.

Function key input

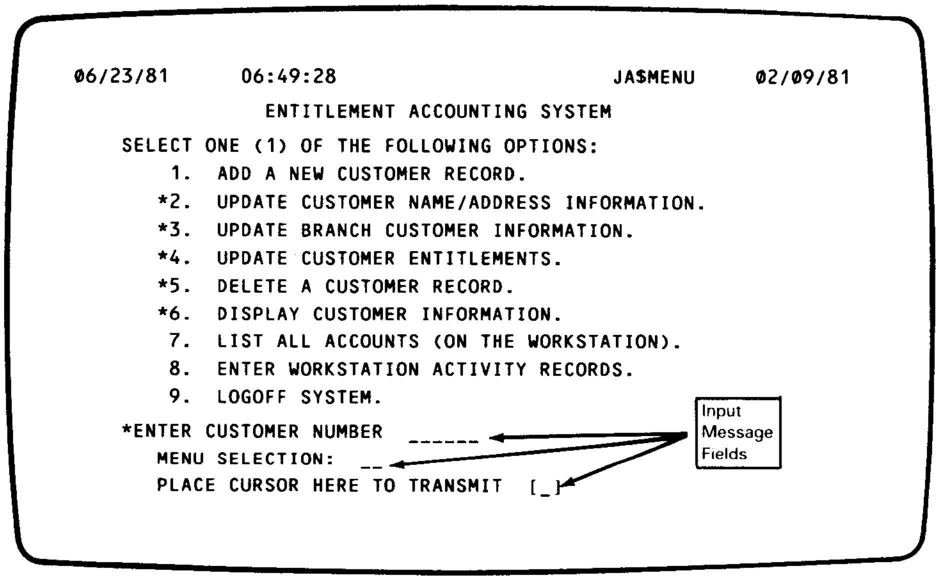
The operator can enter a function key instead of formatted input, if the action program is prepared to accept it. A function key cancels the screen format.

External succession

When the action program displaying the screen format terminates with external succession, IMS schedules the action program named in the SUCCESSOR-ID field of the program information block and sends the input data entries to the successor program's input message area.

Receiving formatted input

In the JAMENU action program in Appendix B, the same COBOL action program displays a screen format and also accepts input entered on the format. After building the screen format, JAMENU terminates with external succession, naming itself as successor. Figure 7-6 shows the screen format JAMENU displays, and Figure 7-7 shows the input message fields to receive the formatted input.



```
06/23/81        06:49:28                    JA$MENU     02/09/81
                   ENTITLEMENT ACCOUNTING SYSTEM
    SELECT ONE (1) OF THE FOLLOWING OPTIONS:
        1.  ADD A NEW CUSTOMER RECORD.
       *2.  UPDATE CUSTOMER NAME/ADDRESS INFORMATION.
       *3.  UPDATE BRANCH CUSTOMER INFORMATION.
       *4.  UPDATE CUSTOMER ENTITLEMENTS.
       *5.  DELETE A CUSTOMER RECORD.
       *6.  DISPLAY CUSTOMER INFORMATION.
        7.  LIST ALL ACCOUNTS (ON THE WORKSTATION).
        8.  ENTER WORKSTATION ACTIVITY RECORDS.
        9.  LOGOFF SYSTEM.
    *ENTER CUSTOMER NUMBER  ______
       MENU SELECTION:  __
       PLACE CURSOR HERE TO TRANSMIT  [_]
```

Figure 7-6. Screen Format Displayed by JAMENU Action Program

```
01  INPUT-MESSAGE-AREA.    COPY IMA.
    .
    .
    .
    05  IMA-SCREEN-REC REDEFINES IMA-PASS-1.
        10  SR-CUST-NBR              PIC 9(6).
        10  SR-MENU                  PIC 99.
        10  SR-TRSMIT                PIC X.
        10  FILLER                   PIC X(4).
```

Figure 7-7. Input Message Area Fields for Formatted Input

Normal termination

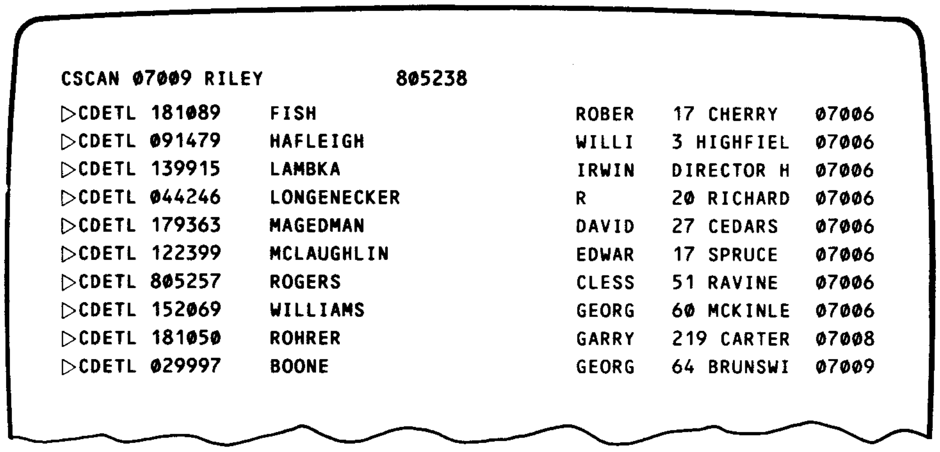
In the case of normal termination, the first input field in the format must contain a valid transaction code, because IMS must schedule a new transaction to receive the input data. IMS sends the input data, including the transaction code, to the action program named in the configurator TRANSACT section.

Defining a transaction code as input/output variable

A convenient way to ensure that the terminal operator enters the appropriate transaction code in the first input field is to define that field as an input/output variable. Display the transaction code, and when the terminal operator transmits the screen the transaction code is automatically entered as input data.

Example screen format displaying transaction codes

Figure 7-8 shows an input/output screen format displayed in response to the CSCAN transaction code. Initially, the cursor is positioned after the CSCAN transaction code. To list more names and addresses, the terminal operator simply presses the transmit key and the CSCAN transaction is rescheduled. To get details about a certain customer, the operator positions the start-of-entry character and cursor on the line for that customer and transmits. This schedules the CDETL transaction. (The CSCAN and CDETL action programs in Appendix B do not use screen format services but could have generated the same screens with screen format services.)



```
CSCAN Ø7ØØ9 RILEY           8Ø5238
▷CDETL 181Ø89    FISH                        ROBER   17 CHERRY    Ø7ØØ6
▷CDETL Ø91479    HAFLEIGH                    WILLI   3 HIGHFIEL   Ø7ØØ6
▷CDETL 139915    LAMBKA                      IRWIN   DIRECTOR H   Ø7ØØ6
▷CDETL Ø44246    LONGENECKER                 R       2Ø RICHARD   Ø7ØØ6
▷CDETL 179363    MAGEDMAN                    DAVID   27 CEDARS    Ø7ØØ6
▷CDETL 122399    MCLAUGHLIN                  EDWAR   17 SPRUCE    Ø7ØØ6
▷CDETL 8Ø5257    ROGERS                      CLESS   51 RAVINE    Ø7ØØ6
▷CDETL 152Ø69    WILLIAMS                    GEORG   6Ø MCKINLE   Ø7ØØ6
▷CDETL 181Ø5Ø    ROHRER                      GARRY   219 CARTER   Ø7ØØ8
▷CDETL Ø29997    BOONE                       GEORG   64 BRUNSWI   Ø7ØØ9
```

Figure 7-8. Displaying Transaction Codes in Input/Output Fields

Programming efficiency

Although you can display an input/output screen format using either external succession or normal termination, external succession is more efficient. For a complete example of an action program using a screen format with external succession, see the JAMENU program in Appendix B. JAMENU also uses immediate internal succession to pass control to succeeding action programs that process the menu selection entered by the terminal operator.

NOTE:

Input option indicators

You can define certain input option indicators at screen format generation time. IMS does not support these input option indicators. However, if you defined any input option indicators for this screen format, perhaps for use with another program, you must code option indicator bytes as the first entries in the input message area.

7.8. VALIDATING INPUT DATA

Invalid entries

The screen format coordinator validates the input data entered at the terminal and blinks invalid fields. The terminal operator can correct the invalid entries until the retry count specified at screen format generation time is reached. At that point, IMS schedules the successor program and places a 7 in the STATUS-CODE field and 0 in the DETAILED-STATUS-CODE field in the program information block.

IMS sets status and detailed status codes

Input status bytes

The input data is followed by one status byte for each input field. You must allow space for these fields in your input message area, but the length field in the input message header includes only the input data items and not their status bytes. When validation errors occur, the screen format coordinator places an error code into the status byte for the invalid fields and replaces the invalid fields with X'FF'. The input validation error codes are:

Error fields filled with X'FF'

Input validation error codes

Input Validation Error Code	Explanation
1	Nonnumeric keyin for a numeric field
2	Nonalphabetic keyin for an alphabetic field
3	Incorrect number of characters entered
4	Decimal point alignment error
5	Range check failure

When your program receives a validation error, you will probably want it to send a message to the terminal operator and terminate the transaction.

7.9. DISPLAYING AN ERROR FORMAT OR REPLENISH SCREEN

Using the REBUILD function

After the terminal operator enters input on a screen format and the screen format coordinator validates the input, you can retain the format at the terminal and make changes to it by issuing a REBUILD function call. You can use the REBUILD function in two different ways:

Constructing error screen

1. Construct an error screen. Your action program performs additional validation of input fields and fills the input fields that are in error with X'FF' (HIGH-VALUES). When you issue the REBUILD, the screen format generator blinks any input fields filled with X'FF'.

Constructing replenish screen

2. Construct a replenish screen to prompt the terminal operator for the next input. When you issue the REBUILD function call, the screen format generator replaces input and input/output fields with underlines or other replenish value defined at screen format generation.

Identifying error fields

When you want to build an error screen, identify the area containing the error fields (usually the input message area) with the *variable-data* parameter on the REBUILD function. Omit this parameter when you want to build a replenish screen.

Defining output buffer

As with the BUILD function, you must define an output buffer, full-word aligned and starting with a 16-byte output message header.

Where screen is built

You can request that the error or replenish screen be built in the output buffer or in dynamic main storage. However, because of the smaller size of the message you send with the REBUILD function, you may want to use the output buffer instead of dynamic main storage.

Output buffer length

Dynamic main storage

If you want the screen built in the output buffer, move the output buffer length into the TEXT-LENGTH field of the output message header. (To determine the output buffer length, allow approximately 10 bytes per blinking field or replenish field plus 25 bytes for overhead.) To build the screen in dynamic main storage, move C'D' to SFS-LOCATION (set ZA#SFDYN in ZA#FLOC).

Use RETURN function, not SEND function

Termination types allowed

After issuing the REBUILD function to construct an error or replenish screen, issue the RETURN function to send the screen to the terminal. Never use the SEND function with a CALL REBUILD, because the error or replenish screen requests input from the terminal operator. For the same reason, you must terminate the action program with external succession or normal termination.

Using option indicators instead of REBUILD function

You can also build an error or replenish screen (or a combination) by using option indicators and issuing a second BUILD function call instead of the REBUILD function. When you build an error screen this way, you do not have to fill the error fields with X'FF'. Set the appropriate indicators on by moving C'1' to the option indicator byte locations before issuing the BUILD function call. You cannot use the REBUILD function with a screen format that has any option indicators defined.

7.10. BUILDING AN ERROR OR REPLENISH SCREEN (REBUILD)

REBUILD function constructs error or replenish screen

The REBUILD function call constructs an error or replenish screen in the output buffer or in dynamic main storage. The screen format from the previous BUILD function remains in effect at the terminal, and error fields are blinked or input fields are replenished.

The COBOL and BAL formats for the REBUILD function call are:

- COBOL format

COBOL format

```
CALL 'REBUILD' USING output-buffer [variable-data].
```

- BAL format

BAL format

```
{CALL    }REBUILD, (output-buffer[,variable-data])
{ZG#CALL }
```

Output-buffer

Output-buffer identifies the output area where the error or replenish format is built. This area is full-word aligned and begins with a 16-byte output message header. When you use the dynamic main storage option, this area contains only the output message header.

Variable-data

Variable-data identifies an area containing the input message fields including error fields. This is usually the input message area.

Include for error screen, omit for replenish screen

When you include the *variable-data* parameter, the screen format coordinator blinks all fields filled with X'FF'. When you omit this parameter, the screen format coordinator replaces all input and input/output fields with the replenish value you defined at screen format generation, usually underlines.

7.11. EXAMPLE CODING TO DISPLAY AN ERROR OR REPLENISH SCREEN

Displaying an error screen

Assuming you displayed the screen format shown in Figure 7-6 using the BUILD function, Figure 7-9 shows an example of the COBOL coding to validate the menu selection field and display an error screen using the REBUILD function. Figure 7-10 shows this coding in a BAL action program.

Note in the COBOL coding that the input fields are redefined as alphanumeric. This is necessary because you cannot move HIGH-VALUES to a numeric field.

```
Ø1  INPUT-MESSAGE-AREA.    COPY IMA.
    .
    .
    .
    Ø5  IMA-SCREEN-REC REDEFINES IMA-PASS-1.
        1Ø  SR-CUST-NBR                     PIC 9(6).
        1Ø  SR-CUST-NBR-ERR REDEFINES SR-CUST-NBR  PIC X(6).
        1Ø  SR-MENU                         PIC 99.
        1Ø  SR-MENU-ERR REDEFINES SR-MENU  PIC XX.
        1Ø  SR-TRSMIT                       PIC X.
        1Ø  FILLER                          PIC X(4).
    .
    .
    .
Ø1  OUTPUT-MESSAGE-AREA.    COPY OMA.
    Ø5  OMA-TEXT                            PIC X(3ØØØ).
    .
    .
    .
PROCEDURE DIVISION          USING PROGRAM-INFORMATION-BLOCK
                                  INPUT-MESSAGE-AREA
                                  WORK-AREA
                                  OUTPUT-MESSAGE-AREA
                                  CONTINUITY-DATA-AREA.
    .
    .
    .
255-VALIDATE-MENU-SEL.
    IF  SR-MENU < 1 OR > 9
        MOVE HIGH-VALUES TO SR-MENU-ERR
        PERFORM 5Ø6-REBUILD
    ELSE
        PERFORM SET-MENU.
    .
    .
    .
```

Figure 7-9. Building an Error Screen in a COBOL Action Program (Part 1 of 2)

```
506-REBUILD.
    MOVE 100 TO TEXT-LENGTH.
    CALL 'REBUILD'          USING OUTPUT-MESSAGE-AREA
                                  IMA-SCREEN-REC.
    IF STATUS-CODE IS GREATER THAN 0
       MOVE '3' TO ERR-FLAG.
507-RETURN.
    CALL 'RETURN'.
```

Figure 7-9. Building an Error Screen in a COBOL Action Program (Part 2 of 2)

```
1        10     16

*  VALIDATE MENU SELECTION
         CLI    SRMENU,X'F1'
         BL     REBLD
         CLI    SRMENU,X'F9'
         BH     REBLD
         .
         .
         .
*  BUILD ERROR SCREEN
REBLD    MVC    ZA#OTL,MSGSIZE        SET TEXT-LENGTH FIELD
         ZG#CALL REBUILD,(OMAREA,IMAREC)
         CLI    ZA#PSC+1,X'00'        ERROR CHECKING
         BNE    BLDERR
         B      TERM
BLDERR
         .
         .
         .
TERM     ZG#CALL RETURN
*
*  CONSTANTS
MSGSIZE  DC     H'100'
         .
         .
         .
*  ACTIVATION RECORD DEFINITION
         .
         .
         .
         ZM#DIMH
```

Figure 7-10. Building an Error Screen in a BAL Action Program (Part 1 of 2)

```
1        10    16

IMAREC   EQU   *
SRCUST   DS    CL6                      INPUT MESSAGE FIELDS
SRMENU   DS    CL2
SRXMIT   DS    CL5
OMAREA   ZM#DOMH
OMATEXT  DS    CL3ØØØ
```

Figure 7-10. Building an Error Screen in a BAL Action Program (Part 2 of 2)

Coding to display a replenish screen

To build a replenish screen, you need only move a value to the TEXT-LENGTH field (or move C'D' to SFS-LOCATION to build the screen in dynamic main storage) and issue the REBUILD function call without the *variable-data* parameter:

```
MOVE 1ØØ TO TEXT-LENGTH.
CALL 'REBUILD' USING OUTPUT-MESSAGE-AREA.
```

Setting option indicators

To build an error or replenish screen using option indicators and the BUILD function, use the same coding used to display the screen format initially, except that you move C'1' to the appropriate option indicator bytes before issuing the BUILD function. (See 7.5.)

7.12. ERROR RETURNS FROM THE REBUILD FUNCTION

Unsuccessful REBUILD function

When the REBUILD function call is unsuccessful, no error format or replenish screen is constructed and IMS returns one of the following pairs of status and detailed status codes to the program information block:

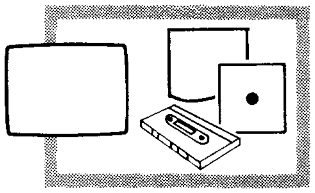
Status and detailed status codes

Status Code (Decimal)	Detailed Status Code (Decimal)	Explanation
1		Internal error
7	1	Buffer area not large enough; IMS places the actual length required for the format in the TEXT-LENGTH field.
7	5	Internal error
7	6	I/O error reading screen format file
7	7	REBUILD not allowed because screen format has no input fields
7	8	Invalid field in variable data area
7	9	Variable-data parameter specified but no error field detected
7	11	System error

See Appendix D for a complete listing of status and detailed status codes in hexadecimal.

7.13. DISPLAYING A SCREEN FORMAT ON AN AUXILIARY DEVICE

You can use the BUILD function call to output a screen format to an auxiliary device - printer, cassette, or diskette - attached to a display terminal.



Setting output message header fields

To output a screen format to an auxiliary device, you place values in the AUX-FUNCTION and AUX-DEVICE-NO fields in the output message header before issuing the BUILD function call. The AUX-FUNCTION setting tells IMS which print or transfer option to use, and the AUX-DEVICE-NO identifies the auxiliary device.

Print and transfer options

Table 7-1 lists the print and transfer options IMS supports for writing of screen formats and the settings for the AUX-FUNCTION field in continuous and noncontinuous output modes. For an explanation of the print and transfer options, see 6.19.

Restrictions

Because the terminal operator cannot enter input at an auxiliary device, the screen format must be output-only. For the same reason, you cannot use the REBUILD function call to write an error or replenish screen to an auxiliary device.

NOTE:

When you build a screen in dynamic main storage, all values, including auxiliary device numbers and functions, must be present in the output message header before you issue the CALL BUILD. If any header values (except SFS-options) are changed after the CALL BUILD, the new values are ignored.

Table 7-1. Print/Transfer Options for Writing Screen Formats to Auxiliary Devices

Input/Output Options			Contents of aux-function Field				Auxiliary Devices			
			Continuous Output		No Continuous Output		UTS 400		UNISCOPE 100/200	
Name	Suppression	Inhibit Space Suppression	Hex	Character	Hex	Character	Supported	Not Supported	Supported	Not Supported
Print Mode	X		F3	3	F0	0	X (recommended) ① ③		X (recommended) ①	
		X	F5	5	F2	2	X (recommended) ① ③			X (unpredictable output at screen and auxiliary device)
Print Transparent	X		F7	7	F4	4	X② ③		X②	
		X	F9	9	F6	6	X② ③			X (unpredictable output at screen and auxiliary device)
Print Form (ESC H)	X		C1	A	D1	J	X④			X⑥
		X	C6	F	D6	O	X④			X⑥
Transfer All (ESC G)	X		C2	B	D2	K	X (recommended) ⑤			X⑥
		X	C7	G	D7	P	X⑤			X⑥
Transfer Variable (ESC F)	X		C4	D	D4	M	X④			X⑥
		X	C8	H	D8	Q	X④			X⑥
Transfer Changed (ESC E)	X		C5	E	D5	N		X (field control characters not supported)		X⑥
		X	E8	Y	F8	8		X (field control characters not supported)		X⑥

LEGEND:

① Printer - same format as screen

② Printer - same information as screen; no carriage returns

③ Cassette/diskette - same format as screen; no field control characters

④ Cassette/diskette - same format as screen; only records unprotected fields

⑤ Cassette/diskette - same format as screen; records all fields and all field control characters

⑥ Cassette/diskette - not available

7.14. USING SCREEN FORMATS IN A DISTRIBUTED DATA PROCESSING ENVIRONMENT

Your action programs can call on screen format services in a distributed data processing environment using the IMS transaction facility. (See Section 9.)

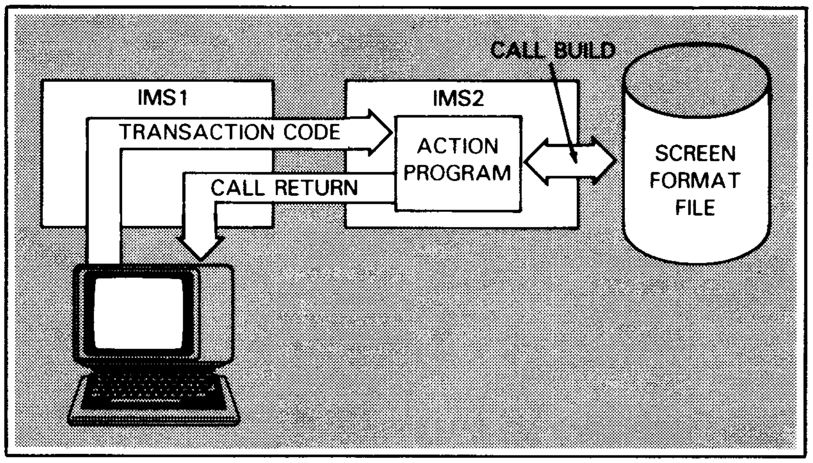
When your action program processes a transaction that is initiated by a terminal operator at a remote system . . .

You can

Displaying format at initiating terminal

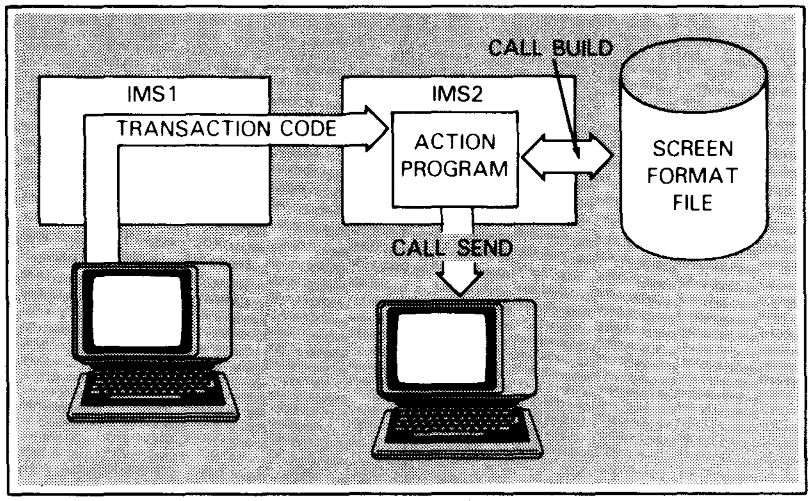
Restrictions

1. Issue a CALL BUILD followed by a CALL RETURN to display a screen format at the terminal that initiated the transaction at the remote system. You cannot output a screen format to an auxiliary device at the remote system (primary IMS) or to an action program initiating a remote transaction.



Displaying format at local terminal

2. Issue a CALL BUILD followed by a CALL SEND to display a screen format at a terminal (or auxiliary device) attached to your local IMS system. You cannot use a CALL SEND to display a screen format at the remote system (primary IMS).



Identifying remote system

When an action is initiated at a remote system, the SOURCE-TERMINAL-ID field (ZA#ISTID) of the input message area contains the locap-name of the remote system instead of a terminal identification. To display a screen at the source terminal, you can move the locap-name to the DESTINATION-TERMINAL-ID field (ZA#ODTID) of the output message area or leave binary zeros in this field.

Identifying local terminal

SEND function considerations

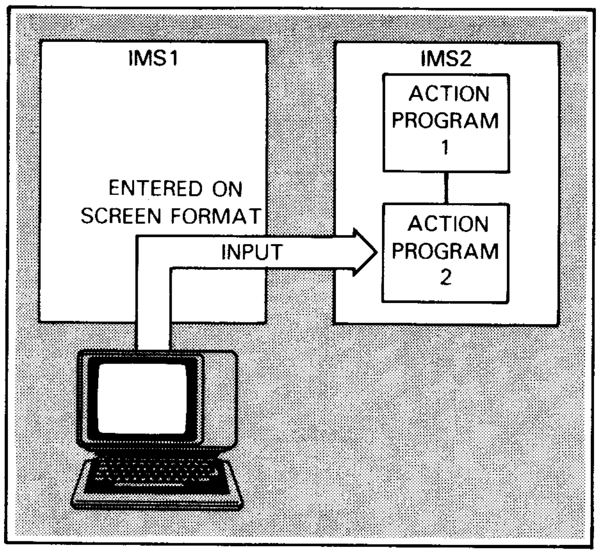
To display a screen at a terminal attached to your local IMS system, move the terminal-id to the DESTINATION-TERMINAL-ID field and issue a SEND function. Remember, you can display only an output format when you use the SEND function. Afterwards, clear the DESTINATION-TERMINAL-ID field or move the locap-name to that field before issuing a CALL RETURN to send an output message to the source terminal.

Termination types allowed

When you display an input/output screen format at the source terminal (at the remote system), you can terminate your program normally or with external succession. We recommend external succession.

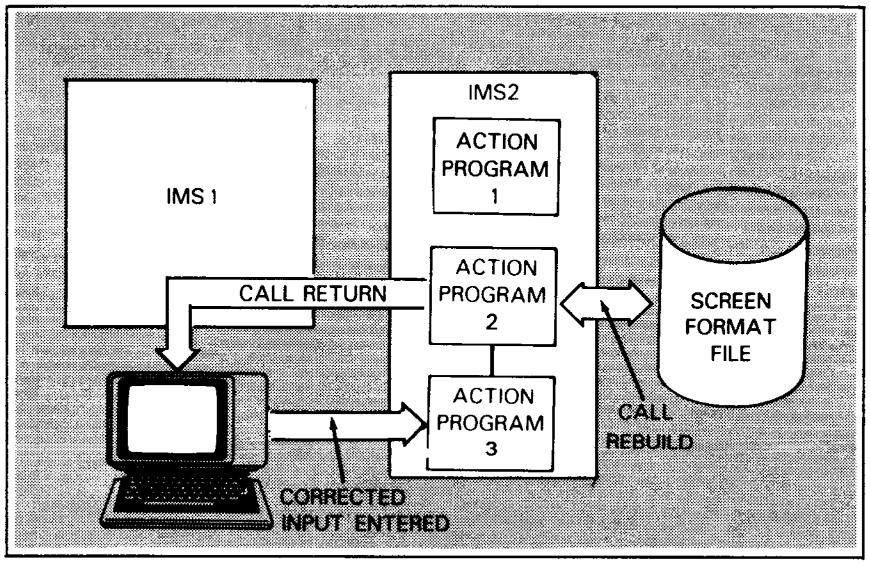
Receiving formatted input

When the terminal operator at the remote system enters input on the screen format, the successor program you name at your local IMS system (which could be the same action program) takes control and receives the input.



Displaying error or replenish screen

The successor action program can issue a CALL REBUILD, followed by a CALL RETURN, to build an error or replenish screen at the source terminal. Again, you can move the locap-name from the SOURCE-TERMINAL-ID field to the DESTINATION-TERMINAL-ID field or leave binary zeros in that field. This action program should also terminate with external succession and name a successor program to process the corrected input.



8. Calling Subprograms from Action Programs

8.1. WHEN TO USE SUBPROGRAMS

Subprograms must reside in main storage

You can call subprograms from action programs to perform common functions or repetitive computations. Subprograms must reside in main storage to be called by an action program. This guarantees their efficient use by not requiring that they be loaded into main storage each time they are called. They are loaded with IMS during start-up.

No SUBPROG call from subroutine

When a calling action program uses linked subroutines, only the main action program may issue a subprogram call.

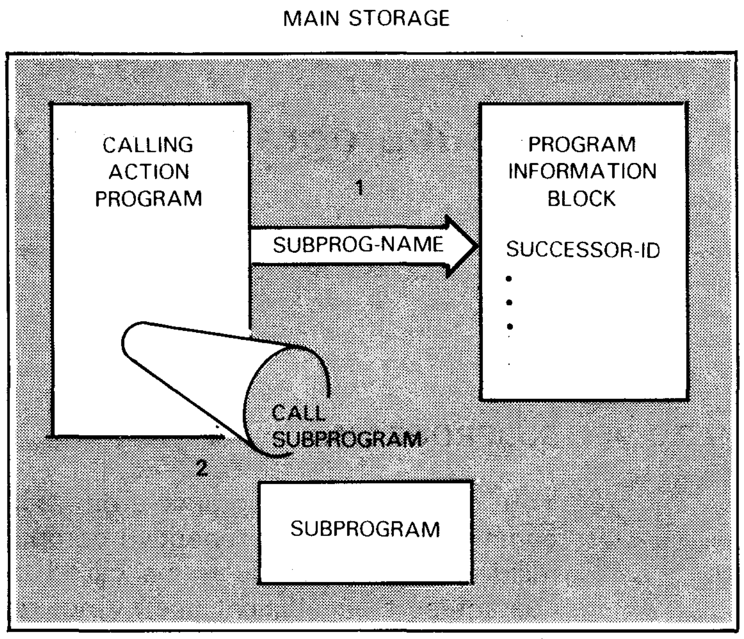
8.2. HOW TO USE SUBPROGRAMS

Configuration parameters

When you use subprograms, configure SUBPROG=YES in the OPTIONS section. Also, name the subprograms on the *program-name* parameter of the PROGRAM section and specify SUBPROG=YES in the same section.

Successor-id subprogram name

To use a subprogram, the calling action program must place the subprogram name in the SUCCESSOR-ID field of the program information block before calling the resident subprogram.



Serially reusable subprograms

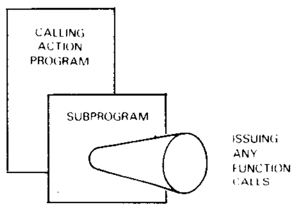
Subprograms may be coded as either serially reusable or reentrant modules. If a subprogram is accessed by one action program at a time during a transaction or is written in COBOL, make it serially reusable. The subprogram code can be modified but must be reset or restored before it is accessed again by another action program. A serially reusable subprogram can read and write into its own area nonreentrant calling action programs and the activation record.

Reentrant subprograms

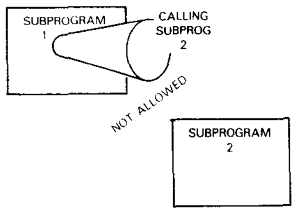
If several action programs access a subprogram concurrently, code the subprogram as a reentrant BAL module to increase throughput. Reentrant subprograms are executed as read-only. They may modify only the activation record and nonreentrant calling action programs.

Subprogram function calls

Subprograms can issue all the function calls that regular action programs use.



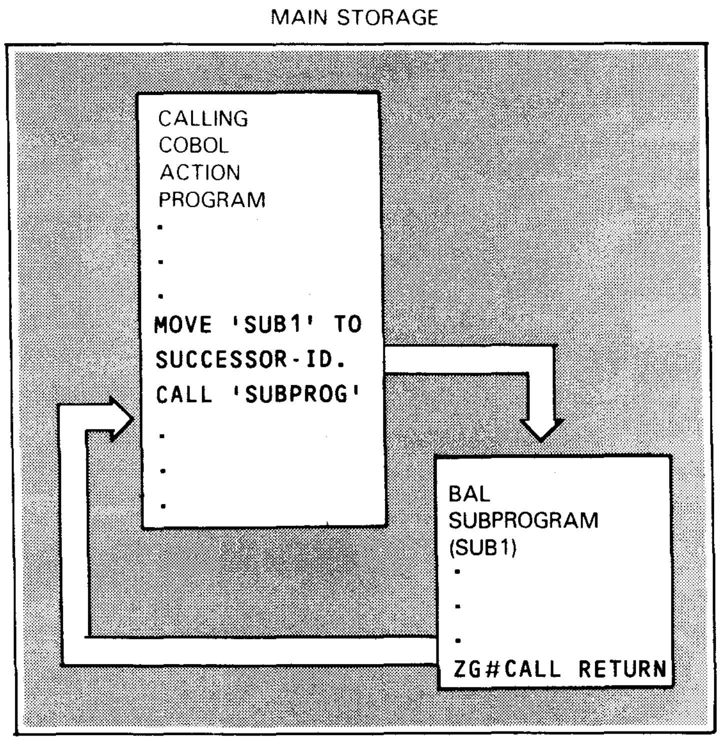
Subprograms may not call other subprograms.



Action program/subprogram interface A parameter list provides the means of transferring information from action program to subprogram.

Accessing files The called subprogram can access only those files allocated for the calling action program.

Calling and called program languages Your calling action program may be in COBOL while a subprogram may be in BAL, or both calling program and subprogram may be in the same language.



8.3. COBOL ACTION PROGRAM AND SUBPROGRAM INTERFACE

A COBOL action program calls a resident subprogram with the following sequence:

COBOL subprogram call format

```
MOVE subprogram-name TO SUCCESSOR-ID.
CALL 'SUBPROG' [USING data-name-1...data-name-n].
```

where:

`data-name-1...data-name-n`
Refer to data items in the data division of the calling COBOL action program. No more than 12 data-names can be specified.

COBOL return call format

A subprogram written in COBOL returns control to the calling action program as follows:

```
CALL 'RETURN'.
```

Saving status and detailed status codes from main program to subprogram

When the calling action program issues the SUBPROG CALL function, IMS clears the status and detailed status code fields in the program information block. Be sure to save status and detailed status codes in your calling program's work area before issuing a SUBPROG call. Otherwise, you lose the status of the latest function call issued.

When you issue the SUBPROG call, IMS transfers the contents of the calling program's work area to the subprogram's work area and your saved status codes are received in the subprogram's work area.

Saving status and detailed status codes from subprogram to main program

Also, depending on your application, when returning to the main program, you may want to save the latest status and detailed status codes from the subprogram. When the main program needs the status of the latest function call, you move these program information block values to the subprogram work area. When the CALL RETURN function executes, IMS returns these values to the main program work area. Otherwise, IMS clears the status and detailed status codes in the program information block and they are lost.

8.4. BAL ACTION PROGRAM AND SUBPROGRAM INTERFACE

A BAL action program calls a resident subprogram via the following macroinstruction:

BAL subprogram call format

```
{CALL   }SUBPROG,(param-1,...,param-n)
{ZG#CALL}
```

where:

param-1,...,param-n
: Refer to labels of storage locations in the BAL action program. Up to 12 parameters can be specified.

A subprogram written in BAL returns control to the calling action program via the following macroinstruction:

BAL return call format

```
{CALL   }RETURN
{ZG#CALL}
```

Setting successor-id location

Remember to place the name of the called subprogram in the program information block at location ZA#PSID before issuing the CALL function. The subprogram name must be left-justified and zero filled (X'F0') in a 6-byte area.

Parameter list location

Register contents

When the calling action program transfers control to the called subprogram, register 1 points to the specified parameter list. If the subprogram requires working storage, the calling program can pass the address of the working storage to the subprogram either in the parameter list or in a register. Other register contents are as follows:

REGISTERS		CONTENTS
Register 0	>	Unpredictable
Register 1	>	Parameter list address
Registers 2-12	>	Address of calling action program contents
Register 13	>	72-byte save area supplied by calling action program. Subprogram must save caller's registers using standard linkages.
Register 14	>	Return address
Register 15	>	Entry point address of subprogram

Saving status and detailed status codes

Because IMS clears the status and detailed status codes after the main program issues the SUBPROG call, your main program must save these codes before issuing the SUBPROG call. Depending on your application, saving these codes may also be necessary before issuing the CALL RETURN from the subprogram.

8.5. SUBPROGRAM SAMPLE APPLICATION

Application possibilities

Consider how often you test the performance of an I/O function call for various error conditions and consequently issue an error message to the terminal. After each function call you check status. All of the error conditions and error messages could be coded in a subprogram so that each time the calling action program issues a function call, it could call the subprogram to test the status of that function call and move the appropriate error message into an area of the calling action program. After returning to the calling program, that program could issue the error message to the terminal.

In this case, you can handle all the error testing and error message processing in your subprogram instead of duplicating the code in several action programs. Other routines suited to subprograms might be a frequently calculated inventory or payment total or cursor positioning used often in generating output messages to the terminal.

Sample subprogram application

Probably the most common subprogram call application is to a COBOL subprogram. Figure 8-1 is an example of a COBOL action program (GRP4D) that calls the COBOL subprogram (NUMPRG) to determine the status of function calls issued by GRP4D. Figure 8-2 shows the subprogram, NUMPRG.

Explanation of sample

In Figure 8-1, the calling program (GRP4D) retrieves the customer record of the customer named at the terminal. This customer record is on the file, TEST4, identified on line 9.

Once GRP4D retrieves the customer record (I-REC), it tests the status code for the GET function call. If the GET is successful (line 56), GRP4D processes a customer record (lines 72-82) sending it to the source terminal upon normal termination (lines 83-84).

Explanation of sample

If the GET is unsuccessful, GPR4D saves the status and detailed status codes and moves the suprogram name, NUMPRG, to the SUCCESSOR-ID field in the program information block (line 59) and calls the subprogram (line 60). Notice particularly that the USING clause in the procedure division of the subprogram (line 15) must match the USING clause on the CALL 'SUBPROG' statement in the calling program (line 60). This establishes the parameter list.

NUMPRG (Figure 8–2) tests status codes, moves the appropriate error messages to the work area (lines 9–14, Figure 8–2), and returns to GRP4D (line 26, Figure 8–2). Following the SUBPROG call, GRP4D receives the error message returned by NUMPRG, moves it to the output message area (lines 41–52, Figure 8–1), and issues the output message to the terminal (lines 61–70, Figure 8–1). GPR4D terminates normally with the CALL 'RETURN' (line 84, Figure 8–1).

When the status code being tested in NUMPRG is satisfied, NUMPRG returns to GRP4D. GRP4D processes the error message by sending it to the source terminal on normal termination.

Note that the activation record areas described in the subprogram linkage section must correspond in size and layout to their like areas in the main program. (See Figure 8–1, lines 18–26, and Figure 8–2, lines 9–14.)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. GRP4D.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. UNIVAC-OS3.
OBJECT-COMPUTER. UNIVAC-OS3.
DATA DIVISION.
WORKING-STORAGE SECTION.
77  TEST4     PIC X(7) VALUE 'TEST4  '.
77  DICE1     PIC X(4) VALUE ='1003050A'.
77  DICE2     PIC X(4) VALUE ='10060200'.
77  DICE3     PIC X(4) VALUE ='10060003'.
LINKAGE SECTION.
01  PIB. COPY PIB74.
01  IMA. COPY IMA74.
    02 FILLER    PIC X(11).
    02 PHONE-IN    PIC 999.
01  WORK-AREA.
    02 I-REC.
        03  PHONE-O      PIC 999.
        03  NAME-O    PIC X(15).
        03  ADDRESS-O    PIC X(6).
    02  ERR-DATA.
        03  MSG     PIC X(14).
        03  S-CODE      PIC 9999.
        03  D-CODE      PIC 9999.
01  OMA. COPY OMA74.
    02  DATA-LINE.
        03  DICE-1  PIC X(4).
        03  MSG1    PIC X(4).
        03  DICE-3  PIC X(4).
        03  NAMEO   PIC X(15).
        03  DICE-2  PIC X(4).
        03  MSG2    PIC X(7).
        03  DICE-4  PIC X(4).
        03  ADDRESSO   PIC X(6).
        03  DICE-5  PIC X(4).
        03  MSG3    PIC X(3).
        03  DICE-6  PIC X(4).
        03  PHONEO     PIC 999.
    02  ERR-MSG-LINE REDEFINES DATA-LINE.
        03  DICE-7  PIC X(4).
        03  MSGO    PIC X(14).
        03  DICE-8  PIC X(4).
        03  MSG4    PIC X(11).
        03  DICE-9  PIC X(4).
        03  CODE10  PIC 9999.
        03  DICE-10    PIC X(4).
        03  MSG5       PIC X(8).
        03  DICE-11    PIC X(4).
```

Figure 8–1. Sample Action Program (GRP4D) Calling Subprogram (NUMPRG) (Part 1 of 2)

```
                03  CODE20  PIC 9999.
                03  FILLER  PIC X.
       PROCEDURE DIVISION USING PIB IMA WORK-AREA OMA.
       BEGIN.
           CALL 'GET' USING TEST4 I-REC PHONE-IN.
           IF STATUS-CODE EQUAL ZERO GO TO PROCESS-MSG.
           MOVE STATUS-CODE TO S-CODE.
           MOVE DETAILED-STATUS-CODE TO D-CODE.
           MOVE 'NUMPRG' TO SUCCESSOR-ID.
           CALL 'SUBPROG' USING WORK-AREA.
       PROCESS-ERROR.
           MOVE 80 TO TEXT-LENGTH OF OMA.
           MOVE DICE1 TO DICE-7.
           MOVE DICE2 TO DICE-8, DICE-10.
           MOVE DICE3 TO DICE-9, DICE-11.
           MOVE 'STATUS CODE' TO MSG4.
           MOVE 'DETAILED' TO MSG5.
           MOVE S-CODE TO CODE10.
           MOVE D-CODE TO CODE20.
           MOVE MSG TO MSGO.
           GO TO E-O-J.
       PROCESS-MSG.
           MOVE 80 TO TEXT-LENGTH OF OMA.
           MOVE DICE1 TO DICE-1.
           MOVE DICE3 TO DICE-3, DICE-4, DICE-6.
           MOVE DICE2 TO DICE-2, DICE-5.
           MOVE 'NAME' TO MSG1.
           MOVE 'ADDRESS' TO MSG2.
           MOVE 'KEY' TO MSG3.
           MOVE NAME-O TO NAMEO.
           MOVE ADDRESS-O TO ADDRESSO.
           MOVE PHONE-O TO PHONEO.
       E-O-J.
           CALL 'RETURN'.
```

Figure 8-1. Sample Action Program (GRP4D) Calling Subprogram (NUMPRG) (Part 2 of 2)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. NUMPRG.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. UNIVAC-OS3.
OBJECT-COMPUTER. UNIVAC-OS3.
DATA DIVISION.
LINKAGE SECTION.
01  WORK-AREA.
    02  FILLER  PIC X(24).
    02  ERR-DATA.
        03 MSG    PIC X(14).
        03  S-CODE   PIC 9999.
        03  D-CODE   PIC 9999.
PROCEDURE DIVISION USING WORK-AREA.
BEGIN.
    IF S-CODE EQUAL 1
        MOVE 'INVALID    KEY' TO MSG ELSE
    IF S-CODE EQUAL 2
        MOVE 'UNALLOCATED FI' TO MSG ELSE
    IF S-CODE EQUAL 3
        MOVE 'INVALID    REQ' TO MSG ELSE
    IF S-CODE EQUAL 4
        MOVE 'I/O     ERRCR' TO MSG ELSE
    MOVE 'PROBLEM IN SUB' TO MSG.
    CALL 'RETURN'.
```

Figure 8-2. Sample Subprogram (NUMPRG)

9. Action Programming in a Distributed Data Processing Environment

9.1. BASIC DDP REQUIREMENTS AND TERMINOLOGY

Configuration and network definition requirements

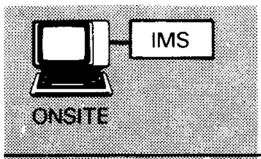
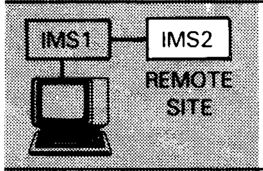
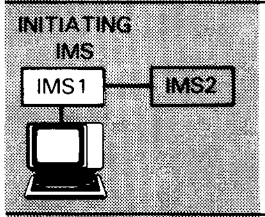
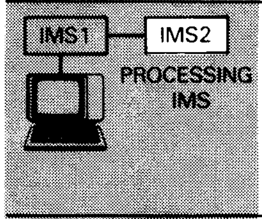
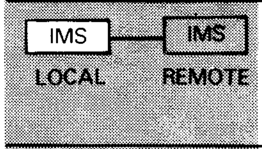
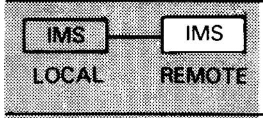
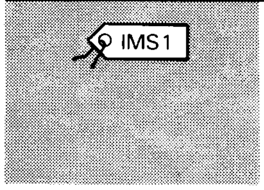
IMS handles distributed data processing (DDP) transactions through the IMS transaction facility. To use distributed data processing with IMS, you must include the IMS transaction facility in your software at each OS/3 system and must configure multithread IMS at each system. Also, you must define a global ICAM network that supports distributed data processing and include a LOCAP section in the IMS configuration for each IMS system where you want to route transactions or which will route transactions to you. Consult the IMS system support functions user guide UP-8364 (current version) for configuration and network definition requirements.

DDP terminology

Let's define some terms we'll be using throughout the discussion of DDP transaction processing:

DDP REQUIREMENTS AND TERMS

DDP terminology

	LOCAL TRANSACTION Transaction that is processed at the same IMS system where it is initiated
	REMOTE TRANSACTION Transaction that is initiated at one IMS system and processed at another
	PRIMARY IMS IMS system where a remote transaction is initiated. In our illustrations we call this system IMS1.
	SECONDARY IMS IMS system where a remote transaction is processed. The action programs processing the transaction and any files they access are located here. In our illustrations we call this system IMS2.
	LOCAL IMS Your IMS system, regardless of whether your system is primary or secondary for a particular transaction
	REMOTE IMS IMS system at another computer
	LOCAP-NAME The 4-character label of a LOCAP macroinstruction in your ICAM network definition, identifying a local or remote IMS system

9.2. HOW IMS ROUTES REMOTE TRANSACTIONS

Transaction routing types

There are three different ways in which the primary IMS can route a transaction to a secondary system:

Routing a Transaction To Secondary System

1. Directory routing

The terminal operator enters a transaction code that identifies a transaction at a secondary system. The transaction code is defined in the configurator TRANSACT section.

2. Operator routing

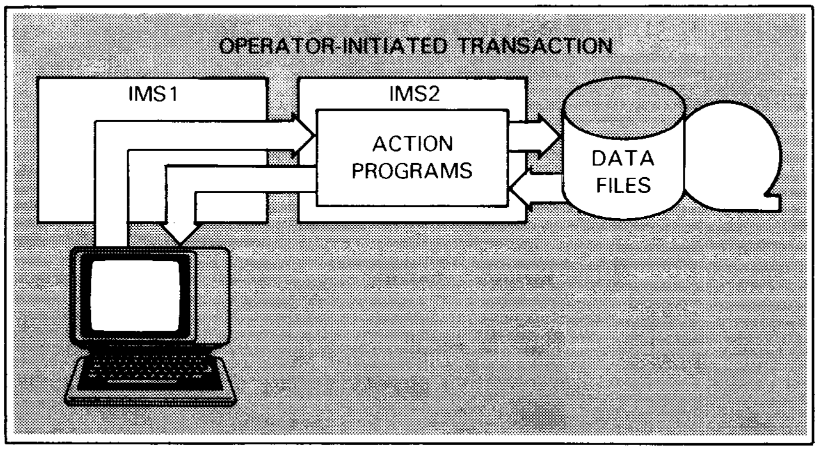
The terminal operator prefixes the transaction code with a route character (followed by a period) that routes the transaction to a secondary system. This route character is defined in the configurator LOCAP section or in a PARAM job control statement at IMS start-up.

3. Action program routing

The terminal operator enters a transaction code that initiates a transaction at the primary system. The action program processing this local transaction issues an ACTIVATE function call to initiate a transaction at a secondary system.

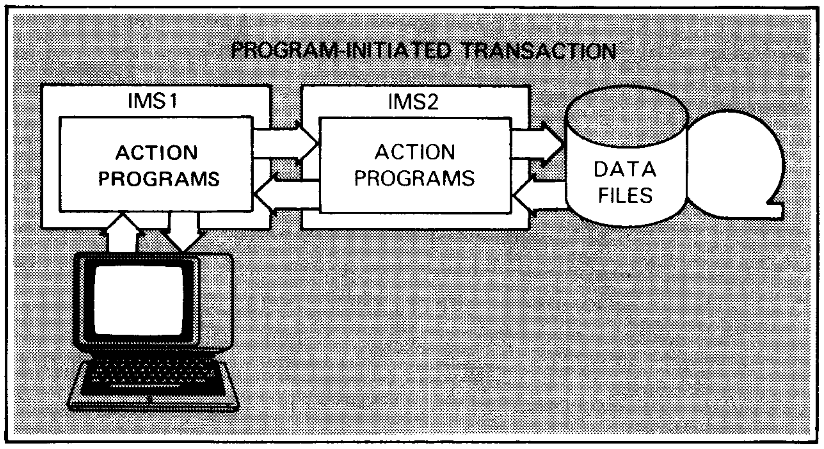
Operator-initiated transactions

From the programmer's viewpoint, directory and operator routing are the same, because they are both initiated by a terminal operator. Once the transaction is routed to the secondary system, an action program or series of action programs at that system interacts with the terminal operator the same way as in a local transaction. No action programs are involved at the primary system.



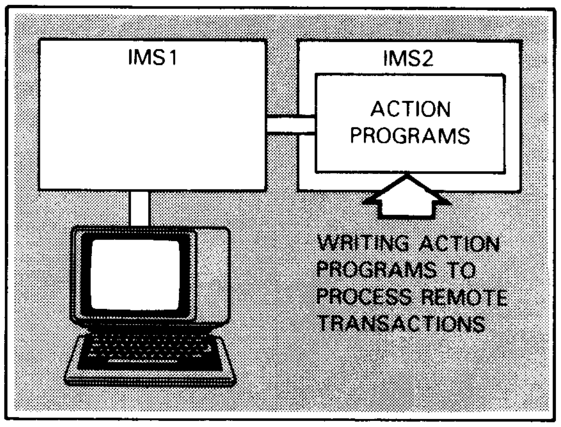
Program-routed transactions

With action program routing, action programs at the secondary system do not interact directly with the terminal operator. They return a message to the initiating action program or its successor, which in turn outputs a message to the terminal operator. As a programmer, you may be writing action programs at either the primary or secondary system.



9.3. PROCESSING A REMOTE TRANSACTION

First, we'll assume that you are at a secondary IMS, writing action programs to process transactions initiated by an operator or an action program at a primary IMS system.



Similar to processing local transaction

There is little difference between the way you process a remote transaction and the way you process a local transaction. You can use the same action programs to process both local and remote transactions.

Receiving input message

When the transaction begins, you receive an input message starting with a 1- to 8-character transaction code, just as with a local transaction.

Determining input message source

You can determine the source of the input message by testing the DDP-MODE field (ZA#DDPMD) of the program information block and the SOURCE-TERMINAL-ID field (ZA#ISTID) of the input message header.

DDP-MODE field

The DDP-MODE field contains the value 'R' (ZA#DTR) when the transaction is operator-initiated (either directory routing or operator routing). It contains the value 'A' (ZA#PTRA) when the transaction is initiated by an action program. When a transaction is local, the DDP-MODE field contains zeros (X'00'). This field has other possible values, but they apply to action programs at the primary IMS system (see 9.8).

SOURCE-TERMINAL-ID field

When an action is scheduled to process a transaction at a secondary IMS, the SOURCE-TERMINAL-ID field contains the locap-name of the IMS system originating the transaction rather than a terminal-id. You cannot test for the actual terminal initiating a remote transaction.

General restrictions

There are a few general restrictions on processing remote transactions. (There are several additional restrictions for program-initiated remote transactions, which we'll discuss a little later in this section.)

SEND function restriction

1. You cannot use the SEND function to output a message to the originating terminal (or any terminal at the remote IMS). However, you can use the SEND function to output a message to a terminal at your local IMS. Afterwards, clear the DESTINATION-TERMINAL-ID field (ZA#OTID) or move the source locap-name to that field before issuing a CALL RETURN to send an output message to the originating terminal.

Continuous output restriction

2. You cannot send continuous output to the originating terminal. Again, you can use the SEND function to initiate continuous output at a local terminal using output-for-input queueing.

Auxiliary device restriction

3. You cannot send output to an auxiliary device attached to the originating terminal. However, you can output to local auxiliary devices using the SEND function.

9.4. PROCESSING AN OPERATOR-INITIATED REMOTE TRANSACTION

With the few exceptions we've already mentioned, you process an operator-initiated remote transaction the same way as a local transaction.

Action program succession

You can use any type of action program succession with operator-initiated transactions. Once the transaction begins, the IMS transaction facility establishes a communications link which stays in effect until the transaction ends. When you use external succession, the terminal operator receives and responds to your output messages without entering any additional codes.

Figure 9-1 illustrates a remote dialog transaction, using both internal (either immediate or delayed) and external succession.

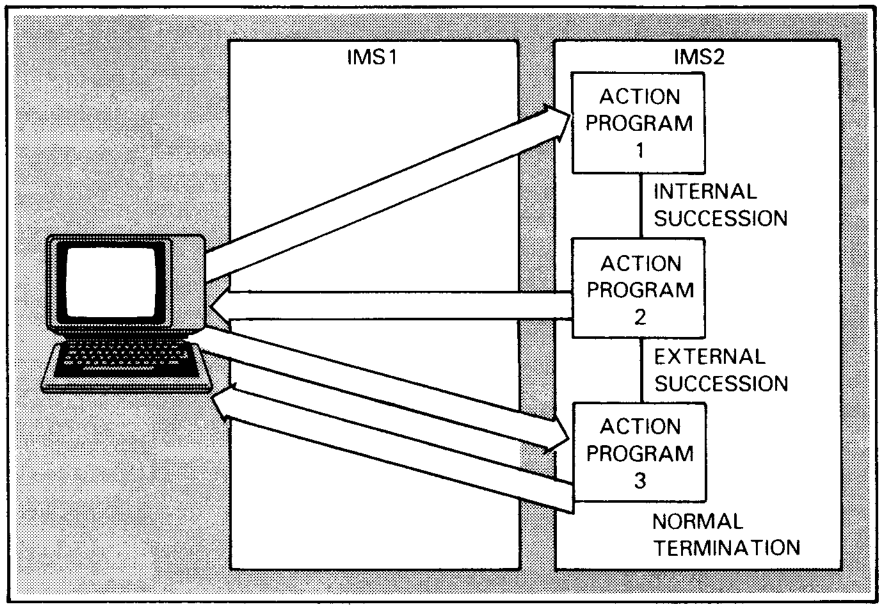


Figure 9–1. Processing an Operator-Initiated Remote Dialog Transaction

Screen format services in DDP

You can use screen format services with operator-initiated remote transactions. See 7.14 for details.

9.5. PROCESSING A PROGRAM-INITIATED REMOTE TRANSACTION

When a remote transaction is initiated by an action program, you send an output message back to the originating action program's successor. That action program in turn outputs a message to the terminal operator.

Considerations and restrictions

Because your output message goes to an action program rather than to a terminal, there are a few additional considerations and restrictions:

Output message formatting

1. You may want to format the output message differently; you do not need control characters. Of course, you may want to use the same output message for either operator- or program-initiated transactions. In this case, the action program receiving your message must be prepared to receive your control characters.

Screen formatting restriction

2. You cannot use a screen format for the output message you return to the originating action program or its successor (see 7.14). However, you can use the SEND function to display a screen format at a local terminal.

Allowable termination types

3. You must use normal termination when you return an output message to the originating action program's successor. You cannot use external succession. You can, however, use immediate or delayed internal succession and have your successor program return the output message (Figure 9-2).

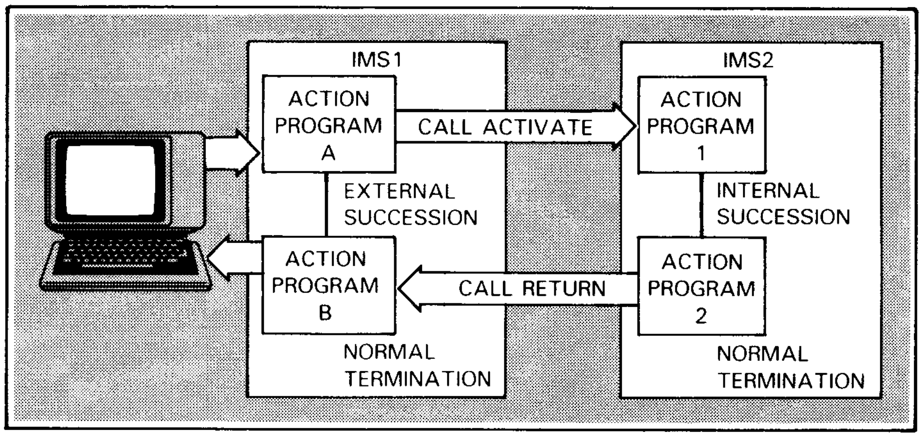


Figure 9-2. Processing a Program-Initiated Remote Transaction

Dialog with terminal operator

Although a program-initiated remote transaction always has just one input message and one response, a dialog with the terminal operator can still take place. The initiating series of action programs at the primary IMS can use external succession to output messages and receive responses from the terminal and can issue repeated ACTIVATE function calls to communicate with your action programs and access your files. Figure 9-3 shows how you might process successive program-initiated remote transactions while the initiating action programs carry on a dialog with the terminal operator.

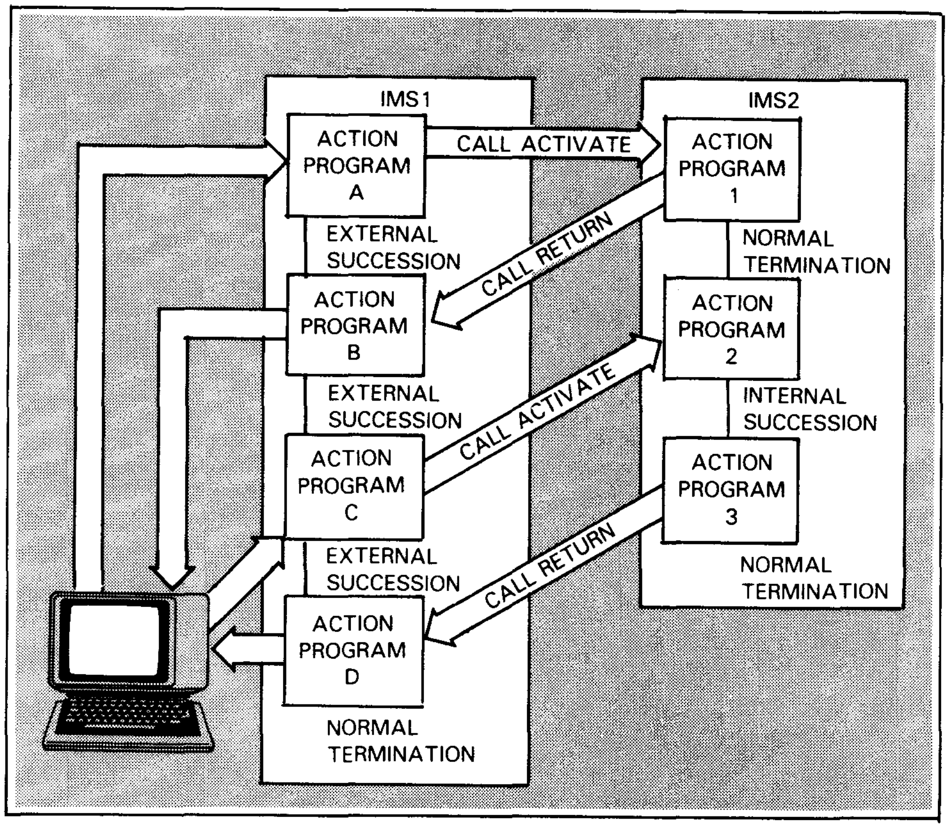
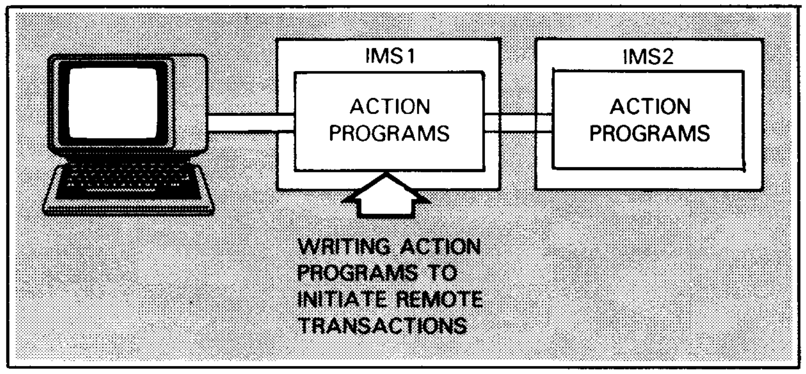


Figure 9–3. Processing Successive Program-Initiated Remote Transactions

9.6. ROUTING TRANSACTIONS TO A REMOTE IMS SYSTEM

Now, assume that you are at a primary IMS, writing action programs to initiate remote transactions and receive response messages from a remote system.



In a program-initiated remote transaction, you make the decision whether to route the transaction to a remote system on the basis of some data the terminal operator enters or perhaps something you discover when you access your files or make some computations.

Initiating remote transaction

External succession required

You initiate a remote transaction by identifying the remote IMS system (locap-name) in the output message header, building a message containing a transaction code in your output message area, and issuing an ACTIVATE function call. You must terminate your action program externally, naming a successor program at your local IMS system. Of course, you can reschedule the same action program as the successor.

Processing response message

Action programs at the remote IMS system process your message and send a response. Your successor program receives the response message in its input message area. You can then send an output message to the originating terminal. (See Figures 9-2 and 9-3.) If you wish, you can issue another ACTIVATE call instead of outputting a message to the terminal (Figure 9-4).

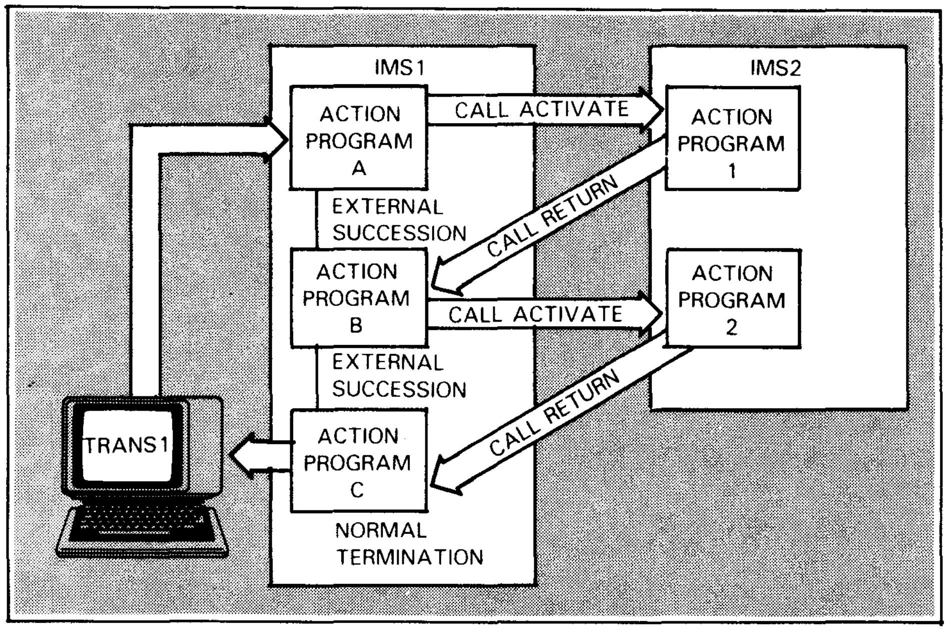


Figure 9-4. Issuing Muliple ACTIVATE Calls without Operator Intervention

9.7. INITIATING A REMOTE TRANSACTION (ACTIVATE)

The ACTIVATE function call initiates a remote transaction and terminates the action program. It has no parameters. The COBOL and BAL formats for the ACTIVATE function call follow.

- COBOL format

COBOL format

```
CALL 'ACTIVATE'
```

- BAL format

BAL format

```
{CALL    }ACTIVATE
{ZG#CALL }
```

Here is a step-by-step procedure for initiating a remote transaction:

Identifying remote system

1. **Identify** the remote IMS system where you want the transaction processed by placing its locap-name in the DESTINATION-TERMINAL-ID field (ZA#ODTID) of the output message header.

Building output message

2. **Build** the output message you want to send to the remote system in the output message area. The message must begin with a transaction code that is acceptable to the remote IMS system.

Setting text length

3. **Move** the message length to the TEXT-LENGTH field (ZA#OTL) of the output message header.

Naming external successor

4. **Specify** external termination and the name of a successor program at your IMS system. The successor program can be the same program.

Issuing ACTIVATE call

5. **Issue** the ACTIVATE function call.

RETURN function not used

You don't issue a RETURN function call when you initiate a remote transaction. The ACTIVATE function call terminates the action program and sends the output message to the remote system.

9.8. RECEIVING A RESPONSE MESSAGE IN THE SUCCESSOR ACTION PROGRAM

Successor program receives message

When remote transaction is successful

When remote transaction is unsuccessful

When an action program issues an ACTIVATE function call and terminates in external succession, its successor program receives a message in the input message area regardless of whether the remote transaction is successful. When the remote transaction is successful, the successor program receives a response from the action program processing the transaction at the secondary IMS. When the remote transaction is unsuccessful, the successor program receives error codes in the input message area.

DDP-MODE field

To determine whether the transaction was successful, test the DDP-MODE field (ZA#DDPMD) of the program information block. The DDP-MODE field contains the value 'E' (ZA#PTRE) when the remote transaction ends normally and returns a message to your program. It contains the value 'C' (ZA#PTRC) when the remote transaction is unsuccessful. This field has other possible values, but they apply to action programs processing a remote transaction at a secondary IMS system.

Processing successful response

When the remote transaction is successful (value 'E'), you can send a message to the originating terminal or issue another ACTIVATE call to initiate another remote transaction.

IMS sets the DDP-MODE field to 'C' and places an error code in the input message area when:

Error causes

- your output message cannot be sent to the remote IMS;
- your output message arrives at the remote IMS but the transaction cannot be scheduled;
- the remote transaction is scheduled but terminates abnormally; or
- the remote transaction terminates normally but your program does not receive the response message.

Processing unsuccessful response

You can continue processing your local transaction, perhaps issuing an error message to the source terminal.

Errors causing cancellation of initiating transaction

The only errors causing cancellation of the initiating transaction are succession errors. If an action program issuing a CALL ACTIVATE specifies an invalid termination indicator or successor id, IMS cancels the transaction and sends an error message to the source terminal. Also, if the terminal operator keys in the ZZCNC terminal command, the transaction is canceled.

9.9. ERROR RETURNS FROM UNSUCCESSFUL REMOTE TRANSACTION

IMS sets error code in input message area

When the remote transaction is unsuccessful, IMS places the value 'C' in the DDP-MODE field and also sets an error code in the input message area. The error code consists of 2-byte class code and a 2-byte reason code. When the class code is 0081, an error message follows the error code.

The format of the input message area when IMS returns an error is:

Input message area error format

Input Message Header	Error Class Code	Error Reason Code	Message-Text (Optional)
16 bytes	2 bytes	2 bytes	Variable

Table 9-1 describes the error codes and their meanings.

Table 9-1. Errors Returned to Input Message Area when Remote Transaction Is Unsuccessful (Part 1 of 2)

Error codes

Class Code (Hexadecimal)	Reason Code (Hexadecimal)	Explanation
0003	000C	Distributed data processing not configured
0006	0004	Destination locap-name invalid or auxiliary function specified
0006	0005	No ICAM buffer available for switched message
0006	0006	Disk error on switched message
0006	0007	Invalid length specification for switched message
0006	0009	CALL ACTIVATE requested by action program at remote IMS
000A	0001	Invalid function code. Submit software user report (SUR).
000A	0002	Invalid name. Submit SUR.
000A	0003	Buffer not available. Retry.
000A	0004	Invalid data type. Submit SUR.
000A	0005	Invalid data length. Submit SUR.
0080	0100	Required header item missing. Submit SUR.
0080	0700	Message sequence error. Submit SUR.
0080	0800	Invalid mode of operation. Submit SUR.

Table 9–1. Errors Returned to Input Message Area when Remote Transaction Is Unsuccessful (Part 2 of 2)

Class Code (Hexadecimal)	Reason Code (Hexadecimal)	Explanation
0080	0A00	Protocol procedure error. Submit SUR.
0080	0B00	Invalid header item. Submit SUR.
0080	0C00	Version not supported. Submit SUR.
0080	0D00	Class of procedure not supported. Submit SUR.
0081	0000	Action program or IMS error at remote system. Message text indicates specific error.
008C	0001	Error in transaction presentation control header. Submit SUR.
0400	0001	Invalid transaction code specified
0400	0002	Shutdown in process at remote IMS
1000	0100	No sessions available. Increase DDPSESS specification.
1100	1800	No ICAM buffer available. Increase buffers in ICAM network definition.
1100	1900	No session established. Submit software user report (SUR).
1200	9900	Invalid request. Submit SUR.
1400	0000	Remote system shut down. Could be normal or error condition.

NOTE:

If TRANSLAT=YES is configured for the action receiving the input message, class and reason codes containing the values 81-89, 91-99, and A2-A9 are translated to the values C1-C9, D1-D9, and E2-E9.

Abnormal termination of remote transaction

Three-line termination message

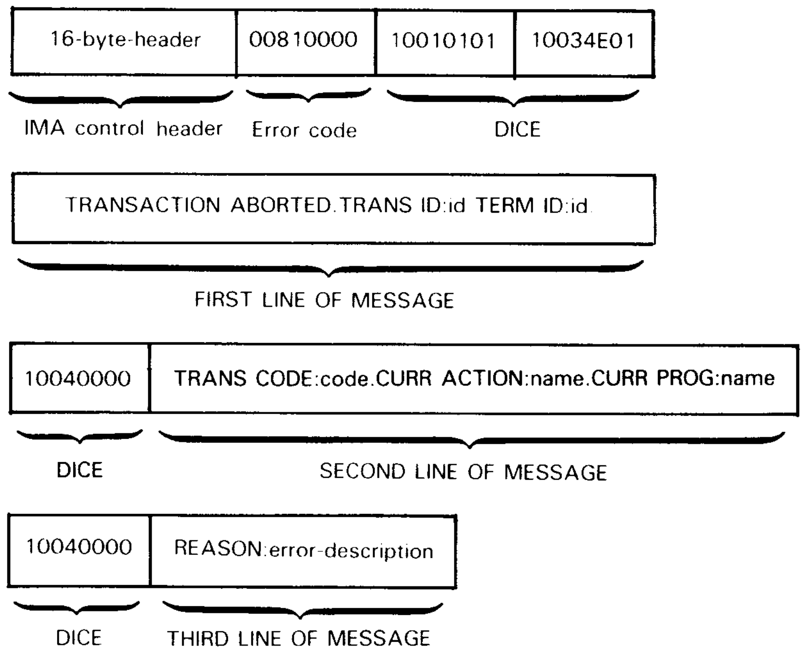
The class code 0081 indicates that the remote transaction abnormally terminated because of an IMS or action program error. This class code is always followed by a reason code of 0000 and a message text. The message text is one of the 3-line multithread IMS transaction termination messages documented in the system messages programmer/operator reference, UP-8076 (current version).

Message formatted for output to terminal

The 3-line transaction termination message is formatted for output to the source terminal. You can move this message to your output message area and send it to the source terminal without additional formatting.

An example of the input message area contents when IMS returns an error code of 0081 is:

Input message area contents

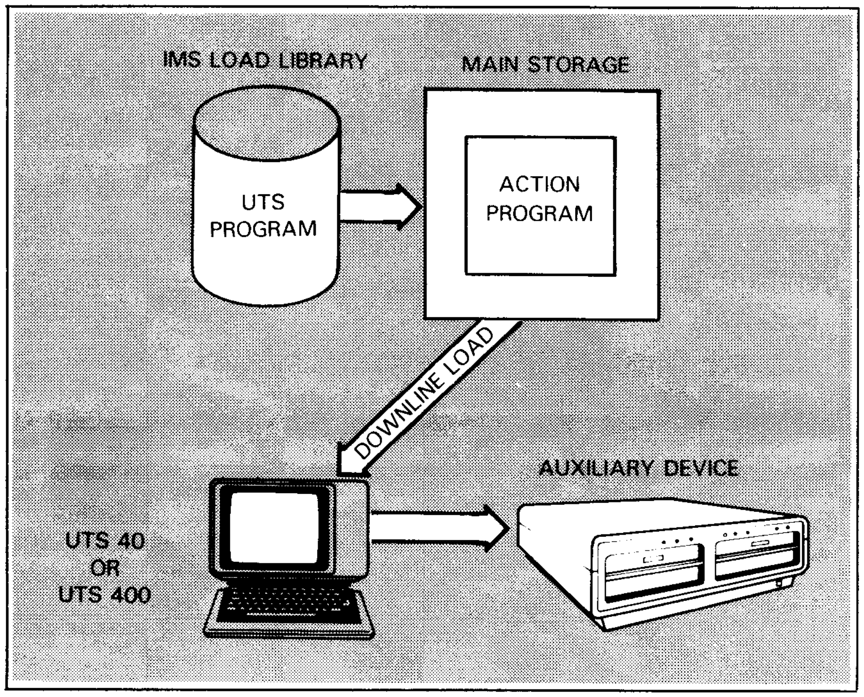


10. Additional Special Features

10.1. DOWNLINE LOAD FEATURE

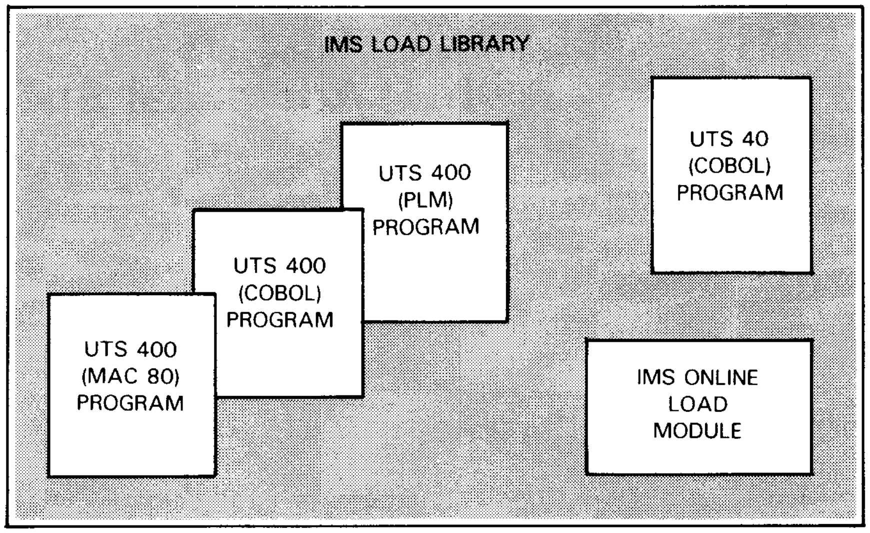
UTS 40/UTS 400 programs

Downline load action programs load COBOL, MAC 80, or PLM programs into the storage area of a Universal Terminal System 400 (UTS 400) or COBOL programs into the storage area of a UTS 40 for immediate execution. They can also load these UTS programs to auxiliary storage devices (diskette or cassette) attached to the UTS 40 and UTS 400.



Store UTS programs in load library

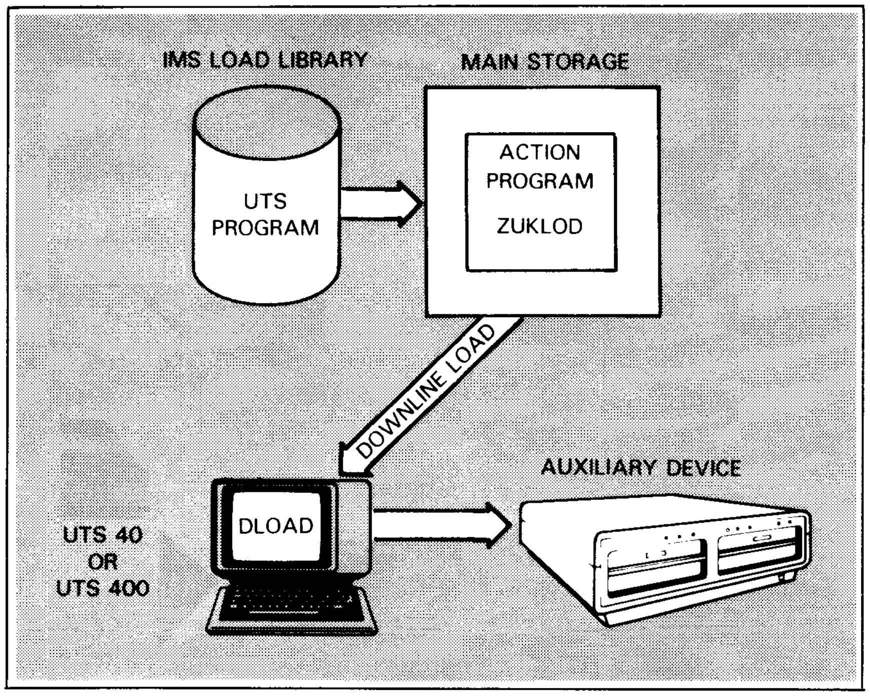
These UTS programs must be stored in the IMS load library – the same load library that contains your online IMS load module and action programs. If you configure the fast load feature, do not store UTS programs in the action program load library. Store them in the library containing the IMS load module or in the system load library, $Y$LOD.



There are two ways of downline loading:

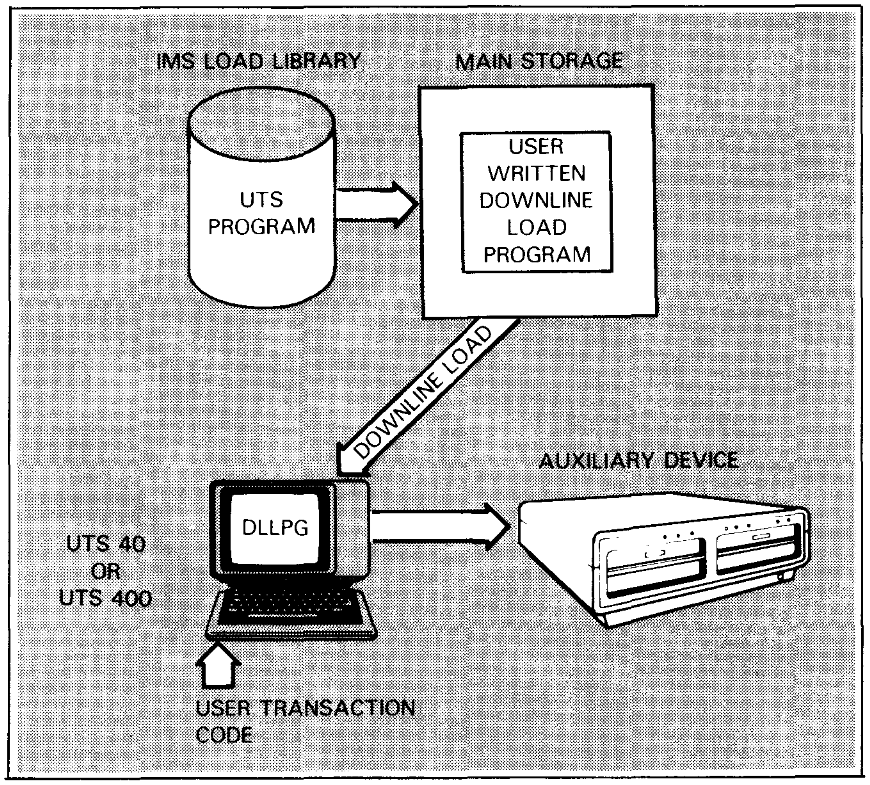
DLOAD downline load

1. **Enter** the transaction code, DLOAD, to activate the IMS downline load action program, ZUKLOD.



Action program downline load

2. Write your own downline load action program.



DLOAD details

For details of the DLOAD transaction code, see the IMS terminal user guide, UP-9208 (current version).

Downline load applications

Downline loading programs can be useful in numerous applications. One use is for editing and validating IMS input messages. If errors occur in input editing and validation, you can handle them directly at the UTS terminal without transmitting the message to the host computer.

Downline load environments

To use the downline loading feature, generate a resident ICAM that supports unsolicited output and specify DLLOAD=YES in the OPTIONS section of the configurator input.

The UTS terminal accepting a downline load must be a master or primary station and not a slave station.

Other UTS information

Before using the downline loading feature, you should be familiar with the UTS 40 or UTS 400 terminal description found in the ICAM concepts and facilities, UP-8194 (current version), the Universal Terminal System 400 programmer reference, UP-8359 (current version), and the Universal Terminal System 40 COBOL programmer reference, UP-8481 (current version).

10.2. WRITING DOWNLINE LOAD ACTION PROGRAMS

Load to UTS main storage or auxiliary storage

Suppose you decide not to call the ZUKLOD action program via the DLOAD transaction code to downline load UTS programs. You can write your own downline load action program to read blocks of UTS program code from the IMS load library to a UTS terminal or auxiliary device. Figure 10–1 is a sketch of a downline load action program that loads a UTS program, stored in the IMS load library, downline to a UTS 400 main storage.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LODPRG.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. UNIVAC-0S3.
OBJECT-COMPUTER. UNIVAC-0S3.
DATA DIVISION.
WORKING-STORAGE SECTION.
77    LOD-MOD-NAME          PIC X(8)  VALUE   'MACPROG1'.
77    BUF-SIZE              PIC 9999  USAGE COMP  VALUE 1000.
LINKAGE SECTION.
01    PROGRAM-INFORMATION-BLOCK. COPY PIB74.
01    INPUT-MESSAGE-AREA. COPY IMA74.
      02   UTS400-RESPONSE-MESSAGE.
           03   UTS400-RESPONSE-DICE       PIC X(4).
           03   UTS400-RESPONSE            PIC X(4).
      02   DEL-NOTICE-MSG REDEFINES UTS400-RESPONSE-MESSAGE.
           03   CONT-CODE                   PIC X(4).
           03   DEL-NOT-CODE                PIC X.
           03   FILLER                      PIC XXX.
      02   TRANS-CODE-ENTRY  REDEFINES UTS400-RESPONSE-MESSAGE.
           03   TR-CODE                     PIC X(5).
           03   FILLER                      PIC XXX.
01    OUTPUT-MESSAGE-AREA. COPY OMA74.
      02   DOWNLINE-LOAD-MESSAGE.
           03   DOWNLINE-LOAD-HEADER        PIC X(6).
           03   DOWNLINE-LOAD-TEXT          PIC X(1000).
01    CONTINUITY-DATA-AREA.
      02   GET-SET-AREA                     PIC X(400)  SYNC.
PROCEDURE DIVISION  USING  PROGRAM-INFORMATION-BLOCK
                           INPUT-MESSAGE-AREA
                           OUTPUT-MESSAGE-AREA
                           CONTINUITY-DATA-AREA.
START-PROG.
      IF TRANS-CODE = 'DLLPG' GO TO SET-PARA
      ELSE
```

Figure 10–1. User-written Downline Load Action Program Sketch (Part 1 of 3)

```
            IF CONT-CODE = 'CONT' GO TO TEST-DEL-NOTICE
            ELSE
                    GO TO LOAD-STATUS-CHECK.
SET-PARA.
      CALL 'SETLOAD' USING LOD-MOD-NAME  GET-SET-AREA.
                (Status code tests)
GET-PROG-CODE.
      CALL 'GETLOAD' USING GET-SET-AREA  DOWNLINE-LOAD-TEXT  BUF-SIZE.
      IF STATUS-CODE > 0  GO TO STAT-TEST
      ELSE MOVE 'C' TO AUX-FUNCTION
           MOVE 'CONT' TO CONTINUOUS-OUTPUT-CODE
      GO TO EXTERNAL-TERMINATION.
STAT-TEST.
      IF STATUS-CODE = 2 GO TO EXTERNAL-TERM
      ELSE
            IF STATUS-CODE = 3 AND DETAILED-STATUS-CODE = 20
                      GO TO INVAL-REQ
     ELSE
          IF STATUS-CODE = 3 AND DETAILED-STATUS-CODE = 21
                 GO TO SMALL-DATA-BUF
          ELSE
                 IF STATUS-CODE = 4 GO TO I/O-ERR.
EXTERNAL-TERM.
      MOVE '1B0E30323130' TO DOWNLINE-LOAD-HEADER.
      MOVE 'E' TO TERMINATION-INDICATOR.
      MOVE 'LODPRG' TO SUCCESSOR-ID.
      CALL 'RETURN'.
AB-TERM.
      MOVE 'S' TO TERMINATION-INDICATOR.
      CALL 'RETURN'.
NORM-TERM.
      (Send message to terminal)
      CALL 'RETURN'.
INVAL-REQ.
      (Send unsuccessful message to terminal)
      CALL 'RETURN'.
TEST-DEL-NOTICE.
      IF DEL-NOT-CODE = '81' GO TO GET-PROG-CODE  ELSE GO TO ERR-ROUT.
LOAD-STATUS-CHECK.
      IF UTS400-RESPONSE = '39303030' GO TO NORM-TERM.
UNSUCCESSFUL-LOD.
      (Generate error message)
      GO TO NORM-TERM.
SMALL-DATA-BUF.
      (Generate error message)
      GO TO NORM-TERM.
I/O-ERR.
```

Figure 10–1. User-written Downline Load Action Program Sketch (Part 2 of 3)

```
00084        (Generate error message)
00085        GO TO NORM-TERM.
00086    ERR-ROUT.
00087        (Generate error message)
00088        GO TO NORM-TERM.
```

Figure 10–1. User-written Downline Load Action Program Sketch (Part 3 of 3)

Downline load action programs must contain the following:

UTS load module name

▶ An 8-byte field defined for the UTS load-module-name (line 9 of Figure 10–1). The data-name used to describe this 8-byte field is the same name you must use on the SETLOAD function call.

SETLOAD function call

▶ One SETLOAD function call for each downline load (line 41). Issue the SETLOAD function before any GETLOAD function call because initialization must occur before you read a block of code from a UTS load module.

GETLOAD function call

▶ GETLOAD function calls issued to read blocks of code from the UTS load module into the data buffer in the output message area of your calling downline load action program (line 44).

Work area for SETLOAD and GETLOAD

▶ A 400-byte area defined on the word boundary in the continuity data area (line 29). This area is used as a work area by the SETLOAD and GETLOAD function calls.

Data buffer area and size field

▶ The data-buffer (line 27) and 2-byte field indicating its size (line 10). The data-buffer contains a block of code read from the load module.

Size field contents

Before the downline load program issues the GETLOAD function call, the size field (lines 10 and 44) should have the length of the buffer area in binary format. After the return from the GETLOAD call, the size field has the number of bytes actually moved into the buffer area. This number is also in the binary format.

After issuing the GETLOAD function call, the downline load program must:

End-of-file test

- check for end-of-file (02) in the STATUS-CODE field of the program information block (lines 50 and 59–63); and

Process status code

- process the status code in the program information block for successful completion of the GETLOAD function call (lines 46–48 and 59–63).

Successful GETLOAD processing

If the GETLOAD function is successful, the downline load program should:

Character in AUX-FUNCTION FIELD

1. Move 'C' to the AUX-FUNCTION field (the first byte of the AUXILIARY-DEVICE-ID field) of the output message header (line 46) if you are sending the block of UTS program code to the terminal (primary device) main storage. Otherwise, see Table 6-1 for the continuous output character needed by your application.

Load code prefix

2. Prefix the data block received from the GETLOAD function call with a proper heading to load this block either directly into the UTS main storage or to an auxiliary storage device. This prefixed data block becomes the text in the downline load program's output message area. This text length can be calculated using the length returned in the size parameter of the GETLOAD function call. See Figure 10-1, lines 25-27 and 60 for an example of the output message area and the prefixing description required to format the text part of the output message area.

   Prefix for main storage load

   Your downline load action program should move the 6-byte prefix, X'1B0E30323130', into the prefix header (DOWNLINE-LOAD-HEADER) to provide the header information for loading the UTS main storage.

   Prefix for auxiliary storage load

   If the downline load is intended for the auxiliary storage device, your action program should instead move X'1313nnnnnnnn' into the prefix header (DOWNLINE-LOAD-HEADER). Here 'nnnnnnnn' is a 4-character ASCII sequence naming the UTS load program.

   Figure 10-1, line 60 shows that the UTS MAC 80 program (MACPROG1) is downline loaded into the UTS main storage device.

Sending UTS program code

3. Send the message from the downline load action program output message to the UTS terminal or auxiliary device using the continuous output feature (lines 46 and 47).

Terminate downline load program with external succession

4. Terminate the downline load action program with external succession (i.e., place 'E' in the TERMINATION-INDICATOR field of the program information block) and name the downline load action program as the successor. The successor action program must then be prepared to handle a delivery notice in the form of an input message (lines 17-20). This includes testing the delivery notice for error and if an error occurs, moving an error message to the output message area before terminating the program normally (lines 73 and 86-88).

Unsuccessful SETLOAD/GETLOAD

If the SETLOAD or GETLOAD function is unsuccessful and you configured ERET=YES in the PROGRAM section of the configurator, your downline load action program receives control with error indications set in the STATUS-CODE field of the program information block. For status code settings in this case, see status codes 3 and 4 in 10.3. and 10.4. The action program should then send an appropriate error message to the terminal (lines 49–58).

If the SETLOAD or GETLOAD function is unsuccessful and you didn't configure ERET=YES, IMS cancels the transaction and sends the following message to the terminal:

```
DOWN LINE LOAD ERROR.
```

Transfer record

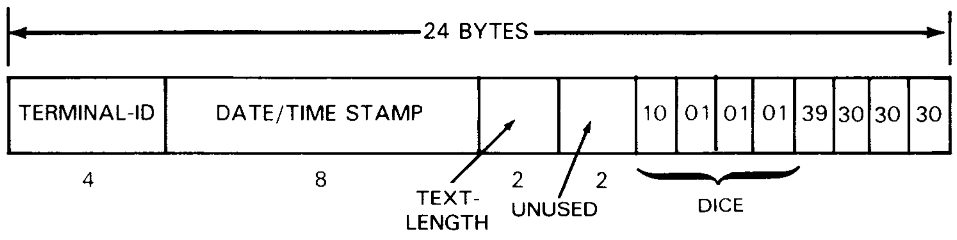
If the GETLOAD function returns an end-of-file condition (STATUS-CODE set to X'02' in the program information block), the buffer area contains the transfer record. This is the last block that should be sent to the UTS terminal; thus, your action program should issue no more GETLOAD functions for this load module.

Response message from UTS terminal

If the blocks of code are sent to the UTS main storage for immediate execution of the program, then when the UTS terminal receives a transfer record it automatically transmits a response (input message) indicating whether or not the downline load was successful. Therefore, the downline load action program should not use continuous output to send this last block. It should follow the same procedure as for a successful GETLOAD function, except it should not move 'C' into the AUX-FUNCTION field of the output message header. The successor action program then receives in its input message area the 24-byte message header from a UTS in the following formats:

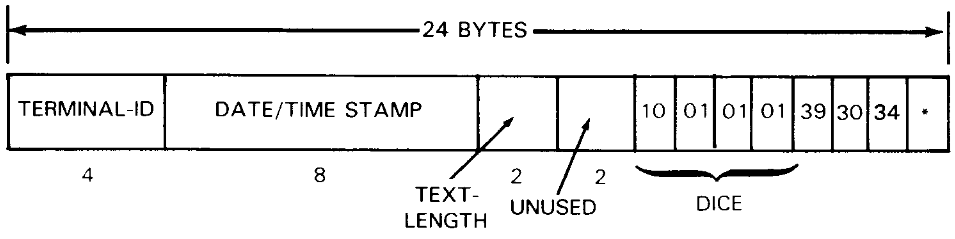
Message header for successful load

SUCCESSFUL LOAD



Message header for unsuccessful load

UNSUCCESSFUL LOAD



NOTE:

If you specify EDIT=NONE in the ACTION section, your program receives these DICE characters. If you specify EDIT=c or EDIT=tablename, or if you omit the EDIT parameter, these characters are stripped from the message header before it is sent to the program.

Unsuccessful load error byte values

Table 10-1 defines the various error bit configurations (*) that can be returned in the last byte of the message from the UTS terminal.

Table 10-1. Rejected Load Error Byte Definition

Bit Number*	Error Type	Probable Cause/Recovery
7	Never set	
6	Always set	
5	Program cannot be loaded because previous program did not clear program-loaded flag (LOADFL)	The UTS operator should initiate a power-on confidence test from the controller or master station and, upon completion of the test, the load should be retried.
4	Load addressed to a UTS slave station instead of a master station	The load should be retried and addressed to the UTS master station.
3	Illegal control code encountered in program	IMS error - submit SUR
2	Block overflow occurred in available/assigned main storage	If main storage is available, the UTS operator should assign the appropriate storage to the program. The load should be retried. If main storage is not available, the program should be recompiled, addressing available storage.
1	Start address of block is not in available/assigned main storage	Use the control page to assign more main storage, and reenter your transaction code. If insufficient main storage is available, the program must be recompiled.
0	Addresses A and B not equal	IMS error - submit SUR

*Numbered from right to left; i.e., bit 7 is the most significant bit; bit 0 is the rightmost or least significant bit.

See Figure 10-1, lines 14-16 for an example of the input message area description to receive the UTS 400 response message after the last block of UTS program code is transferred downline.

Handling UTS response message

After receiving the response message, the downline load action program should:

1. interrogate the response message (lines 75–76) and send an appropriate output message to the terminal indicating the success or failure of the downline load; and

2. terminate normally, i.e., place 'N' in the TERMINATION-INDICATOR of the program information block.

UTS response after auxiliary device load

When the action program downline loads a UTS program to an auxiliary device, the UTS terminal does not generate a response message after it receives the last block of code. Therefore, the status of the downline load is not known until the program code is read into the UTS main storage.

10.3. INITIALIZING DOWNLINE LOAD (SETLOAD)

SETLOAD format

The SETLOAD function call is the first function called by a downline load action program. The COBOL and BAL formats for the SETLOAD function code are:

■ COBOL format

```
CALL 'SETLOAD' USING module-name save-area.
```

■ BAL format

```
{CALL   };SETLOAD,(module-name,save-area)
{ZG#CALL}
```

UTS program module name

Module-name is an 8-byte field containing the name of the UTS program load module to be downline loaded.

Save-area

Save-area is a 400-byte area defined in the continuity data area. IMS uses the save-area to process the SETLOAD and GETLOAD function calls. This area must be word-aligned.

SETLOAD status codes

When a SETLOAD function call is issued, IMS returns one of the following status codes with corresponding detailed status codes in the program information block.

Status Codes (Decimal)	Detailed Status Codes (Decimal)	Description
0	0	Successful SETLOAD
3	1	Invalid request; invalid number of parameters
3	7	Invalid request; function invalid for type of request
3	22	Invalid request; after the initial SETLOAD is issued, SETLOAD may not be issued again until the downline load action program receives the transfer record via the GETLOAD call.

10.4. LOADING THE UTS PROGRAM (GETLOAD)

GETLOAD format

Your downline load action program issues the GETLOAD function call immediately after the SETLOAD function and repeatedly issues the GETLOAD function until end-of-file is reached for the UTS program load module. The COBOL and BAL formats for the GETLOAD function call are:

- COBOL format

COBOL format

```
CALL 'GETLOAD' USING save-area buffer-area size.
```

- BAL format

BAL format

```
{CALL   }GETLOAD,(save-area,buffer-area,size)
{ZG#CALL}
```

Save-area

Save-area is the 400-byte word-aligned area previously defined in the SETLOAD function. IMS uses the save-area to process the SETLOAD and GETLOAD function calls.

Buffer-area

Buffer-area is the data-buffer in the output message area where your program receives a block of code from the UTS load module.

Size field

Size is a 2-byte field where the length (size) of the buffer-area is stored.

GETLOAD status codes

When your downline load action program issues a GETLOAD function call, IMS returns one of the following status codes and corresponding detailed status codes in the program information block.

Status Codes (Decimal)	Detailed Status Codes (Decimal)	Description
0	0	Successful GETLOAD
2	0	End-of-load module (transfer record received). Note that end-of-file is set at the time the last block of data (transfer record) is passed to the action program.
3	20	Invalid request; save-area address invalid or SETLOAD was not issued before GETLOAD.
3	21	Invalid request; data buffer too small (less than 10 bytes).
4	XX	I/O error. XX is the error code (in binary) returned by the OS/3 loader. Note that these error codes are explained in the system messages programmer/operator reference, UP-8076.

10.5. DISCONNECTING A LINE FROM AN ACTION PROGRAM

Line disconnect feature

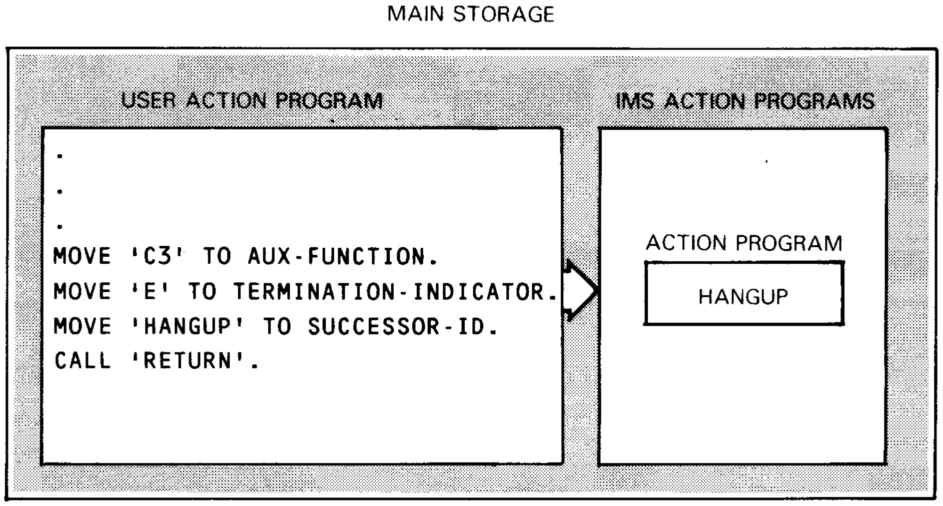
The line disconnect feature allows an action program to disconnect a single-station dial-in line following the delivery of its output message to enable another terminal to dial in on the same line. To use the line disconnect feature, include the continuous output capability in your configuration by specifying CONTOUT=YES in the OPTIONS section. The line disconnect feature is available only in a dedicated ICAM network, not a global network.

Action program operations

To disconnect a line after message transmission, the action program must:

- place a continuous output flag (X'C3') in the AUX-FUNCTION byte (ZA#OAUX field) of the output message header; and
- specify external succession with 'HANGUP' as the successor by setting the TERMINATION-INDICATOR field (ZA#PSIND) in the program information block to E and the SUCCESSOR-ID field (ZA#PSID) to 'HANGUP'.

HANGUP is an action program supplied by IMS that terminates with a special code causing IMS to issue a line release/line request sequence to ICAM to disconnect the line.



```
.
.
.
MOVE 'C3' TO AUX-FUNCTION.
MOVE 'E' TO TERMINATION-INDICATOR.
MOVE 'HANGUP' TO SUCCESSOR-ID.
CALL 'RETURN'.
```

After the output message is sent, no further input is required from the terminal operator. IMS waits for ICAM notification of message delivery before scheduling the external successor, HANGUP. In this way, delivery of the message prior to the line disconnect is ensured.

10.6. INITIATING AN OS/3 JOB FROM AN ACTION PROGRAM (RUN)

RUN function call

You can initiate background batch jobs from your action program by issuing the RUN function call. The RUN function initiates a system command that reads a job control stream and schedules that job for execution. The COBOL and BAL formats for the RUN function call are:

- COBOL Format

COBOL format

```
CALL 'RUN' USING command-text.
```

- BAL Format

BAL format

```
{CALL    } RUN,(command-text)
{ZG#CALL }
```

Command-text

Command-text is the symbolic address of a character string that consists of a valid command and its associated parameters. Valid commands are RUN, RU, RV, SI, SC, OCL, OC, or OV. The command text may not exceed 64 characters. The following COBOL coding illustrates the statements needed in the action program to use the RUN function call:

COBOL example

```
WORKING-STORAGE SECTION.
77    CMD-TEXT        PIC X(18)    VALUE 'RV JOBN(JOBC),HIGH'.
.
.
.
PROCEDURE DIVISION.
.
.
.
PARA-10.
           CALL 'RUN' USING CMD-TEXT.
```

The following coding illustrates the same statement in BAL:

BAL example

```
1          10     16
-------------------------------------------
           CALL   RUN,(CMDTXT)
           .
           .
           .
CMDTXT     DC     CL18'RV JOBN(JOBC),HIGH'
```

P
A
R
T
4.

PREPARING ACTION PROGRAMS FOR EXECUTION

11. Compiling, Linking, and Storing Action Programs

11.1. PREPARING ACTION PROGRAMS FOR ONLINE PROCESSING

After you write a COBOL or BAL action program or subprogram, you must. . .

DO the following

What you must do

1. Compile or assemble the action program or subprogram (11.1).
2. Link edit the program to create a load module (11.2).
3. Store the program in the appropriate load library (11.3).
4. Identify the program to IMS in a PROGRAM section of the configuration. (See the IMS system support functions user guide, UP-8364 (current version).)
5. Identify the load library in the job control stream at IMS start-up, unless programs are stored in the system load library, $Y$LOD. (See UP-8364.)

Scope of section

This section tells you how to compile (or assemble) and link your action programs and subprograms and where to store them for use during the online IMS session. For additional information on the job control statements and procedures shown in the examples, refer to the current versions of the job control user guide, UP-8065, and the appropriate language manual.

11.2. COMPILING OR ASSEMBLING ACTION PROGRAMS

Assembling BAL program

You assemble a basic assembly language action program or subprogram the same way as any other BAL program.

Compiling COBOL program

You compile a COBOL action program or subprogram the same way as other COBOL programs, with one exception. That exception is different for 1974 American National Standard COBOL and extended COBOL and also depends on whether or not the program is sharable.

Sharable and Nonsharable COBOL Programs

Sharable 1974 COBOL program

To compile a sharable 1974 COBOL program, include the job control statement:

```
// PARAM IMSCOD=YES
```

Sharable extended COBOL program

To compile a sharable extended COBOL program, include the job control statement:

```
// PARAM OUT=(M)
```

IMS language restrictions

When you specify IMSCOD=YES or OUT=(M), the COBOL compiler checks for IMS language restrictions and issues diagnostics. For this reason, you should include this PARAM statement even if you don't need a sharable program. However, if your program is not written to sharable standards (for instance, the procedure division contains statements that move data to the working-storage section), you cannot compile it with IMSCOD=YES or OUT=(M).

Configuration requirements

To share COBOL action programs or subprograms, you must specify the TYPE=SHR and SHRDSIZE parameters in your IMS configuration in addition to including the shared code PARAM statement at compilation time. You can share action programs and subprograms only in multithread IMS.

Nonsharable 1974 COBOL program

To compile a nonsharable 1974 COBOL program, include the job control statement:

```
// PARAM CALLST=YES
```

to assure the proper linkages to IMS at CALL interrupts. However, the compiler does not check for IMS language restrictions when you use CALLST=YES instead of IMSCOD=YES.

Nonsharable extended COBOL program

There is no special PARAM statement for compiling nonsharable extended COBOL action programs. When you omit PARAM OUT=(M), the compiler does not check for IMS language restrictions and you receive the COBOL error message:

```
140  NO EXIT PROGRAM NOR RETURN STATEMENT ASSOCIATED WITH
     ENTRY OR USING STATEMENT
```

You can ignore this message.

Table 11-1 summarizes the use of PARAM statements for sharable and nonsharable COBOL action programs.

Table 11-1. Compiling Sharable and Nonsharable COBOL Action Programs

Sharable Action Program		Nonsharable Action Program
1974 COBOL	Include // PARAM IMSCOD=YES. Compiler checks for IMS language restrictions.	Include // PARAM CALLST=YES. Assures proper linkages to IMS at CALL interrupts. Compiler does not check for IMS language restrictions.
Extended COBOL	Include // PARAM OUT=(M). Compiler checks for IMS language restrictions.	No substitute for // PARAM OUT=(M). Compiler does not check for IMS language restrictions, generates error message which can be ignored.

Volatile data area

In the listing for a shared COBOL action program, the size of the volatile data area is printed in decimal just before the COBOL COMPILATION COMPLETE message. The format of this message is:

```
SHARED CODE VOLATILE DATA AREA=nnnn BYTES
```

Multithread IMS uses the shared code volatile data area to save and restore data at CALL interrupts. It is not used in single-thread IMS.

Size used for SHRDSIZE specification

Use this size for the SHRDSIZE parameter specification in the ACTION section of your IMS configuration. If the action includes more than one COBOL action program, use the largest shared code volatile data area for this specification.

Job Control for Compiling COBOL Action Programs

To compile a 1974 COBOL action program or subprogram, you can use either the COBL74 job control procedure (jproc) or the EXEC COBL74 job control statement.

COBL74 jproc

Figure 11-1 uses the jproc and assumes that the source program, MYPROG, is filed in the system source library, $Y$SRC. The program is sharable.

```
// JOB PROG1
//MYPROG COBL74 IN=(RES)
// PARAM IMSCOD=YES
/&
// FIN
```

Figure 11-1. Compiling a 1974 COBOL Action Program Using Jproc

EXEC COBL74 statement

When you use the EXEC COBL74 job control statement, you must allocate a printer and three work files for the COBOL compiler. In Figure 11-2, the source program is embedded in the job control stream. The program is not sharable.

```
// JOB PROG2
// DVC 2Ø  // LFD PRNTR
// WORK1
// WORK2
// WORK3
// EXEC COBL74
// PARAM CALLST=YES
/$
     .
     .  source program
     .
/*
/&
// FIN
```

Figure 11-2. Compiling a 1974 COBOL Action Program Using Standard Job Control

To compile an extended COBOL action program or subprogram, you can use either the COBOL jproc or the EXEC COBOL job control statement.

COBOL jproc

Figure 11-3 executes the extended COBOL compiler using the COBOL jproc. In this example, the source program is embedded in the job control stream, and the program is sharable.

```
// JOB PROG3
// COBOL
// PARAM OUT=(M)
/$
     .
     .  source program
     .
/*
/&
// FIN
```

Figure 11-3. Compiling an Extended COBOL Action Program Using Jproc

EXEC COBOL statement

Figure 11-4 uses the EXEC COBOL job control statement and assumes that the source program, MYPROG, is filed in a user source library, SRCIN. Notice that a device assignment set is required for the user source library. The program is sharable.

```
// JOB PROG4
// DVC 20  // LFD PRNTR
// DVC 50  // VOL DISK01  // LBL SRCLIB  // LFD SRCIN
// WORK1
// WORK2
// WORK3
// EXEC COBOL
// PARAM IN=MYPROG/SRCIN
// PARAM OUT=(M)
/&
// FIN
```

Figure 11-4. Compiling an Extended COBOL Action Program Using Standard Job Control.

Job Control for Assembling BAL Action Programs

You assemble BAL action programs and subprograms the same way as other BAL programs, using the ASM jproc or the EXEC ASM job control statement.

ASM jproc

Figure 11-5 uses the ASM jproc and assumes the source program, ASMPRG, is filed in the system source library, $Y$SRC.

```
// JOB PROG5
//ASMPRG ASM IN=(RES)
/&
// FIN
```

Figure 11-5. Assembling a BAL Action Program Using Jproc

EXEC ASM statement

Figure 11-6 uses the EXEC ASM job control statement and takes source input from the job control stream. You must allocate a printer and two work files for the assembler.

```
// JOB PROG6
// DVC 2Ø  // LFD PRNTR
// WORK1
// WORK2
// EXEC ASM
/$
    .
    .  source program
    .
/*
/&
// FIN
```

Figure 11-6. Assembling a BAL Action Program Using Standard Job Control

11.3. LINK EDITING ACTION PROGRAMS

After you obtain a clean action program compilation or assembly, you must link edit the program and store it in the appropriate load library. We discuss load libraries in 11.4.

When you can use LINK jproc

You can use the LINK job control procedure for a BAL program or for a COBOL program compiled with PARAM IMSCOD=YES or PARAM OUT=(M). You must use the EXEC LNKEDT job control statement for nonsharable COBOL action programs.

On the LINK jproc, you must specify the OUT parameter to store the action program in a load library:

LINK jproc format

```
// LINK action-program-name, OUT={(vol-ser-no,label)}
                                 {(RES,$Y$LOD)      }
```

For example:

```
// LINK MYPROG,OUT=(RES,$Y$LOD)
```

If you want to give the action program load module a different name than the object module, use this format:

Format for naming load module

```
//load-module-name LINK object-module-name,
   OUT={(vol-ser-no,label)}
       {(RES,$Y$LOD)      }
```

LINK jproc example

Figure 11-7 uses the jproc to link edit an object module called MYPROG and create a load module called CREDIT. Output is to LOADLIB. You do not need a device assignment for LOADLIB because the LINK jproc generates it from your OUT specification.

```
// JOB LINK
//CREDIT LINK MYPROG,OUT=(IMSVOL,LOADLIB)
/&
// FIN
```

Figure 11-7. Link Editing an Action Program Using Jproc

Using standard job control

When you execute the linkage editor using standard job control, you need a LOADM statement to name the load module and INCLUDE statements for the action program object module and the IMS link module, ZF#LINK.

ENTER statement

A nonsharable extended COBOL action program or subprogram also requires an ENTER statement. The ENTER statement must be the last linkage editor control statement in your job control stream.

Example using EXEC LNKEDT

Figure 11-8 shows a standard job control stream for the linkage editor. The linkage editor requires a printer file and one work file. You can omit the printer file if you assigned one to the compiler in the same job control stream. Output is to the system load library, $Y$LOD; a device assignment is not needed for this file.

```
// JOB LNKEDT
// DVC 20  // LFD PRNTR
// WORK1
// EXEC LNKEDT
// PARAM OUT=$Y$LOD
/$
   LOADM CREDIT
   INCLUDE MYPROG①
   INCLUDE ZF#LINK,$Y$OBJ
   ENTER MYPROG②
/*
/&
// FIN
```

NOTES:

① For extended COBOL, the object module name is appended with 00.

② Required only for nonsharable extended COBOL programs.

Figure 11-8. Link Editing an Action Program Using Standard Job Control

Compile and link example using jprocs

Figure 11-9 shows a job control stream for compiling and linking a 1974 COBOL action program, using both the COBL74 and LINK jprocs. The action program is stored in the LOAD action program library (see 11.4). The LINK jproc generates a device assignment for the load library.

```
// JOB COBL
//MYPROG COBL74 IN=(RES)
// PARAM IMSCOD=YES
//CREDIT LINK MYPROG,OUT=(IMSVOL,LOAD)
/&
// FIN
```

Figure 11-9. Compiling and Linking a COBOL Action Program Using Jprocs

Assemble and link example using standard job control

Figure 11-10 shows a job control stream for assembling and linking a BAL action program, using standard job control. A device assignment set is required for the output file, LOADLIB.

```
// JOB ASML
// DVC 2Ø  // LFD PRNTR
// DVC 5Ø  // VOL IMSVOL  // LBL LOADLIB  // LFD LOADLIB
// WORK1
// WORK2
// EXEC ASM
    .
    .  source program
    .
/*
// WORK1
// EXEC LNKEDT
// PARAM OUT=LOADLIB
/$
   LOADM PAYROL
   INCLUDE ASMPRG
   INCLUDE ZF#LINK,$Y$OBJ
/*
/&
// FIN
```

Figure 11-10. Assembling and Linking a BAL Action Program Using Standard Job Control

11.4. STORING ACTION PROGRAMS IN A LOAD LIBRARY

When you link edit an action program, you must specify the load library where you want it stored. IMS has specific requirements for storing action programs.

One library for action programs

The first requirement is that all your action programs must reside in the same load library.

When you use fast load feature

The load library you choose depends on whether or not you configure the fast load feature by specifying FASTLOAD=YES in the OPTIONS section of your IMS configuration. (See the IMS system support functions user guide, UP-8364 (current version).)

Improves performance

The fast load feature improves online performance in applications with large action programs or frequent action program loading.

Fast loading requires LOAD library

Action programs loaded from fast load file

If you configure fast loading, place all action programs in a separate action program load library in unblocked format. You assign this library at IMS start-up with the LFD-name LOAD. At start-up, you also assign the fast load file, LDPFILE. The first time a transaction calls on a particular action program, IMS copies the program from LOAD to the LDPFILE. After that, action programs are loaded from LDPFILE.

When you do not use fast load feature

If you do not want fast loading, you can store your action programs in either of two libraries (but all in the same library):

1. the system load library, $Y$LOD; or
2. the library containing your online IMS load module. This library is identified at configuration time by the LIBL parameter of the IMSCONF jproc.

NOTE:

Where to store UTS programs

If you use downline loading (10.1), store your universal termination system (UTS) programs in $Y$LOD or in the library containing the online IMS load module. Do not store UTS programs in the LOAD action program library.

11.5. REPLACING ACTION PROGRAMS IN THE LOAD LIBRARY DURING ONLINE PROCESSING

Subprogram restriction

You can replace action programs in the load library while IMS is online, whether or not you use the fast load feature. However, you cannot replace resident subprograms during online processing.

How to replace programs

You replace an action program in the $Y$LOD, LOAD, or other load library by recompiling (or reassembling) and relinking, or by applying a patch (COR). For an explanation of the COR function, see the system service programs user guide, UP-8062 (current version).

Fast load requirement

When you use the fast load feature, you must insert the statement:

```
// DD ACCESS=EXCR
```

in the device assignment set for the LOAD library in the compile and link or COR job control stream.

Recompile and link example

The job control stream in Figure 11-11 recompiles and links a 1974 COBOL action program for output to the LOAD file. This example assumes you use the fast load feature.

```
// JOB RECOMP
// DVC 50  // VOL IMSVOL  // DD ACCESS=EXCR  // LBL LOAD  // LFD LOAD
//MYPROG COBL74 IN=(RES)
// PARAM IMSCOD=YES
//CREDIT LINK MYPROG,OUT=(IMSVOL,LOAD)
/&
// FIN
```

Figure 11-11. Recompiling and Linking an Action Program During Online Processing

ZZPCH command

After replacing the action program in the load library, issue the ZZPCH master terminal command. The next time a transaction calls on the action program, IMS loads the new version from the load library. When you use the fast load feature, IMS copies the new version to the LDPFILE. The ZZPCH master terminal command is described in the IMS terminal users guide, UP-9208 (current version).

Adding action program to library

Follow the same procedure to add an action program to the load library that is missing at start-up. Of course, the program must be defined in a PROGRAM section of the IMS configuration.

ALTER statement restricted when using fast loading

When you use the fast load feature, do not use ALTER statements in the job control steam at IMS start-up. When you do not use fast loading, you can insert ALTER statements in the start-up job control stream to make temporary changes to action programs.

PART 5.

SNAP DUMP ANALYSIS

12. Debugging Action Programs

Though error-free programs are every programmer's dream, in reality they never seem to materialize. After all the explanations are made about how to program applications correctly, probably the most important tool a programmer has is his working knowledge of debugging procedures. Consequently, it's important to know how to debug your action program using the snap dump feature provided by IMS.

12.1. TYPES OF SNAP DUMPS

Termination and CALL SNAP dumps

You can obtain two types of snap dumps:

1. the termination snap dump
2. the CALL SNAP dump

Obtaining termination snap

A termination snap is caused by action program termination either by voluntarily moving an S to the termination indicator or by abnormally terminating due to program check or timer-check (time out due to a loop in the action program).

Obtaining CALL SNAP dump

A CALL SNAP dump is caused by your program voluntarily issuing the CALL SNAP statement in a COBOL action program or the ZG#CALL SNAP macroinstruction in a BAL action program. The action program does not terminate to produce this dump.

Edited and unedited snaps

IMS provides both edited and unedited snap dumps. In single-thread IMS, termination snaps are always edited; however, for CALL SNAP dumps only unedited snap dumps are available. In multithread IMS, users must specify SNAPED=YES in the OPTIONS section of the IMS configuration to obtain edited snap dumps.

12.2. TERMINATION SNAPS

General breakdown

Figure 12-1 illustrates the general layout of a termination snap dump caused by S termination indicator or abnormal termination.

This same general layout applies to single and multithread IMS.

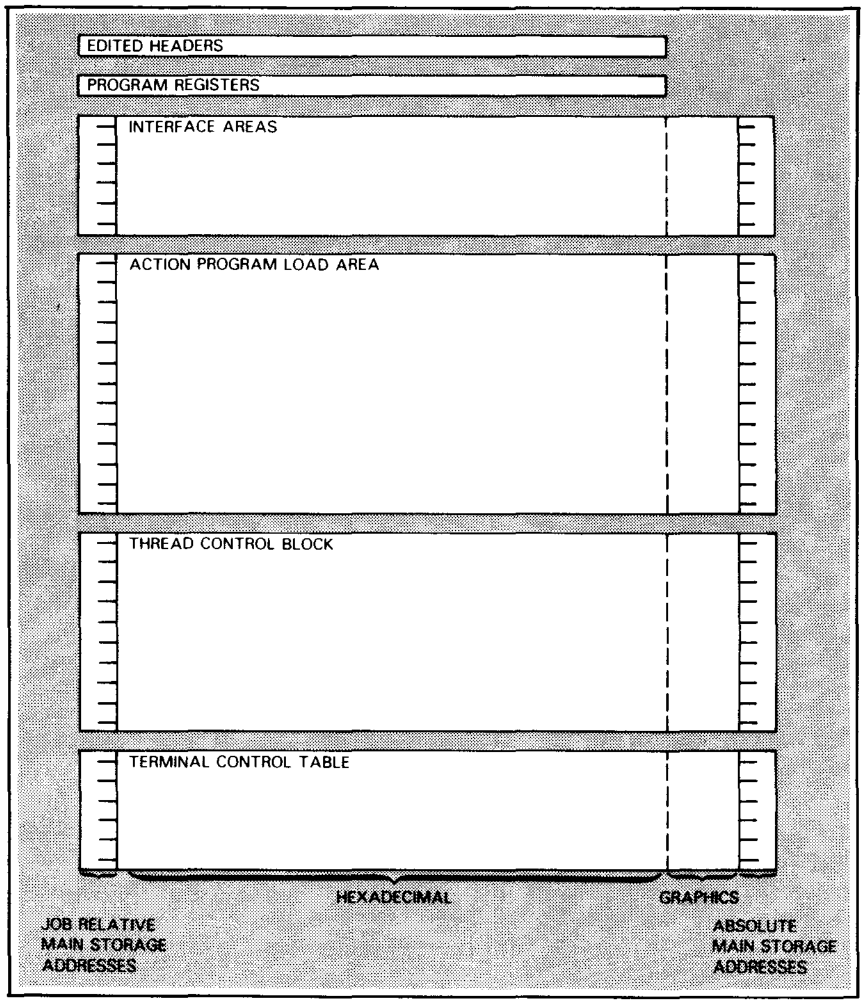


Figure 12-1. Layout of a Termination Snap Dump

There are six sections to each termination snap dump: edited headers, IMS and action program registers, interface areas, action program load area, the thread control block, and the terminal control table.

Edited header data

The edited header section contains information about the action program that was running when the snap occurred. Included is the name of the action program load module that was executing, an allocation map that provides the relative addresses of action programs and IMS areas needed in debugging the program, and a general statement of why the snap dump occurred: e.g., USER REQUESTED VOLUNTARY TERMINATION.

Register section

Registers with a voluntary termination snap

The next section contains registers and their contents. Here, you'll find one or two sets of registers depending on the reason for the snap dump. If your action program voluntarily terminated with a snap, i.e., S termination indicator, your snap dump contains one set of registers - IMS registers. These registers are of little use to you.

When you voluntarily terminate your action program to obtain a snap dump, you're usually checking contents of interface areas that are easily locatable from the allocation map in your snap dump. In this situation, you do not need to obtain a program status word from the save area. Furthermore, no program status word is passed to the save area on a termination snap.

If, however, your action programs are in BAL and you do need to know your action program's register contents on a termination snap, look in your action program's save area plus C_{16} bytes to find registers 14, 15, and 0-12 in that order.

To arrive at the save area plus C_{16}, locate the BAL program information block DSECT field, ZA#PSAVE, which contains the address of your action program save area. (See Figure 3-3 for the BAL program information block DSECT.)

Registers with an abnormal termination snap

On the other hand, if IMS terminates your action program abnormally, the snap dump contains two sets of registers - user action program registers and IMS registers.

User registers precede IMS registers and are labeled so they are easily identifiable. Just above the user registers 0-F is the 8-byte program status word indicating in its last three bytes the address of the instruction immediately following the one that caused the abnormal termination. (See Figure 12-6 program status word, E0E60E01 40$\boxed{034C5C}_{16}$.)

Interface areas

Following the register section, you find the interface areas - program information block, output message area, input message area, work area, continuity data area, and defined record area.

Program area

The next section of the snap dump is the action program load area. It contains the executable load module generated by the linkage editor.

Thread control block

Following the action program area is a section used for the action program's thread control block. In the third control block, most pointers and flags required to control the user environment are stored for use by IMS and indirectly by the user action program.

Figure 12-2 illustrates the relationship between the IMS thread control block and the user interface areas for both single-thread and multithread IMS.

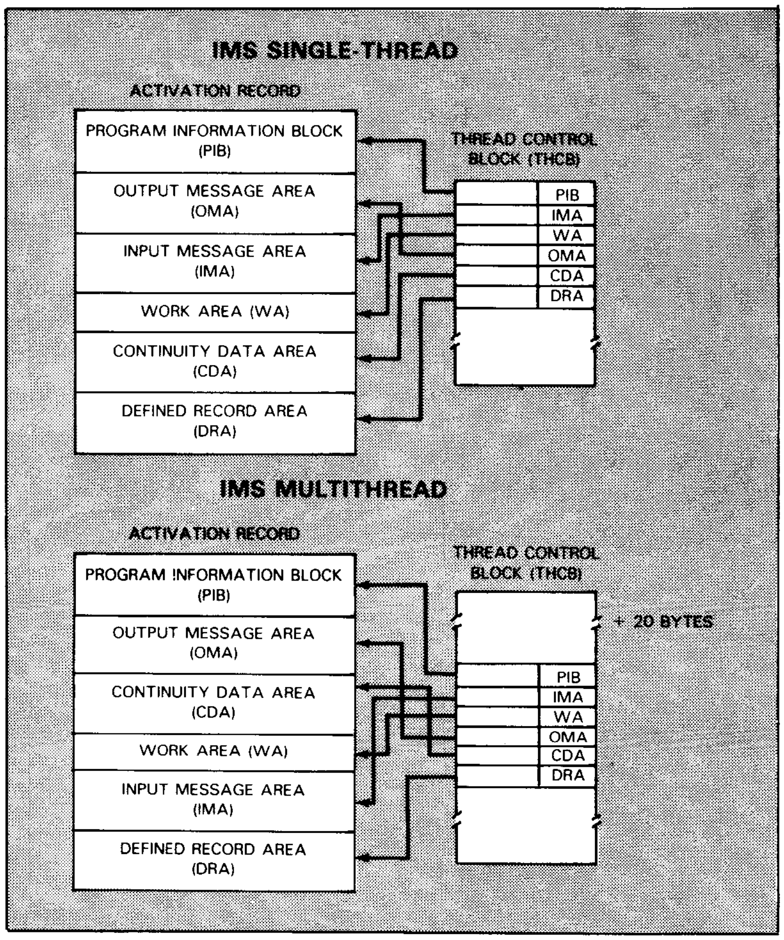

IMS SINGLE-THREAD
ACTIVATION RECORD
PROGRAM INFORMATION BLOCK (PIB)
OUTPUT MESSAGE AREA (OMA)
INPUT MESSAGE AREA (IMA)
WORK AREA (WA)
CONTINUITY DATA AREA (CDA)
DEFINED RECORD AREA (DRA)
THREAD CONTROL BLOCK (THCB)
PIB
IMA
WA
OMA
CDA
DRA
IMS MULTITHREAD
ACTIVATION RECORD
PROGRAM INFORMATION BLOCK (PIB)
OUTPUT MESSAGE AREA (OMA)
CONTINUITY DATA AREA (CDA)
WORK AREA (WA)
INPUT MESSAGE AREA (IMA)
DEFINED RECORD AREA (DRA)
THREAD CONTROL BLOCK (THCB)
+ 20 BYTES
PIB
IMA
WA
OMA
CDA
DRA

Figure 12-2. Relation between THCB and Interface Areas

Notice that pointers within the thread control block point to each interface area. Single-thread and multithread IMS differ only in the location of these pointers and in the relative order of the interface areas themselves.

Thread control block locations for single and multithread

Also, the program information block (first interface area) in the thread control block is located 20 bytes into the thread control block in a multithread termination snap. In a sinlge-thread termination snap, the program information block begins at the first byte of the thread control block.

Terminal control block

The last section in the snap dump is the terminal control table. Data in this area is relevant to the terminal that initiated the action and is the least useful section of the dump to the IMS programmer.

12.3. CALL SNAP DUMPS

Layout Description

General breakdown

Figure 12-3 illustrates the general layout of CALL SNAP dump. Except for the edited headers, this layout pertains to single and multithread CALL SNAP dumps. All single-thread CALL SNAP dumps are unedited.

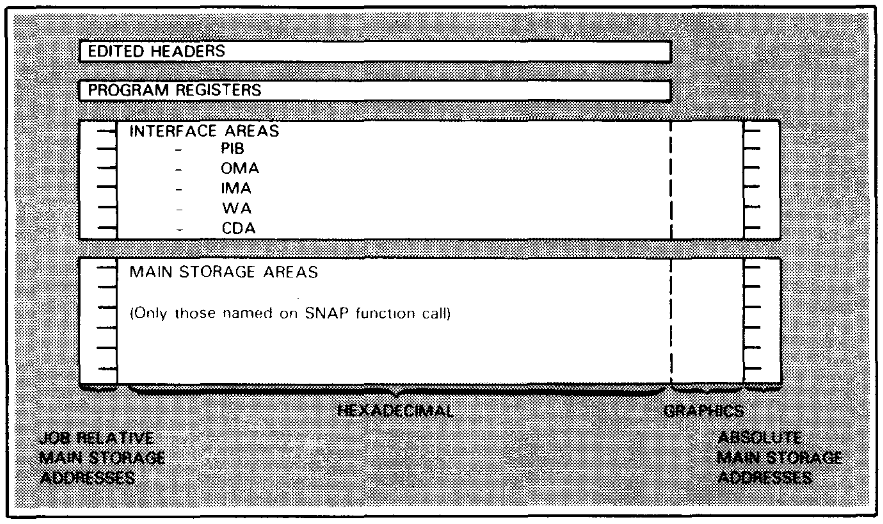


Figure 12-3. Layout of a CALL SNAP Dump

There are three sections in each CALL SNAP dump:

1. Edited headers (for edited dumps)
2. IMS registers
3. Requested main storage areas

Edited headers data

The edited Header Section contains information about the action program that was running when the CALL SNAP occurred. Included is the name of the action program load module that was executing, an allocation map that provides the relative addresses of action programs and IMS areas needed in debugging the program, and a general statement of why the snap dump occurred; e.g., USER INLINE SNAP.

Register section

The register section contains IMS registers only. No program registers are shown. These registers are of little use to you.

Interface areas

Following the register section, you find the main storage area. The main storage areas included in the CALL SNAP dump are only those you named on the SNAP function call in your action program. You can dump up to six main storage areas including interface areas.

SNAP Function Call

Purpose of SNAP function call

When you want to debug your action program without terminating the program, use the SNAP function call. The SNAP function dumps up to six noncontiguous main storage areas in hexadecimal. Output is to the printer. COBOL and BAL formats for the SNAP function calls are:

▶ COBOL Format

```
CALL 'SNAP' USING start-area-1 end-area-1
                  [...start-area-6 end-area-6].
```

▶ BAL Format

```
ZG#CALL SNAP,(start-addr-1, end-addr-1[,...start-addr-6,
               end-addr-6])
```

Start-area and end-area

The *start-area-1* and *end-area-1* parameters are paired for the COBOL CALL statement just as the *start-addr-1* and *end-addr-1* parameters are paired for the BAL CALL statement. The *start-area-1* is the data name of the beginning of the area to be snapped and the *end-area-1* is the data name of the end of the area to be snapped.

Start-addr and end-addr

For the BAL CALL macroinstruction, the *start-addr-1* and *end-addr-1* parameters indicate the start and end addresses of the area being snapped.

Six noncontiguous areas snapped

The SNAP function dumps up to six areas including the program information block, input message area, work area, output message area, continuity data area, working-storage (COBOL), and defined storage area (BAL).

SNAP function and naming areas to be snapped

In the FIXSAM action program (Figure 12-7, line 312) the SNAP function call shows how the start areas and end areas are paired and their data names defined elsewhere in the program. Though the beginning and ending identification of these snapped areas may occur on the SNAP function call in any order as long as they are paired, the interface areas take their beginning and ending identification from the single and multithread activation record layouts shown in Figure 12-2.

12.4. SINGLE AND MULTITHREAD SNAPS

Order of interface areas

There are three major differences between single-thread and multithread snap dumps. First, the order of the interface areas is different. In a single-thread dump, it is: program information block; output message area; input message area; work area; continuity data area; and defined record area if defined files are used. On a multithread dump, it is: program information block; output message area; continuity data area; work area; input message area; and defined record area if defined files are used. Since the allocation map in an edited dump points directly to these areas, there should be no difficulty in locating them in either single or multithread IMS dumps.

Different DSECTs

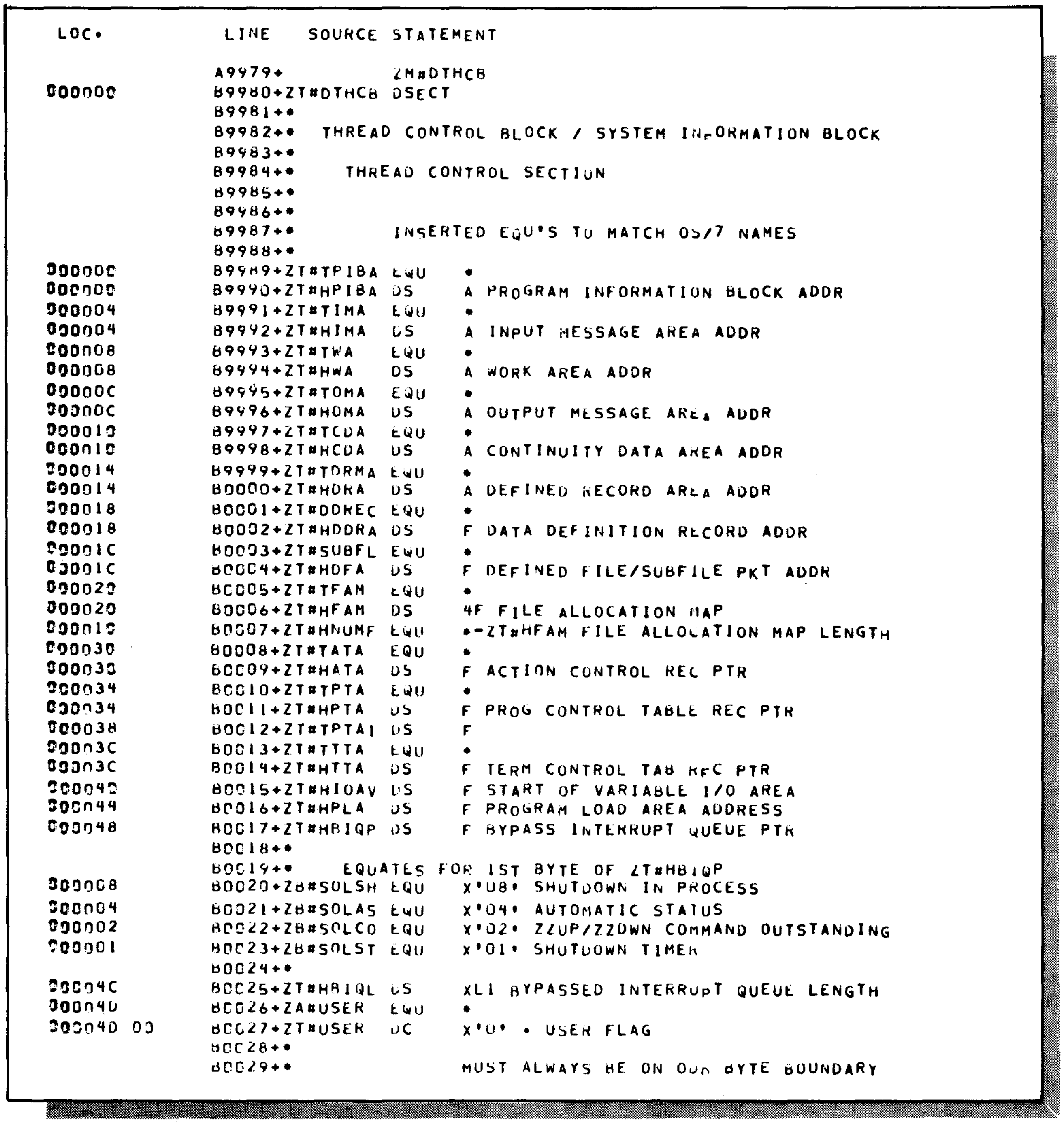
The second major difference concerns the thread control block. The format for single-thread and multithread is totally different. Figures 12-4 and 12-5 provide listings of the thread control block DSECTs for both single-thread and multithread IMS. By examining these figures, notice that although the format is different, the data they contain is basically the same.

Shared code differences

The third difference is if the action program is a shared code COBOL program, in multithread the termination snap dump shows an additional area appended to the end of the program information block. This is the shared code volatile save area used by IMS and COBOL to make COBOL reentrant. This portion of the dump is of little use to an action programmer.

The terminal control table for single and multithread IMS is also a valuable debugging aid. Figure 12-6 shows this table.

SINGLE THREAD CONTROL BLOCK



```
  LOC.          LINE    SOURCE STATEMENT

               A9979+          ZM#DTHCB
000000         B9980+ZT#DTHCB DSECT
               B9981+*
               B9982+*   THREAD CONTROL BLOCK / SYSTEM INFORMATION BLOCK
               B9983+*
               B9984+*     THREAD CONTROL SECTION
               B9985+*
               B9986+*
               B9987+*           INSERTED EQU'S TO MATCH OS/7 NAMES
               B9988+*
000000         B9989+ZT#TPIBA EQU    *
000000         B9990+ZT#HPIBA DS     A PROGRAM INFORMATION BLOCK ADDR
000004         B9991+ZT#TIMA  EQU    *
000004         B9992+ZT#HIMA  DS     A INPUT MESSAGE AREA ADDR
000008         B9993+ZT#TWA   EQU    *
000008         B9994+ZT#HWA   DS     A WORK AREA ADDR
00000C         B9995+ZT#TOMA  EQU    *
00000C         B9996+ZT#HOMA  DS     A OUTPUT MESSAGE AREA ADDR
000010         B9997+ZT#TCDA  EQU    *
000010         B9998+ZT#HCDA  DS     A CONTINUITY DATA AREA ADDR
000014         B9999+ZT#TDRMA EQU    *
000014         B0000+ZT#HDRA  DS     A DEFINED RECORD AREA ADDR
000018         B0001+ZT#DDREC EQU    *
000018         B0002+ZT#HDDRA DS     F DATA DEFINITION RECORD ADDR
00001C         B0003+ZT#SUBFL EQU    *
00001C         B0004+ZT#HDFA  DS     F DEFINED FILE/SUBFILE PKT ADDR
000020         B0005+ZT#TFAM  EQU    *
000020         B0006+ZT#HFAM  DS     4F FILE ALLOCATION MAP
000010         B0007+ZT#HNUMF EQU    *-ZT#HFAM FILE ALLOCATION MAP LENGTH
000030         B0008+ZT#TATA  EQU    *
000030         B0009+ZT#HATA  DS     F ACTION CONTROL REC PTR
000034         B0010+ZT#TPTA  EQU    *
000034         B0011+ZT#HPTA  DS     F PROG CONTROL TABLE REC PTR
000038         B0012+ZT#TPTA1 DS     F
00003C         B0013+ZT#TTTA  EQU    *
00003C         B0014+ZT#HTTA  DS     F TERM CONTROL TAB REC PTR
000040         B0015+ZT#HIOAV DS     F START OF VARIABLE I/O AREA
000044         B0016+ZT#HPLA  DS     F PROGRAM LOAD AREA ADDRESS
000048         B0017+ZT#HBIQP DS     F BYPASS INTERRUPT QUEUE PTR
               B0018+*
               B0019+*     EQUATES FOR 1ST BYTE OF ZT#HBIQP
000008         B0020+ZB#SOLSH EQU    X'08' SHUTDOWN IN PROCESS
000004         B0021+ZB#SOLAS EQU    X'04' AUTOMATIC STATUS
000002         B0022+ZB#SOLCO EQU    X'02' ZZUP/ZZDWN COMMAND OUTSTANDING
000001         B0023+ZB#SOLST EQU    X'01' SHUTDOWN TIMER
               B0024+*
00004C         B0025+ZT#HBIQL DS     XL1 BYPASSED INTERRUPT QUEUE LENGTH
00004D         B0026+ZA#USER  EQU    *
00004D 00      B0027+ZT#USER  DC     X'0' . USER FLAG
               B0028+*
               B0029+*               MUST ALWAYS BE ON ODD BYTE BOUNDARY
```

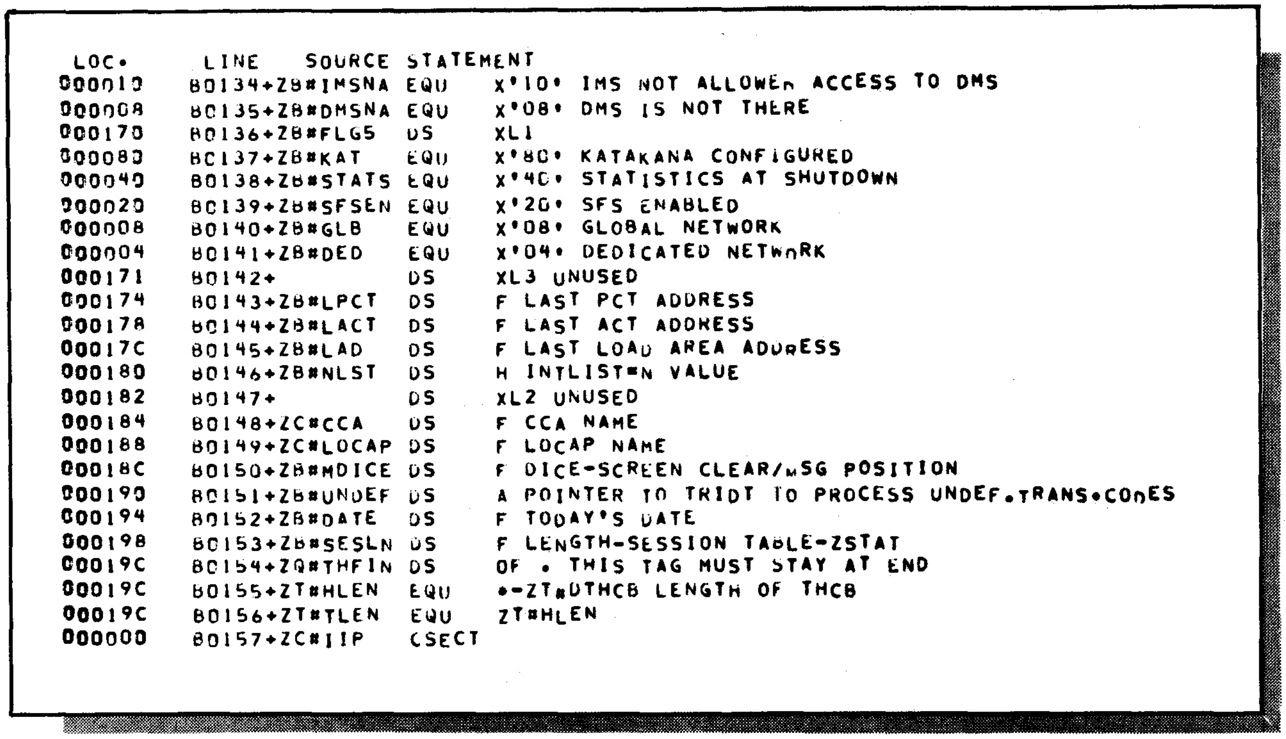
Figure 12–4. Single-Thread Thread Control Block (Part 1 of 4)

```
  LOC.      LINE    SOURCE STATEMENT
           B0030+*
           B0031+*                                                  80 - I/O HAS OCCURRED
           B0032+*                                                  40 - INITIAL SETTING FOR USER
           B0033+*                                                  00 - IMS ACTIVE
           B0034+*                                                     - COUNT FOR TOTAL TIME
00004E     B0035+ZT#TIND  EQU    *
00004E     B0036+ZT#HIND  DS     XL1 CONTROL INDICATORS
           B0037+*
           B0038+*     EQUATES FOR ZT#HIND
           B0039+*
000080     B0040+ZT#HINSP EQU    X'80' SNAP INDICATOR
000040     B0041+ZT#HINER EQU    X'40' ERROR RETURN
000020     B0042+ZT#HINDI EQU    X'20' DELAYED INTERNAL SUCCESSION
000010     B0043+ZT#HINEO EQU    X'10' EXPLICIT OUTPUT
000008     B0044+ZT#HINEX EQU    X'08' EXTERNAL SUCCESSION
000004     B0045+ZT#HINCN EQU    X'04' CANCELLED
000002     B0046+ZT#HINIR EQU    X'02' INTERNAL REQUEST TO FILE MGMT
000001     B0047+ZT#HINUP EQU    X'01' UPDATE PERFORMED BY THIS ACTION
           B0048+*
00004F     B0049+ZT#SYIND DS     XL1 CONTROL INDICATORS
000080     B0050+ZT#ILIST EQU    X'80' INTERRUPT LIST IF SET
000040     B0051+ZT#TOMRD EQU    X'40' . IF ON INDICATES READ FROM TOMFOLE
000020     B0052+ZT#TRSD  EQU    X'20' . RESEND = NO
000010     B0053+ZT#UTOUT EQU    X'10' USER TIME OUT
000008     B0054+ZT#ESETL EQU    X'08'
000004     B0055+ZT#USETX EQU    X'04' USE THE TEXT IN OMA ALTHOUGH TRANS WAS CNC
000002     B0056+ZT#ZZOPN EQU    X'02' INDICATES TO WRITE ZZOPN TERM. RECORD
000050     B0057+ZT#PSSK  DS     9F
           B0058+*
           B0059+*     FILE MANAGEMENT ENTRIES
           B0060+*
000074     B0061+ZT#TFC   EQU    *
000074     B0062+ZT#HFC   DS     F BYTE 0 :# OF PARAMS
           B0063+*                                      BYTE 3 : FUNCTION CODE
000078     B0064+ZT#TUPDA EQU    *
000078     B0065+ZT#HUPDA DS     F UNPROTECTED DTF ADDR
00007C     B0066+ZT#TCR   EQU    *
00007C     B0067+ZT#HRPLA DS     F PARAM LIST ADDR
000080     B0068+ZT#TFWA  EQU    *
000080     B0069+ZT#HFWA  DS     3A FILE MGMT WORK AREA
00008C     B0070+ZT#DMSL  DS     A TCT ADDR OF DMS RUN-UNIT
000090     B0071+ZT#DMCA  DS     A DMS - DMCA ADDRESS
           B0072+*
           B0073+*     SAVE AREAS
           B0074+*
           B0075+*
           B0076+*
000094     B0077+ZT#HSADM DS     18F DATA MANAGEMENT SAVE AREA
0000DC     B0078+ZT#HSAIR DS     18F INTERNAL REQUEST SAVE AREA
           B0079+*
           B0080+*     SYSTEM INFORMATION SECTION
           B0081+*
```

Figure 12-4. Single-Thread Thread Control Block (Part 2 of 4)

```
 LOC.      LINE    SOURCE STATEMENT
000124    B0082+ZB#STIDT DS    F TRANSACTION CODE TABLE
000128    B0083+ZB#SACT  DS    F ACTION CONTROL TABLE
00012C    B0084+ZB#SPCT  DS    F PROGRAM CONTROL TABLE
000130    B0085+ZB#SFCTI DS    F FILE CONTROL TABLE INDEX
000134    B0086+ZB#STERM DS    F TERMINAL CNTL TBL ADDR
000138    B0087+ZB#SDCTI DS    F DEF FILE CONTROL TABLE
00013C    B0088+ZB#SFADR DS    F IMS LOAD ADDRESS
000140    B0089+ZB#SAVAL DS    F AVAILABLE LIST ADDRESS
000144    B0090+ZB#STCS  DS    F TERM. CONTROL SECTION
000148    B0091+ZB#SIMB  DS    F INPUT MESSAGE BUFFER
00014C    B0092+ZB#SIOAE DS    F I/O AREA END ADDR
000150    B0093+ZB#SFSAD DS    A ADDR IMS SESSION STATISTICS
000154    B0094+ZB#LOUTM DS    H LARGEST OUTPUT MSG.
000156    B0095+ZB#LINM  DS    H LARGEST INPUT MSG.
000158    B0096+ZB#LOMTI DS    4C LARGEST OUTPUT MSG.-TERM ID. NAME
00015C    B0097+ZB#LIMTI DS    4C LARGEST INPUT  MSG.-TERM ID. NAME
000160    B0098+ZB#SMLL  DS    H STANDARD MESSAGE LINE LENGTH
000162    B0099+ZB#SMNL  DS    H STANDARD MESSAGE NUMBER OF LINES
000164    B0100+ZB#SIMBL DS    H INPUT MESSAGE BUFFER LENGTH
000166    B0101+ZB#TMCCA DS    H NUMBER OF TERMS IN ICAM CCA
000168    B0102+ZB#STOF  DS    XL1 . USER TIMEOUT FLAG
000169    B0103+ZB#SOLOF DS    XL1 CONTROL INDICATORS FOR AUDIT
          B0104+*
          B0105+*     EQUATES FOR ZB#SOLOF
000080    B0106+ZB#SOLUP EQU   X'80' UPDATING PERMITTED
000040    B0107+ZB#SOLAI EQU   X'40' AUDIT MODULE INCLUDED
          B0108+*                                (BEF IMAGES, TR FILES)
000020    B0109+ZB#SOLRD EQU   X'20' ROLLBACK PROGRAM / FILE DOWN
000010    B0110+ZB#SOLSU EQU   X'10' SUPPRESS UPDATES
000008    B0111+ZB#SOLTB EQU   X'08' BEFORE IMAGES TRACED
000004    B0112+ZB#SOLTA EQU   X'04' AFTER IMAGES TRACED
000002    B0113+ZB#SOLTI EQU   X'02' INPUT MESSAGES TRACED
000001    B0114+ZB#SOLTE EQU   X'01' I/O ERROR TRACE FILE
          B0115+*
00016C    B0116+         DS    0F
          B0117+*
00016C    B0118+ZB#FLG1  DS    X . FLAG1 OF STARTUP
000080    B0119+ZB#STRIN EQU   X'80' . STARTUP ACTIVE
000040    B0120+ZB#TCRSH EQU   X'40' ..TRCFILE=CRASH
000020    B0121+ZB#TEXT  EQU   X'20' ..TRCFILE=EXT
00016D    B0122+ZB#FLG2  DS    X .FLAG FOR TOMFILE
000080    B0123+ZB#TOMUP EQU   X'80' . TOMFILE CONFIGURED
000001    B0124+ZB#TOMER EQU   X'01' . ERROR ON TOM FILE
000002    B0125+ZB#TOMNT EQU   X'02' . DO NOT TRACE TOMFILE
00016E    B0126+ZB#FLG3  DS    X .FLAG FOR TYPE OF RESTART
000001    B0127+ZB#INDCL EQU   X'01' .START=CLEAN
000002    B0128+ZB#INDWA EQU   X'02' .START=WARM
000004    B0129+ZB#INDCO EQU   X'04' .START=COLD
00016F    B0130+ZB#FLG4  DS    X DMS FLAG BYTE
000080    B0131+ZB#IMSDM EQU   X'80' IMS HAS MADE A REQUEST TO DMS
000040    B0132+ZB#DMSDC EQU   X'40' DMS HAS TERMINATED
000020    B0133+ZB#DMSRU EQU   X'20' DMS RUN-UNIT EXISTS
```

Figure 12–4. Single-Thread Thread Control Block (Part 3 of 4)

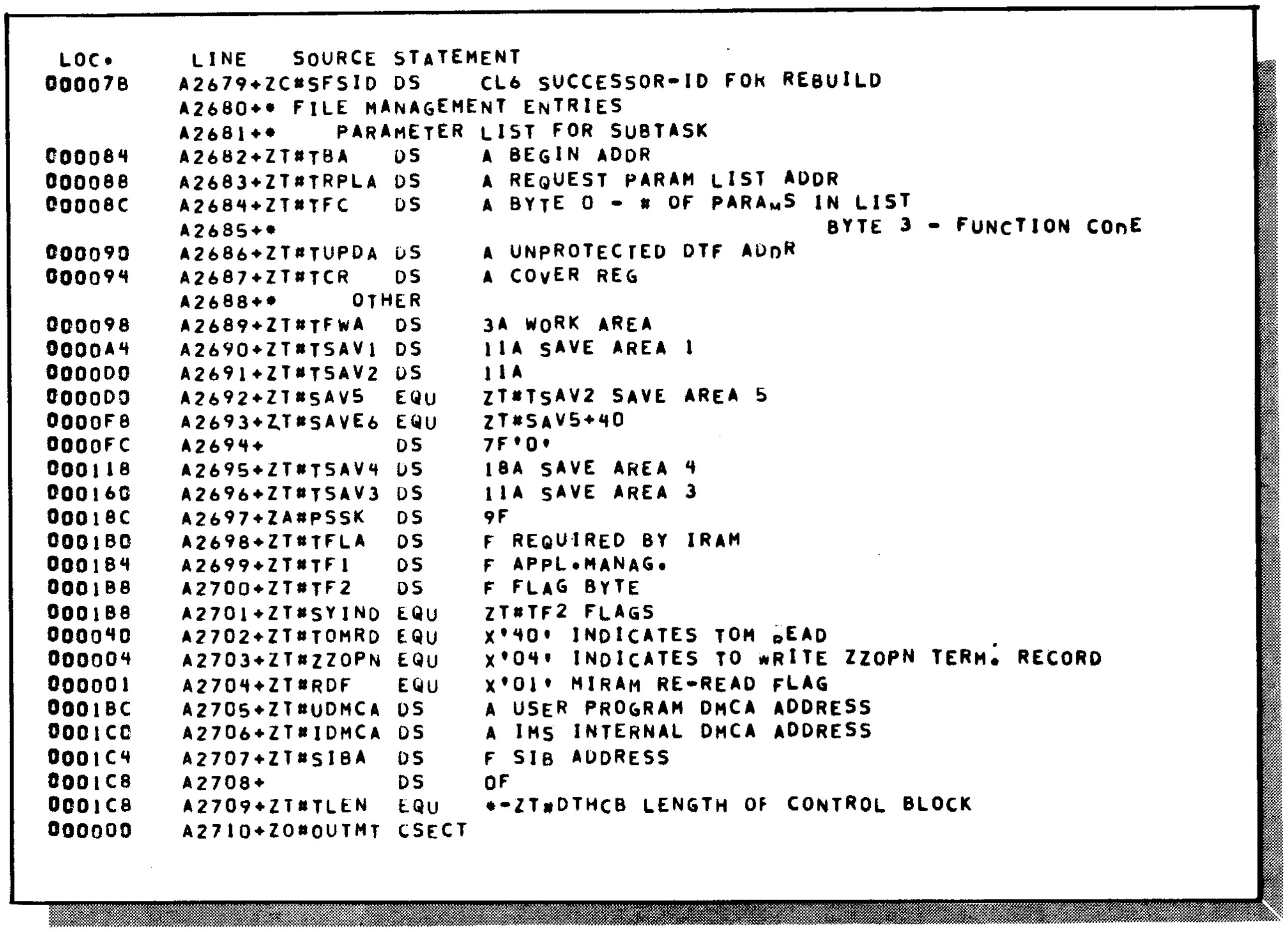


```
 LOC.     LINE    SOURCE STATEMENT
000010   80134+ZB#IMSNA EQU   X'10' IMS NOT ALLOWED ACCESS TO DMS
000008   80135+ZB#DMSNA EQU   X'08' DMS IS NOT THERE
000170   80136+ZB#FLG5  DS    XL1
000080   80137+ZB#KAT   EQU   X'80' KATAKANA CONFIGURED
000040   80138+ZB#STATS EQU   X'40' STATISTICS AT SHUTDOWN
000020   80139+ZB#SFSEN EQU   X'20' SFS ENABLED
000008   80140+ZB#GLB   EQU   X'08' GLOBAL NETWORK
000004   80141+ZB#DED   EQU   X'04' DEDICATED NETWORK
000171   80142+         DS    XL3 UNUSED
000174   80143+ZB#LPCT  DS    F LAST PCT ADDRESS
000178   80144+ZB#LACT  DS    F LAST ACT ADDRESS
00017C   80145+ZB#LAD   DS    F LAST LOAD AREA ADDRESS
000180   80146+ZB#NLST  DS    H INTLIST=N VALUE
000182   80147+         DS    XL2 UNUSED
000184   80148+ZC#CCA   DS    F CCA NAME
000188   80149+ZC#LOCAP DS    F LOCAP NAME
00018C   80150+ZB#MDICE DS    F DICE-SCREEN CLEAR/MSG POSITION
000190   80151+ZB#UNDEF DS    A POINTER TO TRIDT TO PROCESS UNDEF.TRANS.CODES
000194   80152+ZB#DATE  DS    F TODAY'S DATE
000198   80153+ZB#SESLN DS    F LENGTH-SESSION TABLE-ZSTAT
00019C   80154+ZQ#THFIN DS    OF . THIS TAG MUST STAY AT END
00019C   80155+ZT#HLEN  EQU   *-ZT#DTHCB LENGTH OF THCB
00019C   80156+ZT#TLEN  EQU   ZT#HLEN
000000   80157+ZC#IIP   CSECT
```

Figure 12-4. Single-Thread Thread Control Block (Part 4 of 4)

```
  LOC.     LINE    SOURCE STATEMENT

           2628           PRINT GEN
           2629           ZM#DTHCB
000000   A2630+ZT#DTHCB DSECT
000000   A2631+ZT#THQPT DS    F . NEXT THREAD IN QUEUE POINTER
000004   A2632+ZT#NTHCB DS    F . NEXT THREAD FOR SCHEDULING
000008   A2633+ZT#THURF DS    X . URGENT FLAG   0 - ROUTINE
000009   A2634+ZT#THRDF DS    X . THREAD READY FLAG  1 - READY
00000A   A2635+ZT#DWAIT DS    0X BIT 0 INITIAL THREAD WAIT FLAG - WAIT
00000A   A2636+ZT#REGRS DS    X BIT 7 RESTORE REGISTER FLAG  0 - YES
00000B   A2637+ZT#IECB3 DS    X BIT 0 CANCEL FLAG  1 - CANCEL
         A2638+*                        BIT 2 OUTPUT MESSAGE GENERATED BY ZG#MTMSO
         A2639+*                        BIT 3 INTERNAL CANCEL INITIATED
         A2640+*                        BIT 7 IECB FLAG     1 - 3WORD
00000C   A2641+ZT#THSVR DS    F . THREAD SAVE AREA REGISTER
000010   A2642+ZT#THRAD DS    F . THREAD RETURN ADDRESS
000014   A2643+ZT#TPIBA DS    A PROGRAM INFORMATION BLOCK ADDR
000018   A2644+ZT#TIMA  DS    A INPUT MESSAGE AREA ADDR
00001C   A2645+ZT#TWA   DS    A WORK AREA ADDR
000020   A2646+ZT#TOMA  DS    A OUTPUT MESSAGE AREA ADDR
000024   A2647+ZT#TCDA  DS    A CONTINUITY DATA AREA ADDR
000028   A2648+ZT#TDRMA DS    A DEFINED RECORD AREA ADDR
00002C   A2649+ZT#DDREC DS    A DATA DEFINITION RECORD ADDR
000030   A2650+ZT#SUBFL DS    A DEFINED FILE SUB-FILE DESC ADDR
000034   A2651+ZT#TFAM  DS    8F FILE ALLOCATION MAP
000020   A2652+ZT#TNUMF EQU   *-ZT#TFAM FILE ALLOCATION MAP LENGTH
000054   A2653+ZT#TATA  DS    A ACTION CONTROL TABLE RECORD ADDR
000058   A2654+ZT#TPTA  DS    A PROGRAM CONTROL TABLE RECORD ADDR
00005C   A2655+ZT#TPTA1 DS    F
000060   A2656+ZT#TTTA  DS    A TERMINAL CONTROL TABLE RECORD ADDR
000064   A2657+ZT#TIMB  DS    A INPUT MSG BUFFER ADDR
000068   A2658+ZT#TEDIT DS    A EDIT TABLE ADDR
00006C   A2659+ZT#TRID  DS    CL8 TRANSACTION ID
000074   A2660+ZT#TIND  DS    XL1 CONTROL INDICATORS
         A2661+*                        BIT 0      TERMINATION TYPE     0    NORMAL
         A2662+*                                                        1    ABNORMAL
         A2663+*                        BIT 2      ERROR RETURN         0    NO
         A2664+*                                                        1    YES
         A2665+*                        BIT 3-4    INTERNAL MESSAGE CONTROL:
         A2666+*                                   00   END ACTION OR END TRANSACTION
         A2667+*                                   01   EXPLICIT OUTPUT
         A2668+*                                   10   DELAYED INTERNAL SUCCESSION
         A2669+*                                   11   CANCELLED
         A2670+*                        BIT 5      INTERNAL REQUEST INDIC FOR FM
         A2671+*                                                        0    NO
         A2672+*                                                        1    YES
         A2673+*                        BIT 6  OUTPUT IN PROCESS
         A2674+*                        BIT 7  OUTPUT WAITED
000075   A2675+ZT#TER#  DS    X ERROR CODE NUMBER
000076   A2676+ZT#TES   DS    H RELATIVE ACT RECORD ADDR
000078   A2677+ZC#SFSSC DS    H INPUT STATUS BYTE COUNT
00007A   A2678+ZC#ITLN  DS    XL1 XTION FLD LEN CTR-INVALID TRANSACTION
```

Figure 12-5. Multithread Thread Control Block (Part 1 of 2)



```
 LOC.      LINE   SOURCE STATEMENT
000078    A2679+ZC#SFSID DS    CL6 SUCCESSOR-ID FOR REBUILD
          A2680+* FILE MANAGEMENT ENTRIES
          A2681+*     PARAMETER LIST FOR SUBTASK
000084    A2682+ZT#TBA   DS    A BEGIN ADDR
000088    A2683+ZT#TRPLA DS    A REQUEST PARAM LIST ADDR
00008C    A2684+ZT#TFC   DS    A BYTE 0 - # OF PARAMS IN LIST
          A2685+*                                    BYTE 3 - FUNCTION CODE
000090    A2686+ZT#TUPDA DS    A UNPROTECTED DTF ADDR
000094    A2687+ZT#TCR   DS    A COVER REG
          A2688+*       OTHER
000098    A2689+ZT#TFWA  DS    3A WORK AREA
0000A4    A2690+ZT#TSAV1 DS    11A SAVE AREA 1
0000D0    A2691+ZT#TSAV2 DS    11A
0000D0    A2692+ZT#SAV5  EQU   ZT#TSAV2 SAVE AREA 5
0000F8    A2693+ZT#SAVE6 EQU   ZT#SAV5+40
0000FC    A2694+         DS    7F'0'
000118    A2695+ZT#TSAV4 DS    18A SAVE AREA 4
000160    A2696+ZT#TSAV3 DS    11A SAVE AREA 3
00018C    A2697+ZA#PSSK  DS    9F
0001B0    A2698+ZT#TFLA  DS    F REQUIRED BY IRAM
0001B4    A2699+ZT#TF1   DS    F APPL.MANAG.
0001B8    A2700+ZT#TF2   DS    F FLAG BYTE
0001B8    A2701+ZT#SYIND EQU   ZT#TF2 FLAGS
000040    A2702+ZT#TOMRD EQU   X'40' INDICATES TOM READ
000004    A2703+ZT#ZZOPN EQU   X'04' INDICATES TO WRITE ZZOPN TERM. RECORD
000001    A2704+ZT#RDF   EQU   X'01' MIRAM RE-READ FLAG
0001BC    A2705+ZT#UDMCA DS    A USER PROGRAM DMCA ADDRESS
0001C0    A2706+ZT#IDMCA DS    A IMS INTERNAL DMCA ADDRESS
0001C4    A2707+ZT#SIBA  DS    F SIB ADDRESS
0001C8    A2708+         DS    0F
0001C8    A2709+ZT#TLEN  EQU   *-ZT#DTHCB LENGTH OF CONTROL BLOCK
000000    A2710+ZO#OUTMT CSECT
```

Figure 12-5. Multithread Thread Control Block (Part 2 of 2)

SINGLE AND MULTITHREAD TERMINAL CONTROL TABLE

```
LOC•        LINE   SOURCE STATEMENT

             2712          ZM#DTCT
000000      A2713+ZC#DTCT  DSECT  •••• TERMINAL CONTROL TABLE RECORD ••••
            A2714+•
000000      A2715+ZC#LINK  DS     F ACT LINK TO NEXT TCT IN QUEUE
000004      A2716+ZC#TID   DS     XL4 TERMINAL ID
000008      A2717+ZC#TAL   DS     F REL ADDR SOURCE TCT  (OS/3)
00000C      A2718+ZC#TALT  DS     F REL ADDR ALTERNATE TCT (OS/3)
000010      A2719+ZC#TTTA  DS     F CORRESPONDING TTT ADDRESS
000014      A2720+ZC#TESR  DS     F SUCC ACT REL ADDR - ROLLBACK
000018      A2721+ZC#TCOL  DS     H CONTINUITY DATA LENGTH
00001A      A2722+ZC#TLN   DS     XL1 LINE NUMBER
00001B      A2723+ZC#TTST  DS     XL7 STATUS BYTES
00001B      A2724+ZC#TST   EQU    ZC#TTST
            A2725+•
            A2726+•    EQUATES FOR ZC#TTST/ZC#TST
            A2727+•
000080      A2728+ZC#TTLST EQU    X'80' LAST TCT
000040      A2729+ZC#TTTMD EQU    X'40' TEST MODE
000020      A2730+ZC#TTUM  EQU    X'20' URGENT MESSAGE, ACTION
000010      A2731+ZC#TTDWN EQU    X'10' TERMINAL DOWN
000008      A2732+ZC#TTHLD EQU    X'08' HOLD TERMINAL
000004      A2733+ZC#TTUT  EQU    X'04' URGENT TERMINAL
000002      A2734+ZC#TMWR  EQU    X'02' MSG WAIT (FOR ZZTST) RECEIVED
000001      A2735+ZC#TMTC  EQU    X'01' MWRITE FOR ZZTST (SINLGE THREAD)
000001      A2736+ZC#TOMW  EQU    X'01' OUTSTANDING MWRITE (MULTI THREAD)
            A2737+•
00001C      A2738+ZC#TST1  EQU    ZC#TST+1,1
            A2739+•
            A2740+•    EQUATES FOR ZC#TST1
            A2741+•
000080      A2742+ZC#TTIM  EQU    X'80' INTERACTIVE MODE
000040      A2743+ZC#TTMT  EQU    X'40' MASTER TERMINAL
000020      A2744+ZC#TALTS EQU    X'20' ALTERNATE TERM SPECIFIED
000010      A2745+ZC#TTRC  EQU    X'10' ROLLBACK COMPLETE
000008      A2746+ZC#TTMWS EQU    X'08' IMS SENT MSG WAIT
000004      A2747+ZC#TTBTH EQU    X'04' BATCH TERMINAL
000002      A2748+ZC#TTRP  EQU    X'02' ROLLBACK IN PROCESS
000001      A2749+ZC#TTMS  EQU    X'01' MSG TO ORIG TERM SENT
            A2750+•
00001D      A2751+ZC#TST2  EQU    ZC#TST1+1,1
00001D      A2752+ZC#TPRSF EQU    ZC#TST2
            A2753+•
            A2754+•    EQUATES FOR ZC#TST2
            A2755+•
000080      A2756+ZC#TTUNS EQU    X'80' MWRITE ISSUED FROM ZO#UNSMT MODULE
000040      A2757+ZC#TTREL EQU    X'40' RELEASE BUFFER AT MWRITE COMPL
000020      A2758+ZC#TPRMQ EQU    X'20' MSG IN QUEUE
000010      A2759+ZC#TPRMP EQU    X'10' MSG IN PROCESS
000008      A2760+ZC#TTSTA EQU    X'08' SEND AUTO STATUS MESSAGE
000004      A2761+ZC#TCONT EQU    X'04' CONTINUOUS OUTPUT REQUESTED
000002      A2762+ZC#TDELN EQU    X'02' DEL NOTICE - ACTION TO BE SCHED
```

Figure 12–6. Single-Thread and Multithread Terminal Control Table (Part 1 of 5)

```
 LOC.          LINE   SOURCE STATEMENT
000001         A2763+ZC#TOIQ  EQU    X'01' OUTPUT GENERATED FOR INPUT QUEUING
               A2764+*
00001E         A2765+ZC#TST3  EQU    ZC#TST2+1,1
               A2766+*
               A2767+*     EQUATES FOR ZC#TST3
               A2768+*
000080         A2769+ZC#TTDR  EQU    X'80' DISCONNECT REQUESTED (S/T)
000040         A2770+ZC#TTQNE EQU    X'40' TERMINAL'S LOW QUEUE NOT EMPTY
000020         A2771+ZC#THDRS EQU    X'20' OUTPUT HEADER SAVED
000010         A2772+ZC#TIDN  EQU    X'10' INTERNAL DELIVERY NOTICE
000008         A2773+ZC#TIGM  EQU    X'08' IMS GENERATED ERROR MSG
000004         A2774+ZC#COIP  EQU    X'04' CONTINUOUS OUTPUT IN PROCESS (M/T)
000002         A2775+ZC#TNRDY EQU    X'02' NO IMS READY MSG TO THIS TERMINAL
000001         A2776+ZC#TUNAC EQU    X'01' SEND UNSOLICITED OUTPUT INDICATOR
               A2777+*                     FOR SWITCHED MESSAGES AT ACTION END
               A2778+*
00001F         A2779+ZC#TST4  EQU    ZC#TST3+1,1
               A2780+*
               A2781+*     EQUATES FOR ZC#TST4
               A2782+*
000080         A2783+ZC#ERMEX EQU    X'80' A/M GENERATED ERROR MSG.
000040         A2784+ZC#SFSRB EQU    X'40' REBUILD ALLOWED BY A/P
000020         A2785+ZC#ABTDY EQU    X'20' ABORT DYNAMIC SESSION
000010         A2786+ZC#DYTWD EQU    X'10' ABORT TERM WINDOW
000008         A2787+ZC#SIGN  EQU    X'08' SIGN ON FOR DYNAMIC SESSION
000004         A2788+ZC#ATTRI EQU    X'04' TERM HAS CONFIG. ATTRIBUTES
000002         A2789+ZC#CONSL EQU    X'02' CONSOLE TERMINAL
000001         A2790+ZC#CNTRD EQU    X'01' OUTSTANDING TCS/DISKETTE READ FUNCTION
               A2791+*
000020         A2792+ZC#TST5  EQU    ZC#TST4+1,1 DMS FLAGS
               A2793+*
               A2794+*     EQUATES FOR ZC#TST5
               A2795+*
000080         A2796+ZC#IMPRT EQU    X'80' ISSUED IMPACT FOR ACTION
000040         A2797+ZC#DEPND EQU    X'40' DEPART PENDING
000040         A2798+ZC#DEPRT EQU    X'40' ACTION ISSUED DEPART
000020         A2799+ZC#DMSUP EQU    X'20' ISSUED DSM OPEN FOR UPDATE
000020         A2800+ZC#BND   EQU    X'20' BOUND/UNBOUND STATE
000010         A2801+ZC#UBPND EQU    X'10' UNBIND PENDING
000008         A2802+ZC#DMSRO EQU    X'08' DMS FORCED DEPART WITH ROLLBACK
000004         A2803+ZC#DMSUB EQU    X'04' DMS RUN UNIT UNBOUND
000008         A2804+ZC#UPDRU EQU    X'08' OPENED FOR UPDATE IN THIS RUN-UNIT
000004         A2805+ZC#UPDTD EQU    X'04' UPDATING RUN-UNIT IN THIS SUCCESS UNIT
000002         A2806+ZC#TCALL EQU    X'02' FUNCTION CALL/TERMINATION CALL
000001         A2807+ZC#DMSDR EQU    X'01' DMS REQUEST VIA D.R.M.
               A2808+*
000021         A2809+ZC#TST6  EQU    ZC#TST5+1,1 DMS FLAGS EXTENSION
               A2810+*
               A2811+*     EQUATES FOR ZC#TST6
               A2812+*
000080         A2813+ZC#DMSER EQU    X'80' DMS ERROR IN RUN-UNIT
000040         A2814+ZC#WRK1  EQU    X'40' TEMPORARY FLAG #1
```

Figure 12-6. Single-Thread and Multithread Terminal Control Table (Part 2 of 5)

```
 LOC.      LINE    SOURCE STATEMENT
000020    A2815+ZC#WRK2  EQU    X'20' TEMPORARY FLAG #2
000010    A2816+ZC#TTMDF EQU    X'10' MOEFER ISSUED FOR THIS TERMINAL
          A2817+*   THE FOLLOWING STATUS  BYTE TAGS ARE NOT CLEARED WHEN A GLOBAL
          A2818+*   NETWORK DYNAMIC TERMINAL DOES A $$SOFF
          A2819+*          ZC#TTLST
          A2820+*          ZC#TTUT
          A2821+*          ZC#TTMT
          A2822+*          ZC#TNRDY
          A2823+*          ZC#TUNAC
          A2824+*          ZC#ATTRI
          A2825+*
          A2826+*
000022    A2827+ZC#DDPST DS     X DDP STATUS BYTE
          A2828+*
          A2829+*     EQUATES FOR ZC#DDPST
          A2830+*
000080    A2831+ZC#REMTR EQU    X'80' REMOTE TRANS
000040    A2832+ZC#FSOUT EQU    X'40' FIND SESSION OUTSTANDING
000020    A2833+ZC#PSEDO EQU    X'20' PSEUDO TCT
000010    A2834+ZC#DDPOT EQU    X'10' MWRITE FOR DDP
          A2835+*
000023    A2836+ZC#DDPMD DS     X DDP MODE
          A2837+*
          A2838+*     EQUATES FOR ZC#DDP MODE
          A2839+*
0000D9    A2840+ZC#DTR   EQU    C'R' DIRECTORY TRANS. ROUTING
0000C1    A2841+ZC#PTRA  EQU    C'A' PROGRAM   TRANS. ROUTING - ACTIVATE
0000C3    A2842+ZC#PTRC  EQU    C'C' PROGRAM   TRANS. ROUTING - ABORT/CANCEL
0000C5    A2843+ZC#PTRE  EQU    C'E' PROGRAM   TRANS. ROUTING - END
          A2844+*
000024    A2845+ZC#SFLAG DS     XL1 GENERAL SFS FLAG  BYTE
          A2846+*
          A2847+*     EQUATES FOR ZC#SFLAG
          A2848+*
000080    A2849+ZC#INFMT EQU    X'80' INPUT FORMAT
000040    A2850+ZC#DYNM  EQU    X'40' DYNAMIC MEMORY
000020    A2851+ZC#SFBT1 EQU    X'20' SFS FLAG 1
000010    A2852+ZC#ITCF  EQU    X'10' INVALID XTION
000008    A2853+ZC#SFBT2 EQU    X'08' SFS FLAG 2
          A2854+*
000025    A2855+ZC#SFIRC DS     XL1 SFS INPUT RETRY COUNT
          A2856+*
000026    A2857+         DS     XL2 UNUSED
000028    A2858+ZC#TRCTA DS     A TRCT ADDR
00002C    A2859+ZC#TQE   DS     F CANCEL LINK
000030    A2860+ZC#PRFT  DS     F DISPL TO PROCESS FILE TABLE
000034    A2861+ZC#PQCNT DS     H PROCESS QUEUE COUNT
000036    A2862+ZC#MQCNT DS     XL1 LAST ICAM SVC
000037    A2863+ZC#TDELS DS     XL1 DELIVERY NOTICE STATUS
000038    A2864+ZC#LGCNT DS     H LOW QUEUE COUNT
00003A    A2865+ZC#TIN   DS     H TOTAL INPUT COUNT
00003C    A2866+ZC#TINT  DS     H TRANS. INPUT COUNT
```

Figure 12–6. Single-Thread and Multithread Terminal Control Table (Part 3 of 5)

```
 LOC.      LINE   SOURCE STATEMENT
00003E   A2867+ZC#TTCM  DS    H TERM COMMAND COUNT
000040   A2868+ZC#TINCH DS    F TOTAL NO. INPUT  CHARS.
000044   A2869+ZC#TOTCH DS    F TOTAL NO. OUTPUT  CHARS.
000048   A2870+ZC#TOC   DS    H TOTAL OUTPUT COUNT
00004A   A2871+ZC#TOMSZ DS    H SOURCE TERM O/P MSG. SIZE
00004C   A2872+ZC#TON   DS    F TIMER LINK
000050   A2873+ZC#IML   DS    H INPUT MESSAGE LENGTH
000052   A2874+ZC#OML   DS    H OUTPUT MESSAGE LENGTH
000054   A2875+ZC#TML   DS    H TIMER MESSAGE LENGTH (OS/3 M.T.)
         A2876+*   OS/3 S.T. USES ZC#COSEQ INSTEAD OF ZC#TML
000054   A2877+ZC#COSEQ EQU   ZC#TML C/O SEQ COUNT (OS/3 S.T. ONLY)
000056   A2878+ZC#DML   DS    H DDP MSG. LENGTH
000058   A2879+ZC#IBF   DS    A INPUT  BUFFER ADDR
00005C   A2880+ZC#OBF   DS    A OUTPUT BUFFER ADDR
000060   A2881+ZC#TBF   DS    A TIMER  BUFFER ADDR
000064   A2882+ZC#DBF   DS    A DDP    BUFFER ADDR
000068   A2883+ZC#DPREL DS    A DDP BUFFER RELEASE ADDR
00006C   A2884+ZC#TDELC DS    XL4 USER CONTINUOUS OUTPUT CODE
000070   A2885+ZC#SFSTC DS    A SFS TERMINAL CLASS ENTRY ADDR
000074   A2886+ZC#SFSFN DS    CL8 SFS FORMAT NAME
00007C   A2887+ZC#SESAD DS    A SESSION STAT TABLE ADDR
000080   A2888+ZC#SESID DS    F SESSION ID
000084   A2889+ZC#TDMEM DS    F SFS DYNAMIC MEMORY ADDR
000088   A2890+ZC#TTRID DS    CL8 TRANS ID (INITIAL DATE/TIME)
000088   A2891+ZC#TRID  EQU   ZC#TTRID OS/4 TAG
000090   A2892+ZC#DLCNT DS    H IMC DEADLOCK DETECTION COUNT
000092   A2893+         DS    H UNUSED
000094   A2894+ZC#TCB   DS    A THREAD CONTROL BLOCK ADDR
000098   A2895+ZC#TLI   DS    8F TRANS LOCK INDICATOR
0000B8   A2896+ZC#TAUM  DS    8F AUDITED UPDATE MAP
         A2897+*** ZC#TLI AND ZC#TAUM MUST AGREE WITH ZT#TNUMF IN THE THCB
0000D8   A2898+ZC#TTEXT DS    CL8 TRANSLATED TERM CMD/TRANS CODE
0000D8   A2899+ZC#TCODE EQU   ZC#TTEXT OS/4 TAG
0000E0   A2900+ZC#TDDRC DS    CL1 DDR NAME ID CHAR (HIGH BYTE = X'FD')
         A2901+*** THE ABOVE FIELD IS DEFINED IN OS/4 BUT NOT TAGGED
0000E1   A2902+ZC#TDDRN DS    CL7 DATA DEF REC NAME
0000E8   A2903+ZC#TDFN  DS    CL7 DEFINED FILE NAME
0000EF   A2904+         DS    X UNUSED
0000F0   A2905+ZC#TES   DS    F SUCC ACT RECORD RELATIVE ADDR
         A2906+*     MULTI-THREAD SYSTEMS USE ZC#ES & ZC#CDC IN PLACE OF ZC#TES
0000F0   A2907+         ORG   ZC#TES
0000F0   A2908+ZC#ES    DS    H SUCC ACT RECORD RELATIVE ADDR
0000F2   A2909+ZC#CDL   DS    H CONTINUITY DATA LENGTH
         A2910+*
0000F4   A2911+ZC#WAI   DS    H WORK AREA INC
0000F6   A2912+ZC#CDI   DS    H CONTINUITY DATA AREA INC
0000F8   A2913+ZC#TTTN  DS    XL1 TCT RECORD NUMBER
0000F9   A2914+         DS    XL1 UNUSED
0000FA   A2915+         DS    H UNUSED
         A2916+*     MULTI-THREAD USES ZC#CDR & ZC#CES INSTEAD OF ZC#TTTN & ZC#TINT
0000F8   A2917+         ORG   ZC#TTTN
0000F8   A2918+ZC#CDR   DS    H TCT RECORD NUMBER
```

Figure 12-6. Single-Thread and Multithread Terminal Control Table (Part 4 of 5)

```
 LOC.       LINE    SOURCE STATEMENT
0000FA     A2919+ZC#CES    DS     H SUCC ACT REL ADDR _ ROLLBACK
0000FC     A2920+ZC#SCFR   DS     XL4 COUNT FIELD FOR ROLLBACK
           A2921+*
000100     A2922+ZC#TTIR   DS     XL1 TERM IND FOR ACTION PROG USING ROLLBACK
000100     A2923+ZC#TIR    EQU    ZC#TTIR OS/4 TAG
000100     A2924+          ORG    ZC#TIR
000100     A2925+ZC#TRWA   DS     F TRACE WORK AREA
000104     A2926+ZC#FBPA   DS     H * FIRST BLOCK OF PARTITION
000106     A2927+ZC#CBPA   DS     H * CURRENTLY ACCESSED BLOCK
000108     A2928+ZC#LBPA   DS     H * LAST BLOCK OF PARTITION
00010A     A2929+ZC#NRBCB  DS     H *# OF REM.BYTES IN CURR. BLOCK
           A2930+*
00010C     A2931+ZC#TLNAM  DS     CL4 LINE NAME
000110     A2932+ZC#TCHAR  DS     CL4 TERMINAL CHARACTERISTICS
000110     A2933+ZC#TTSL   EQU    ZC#TCHAR SCREEN LENGTH
000111     A2934+ZC#TTSW   EQU    ZC#TTSL+1 SCREEN WIDTH
000112     A2935+ZC#TTTYP  EQU    ZC#TTSW+1 TERMINAL TYPE
           A2936+*
           A2937+*    EQUATES FOR ZC#TTTYP
           A2938+*
000000     A2939+ZC#TTNFC  EQU    X'00' U100/U200/UTS10/TTY
000080     A2940+ZC#TT4PR  EQU    X'80' UTS400 PR
000040     A2941+ZC#TT4U2  EQU    X'40' UTS400 CP (U2 MODE)
000020     A2942+ZC#TT4U4  EQU    X'20' UTS400 CP (U4 MODE) OR UTS400
000010     A2943+ZC#TT327  EQU    X'10' IBM 3271
000008     A2944+ZC#TTU40  EQU    X'08' UTS40
000004     A2945+ZC#TTU20  EQU    X'04' UTS20
000002     A2946+ZC#TT40T  EQU    X'02' UTS400 TEXT EDITOR
           A2947+*
000113     A2948+ZC#TTATT  EQU    ZC#TTTYP+1 TERMINAL ATTRIBUTES
           A2949+*
           A2950+*    EQUATES FOR ZC#TTATT
           A2951+*
000080     A2952+ZC#TTKAN  EQU    X'80' KATAKANA
000040     A2953+ZC#TTNVI  EQU    X'40' NON-VIDEO
000020     A2954+ZC#TTSBT  EQU    X'20' SCREEN BYPASS
000010     A2955+ZC#TTPKT  EQU    X'10' PACKET PDN TERMINAL
000008     A2956+ZC#TTCST  EQU    X'08' CIRCUIT SWITCH PDN TERMINAL
000004     A2957+ZC#TTCCT  EQU    X'04' TERMINAL ON CLUSTER CONTROLLER
           A2958+*
000114     A2959+ZC#TINER  DS     F SFS ERROR FIELD
000118     A2960+ZC#TRIDA  DS     A PTR TO TRIDT ENTRY FOR CURRENT TRANSACTION
00011C     A2961+ZC#ALTID  DS     F ALTERNATE TERM ID
000120     A2962+ZC#TFIN   DS     0F THIS MUST ALWAYS BE AT END
000120     A2963+ZC#TLEN   EQU    *-ZC#DTCT
000000     A2964+ZO#OUTMT  CSECT
```

Figure 12-6. Single-Thread and Multithread Terminal Control Table (Part 5 of 5)

12.5. SAMPLE DUMP ACTION PROGRAM (FIXSAM)

Figure 12-7 shows the sample COBOL action program FIXSAM. This program produces two types of snap dumps depending on values entered at the terminal.

How FIXSAM produces S termination and CALL SNAP dumps

When the operator enters transaction code F#03 followed by the value T (Figure 12-7), line 303), FIXSAM moves an S to the termination indicator to produce a termination snap. Figure 12-8 shows the S termination snap dump.

When the operator enters transaction code F#03 followed by the value Y (Figure 12-7), line 302), FIXSAM issues a CALL SNAP that dumps working storage, the program information block, input message area, output message area, work area, and continuity data area without terminating the program.

Abnormal termination dump

A third type of snap dump is produced if the program terminates abnormally. An abnormal termination snap caused by a program check is shown in Figure 12-10. This dump varies in only a few details from the S termination snap.

```
LINE NO.      SOURCE ENTRY

 00001          IDENTIFICATION DIVISION.
 00002          PROGRAM-ID. FIXSAM.
 00003          ENVIRONMENT DIVISION.
 00004          CONFIGURATION SECTION.
 00005          SOURCE-COMPUTER. UNIVAC-OS3.
 00006          OBJECT-COMPUTER. UNIVAC-OS3.
 00007          DATA DIVISION.
 00008          WORKING-STORAGE SECTION.
 00009          01  DICE-CODES.
 00010         *
 00011         *     SET CURSOR-COORD TO HOME X'10030000'.
 00012               05  CURS-HME                  PIC X(4)    VALUE '    '.
 00013         *
 00014         *     POSITION CURSOR TO A NEW LINE X'10040000'.
 00015               05  NXT-LNE                   PIC X(4)    VALUE '    '.
 00016         *
 00017         *     SKIP 3 LINES AND BEGINNING OF LINE X'10040300'.
 00018               05  SKP-3LN                   PIC X(4)    VALUE '    '.
 00019         *
 00020         *     SKIP 2 LINES AND BEGINNING OF LINE X'10040200'.
 00021               05  SKP-2LN                   PIC X(4)    VALUE '    '.
 00022         *
 00023         *     START OF ENTRY CHARACTER  X'1E'.
 00024               05  SOE-CHAR                  PIC X(1)    VALUE ' '.
 00025         *
 00026          01   NON-NUMB-MSG                  PIC X(49)   VALUE
 00027                   'NON-NUMERIC VALUE ENTERED FOR READS DESIRED FIELD'.
 00028         *
 00029          01   TRANS-CAN-MSG                 PIC X(40)   VALUE
 00030                   'TRANSACTION CANCELLED DUE TO ABOVE ERROR'.
 00031         *
 00032          01   EOF-MSG                       PIC X(40)   VALUE
 00033                   'END OF FILE REACHED DURING READ NUMBER  '.
 00034         *
 00035          01   ERR-MSG                       PIC X(40)   VALUE
 00036                   'ERROR FROM SAM-GET  DURING READ NUMBER  '.
 00037         *
 00038          01   STAT-HDRS                     PIC X(47)   VALUE
 00039                   'STATUS-CODE                  DETAILED STATUS CODE  '.
 00040         *
 00041          01   FSAMTIN                       PIC X(7)    VALUE 'FSMTFIL'.
 00042         *
 00043          01   FSAMDIN                       PIC X(7)    VALUE 'FSMDFIL'.
 00044         *
 00045          01   SUCC-MSG                      PIC X(54)   VALUE
 00046                   'ENTER NUMBER OF READS FOR SAM VAR LENGTH FILES AS F#NN'.
 00047         *
 00048          01   DISCONNECT-MSG                PIC X(25)   VALUE
 00049                   'LINE DISCONNECT REQUESTED'.
 00050          01   HDG-LNE.
```

Figure 12-7. Sample Action Program (FIXSAM) Generating Snap Dumps (Part 1 of 7)

```
LINE NO.      SOURCE ENTRY

  00051              05 HD1                      PIC X(8)     VALUE 'NO. READ'.
  00052              05 FILLER                   PIC X(4)     VALUE SPACES.
  00053              05 HD2                      PIC X(7)     VALUE 'CUST-ID'.
  00054              05 FILLER                   PIC X(8)     VALUE SPACES.
  00055              05 HD3                      PIC X(13)  VALUE 'CUSTOMER NAME'.
  00056              05 FILLER                   PIC X(9)     VALUE SPACES.
  00057              05 HD4                      PIC X(8)     VALUE 'AMT PAID'.
  00058              05 FILLER                   PIC X(6)     VALUE SPACES.
  00059              05 HD5                      PIC X(4)     VALUE 'DATE'.
  00060              05 FILLER                   PIC X(5) VALUE SPACES.
  00061         *
  00062          01  SNP-ERR-MSG                 PIC X(42)    VALUE
  00063                  'ERROR ON SNAP NO. 1  2  3  4  5              '.
  00064          01  END-WS                      PIC X  VALUE '*'.
  00065          LINKAGE SECTION.
  00066          01  PIB. COPY PIB74.
  00067              02  STATUS-CODE               PIC 9(4) COMP-4.
  00068              02  DETAILED-STATUS-CODE      PIC 9(4) COMP-4.
  00069              02  RECORD-TYPE REDEFINES DETAILED-STATUS-CODE.
  00070                  03 PREDICTED-RECORD-TYPE  PIC X.
  00071                  03 DELIVERED-RECORD-TYPE  PIC X.
  00072              02  SUCCESSOR-ID              PIC X(6).
  00073              02  TERMINATION-INDICATOR     PIC X.
  00074              02  LOCK-ROLLBACK-INDICATOR   PIC X.
  00075              02  TRANSACTION-ID.
  00076                  03 YEAR                   PIC 9(4) COMP-4.
  00077                  03 TODAY                  PIC 9(4) COMP-4.
  00078                  03 HR-MIN-SEC             PIC 9(9) COMP-4.
  00079              02  DATA-DEF-REC-NAME         PIC X(7).
  00080              02  DEFINED-FILE-NAME         PIC X(7).
  00081              02  STANDARD-MSG-LINE-LENGTH  PIC 9(4) COMP-4.
  00082              02  STANDARD-MSG-NUMBER-LINES PIC 9(4) COMP-4.
  00083              02  WORK-AREA-LENGTH          PIC 9(4) COMP-4.
  00084              02  CONTINUITY-DATA-INPUT-LENGTH  PIC 9(4) COMP-4.
  00085              02  CONTINUITY-DATA-OUTPUT-LENGTH  PIC 9(4) COMP-4.
  00086              02  WORK-AREA-INC             PIC 9(4) COMP-4.
  00087              02  CONTINUITY-DATA-AREA-INC  PIC 9(4) COMP-4.
  00088              02  SUCCESS-UNIT-ID.
  00089                  03 TRANSACTION-DATE.
  00090                     04 YEAR                PIC 99.
  00091                     04 MONTH               PIC 99.
  00092                     04 TODAY               PIC 99.
  00093                  03 TIME-OF-DAY.
  00094                     04 HOUR                PIC 99.
  00095                     04 MINUTE              PIC 99.
  00096                     04 SECOND              PIC 99.
  00097                  03 FILLER                 PIC XXX.
  00098              02  SOURCE-TERMINAL-CHARS.
  00099                  03 SOURCE-TERMINAL-TYPE  PIC X.
  00100                  03 SOURCE-TERM-MSG-LINE-LENGTH    PIC 9(4) COMP-4.
  00101                  03 SOURCE-TERM-MSG-NUMBER-LINES   PIC 9(4) COMP-4.
  00102                  03 SOURCE-TERM-ATTRIBUTES PIC X.
```

Figure 12-7. Sample Action Program (FIXSAM) Generating Snap Dumps (Part 2 of 7)

```
LINE NO.      SOURCE ENTRY

                          02  DDP-MODE                      PIC X.
                      01  IMA. COPY IMA74.
                          02 SOURCE-TERMINAL-ID                 PIC X(4).
                          02 DATE-TIME-STAMP.
                             03 YEAR                            PIC 9(4) COMP-4.
                             03 TODAY                           PIC 9(4) COMP-4.
                             03 HR-MIN-SEC                      PIC 9(9) COMP-4.
                          02 TEXT-LENGTH                        PIC 9(4) COMP-4.
                          02 AUXILIARY-DEV-ID.
                             03 FILLER                          PIC X.
                             03 AUX-DEV-NO                      PIC X.
                          02  TRANS                     PIC X(2).
                          02  RECTORD                   PIC X(2).
                          02  NORECS REDEFINES RECTORD     PIC 99.
                          02  FILLER                    PIC X.
                          02  DISCONNECT                PIC X.
                          02  FILLER                    PIC X.
                          02  SNAP                      PIC X.
                          02  FILLER                    PIC X.
                          02  EXT-SUCC                  PIC X.
                          02  END-IMA                   PIC X.
                      01  CDA.
                          02  DISCONNECT-SAV            PIC X.
                          02  SNAP-SAV                  PIC X.
                          02  END-CDA                   PIC X.
                      01  OMA. COPY OMA74.
                          02 DESTINATION-TERMINAL-ID       PIC X(4).
                          02 SFS-OPTIONS.
                             03 SFS-TYPE                   PIC X.
                             03 SFS-LOCATION               PIC X.
                          02 FILLER                        PIC X(2).
                          02 CONTINUOUS-OUTPUT-CODE        PIC X(4).
                          02 TEXT-LENGTH                   PIC 9(4)    COMP-4.
                          02 AUXILIARY-DEVICE-ID.
                             03 AUX-FUNCTION               PIC X.
                             03 AUX-DEVICE-NO              PIC X.
                          02 OUT-MSG.
                          03  DICE1                     PIC X(4).
                          03  LINE1.
                              05  FILLER                PIC X(15).
                              05  FILERD                PIC X(7).
                              05  FILLER                PIC X(8).
                              05  TSNP.
                                  10  FILLER            PIC X(19).
                                  10  SNP1              PIC X.
                                  10  FILLER            PIC X(2).
                                  10  SNP2              PIC X.
                                  10  FILLER            PIC X(2).
                                  10  SNP3              PIC X.
                                  10  FILLER            PIC X(2).
                                  10  SNP4              PIC X.
                                  10  FILLER            PIC X(2).
```

Figure 12-7. Sample Action Program (FIXSAM) Generating Snap Dumps (Part 3 of 7)

```
SOURCE ENTRY

       10  SNP5         PIC X.
       10  FILLER       PIC X(10).
   03  DICE2            PIC X(4).
   03  LINE2            PIC X(72).
   03  DICE3            PIC X(4).
   03  LINE3            PIC X(72).
   03  DICE7            PIC X(4).
   03  LINE7.
     05  FILLER         PIC X(15).
     05  FILREAD        PIC X(7).
     05  FILLER         PIC X(50).
   03  DICE8            PIC X(4).
   03  LINE8            PIC X(72).
   03  DICE9            PIC X(4).
   03  LINE9            PIC X(72).
   03  DICE11           PIC X(4).
   03  LINE11           PIC X(72).
   03  DICE12           PIC X(4).
   03  SOE-DICE         PIC X.
   03  END-OMA          PIC X.
 01  WORK-AREA.
   03  REC-IO-AREA-F.
     05  CUST-ID        PIC 9(5).
     05  CUST-NAME      PIC X(20).
     05  AMT-PAID       PIC 9(5)V99.
     05  DATE-PD.
       10  MTH          PIC 9(2).
       10  SLSH-1       PIC X.
       10  DAYC          PIC 9(2).
       10  SLSH-2       PIC X.
       10  YR           PIC 9(2).
     05  FILLER         PIC X(9).
   03  DETAIL-LNE.
     05  FILLER         PIC X(3).
     05  RECS-RD        PIC 9(2).
     05  FILLER         PIC X(8).
     05  CUST-ID        PIC 9(5).
     05  FILLER         PIC X(6).
     05  CUST-NAME      PIC X(20).
     05  FILLER         PIC X(4).
     05  AMT-PAID       PIC $(6).99.
     05  FILLER         PIC X(4).
     05  DATE-PD.
       10  MTH          PIC 9(2).
       10  SLSH-1       PIC X.
       10  DAYC          PIC 9(2).
       10  SLSH-2       PIC X.
       10  YR           PIC 9(2).
     05  FILLER         PIC X(3).
   03  ERR-LNE REDEFINES DETAIL-LNE.
     05  ERROR-BLD      PIC X(40).
     05  RECRD-ERR      PIC Z9.
```

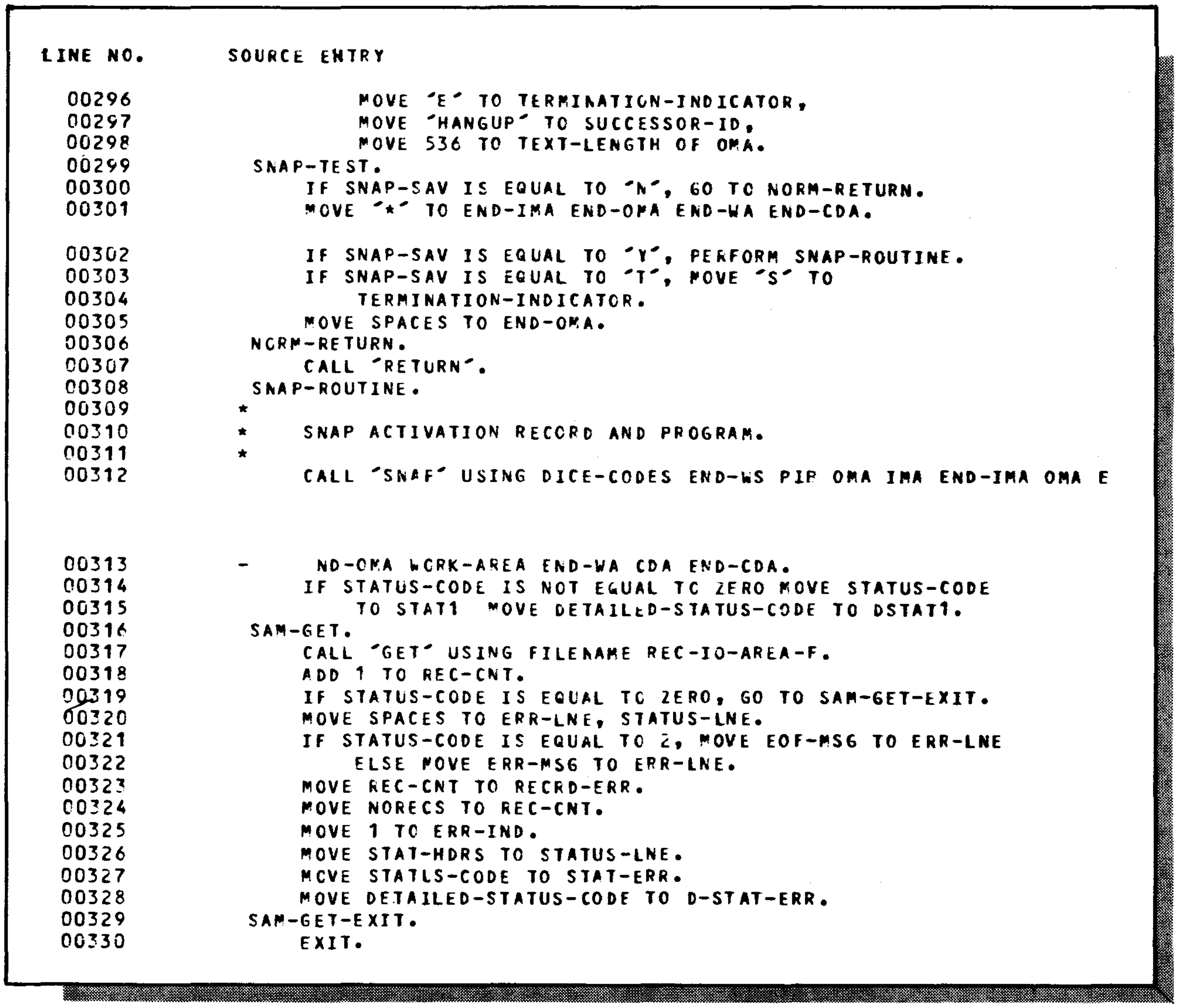
Figure 12-7. Sample Action Program (FIXSAM) Generating Snap Dumps (Part 4 of 7)

```
LINE NO.      SOURCE ENTRY

                         05  FILLER                 PIC X(30).
                     03  STATUS-LNE.
                         05  FILLER                 PIC X(13).
                         05  STAT-ERR               PIC 9(4).
                         05  FILLER                 PIC X(30).
                         05  D-STAT-ERR             PIC 9(4).
                         05  FILLER                 PIC X(21).
                     03  REC-CNT                    PIC 9(2).
                     03  ERR-IND                    PIC 9.
                     03  STAT1                      PIC 9(4).
                     03  DSTAT1                     PIC 9(4).
                     03  STAT2                      PIC 9(4).
                     03  DSTAT2                     PIC 9(4).
                     03  STAT3                      PIC 9(4).
                     03  DSTAT3                     PIC 9(4).
                     03  STAT4                      PIC 9(4).
                     03  DSTAT4                     PIC 9(4).
                     03  STAT5                      PIC 9(4).
                     03  DSTAT5                     PIC 9(4).
                     03  FILENAME                   PIC X(7).
                     03  END-WA                     PIC X.
                PROCEDURE DIVISION USING FIB IMA WORK-AREA OMA CDA.

                OPTIONS-SAVE.
                    MOVE CURS-HME TO DICE1.
                    MOVE NXT-LNE TO DICE2, DICE3.
                    IF SNAP IS EQUAL TO 'Y' OR 'N' OR 'T' MOVE SNAP TO SNAP-SAV
                        ELSE MOVE 'N' TO SNAP-SAV.
                    IF RECTORD IS NOT NUMERIC, MOVE NON-NUMB-MSG TO LINE2,
                        MOVE TRANS-CAN-MSG TO LINE3,
                        MOVE 232 TO TEXT-LENGTH OF OMA,
                        GO TO SNAP-TEST.
                    IF DISCONNECT IS EQUAL TO 'Y' MOVE DISCONNECT TO
                        DISCONNECT-SAV, ELSE MOVE 'N' TO DISCONNECT-SAV.
                TAPE-REC-GET.
                    MOVE ZERO TO ERR-IND, REC-CNT.
                    MOVE 'FILE NAME' TO LINE1, LINE7.
                    MOVE FSAMTIN TO FILENAME, FILERD.
                    IF NORECS IS EQUAL TO ZERO,
                        MOVE HDG-LNE TO LINE2
                        MOVE SPACES TO DETAIL-LNE,
                    MOVE NORECS TO RECS-RD,
                        MOVE DETAIL-LNE TO LINE3,
                        GO TO DISC-REC-GET.
                    MOVE SPACES TO DETAIL-LNE.
                    PERFORM SAM-GET THRU SAM-GET-EXIT UNTIL REC-CNT IS EQUAL TO

                        NORECS.
                    IF ERR-IND IS EQUAL TO ZERO,
                        MOVE CORRESPONDING REC-IO-AREA-F TO DETAIL-LNE,
```

Figure 12-7. Sample Action Program (FIXSAM) Generating Snap Dumps (Part 5 of 7)

```
LINE NO.      SOURCE ENTRY

 00255                 MOVE NORECS TO RECS-RD,
 00256                     MOVE HDG-LNE TO LINE2,
 00257                     MOVE DETAIL-LNE TO LINE3,
 00258                     GO TO DISC-REC-GET.
 00259                 MOVE ERR-LNE TO LINE2.
 00260                 MOVE STATUS-LNE TO LINE3.
 00261             DISC-REC-GET.
 00262                 MOVE ZERO TO ERR-IND, REC-CNT
 00263                 MOVE FSAMDIN TO FILENAME FILREAD.
 00264                 MOVE SKP-2LN TO DICE7.
 00265                 MOVE NXT-LNE TO DICE8, DICE9.
 00266                 IF NORECS IS EQUAL TO ZERO,
 00267                     MOVE HDG-LNE TO LINE8
 00268                 MOVE SPACES TO DETAIL-LNE,
 00269                     MOVE ZEROS TO RECS-RD,
 00270                     MOVE DETAIL-LNE TO LINE9,
 00271                     GO TO SUCC-TEST.
 00272                 MOVE SPACES TO DETAIL-LNE.
 00273                 PERFORM SAM-GET THRU SAM-GET-EXIT UNTIL REC-CNT IS EQUAL TO

 00274                     NORECS.
 00275                 IF ERR-IND IS EQUAL TO ZERO,
 00276                     MOVE CORRESPONDING REC-IO-AREA-F TO DETAIL-LNE,




 00277                     MOVE HDG-LNE TO LINE8,
 00278                     MOVE NORECS TO RECS-RD,
 00279                     MOVE DETAIL-LNE TO LINE9,
 00280                     GO TO SUCC-TEST.
 00281                 MOVE ERR-LNE TO LINE8.
 00282                 MOVE STATUS-LNE TO LINE9.
 00283             SUCC-TEST.
 00284                 MOVE SKP-2LN TO DICE11.
 00285                 IF EXT-SUCC IS NOT EQUAL TO 'N', MOVE 'E' TO
 00286                     TERMINATION-INDICATOR,
 00287                     MOVE 'SAMVIN' TO SUCCESSOR-ID,
 00288                     MOVE SUCC-MSG TO LINE11,
 00289                     MOVE NXT-LNE TO DICE12,
 00290                     MOVE SOE-CHAR TO SOE-DICE,
 00291                 MOVE 541 TO TEXT-LENGTH OF OMA,
 00292                     GO TO SNAP-TEST.
 00293                 MOVE 460 TO TEXT-LENGTH OF OMA.
 00294                 IF DISCONNECT IS EQUAL TO 'Y', MOVE DISCONNECT-MSG TO LINE11,
 00295                     MOVE 'C' TO AUX-FUNCTION,
```

Figure 12–7. Sample Action Program (FIXSAM) Generating Snap Dumps (Part 6 of 7)



```
SOURCE ENTRY

                MOVE 'E' TO TERMINATION-INDICATOR,
                MOVE 'HANGUP' TO SUCCESSOR-ID,
                MOVE 536 TO TEXT-LENGTH OF OMA.
        SNAP-TEST.
            IF SNAP-SAV IS EQUAL TO 'N', GO TO NORM-RETURN.
            MOVE '*' TO END-IMA END-OMA END-WA END-CDA.

            IF SNAP-SAV IS EQUAL TO 'Y', PERFORM SNAP-ROUTINE.
            IF SNAP-SAV IS EQUAL TO 'T', MOVE 'S' TO
                TERMINATION-INDICATOR.
            MOVE SPACES TO END-OMA.
        NORM-RETURN.
            CALL 'RETURN'.
        SNAP-ROUTINE.
       *
       *    SNAP ACTIVATION RECORD AND PROGRAM.
       *
            CALL 'SNAF' USING DICE-CODES END-WS PIP OMA IMA END-IMA OMA E



       -     ND-OMA WORK-AREA END-WA CDA END-CDA.
            IF STATUS-CODE IS NOT EQUAL TO ZERO MOVE STATUS-CODE
                TO STAT1  MOVE DETAILED-STATUS-CODE TO DSTAT1.
        SAM-GET.
            CALL 'GET' USING FILENAME REC-IO-AREA-F.
            ADD 1 TO REC-CNT.
            IF STATUS-CODE IS EQUAL TO ZERO, GO TO SAM-GET-EXIT.
            MOVE SPACES TO ERR-LNE, STATUS-LNE.
            IF STATUS-CODE IS EQUAL TO 2, MOVE EOF-MSG TO ERR-LNE
                ELSE MOVE ERR-MSG TO ERR-LNE.
            MOVE REC-CNT TO RECRD-ERR.
            MOVE NORECS TO REC-CNT.
            MOVE 1 TO ERR-IND.
            MOVE STAT-HDRS TO STATUS-LNE.
            MOVE STATUS-CODE TO STAT-ERR.
            MOVE DETAILED-STATUS-CODE TO D-STAT-ERR.
        SAM-GET-EXIT.
            EXIT.
```

Figure 12–7. Sample Action Program (FIXSAM) Generating Snap Dumps (Part 7 of 7)

12.6. ANALYZING THE TERMINATION SNAP DUMP

Allocation map addresses

The first area of the S termination dump to examine is the edited headers. These include the allocation map that contains the dump addresses of the main storage areas snapped.

The action name is SAMFIN and the action program load module processing that action is also SAMFIN. The term-id (terminal identification) for this transaction is TRMD. This is the way the terminal that initiated the transaction was defined in the communications network definition. The allocation map that follows contains the beginning and end locations as well as the lengths of user interface areas, and other areas included in the snap dump. The locations refer to relative addresses. Relative addresses are printed on the far left side of the snap dump. All addresses are given in hexadecimal.

No address given when area not used

By examining the directory in Figure 12-8 notice that there are no addresses given for action subprogram area. The reason for this is that action program SAMFIN did not call a subprogram.

Thread control block addresses

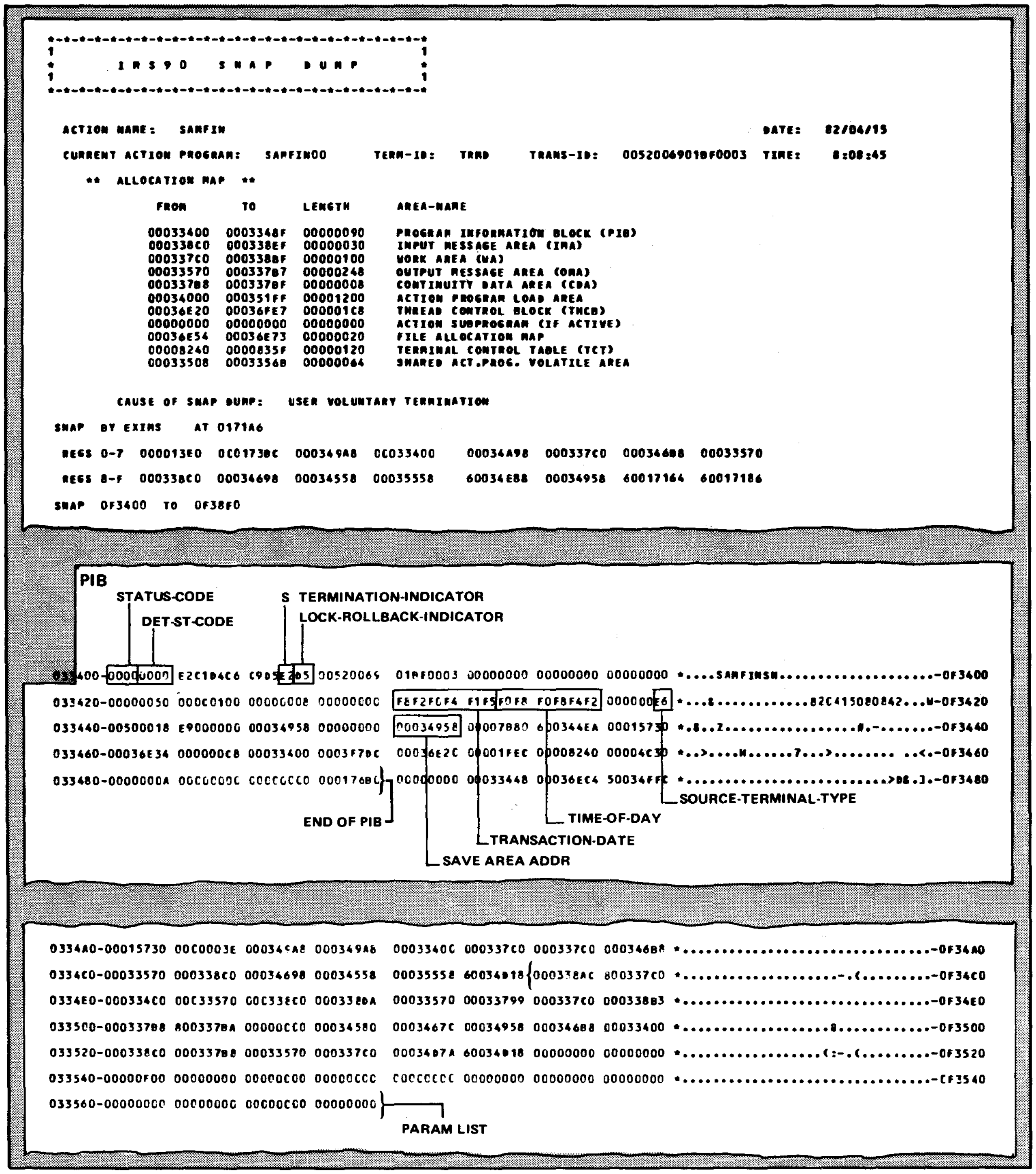
If you are not using an edited snap dump, that is, the snap contains no directory listing, it is still quite easy to locate all your action program's interface areas. Go directly to the thread control block. In this multithread example, it is at location 36E20 plus 15_{16} because the multithread layout begins at the twenty-first byte from the beginning thread control block address. (See Figure 12-8.) The first five full words (40 bytes) contain the relative addresses of the program information block, input message area, work area, output message area, and continuity data area, in that order.

Reason for snap

Following the allocation map on Figure 12-8 is the reason for the snap dump: USER VOLUNTARY TERMINATION. Voluntary termination resulted when the action program moved S to the termination indicator.

One set of registers

In the sample snap dump (Figure 12-8), the register section contains only one set of registers because the action program terminated voluntarily. These are IMS registers. To find SAMFIN's registers, you must go to relative location PIB + 48_{16} (address 33448). Beginning at that location, count three full words. The third word contains the full word address of SAMFIN's save area (34958). The save area contains the action program registers.

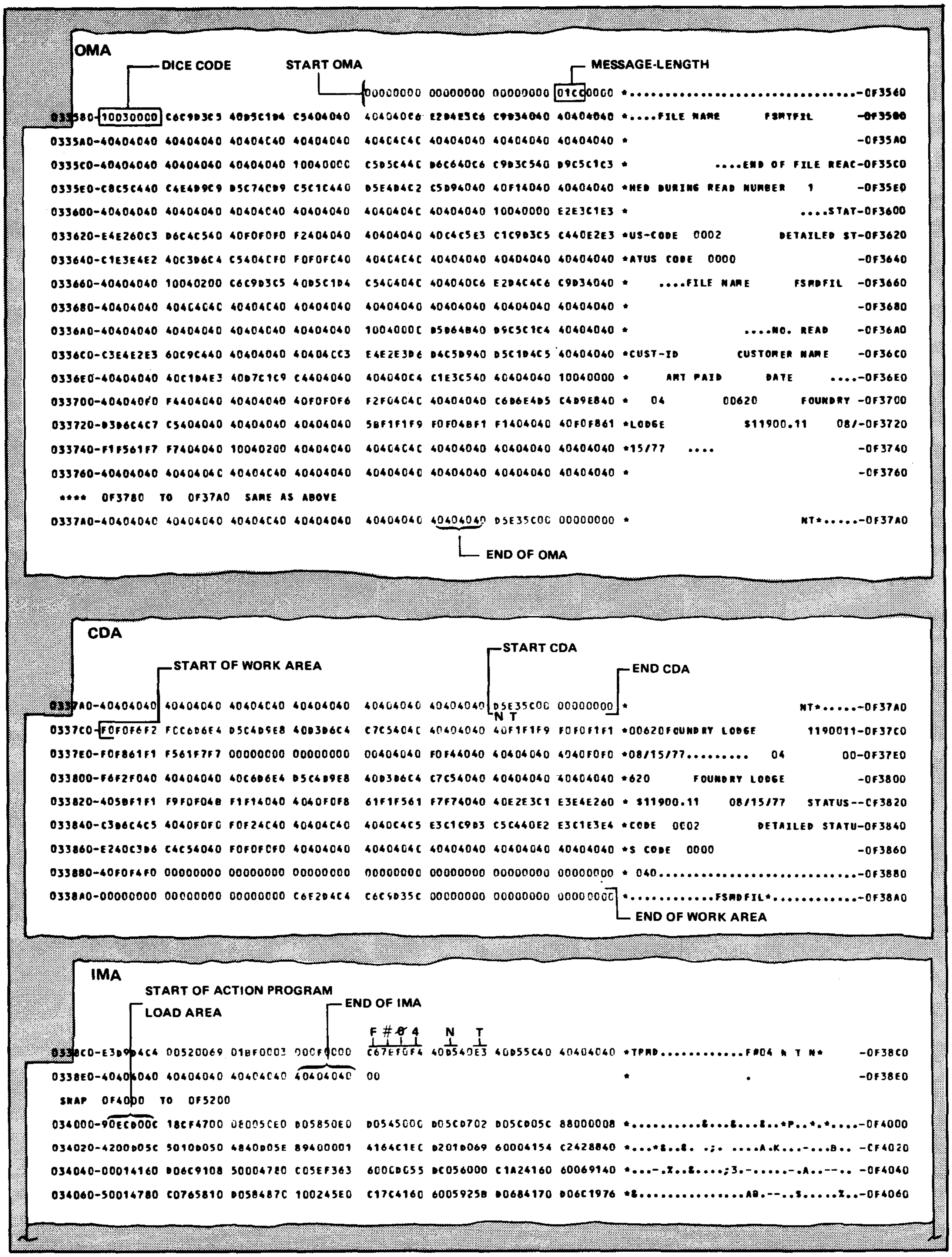


```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
1                                               1
*       I M S 9 0    S N A P    D U M P         *
1                                               1
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

  ACTION NAME:   SAMFIN                                                                   DATE:   82/04/15

  CURRENT ACTION PROGRAM:   SAMFIN00      TERM-ID:   TRMD     TRANS-ID:   00520069010F0003  TIME:    8:08:45

     **  ALLOCATION MAP  **

            FROM        TO        LENGTH      AREA-NAME

          00033400  0003348F  00000090    PROGRAM INFORMATION BLOCK (PIB)
          000338C0  000338EF  00000030    INPUT MESSAGE AREA (IMA)
          000337C0  000338BF  00000100    WORK AREA (WA)
          00033570  000337B7  00000248    OUTPUT MESSAGE AREA (OMA)
          000337B8  000337BF  00000008    CONTINUITY DATA AREA (CDA)
          00034000  000351FF  00001200    ACTION PROGRAM LOAD AREA
          00036E20  00036FE7  000001C8    THREAD CONTROL BLOCK (TMCB)
          00000000  00000000  00000000    ACTION SUBPROGRAM (IF ACTIVE)
          00036E54  00036E73  00000020    FILE ALLOCATION MAP
          00008240  0000835F  00000120    TERMINAL CONTROL TABLE (TCT)
          00033508  0003356B  00000064    SHARED ACT.PROG. VOLATILE AREA

       CAUSE OF SNAP DUMP:   USER VOLUNTARY TERMINATION

 SNAP  BY EXIMS    AT 0171A6

  REGS 0-7  000013E0  0C01730C  000349A8  0C033400   00034A98  000337C0  000346B8  00033570

  REGS 8-F  000338C0  00034698  00034558  00035558   60034E88  00034958  60017164  60017186

 SNAP  0F3400  TO  0F38F0
```

```
033400-00000000 E2C1D4C6 C9D5E2D5 00520069  010F0003 00000000 00000000 00000000 *....SAMFINSN......................-0F3400
033420-00000050 000C0100 0000C008 0000000C  F8F2F0F4 F1F5F0F8 F0F8F4F2 000000E6 *...8............82C415080842...W-0F3420
033440-00500018 E9000000 00034958 00000000  00034958 00007880 600344EA 00015730 *.&..Z................#.-.......-0F3440
033460-00036E34 000000C8 00033400 0003F7DC  00036E2C 00001FEC 00008240 00004C30 *..>....H......7...>.........<.-0F3460
033480-0000000A 00C0000C C0CCCCC0 000176BC  00000000 00033448 00036EC4 50034FFC *.........................>D&.].-0F3480
```

```
0334A0-00015730 00C0003E 000349A8 000349A6  0003340C 000337C0 000337C0 000346B8 *..............................-0F34A0
0334C0-00033570 000338C0 00034698 00034558  00035558 60034D18 000338AC 800337C0 *..................-.(.........-0F34C0
0334E0-000334C0 00C33570 00C338C0 000338DA  00033570 00033799 000337C0 000338B3 *..............................-0F34E0
033500-000337B8 800337BA 00000CC0 00034580  0003467C 00034958 000346B8 00033400 *..................8...........-0F3500
033520-000338C0 000337B8 00033570 000337C0  00034D7A 60034D18 00000000 00000000 *................(:-.(.........-0F3520
033540-00000F00 00000000 0000C000 00000CCC  C0CCCCCC 00000000 00000000 00000000 *..............................-CF3540
033560-00000000 0000000C 00C00C00 00000000
```

Figure 12-8. Termination Snap for SAMFIN Load Module (FIXSAM Action Program) (Part 1 of 3)

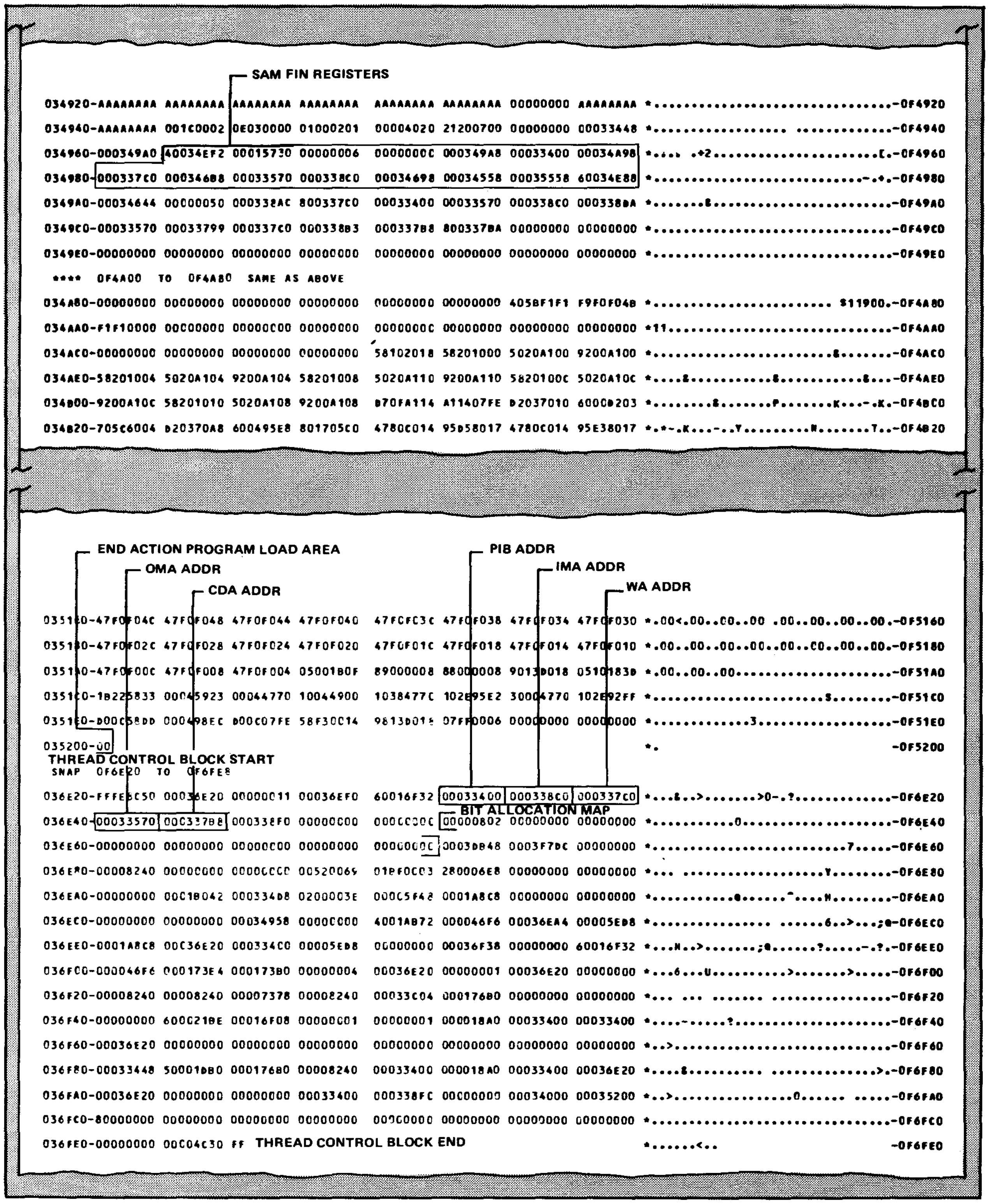

OMA
DICE CODE
START OMA
MESSAGE-LENGTH
END OF OMA
CDA
START OF WORK AREA
START CDA
END CDA
N T
END OF WORK AREA
IMA
START OF ACTION PROGRAM
LOAD AREA
END OF IMA
F # 0 4
N
T

```
                                    00000000 00000000 00000000 01CC0000 *...............................-0F3560
033580-10030000 C6C9D3C5 40D5C1D4 C5404040  404040C6 E2D4E3C6 C9D34040 40404040 *....FILE NAME      FSMTFIL       -0F3580
0335A0-40404040 40404040 40404C40 40404040  40404C4C 40404040 40404040 40404040 *                                -0F35A0
0335C0-40404040 40404040 40404040 1004000C  C5D5C44C D6C640C6 C9D3C540 D9C5C1C3 *                ....END OF FILE REAC-0F35C0
0335E0-C8C5C440 C4E4D9C9 D5C74CD9 C5C1C440  D5E4D4C2 C5D94040 40F14040 40404040 *NED DURING READ NUMBER   1       -0F35E0
033600-40404040 40404040 40404C40 40404040  4040404C 40404040 10040000 E2E3C1E3 *                            ....STAT-0F3600
033620-E4E260C3 D6C4C540 40F0F0F0 F2404040  40404040 40C4C5E3 C1C9D3C5 C440E2E3 *US-CODE  0002        DETAILED ST-0F3620
033640-C1E3E4E2 40C3D6C4 C5404CF0 F0F0FC40  404C4C4C 40404040 40404040 40404040 *ATUS CODE  0000                  -0F3640
033660-40404040 10040200 C6C9D3C5 40D5C1D4  C540404C 404040C6 E2D4C4C6 C9D34040 *    ....FILE NAME      FSMDFIL  -0F3660
033680-40404040 404C4C4C 40404C40 40404040  40404040 40404040 40404040 40404040 *                                -0F3680
0336A0-40404040 40404040 40404C40 40404040  1004000C D5D64B40 D9C5C1C4 40404040 *                    ....NO. READ   -0F36A0
0336C0-C3E4E2E3 60C9C440 40404040 40404CC3  E4E2E3D6 D4C5D940 D5C1D4C5 40404040 *CUST-ID         CUSTOMER NAME    -0F36C0
0336E0-40404040 40C1D4E3 40D7C1C9 C4404040  404040C4 C1E3C540 40404040 10040000 *     AMT PAID       DATE     ....-0F36E0
033700-404040F0 F4404040 40404040 40F0F0F6  F2F04C4C 40404040 C6D6E4D5 C4D9E840 *  04         00620      FOUNDRY -0F3700
033720-D3D6C4C7 C5404040 40404040 40404040  5BF1F1F9 F0F04BF1 F1404040 40F0F861 *LODGE           $11900.11    08/-0F3720
033740-F1F561F7 F7404040 10040200 40404040  40404C4C 40404040 40404040 40404040 *15/77    ....                    -0F3740
033760-40404040 4040404C 40404C40 40404040  40404040 40404040 40404040 40404040 *                                -0F3760
****  0F3780  TO  0F37A0  SAME AS ABOVE
0337A0-40404040 40404040 40404C40 40404040  40404040 40404040 D5E35C0C 00000000 *                         NT*.....-0F37A0
```

```
0337A0-40404040 40404040 40404C40 40404040  40404040 40404040 D5E35C0C 00000000 *                         NT*.....-0F37A0
0337C0-F0F0F6F2 FCC6D6E4 D5C4D9E8 40D3D6C4  C7C5404C 40404040 40F1F1F9 F0F0F1F1 *00620FOUNDRY LODGE         1190011-0F37C0
0337E0-F0F861F1 F561F7F7 00000000 00000000  00404040 F0F44040 40404040 4040F0F0 *08/15/77.........   04          00-0F37E0
033800-F6F2F040 40404040 40C6D6E4 D5C4D9E8  40D3D6C4 C7C54040 40404040 40404040 *620       FOUNDRY LODGE          -0F3800
033820-405BF1F1 F9F0F04B F1F14040 4040F0F8  61F1F561 F7F74040 40E2E3C1 E3E4E260 * $11900.11     08/15/77   STATUS--CF3820
033840-C3D6C4C5 4040F0F0 F0F24C40 40404C40  4040C4C5 E3C1C9D3 C5C440E2 E3C1E3E4 *CODE  0002        DETAILED STATU-0F3840
033860-E240C3D6 C4C54040 F0F0F0F0 40404040  4040404C 40404040 40404040 40404040 *S CODE  0000                     -0F3860
033880-40F0F4F0 00000000 00000000 00000000  00000000 00000000 00000000 00000000 * 040............................-0F3880
0338A0-00000000 00000000 00000000 C6F2D4C4  C6C9D35C 00000000 00000000 00000000 *............FSMDFIL*............-0F38A0
```

```
0338C0-E3D9D4C4 00520069 01BF0003 000F0000  C67EF0F4 40D540E3 40D55C40 40404040 *TPMD............F#04 N T N*     -0F38C0
0338E0-40404040 40404040 40404C40 40404040  00                                  *              .                 -0F38E0
 SNAP  0F4000  TO  0F5200
034000-90ECD00C 18CF4700 08005CE0 D05850E0  D054500C D05CD702 D05CD05C 88000008 *..........&...&...&..*P..*.*....-0F4000
034020-4200D05C 5010D050 4840D05E 89400001  4164C1EC D201D069 60004154 C2428840 *...*&..&. .;. ....A.K...-...B.. -CF4020
034040-00014160 D06C9108 50004780 C05EF363  600CDC55 DC056000 C1A24160 60069140 *...-.%..&....;3.-.....-.A..--.. -0F4040
034060-50014780 C0765810 D058487C 100245E0  C17C4160 6005925B D0684170 D06C1976 *&..............A@.--..$.....%..-0F4060
```

Figure 12-8. Termination Snap for SAMFIN Load Module (FIXSAM Action Program) (Part 2 of 3)


SAM FIN REGISTERS
END ACTION PROGRAM LOAD AREA
OMA ADDR
CDA ADDR
PIB ADDR
IMA ADDR
WA ADDR
THREAD CONTROL BLOCK START
BIT ALLOCATION MAP
THREAD CONTROL BLOCK END

```
034920-AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA  AAAAAAAA AAAAAAAA 00000000 AAAAAAAA *................................-0F4920
034940-AAAAAAAA 001C0002 0E030000 01000201  00004020 21200700 00000000 00033448 *................ ...............-0F4940
034960-000349A0 40034EF2 00015730 00000006  0000000C 000349A8 00033400 00034A98 *.... .+2.....................[.-0F4960
034980-000337C0 000346B8 00033570 000338C0  00034698 00034558 00035558 60034E88 *..........................-.+.-0F4980
0349A0-00034644 00000050 000332AC 800337C0  00033400 00033570 000338C0 000338BA *.......&.......................-0F49A0
0349C0-00033570 00033799 000337C0 000338B3  000337B8 800337BA 00000000 00000000 *................................-0F49C0
0349E0-00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................-0F49E0
 ****  0F4A00  TO  0F4A80  SAME AS ABOVE
034A80-00000000 00000000 00000000 00000000  00000000 00000000 405BF1F1 F9F0F04B *...................... $11900.-0F4A80
034AA0-F1F10000 00000000 00000C00 00000000  0000000C 00000000 00000000 00000000 *11..............................-0F4AA0
034AC0-00000000 00000000 00000000 00000000  58102018 58201000 5020A100 9200A100 *.......................&.......-0F4AC0
034AE0-58201004 5020A104 9200A104 58201008  5020A110 9200A110 5820100C 5020A10C *....&...........&...........&...-0F4AE0
034B00-9200A10C 58201010 5020A108 9200A108  B70FA114 A11407FE B2037010 6000B203 *........&.......P.......K...-.K.-0F4BC0
034B20-705C6004 B20370A8 600495E8 801705C0  4780C014 95B58017 4780C014 95E38017 *.*-.K...-..Y.........N.......T..-0F4B20

035160-47F0F04C 47F0F048 47F0F044 47F0F040  47FCFC3C 47F0F038 47F0F034 47F0F030 *.00<.00..C0..00 .00..00..00..00.-0F5160
035180-47F0F02C 47F0F028 47F0F024 47F0F020  47F0F01C 47F0F018 47F0F014 47F0F010 *.00..00..00..00..00..C0..00..00.-0F5180
0351A0-47F0F00C 47F0F008 47F0F004 05001B0F  89000008 88000008 9013B018 0510183B *.00..00..00.....................-0F51A0
0351C0-1B225833 00045923 00044770 10044900  1038477C 102E95E2 30004770 102E92FF *.....................S.........-0F51C0
0351E0-B00C58DD 000498EC B00C07FE 58F30C14  98130015 07FF0006 00000000 00000000 *.............3.................-0F51E0
035200-00                                                                     *.                              -0F5200
 SNAP   0F6E20   TO   0F6FE8
036E20-FFFEBC50 00036E20 00000011 00036EF0  60016F32 00033400 000338C0 000337C0 *...&..>.......>0-.?...........-0F6E20
036E40-00033570 000337B8 000338F0 00000000  0000C00C 00000802 00000000 00000000 *...........0......................-0F6E40
036E60-00000000 00000000 00000C00 00000000  0000000C 0003D848 0003F7DC 00000000 *.........................7.....-0F6E60
036E80-00008240 00000000 0000CCCC 00520069  01BF0C03 280006E8 00000000 00000000 *... ...................Y........-0F6E80
036EA0-00000000 00C1B042 000334D8 0200003E  000C5F42 0001A8C8 00000000 00000000 *............@......"....H........-0F6EA0
036EC0-00000000 00000000 00034958 0000C000  4001AB72 000046F6 00036EA4 00005ED8 *............... ......6..>...;Q-0F6EC0
036EE0-0001A8C8 00C36E20 000334C0 00005ED8  00000000 00036F38 00000000 60016F32 *...H..>.......;Q......?.....-.?.-0F6EE0
036F00-000046F6 000173E4 000173B0 00000004  00036E20 00000001 00036E20 00000000 *...6...U..........>.......>.....-0F6F00
036F20-00008240 00008240 00007378 00008240  00033C04 000176B0 00000000 00000000 *... ... ...... ...............-0F6F20
036F40-00000000 6000218E 00016F08 00000001  00000001 000018A0 00033400 00033400 *....-.....?.....................-0F6F40
036F60-00036E20 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *..>.............................-0F6F60
036F80-00033448 50001DB0 000176B0 00008240  00033400 000018A0 00033400 00036E20 *....&.......... ..............>.-0F6F80
036FA0-00036E20 00000000 00000000 00033400  000338FC 00000000 00034000 00035200 *..>...............0...... .....-0F6FA0
036FC0-80000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................-0F6FC0
036FE0-00000000 00C04C30 FF                                                   *......<..                      -0F6FE0
```

Figure 12-8. Termination Snap for SAMFIN Load Module (FIXSAM Action Program) (Part 3 of 3)

SAVE area

In Figure 12-8, the save address is 34958. Once you locate this address, which is in the action program load area, advance three full words (C_{16}). At location 34964 you will find your action program's registers 14, 15, and 0-12, in that order.

Finding Error Codes in the Program Information Block

Locating status codes

Looking at Figure 12-8, SAMFIN's program information block begins at address 33400. The first word (4 bytes) contains the STATUS-CODE and DETAILED-STATUS-CODE fields. IMS returns values to these fields indicating the result of action program function calls. If the function call is successful, these fields contain zeros. Figure 12-8 shows that the function call made to IMS was successful because both STATUS-CODE and DETAILED-STATUS-CODE fields indicate a successful function call.

If, for example, IMS returned a status code of 03 and a detailed status code of 0B, it would mean that the action program made an invalid request and that the file requested was not assigned to this action at IMS configuration. Then, to find out exactly which file is involved, you must consult the parameter list address in the thread control block. (See Figures 12-4 and 12-5.)

For a complete listing of the values IMS returns in the STATUS-CODE and DETAILED-STATUS-CODE fields, see Appendix D.

Finding Other Data in the Program Information Block

Locating TERMINATION-INDICATOR field

Still in the program information block at relative location PIB + A_{16} is the TERMINATION-INDICATOR field. If your action program moves an S to this field, this location contains an E2 for voluntary termination snap. The value in this and any other program information block field varies depending on the action program and whether the program terminated voluntarily or involuntarily.

Locating LOCK-ROLLBACK-INDICATOR field

Relative location PIB + B_{16} is the LOCK-ROLLBACK- INDICATOR field. It contains D5 (character N), which is the default value. The value N establishes a new rollback point in the audit file (before-images of records to be updated) and releases all locks for this transaction.

Locating other PIB fields

By comparing the program information block fields listed in Figure 3-2, to the program information block area of the snap dump, you can see exactly what values all these fields contained when the dump occurred. For your convenience, we have noted a few of these fields in Figure 12-8: transaction-date (82.04.15), time-of-day (08.08.42), and source-terminal-type (hexadecimal E6 or character W) indicating a local workstation.

All 90-character positions of the program information block are displayed. Remember, however, that only the first 71 positions are accessible to your action program.

Finding Error Causes in the Output Message Area

Using the allocation map in Figure 12-8, we see that the output message area begins at address 33570. This area contains the 16-byte control header and the output message generated by the action program.

Locating DESTINATION-TERMINAL-ID

The first three words of SAMFIN's output message area (Figure 12-8) including the DESTINATION-TERMINAL-ID and DATE-TIME-STAMP fields contain zeros indicating that the destination terminal is the same as the source terminal.

Locating MESSAGE-LENGTH field

Also, in the output message area at location 3357C or OMA + C_{16} is the 2-byte MESSAGE-LENGTH field. This field indicates the size of the output message to be generated (460 bytes).

Since SAMFIN does not use screen format services and is not a continuous output program, relative locations 3357E and 3357F, respectively, contain zeros.

Following the unused 2-byte AUXILIARY-DEVICE-ID field is the 4-byte DICE field containing the DICE sequence as the first four bytes of the output message text.

Finding Error Causes in the Input Message Area

Locating the input message

The input message area begins at relative address 338C0. Its contents include the input message area control header (16 bytes) and the input data entered by the terminal operator. The terminal input starts at IMA + 10_{16} or 338D0. The terminal operator entered the transaction code, F#04. He didn't wish to test the disconnect feature in this run, so he entered an N. Since he was interested in terminating voluntarily with a snap dump, he entered T in the next position. We've noted these fields to assist you in finding them in the snap dump (Figure 12-8).

Finding Error Causes in the Continuity Data Area

By looking in the allocation map, we find that SAMFIN's continuity data area begins in location 337B8. Here, we see the character D5 or N. This indicates that the value of N was entered at the terminal to indicate that the disconnect feature was not being tested on this run. The next byte indicates an E3 or T meaning that the voluntary termination was used.

Executed instructions

Finding these values tells us that our program executed the instruction which moved these values from the input area to the continuity data area. (See lines 232, 233, and 238-239 in FIXSAM's coding (Figure 12-7).)

Finding Error Causes in the Work Area

Finding executed instructions

Similarly, the work area begins at location 337C0. To find customer identification, name, amount paid, and date paid values in this area of the dump indicates that SAMFIN executed instructions that placed these values there. (See the GET function call (line 317, Figure 12-7) which actually moves these values from the disk or tape file to the work area.

Finding Error Causes in the Action Program Load Area

Now, let's turn our attention to the action program load area. This is by far the lengthiest section of the snap dump. Data contained in the thread control block is equally essential to interpreting the program area so we'll discuss these two areas at the same time.

Using the thread control block

The thread control block is at location 36E20. As we previously mentioned, it contains the addresses of all the interface areas and the action program load area. This data is valuable only if you're using an unedited dump. However, the thread control block does contain other information very useful to the IMS programmer.

Locating the file allocation map

Using the multithread DSECT shown in Figure 12-5, find the ZT#TFAM allocation map tag and its location. Add this value to the thread control block address. In our example at location 36E54, there are four full words (single thread) or eight full words (multithread) used for a file allocation bit map. To use this bit map, you must realize that four full words contain 128 bits and eight full words contain 256 bits. IMS uses these bits to indicate which specific files a user action program can access – one file per bit.

Bits set off

If bits are set to zero, the action program cannot access those files. Examining these locations can be very valuable in determining which files your action program was accessing during execution.

Bits set on

For example, if the high-order bit was on, the action program could access one file – the first file configured. If additional bits were on, additional files could be accessed. These bits are maintained in the same relative order as the actual files were configured.

Three DSECT labels locate and explain parameter list

Three labels from the multithread thread control block DSECT are sometimes helpful in debugging. Using the thread control block DSECT for multithread, Figure 12-5, find three labels:

ZT#TRPLA

ZT#TFC

ZT#TUPDA

In single-thread, the thread control block DSECT labels (Figure 12-4) are:

ZT#HRPLA

ZT#TFC

ZT#TUPDA

Locating the parameter list

To the left of the first label, ZT#TRPLA, find the address that is also the dump address of the parameter list that was passed for the function executed. In this case, the address of the parameter list was 334D8.

Determining the last function call

Next, find the ZT#TFC label representing an address in the dump. This address points to an area in the dump containing the number of parameters in the list and the hexadecimal code representing the last function call. You can go to this address and see the addresses of parameters that were passed. The last valid word in this list will contain a hexadecimal 80 in the first byte. Note that sometimes these function calls are issued by IMS and sometimes by your action program. For this reason, this data is not always useful in debugging.

Determining number of parameters in list

You can determine the number of parameters passed on the last function call by counting the number of words containing valid addresses.

Hexadecimal equivalents for function calls

Table 12-1 lists all the IMS function calls and their corresponding hexadecimal values for use in debugging your action program.

Table 12-1. Hexadecimal Equivalents for Function Calls

Hexadecimal	Function Call
06	RETURN
0A	SEND
26	ESETL
2A	SETL
2E	INSERT
32	DELETE
36	PUT
3A	GETUP
3E	GET
4A	SNAP
8E	SUBPROG
92	SETLOAD
96	GETLOAD
AA	SETK

Locating the DTF or CDIB

Finally, find the label ZT#TUPDA in the DSECT and obtain its address in the same way. This address points to the area in the dump containing the last DTF or CDIB referenced by the last function call executed. This address is not within the range of the user snap dump and is useful only when a job dump is available.

12.7. OTHER DEBUGGING RESOUCES

Using the link map

If your action programs are in COBOL, in addition to their compile and link, a link map is useful. Figure 12-9 shows the link map for action program, FIXSAM.

The link map shows which COBOL object modules are included in the load module. The object module, FIXSAM, is included in the load module, SAMFIN, as well as the IMS interface module, ZF#LINK.

```
UNIVAC SYSTEM OS/3  LINKAGE EDITOR
DATE- 82/04/15 TIME- 07.51

CONTROL STREAM ENCOUNTERED AND PROCESSED AS FOLLOWS-

              // PARAM OUT=LOADLIB
              /$
                          LOADM SAMFIN
                          INCLUDE FIXSAM
                          INCLUDE ZF#LINK,SYSOBJ
                          ENTER FIXSAM
              /*
C@@MSI  *AUTO-INCLUDED*
C@@NUM  *AUTO-INCLUDED*
C@@SME  *AUTO-INCLUDED*


                                              *DEFINITIONS DICTIONARY*

SYMBOL.    TYPE.  PHASE. ADDRESS.       SYMBOL.    TYPE.  PHASE. ADDRESS.       SYMBCL.    TYPE.  PHASE. ADDRESS.

ACTIVATE   ENTRY  ROOT   00C01100       ADDKY      ENTRY  ROOT   00001124       ARETURN    ENTRY  ROOT   00001168
BUILD      ENTRY  ROOT   C000110C       C@@MSI     CSECT  ROOT   00000000       C@@NUM     CSECT  ROOT   000002A8
C@@SME     CSECT  ROOT   C0C00468       CHTBL      ENTRY  ROOT   000010F8       CLOSE      ENTRY  ROOT   0000119C
CMDRB      ENTRY  ROOT   00CC1160       DELETE     ENTRY  ROOT   0000117C       DELKY      ENTRY  ROOT   00001128
DLADR      ENTRY  ROOT   C00C117C       DLKCP      ENTRY  ROOT   00001130       ENDCRL     ENTRY  ROOT   0000115C
ESETL      ENTRY  ROOT   CC001188       ESLMT      ENTRY  ROOT   00001188       FIND       ENTRY  ROOT   000011A0
FIXSAM     CSECT  ROOT   00C004E8       FREE       ENTRY  ROOT   0000118C       GET        ENTRY  ROOT   00001170
GETLOAD    ENTRY  ROOT   000C1118       GETUP      ENTRY  ROOT   00001174       GTADR      ENTRY  ROOT   00001180
INSERT     ENTRY  ROOT   000C1180       KESALM     ENTRY  ROOT   00000000       KESALP     ENTRY  ABS    000011F8
KESRES     ENTRY  ABS    0C0011F8       LNKCP      ENTRY  ROOT   0000112C       OPEN       ENTRY  ROOT   00001198
OPENF      ENTRY  ROOT   C0C01198       PUT        ENTRY  ROOT   00001178       RDID       ENTRY  ROOT   00001170
RDIDC      ENTRY  ROOT   000C11A4       RDIDCL     ENTRY  ROOT   000011A0       RDIDL      ENTRY  ROOT   00001174
RDKEY      ENTRY  ROOT   CCCC1134       RDKEYC     ENTRY  ROOT   00001110       RDKEYCL    ENTRY  ROOT   0000110C
RDKEYL     ENTRY  ROOT   CC001138       RDKYI      ENTRY  ROOT   0000113C       RDKYIC     ENTRY  ROOT   00001108
RDSQ       ENTRY  ROOT   0C001148       RDSQC      ENTRY  ROOT   0000119C       RDSQCL     ENTRY  ROOT   00001194
RDSQI      ENTRY  ROOT   C000114C       RDSQIC     ENTRY  ROOT   00001164       RDSQL      ENTRY  ROOT   0000118C
RDSR       ENTRY  ROOT   C0C01140       RDSRC      ENTRY  ROOT   00001190       RDSRCL     ENTRY  ROOT   00001168
RDSRL      ENTRY  ROOT   C0001144       REBUILD    ENTRY  ROOT   00001110       RELREC     ENTRY  ROOT   00001190
RETURN     ENTRY  ROOT   0C0011A8       RUN        ENTRY  ROOT   000010FC       SEND       ENTRY  ROOT   000011A4
SETK       ENTRY  ROOT   000C1104       SETL       ENTRY  ROOT   00001184       SETLOAD    ENTRY  ROOT   0000111C
SNAP       ENTRY  ROOT   000C1164       SSLOCK     ENTRY  ROOT   00001150       SSUNLK     ENTRY  ROOT   00001154
STCRL      ENTRY  ROOT   0C001158       STLMT      ENTRY  ROOT   00001184       SUB        ENTRY  ROOT   00001120
SUBPROG    ENTRY  ROOT   CC00112C       UNLCCK     ENTRY  ROOT   00001194       WRID       ENTRY  ROOT   00001178
XR3IMS     ENTRY  ROOT   000C1114       ZF#LINK    CSECT  ROOT   000010F8


                                               ** ALLOCATION MAP **

                        LOAD MODULE -   SAMFIN        SIZE -   000011F8

 PHASE NAME  TRANS ADDR    FLAG      LABEL     TYPE      ESID         LNK ORG     HIADDR     LENGTH       OBJ ORG
 SAMFIN00       NODE - ROOT                                           00000000    000011F7   000011F8
 *** START OF AUTO-INCLUDED ELEMENTS -
```

Figure 12-9. Link Map for FIXSAM Action Program (Part 1 of 2)

```
PHASE NAME  TRANS ADDR    FLAG      LABEL     TYPE       ESID        LNK ORG     HIADDR      LENGTH      OBJ ORG
      - 11/10/81 00.00 -            C@@MSI     OBJ
                                    C@@MSI    CSECT       01        00000000    000002A7    000002A8    00000000
      - 10/30/81 00.00 -            C@@NUM     OBJ
                                    C@@NUM    CSECT       01        000002A8    00000465    000001BE    00000000
      - 10/30/81 00.00 -            C@@SME     OBJ
                                    C@@SME    CSECT       01        00000468    000004E1    0000007A    00000000
*** END OF AUTO-INCLUDED ELEMENTS -
      - 82/04/15 07.50 -            FIXSAM     OBJ
                                    FIXSAM    CSECT       01        000004E8    000010F7    00000C10    00000000
      - 81/12/22 06.58 -            ZF#LINK    OBJ
                                    ZF#LINK   CSECT       01        000010F8    000011F7    00000100    00000000
                                    ACTIVATE   ENTRY      01        00001100                            00000008
                                    SETK       ENTRY      01        00001104                            0000000C
                                    CHTBL      ENTRY      01        000010F8                            00000000
                                    RUN        ENTRY      01        000010FC                            00000004
                                    XR3IMS     ENTRY      01        00001114                            0000001C
                                    BUILD      ENTRY      01        0000110C                            00000014
                                    REBUILD    ENTRY      01        00001110                            00000018
                                    GET        ENTRY      01        00001170                            00000078
                                    GETUP      ENTRY      01        00001174                            0000007C
                                    PUT        ENTRY      01        00001178                            00000080
                                    DELETE     ENTRY      01        0000117C                            00000084
                                    INSERT     ENTRY      01        00001180                            00000088
                                    SETL       ENTRY      01        00001184                            0000008C
                                    ESETL      ENTRY      01        00001188                            00000090
                                    FREE       ENTRY      01        0000118C                            00000094
                                    RELREC     ENTRY      01        00001190                            00000098
                                    UNLOCK     ENTRY      01        00001194                            0000009C
                                    OPEN       ENTRY      01        00001198                            000000A0
                                    CLOSE      ENTRY      01        0000119C                            000000A4
                                    FIND       ENTRY      01        000011A0                            000000A8
                                    SEND       ENTRY      01        000011A4                            000000AC
                                    RETURN     ENTRY      01        000011A8                            000000B0
                                    ARETURN    ENTRY      01        00001168                            00000070
                                    SNAP       ENTRY      01        00001164                            0000006C
                                    SUB        ENTRY      01        00001120                            00000028
                                    RDSQL      ENTRY      01        0000118C                            00000094
                                    RDIDC      ENTRY      01        000011A4                            000000AC
                                    RDIDCL     ENTRY      01        000011A0                            000000A8
                                    RDSQC      ENTRY      01        0000119C                            000000A4
                                    RDSQCL     ENTRY      01        00001194                            0000009C
                                    RDSRC      ENTRY      01        00001190                            00000098
                                    RDSRCL     ENTRY      01        00001168                            00000070
                                    RDSQIC     ENTRY      01        00001164                            0000006C
                                    RDKEYC     ENTRY      01        00001110                            00000018
                                    RDKEYCL    ENTRY      01        0000110C                            00000014
                                    RDKYIC     ENTRY      01        00001108                            00000010
                                    GTADR      ENTRY      01        00001180                            00000088
                                    DLADR      ENTRY      01        0000117C                            00000084
                                    ADDKY      ENTRY      01        00001124                            0000002C
                                    DELKY      ENTRY      01        00001128                            00000030
                                    LNKCP      ENTRY      01        0000112C                            00000034
                                    DLKCP      ENTRY      01        00001130                            00000038
                                    WRID       ENTRY      01        00001178                            00000080
                                    RDID       ENTRY      01        00001170                            00000078


PHASE NAME  TRANS ADDR    FLAG      LABEL     TYPE       ESID        LNK ORG     HIADDR      LENGTH      OBJ ORG
                                    RDIDL      ENTRY      01        00001174                            0000007C
                                    RDKEY      ENTRY      01        00001134                            0000003C
                                    RDKEYL     ENTRY      01        00001138                            00000040
                                    RDKYI      ENTRY      01        0000113C                            00000044
                                    RDSR       ENTRY      01        00001140                            00000048
                                    RDSRL      ENTRY      01        00001144                            0000004C
                                    RDSQ       ENTRY      01        00001148                            00000050
                                    RDSQI      ENTRY      01        0000114C                            00000054
                                    STLMT      ENTRY      01        00001184                            0000008C
                                    ESLMT      ENTRY      01        00001188                            00000090
                                    SSLOCK     ENTRY      01        00001150                            00000058
                                    SSUNLK     ENTRY      01        00001154                            0000005C
                                    STCRL      ENTRY      01        00001158                            00000060
                                    ENDCRL     ENTRY      01        0000115C                            00000064
                                    CMDRB      ENTRY      01        00001160                            00000068
                                    OPENF      ENTRY      01        00001198                            000000A0
                                    SUBPROG    ENTRY      01        00001120                            00000028
                                    SETLOAD    ENTRY      01        0000111C                            00000024
                                    GETLOAD    ENTRY      01        00001118                            00000020
             000004E8


                                                   FLAG CODES -
B - BLK DATA CSECT     D - AUTO-DELETED        E - EXCLUSIVE 'A' REF   G - GENERATED EXTRN     I - INCLUSIVE 'V' REF
L - DEFERRED LENGTH    M - MULTIPLY DEFINED    N - NOT INCLUDED        P - PROMOTED COMMON     R - SHARED REC PRODUCED
S - SHARED ITEM        U - UNDEFINED REF       V - VCON ITEM
*ANY OTHER CODES REPRESENT PROCESS ERRORS*


LINK EDIT OF 'SAMFIN'   COMPLETED
DATE- 82/04/15 TIME- 07.53
ERRORS ENCOUNTERED- 0000  UPSI- X'00'
```

Figure 12–9. Link Map for FIXSAM Action Program (Part 2 of 2)

12.8. ANALYZING AN ABNORMAL TERMINATION SNAP DUMP

Abnormal snap dump example

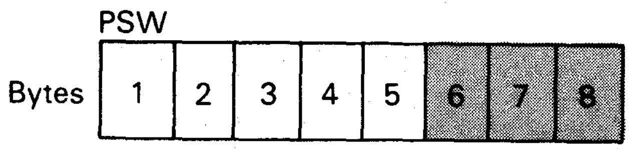
Figure 12-10 shows the dump generated when action program SAMFIN terminates abnormally due to a program check error. This program check occurred because of an invalid instruction code.

Importance of program status word

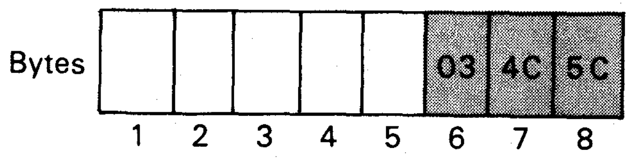
All of the debugging techniques discussed for S termination snaps pertain to abnormal snap dumps except for information about the save area. In addition, the program status word plays an inportant part in determining the cause of an abnormal termination dump.

Locating erroneous instruction address

To find the address of the erroneous instruction, you must first go to the sixth, seventh, and eighth bytes of the program status word.



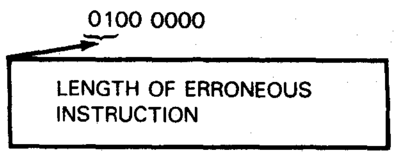
In Figure 12-10, after the allocation map, the address in these bytes is 034C5C.



This is the address of the instruction immediately following the erroneous instruction. You go to address 034C5C and count back one instruction. The next question is: How long is the erroneous instruction? so you know how many bytes to count back from this address.

Interpreting error codes

Once you locate the next sequential instruction after the erroneous one, look at the program-status word in byte 5. The first 4 bits of this byte contain the instruction length code and condition code. You are interested in the two high-order (leftmost) bits of byte 5. Looking at the program status word (Figure 12-10), notice that byte 5 contains 40_{16}. In binary this is:

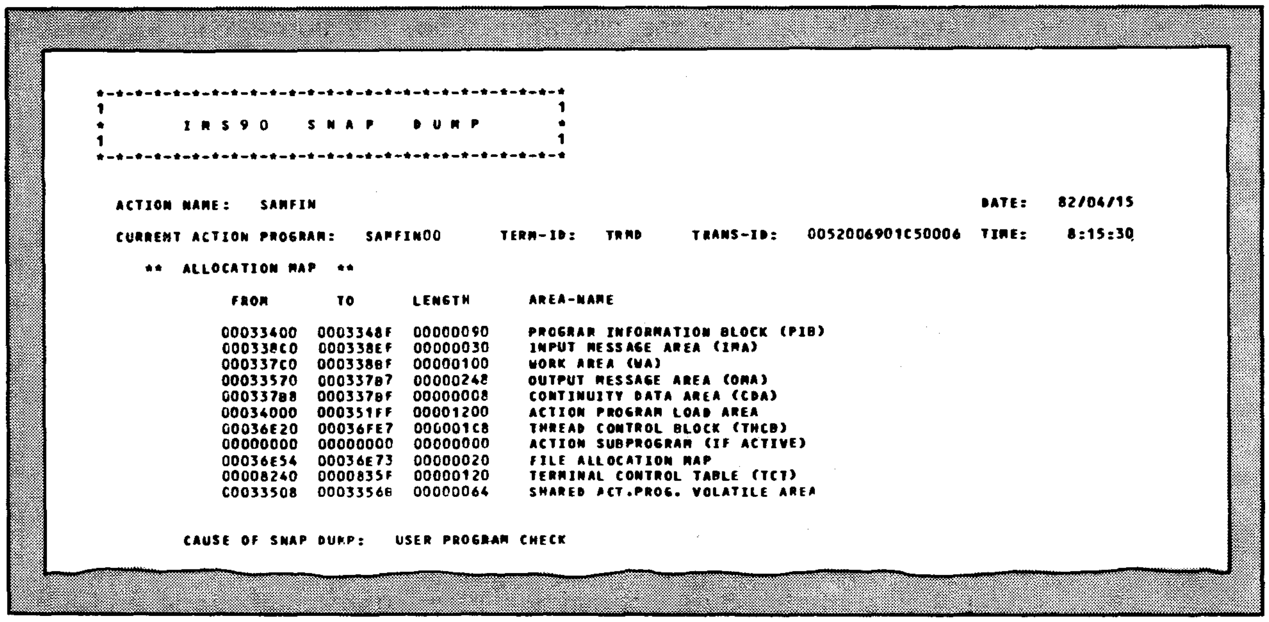


The two high-order bits can have one of the following binary configurations indicating a 2-, 4-, or 6-byte erroneous instruction.

Bit Configuration	Interpretation
01	2-byte instruction
10	4-byte instruction
11	6-byte instruction

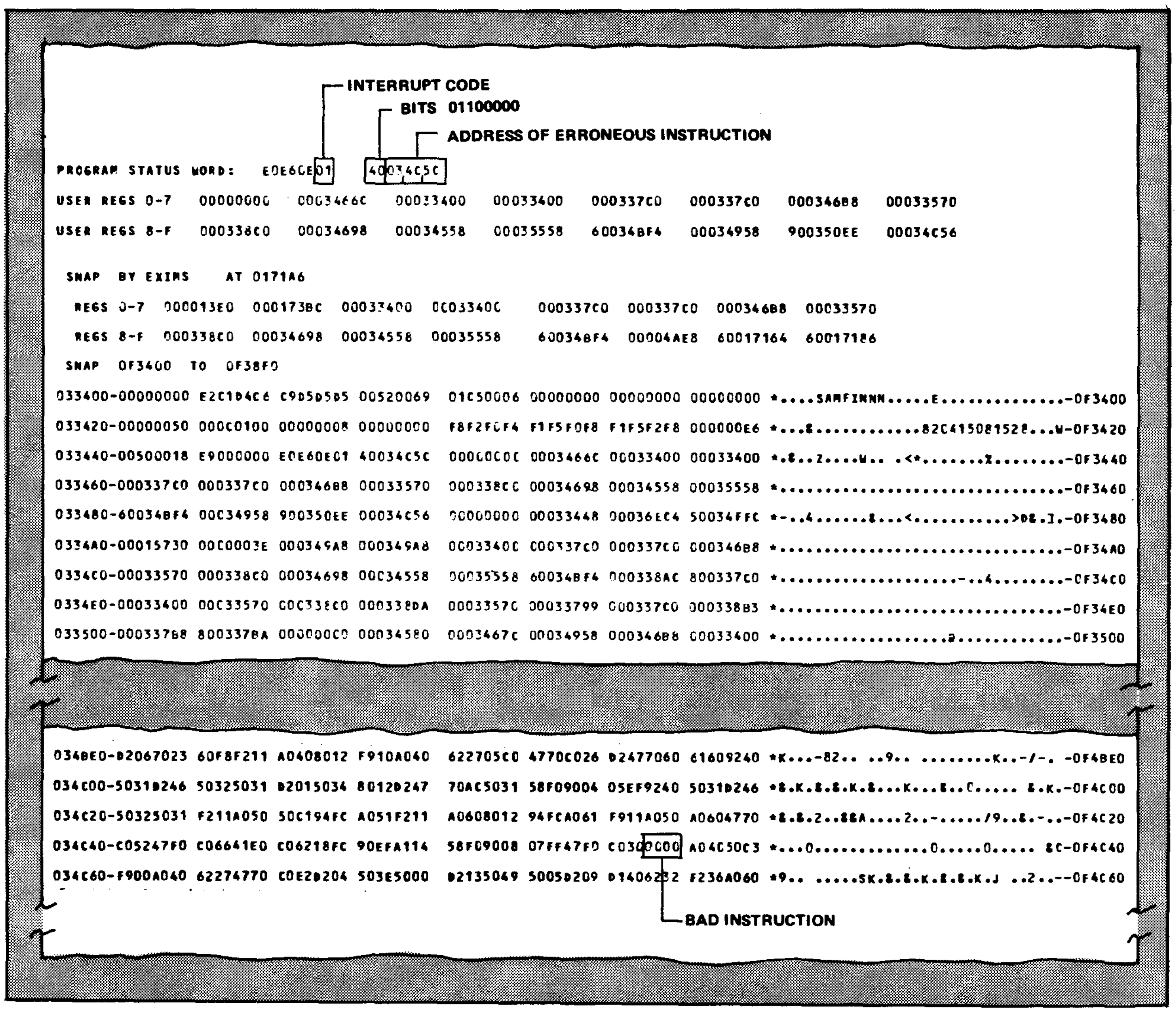
SAMFIN's erroneous instruction has a bit configuration of 01, meaning it is a 2-byte instruction. Counting back from location 034C5C, two bytes show an instruction containing zeros.

Now you go to byte 4 of the program status word to obtain the interrupt code. The interrupt code is 01_{16}, an operation exception. This means that an illegal operation was attempted.



```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
1                                               1
*       I M S 9 0    S N A P    D U M P         *
1                                               1
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

ACTION NAME:   SAMFIN                                                              DATE:    82/04/15

CURRENT ACTION PROGRAM:   SAMFIN00     TERM-ID:   TRMD     TRANS-ID:   0052006901C50006  TIME:     8:15:30

    **  ALLOCATION MAP  **

          FROM       TO       LENGTH       AREA-NAME

        00033400  0003348F  00000090    PROGRAM INFORMATION BLOCK (PIB)
        000338C0  000338EF  00000030    INPUT MESSAGE AREA (IMA)
        000337C0  000338BF  00000100    WORK AREA (WA)
        00033570  000337B7  00000248    OUTPUT MESSAGE AREA (OMA)
        000337B8  000337BF  00000008    CONTINUITY DATA AREA (CDA)
        00034000  000351FF  00001200    ACTION PROGRAM LOAD AREA
        00036E20  00036FE7  000001C8    THREAD CONTROL BLOCK (THCB)
        00000000  00000000  00000000    ACTION SUBPROGRAM (IF ACTIVE)
        00036E54  00036E73  00000020    FILE ALLOCATION MAP
        00008240  0000835F  00000120    TERMINAL CONTROL TABLE (TCT)
        00033508  0003356B  00000064    SHARED ACT.PROG. VOLATILE AREA

      CAUSE OF SNAP DUMP:   USER PROGRAM CHECK
```

Figure 12–10. Program Check Abnormal Termination Snap for SAMFIN Load Module (FIXSAM Action Program) (Part 1 of 2)


INTERRUPT CODE
BITS 01100000
ADDRESS OF ERRONEOUS INSTRUCTION
BAD INSTRUCTION

```
PROGRAM STATUS WORD:   E0E6CE01   40034C5C
USER REGS 0-7   00000000   0003466C   00033400   00033400   000337C0   000337C0   000346B8   00033570
USER REGS 8-F   000338C0   00034698   00034558   00035558   60034BF4   00034958   900350EE   00034C56

 SNAP  BY EXIMS    AT 0171A6
  REGS 0-7  000013E0  000173BC  00033400  00033400    000337C0  000337C0  000346B8  00033570
  REGS 8-F  000338C0  00034698  00034558  00035558    60034BF4  00004AE8  60017164  60017186
 SNAP  0F3400  TO  0F38F0
033400-00000000 E2C1D4C6 C9D5D5D5 00520069  01C50006 00000000 00000000 00000000 *....SAMFINNN.....E..............-0F3400
033420-00000050 000C0100 00000008 00000000  F8F2F0F4 F1F5F0F8 F1F5F2F8 000000E6 *...&............82C41508152&...W-0F3420
033440-00500018 E9000000 E0E60E01 40034C5C  00000C00 0003466C 00033400 00033400 *.&..Z....W.. .<*.......%........-0F3440
033460-000337C0 000337C0 000346B8 00033570  000338C0 00034698 00034558 00035558 *...............................-0F3460
033480-60034BF4 00034958 900350EE 00034C56  00000000 00033448 00036EC4 50034FFC *-..4......&...<...........>D&.].-0F3480
0334A0-00015730 0000003E 000349A8 000349A8  0003340C 000337C0 000337C0 000346B8 *...............................-0F34A0
0334C0-00033570 000338C0 00034698 00034558  00035558 60034BF4 000338AC 800337C0 *..................-..4.........-0F34C0
0334E0-00033400 00033570 000338C0 000338DA  0003357C 00033799 000337C0 000338B3 *...............................-0F34E0
033500-000337B8 800337BA 000000C0 00034580  0003467C 00034958 000346B8 00033400 *..................a............-0F3500

034BE0-D2067023 60F8F211 A0408012 F910A040  622705C0 4770C026 D2477060 61609240 *K...-82.. ..9.. ........K..-/-. -0F4BE0
034C00-5031D246 50325031 D2015034 8012D247  70AC5031 58F09004 05EF9240 5031D246 *&.K.&.&.K.&...K...&..C..... &.K.-0F4C00
034C20-50325031 F211A050 50C194FC A051F211  A0608012 94FCA061 F911A050 A0604770 *&.&.2..&&A....2..-...../9..&.-..-0F4C20
034C40-C05247F0 C06641E0 C06218FC 90EFA114  58F09008 07FF47F0 C0300C00 A04C50C3 *...0............0.....0..... &C-0F4C40
034C60-F900A040 62274770 C0E2D204 503E5000  D2135049 5005D209 D14062B2 F236A060 *9.. .....SK.&.&.K.&.&.K.J ..2..--0F4C60
```

Figure 12-10. Program Check Abnormal Termination Snap for SAMFIN Load Module (FIXSAM Action Program) (Part 2 of 2)

12.9. ANALYZING A CALL SNAP DUMP

Purpose of the CALL SNAP dump

The CALL SNAP dump is useful in action program debugging because the program issuing the SNAP function call can continue processing. By specifying on the SNAP function call only those areas of your program that you want to examine, you obtain the data you want to check without terminating the program.

Sample CALL SNAP dump

Figure 12-11 shows the dump generated by the SNAP function code issued from the SAMFIN action program (Figure 12-7, lines 312 and 313). Notice, each beginning and ending area requested is listed in the dump (Figure 12-11).

```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
1                                                   1
*       I M S 9 0    S N A P    D U M P             *
1                                                   1
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*


  ACTION NAME:   SAMFIN                                                          DATE:   82/04/15

  CURRENT ACTION PROGRAM:   SAMFIN00     TERM-ID:   TRMD     TRANS-ID:   0052006901BB0001  TIME:    8:07:04

     **  ALLOCATION MAP  **

           FROM       TO        LENGTH       AREA-NAME

         00033400  0003348F  00000090    PROGRAM INFORMATION BLOCK (PIB)
         000338C0  000338EF  00000030    INPUT MESSAGE AREA (IMA)
         000337C0  000338BF  00000100    WORK AREA (WA)
         00033570  000337B7  00000248    OUTPUT MESSAGE AREA (OMA)
         000337B8  000337BF  00000008    CONTINUITY DATA AREA (CDA)
         00034000  000351FF  00001200    ACTION PROGRAM LOAD AREA
         00036E20  00036FE7  000001C8    THREAD CONTROL BLOCK (THCB)
         00000000  00000000  00000000    ACTION SUBPROGRAM (IF ACTIVE)
         00036E54  00036E73  00000020    FILE ALLOCATION MAP
         00008240  0000835F  00000120    TERMINAL CONTROL TABLE (TCT)
         00033508  0003356B  00000064    SHARED ACT.PROG. VOLATILE AREA


      CAUSE OF SNAP DUMP:   USER INLINE SNAP CALL

 SNAP  BY EXIMS    AT 0171A6

  REGS 0-7  000013E0  000334D8  000349A8  00033400    000337BA  000337C0  000346B8  00033570

  REGS 8-F  000338C0  00034698  00034558  00035558    60034E88  00033490  60017164  60017186

 SNAP  0F46B8  TO  0F4890

0346B8-10030000 10040000 1C0403C0 10040200  1E000000 00000000 D5D6D560 D5E4D4C5 *.......................NON-NUME-0F46B8

0346D8-D9C9C340 E5C1D3E4 C540C5D5 E3C5D9C5  C440C6D6 D940D9C5 C1C4E240 C4C5E2C9 *RIC VALUE ENTERED FOR READS DESI-0F46D8

0346F8-D9C5C440 C6C9C5D3 C4000C00 00000000  E3D9C1D5 E2C1C3E3 C9D6D540 C3C1D5C3 *RED FIELD.......TRANSACTION CANC-0F46F8

034718-C5D3D3C5 C440C4E4 C540E3D6 40C1C2D6  E5C540C5 D9D9D6D9 C5D5C440 D6C640C6 *ELLED DUE TO ABOVE ERROREND OF F-0F4718

034738-C9D3C540 D9C5C1C3 C8C5C440 C4E4D9C9  D5C740D9 C5C1C440 D5E4D4C2 C5D94040 *ILE REACHED DURING READ NUMBER  -0F4738

034758-C5D9D9D6 D940C6D9 D6D440E2 C1D460C7  C5E3404C C4E4D9C9 D5C740D9 C5C1C440 *ERROR FROM SAM-GET  DURING READ -0F4758

034778-D5E4D4C2 C5D94040 E2E3C1E3 E4E260C3  D6C4C54C 40404040 40404040 40404040 *NUMBER  STATUS-CODE             -0F4778

034798-40C4C5E3 C1C9D3C5 C440E2E3 C1E3E4E2  40C3D6C4 C5404000 C6E2D4E3 C6C9D300 * DETAILED STATUS CODE  .FSMTFIL.-0F4798

0347B8-C6E2D4C4 C6C9D300 C5D5E3C5 D940D5E4  D4C2C5D9 40D6C640 D9C5C1C4 E240C6D6 *FSMDFIL.ENTER NUMBER OF READS FO-0F47B8

0347D8-D940E2C1 D440E5C1 D940D3C5 D5C7E3C8  40C6C9D3 C5E240C1 E240C67E C5D50000 *R SAP VAR LENGTH FILES AS F#NN..-0F47D8

0347F8-D3C9D5C5 40C4C9E2 C3D6D5D5 C5C3E34C  D9C5D8E4 C5E2E3C5 C4000000 00000000 *LINE DISCONNECT REQUESTED.......-0F47F8

034818-D5D64B40 D9C5C1C4 40404040 C3E4E2E3  60C9C44C 40404040 404040C3 E4E2E3D6 *NO. READ    CUST-ID          CUSTO-0F4818
```

Figure 12-11. CALL SNAP Dump for SAMFIN Load Module (FIXSAM Action Program) (Part 1 of 2)

```
034838-D4C5D940 D5C1D4C5 404C4C40 40404040  40C1D4E3 40D7C1C9 C4404040 404040C4 *MER NAME          AMT PAID      D-0F4838
034858-C1E3C540 40404040 C5D9D9D6 D940D6D5  40E2D5C1 D740D5D6 4B40F140 40F24040 *ATE      ERROR ON SNAP NO. 1  2  -0F4858
034878-F34040F4 4040F540 40404040 40404040  40400000 00000000 5C                *3  4  5          ......*         -0F4878
 SNAP  0F3400  TO  0F3570
033400-00000000 E2C1D4C6 C9D5D5D5 00520069  01BD0001 00000000 00000000 00000000 *....SAMFINNN...................-0F3400
033420-00000050 000C0100 00000008 0000C000  F8F2F0F4 F1F5F0F8 F0F7F0F2 000000E6 *...&...........82C415080702...W-0F3420
033440-00500018 E9000000 00034958 00000000  00034958 00007B80 600344EA 00015730 *.&..Z...............#.-.......-0F3440
033460-00036E34 000000C8 00033400 0003F7DC  00036E20 00001FEC 00008240 00004C30 *..>....H......7...>........ ..<.-0F3460
033480-00000000 00C00000 000000C0 00000000  00000000 00033448 00036EC4 40034F6E *.........................>D .]>-0F3480
0334A0-00015730 00C0004A 000349A8 000349A8  0003340C 000337BA 000337CC 000346B8 *.......[.......................-0F34A0
0334C0-00033570 000338C0 00034698 00034558  00035558 60034E88 000346B8 00034890 *..................-.+.........-0F34C0
0334E0-00033400 00033570 000338C0 000338DA  00033570 00033799 000337C0 000338B3 *..............................-0F34E0
033500-000337E8 800337BA 0000CCCC 00034580  0003467C 00034958 000346B8 00033400 *..................@...........-0F3500
033520-000338C0 000337B8 00033570 000337C0  00000000 60034D18 00034EBE 60034E88 *..................-.(...+.-.+.-0F3520
033540-00000F00 00000000 00000C00 00000000  0000C00C 00000000 00000000 00000000 *..............................-0F3540
033560-00000000 00000000 00000000 00000000  00                                  *.................               -0F3560
 SNAP  0F38C0  TO  0F38DA
0338C0-E3D9D4C4 00520069 01BD0001 000F0000  C67BF0F3 40D540E8 40D55C            *TRMD............F#03 N Y N*    -0F38C0
 SNAP  0F3570  TO  0F3799
033570-00000000 00000000 00000000 01CC0000  10030000 C6C9D3C5 40D5C1D4 C5404040 *....................FILE NAME   -0F3570
033590-404040C6 E2D4E3C6 C9D34C40 40404040  40404C4C 40404040 40404040 40404040 *   FSMTFIL                     -0F3590
0335B0-40404040 40404040 40404C40 40404040  4040404C 40404040 40404040 10040000 *                           ....-0F35B0
0335D0-C5D5C440 D6C640C6 C9D3C540 D9C5C1C3  C8C5C440 C4E4D9C9 D5C740D9 C5C1C440 *END OF FILE REACHED DURING READ -0F35D0
0335F0-D5E4D4C2 C5D9404C 40F14C40 40404C40  40404C4C 40404C40 40404040 40404040 *NUMBER   1                    -0F35F0
033610-40404040 40404040 10040000 E2E3C1E3  E4E260C3 D6C4C540 40F0F0F0 F2404040 *         ....STATUS-CODE  0002   -0F3610
033630-40404040 40C4C5E3 C1C9D3C5 C440E2E3  C1E3E4E2 40C3D6C4 C54040F0 F0F0F040 *     DETAILED STATUS CODE  0000 -0F3630
033650-40404040 40404040 40404C40 40404C40  4040404C 10040200 C6C9D3C5 40D5C1D4 *                       ....FILE NAM-0F3650
033670-C5404040 404040C6 E2D4C4C6 C9D34040  40404040 40404040 40404040 40404040 *E       FSMDFIL                  -0F3670
033690-40404040 40404040 40404C40 40404040  404C404C 40404040 40404040 40404040 *                                -0F3690
0336B0-10040000 D5D64B40 D9C5C1C4 4040404C  C3E4E2E3 60C9C440 40404040 404040C3 *....NO. READ    CUST-ID       C-0F36B0
0336D0-E4E2E3D6 D4C5D940 D5C1D4C5 40404040  40404040 40C1D4E3 40D7C1C9 C4404040 *USTOMER NAME          AMT PAID   -0F36D0
0336F0-404040C4 C1E3C540 40404040 10040000  404040F0 F3404040 40404040 40F0F0F0 *   DATE      ....  03          000-0F36F0
033710-F1F04040 40404040 C1D34050 40D2C1E8  7DE240E2 E3C5C1D2 40404040 40404040 *10       AL & KAY'S STEAK        -0F3710
033730-405BF2F1 F7F04BF0 F3404C40 40F1F161  F1F961F7 F6404040 10040200 40404040 * $2170.03     11/19/76   ....    -0F3730
033750-40404040 40404040 40404040 40404040  40404C4C 40404040 40404040 40404040 *                                -0F3750
 ****  0F3770  TO  0F379C  SAME AS ABOVE
033790-40404040 40404040 405C                                                   *         *                      -0F3790
 SNAP  0F37C0  TO  0F38B3
0337C0-F0F0F0F1 F0C1D340 5040D2C1 E87DE240  E2E3C5C1 D2404040 40F0F2F1 F7F0F0F3 *00010AL & KAY'S STEAK    0217003-0F37C0
0337E0-F1F161F1 F961F7F6 00000000 00000000  0040404C F0F34040 40404040 4040F0F0 *11/19/76.........   03          00-0F37E0
033800-F0F1F040 40404040 40C1D340 5040D2C1  E87DE240 E2E3C5C1 D2404040 40404040 *010       AL & KAY'S STEAK       -0F3800
033820-40405BF2 F1F7F04B F0F34C40 4040F1F1  61F1F961 F7F64040 40E2E3C1 E3E4E260 *  $2170.03     11/19/76   STATUS--0F3820
033840-C3D6C4C5 4040F0F0 F0F24C40 40404040  404CC4C5 E3C1C9D3 C5C440E2 E3C1E3E4 *CODE  0002        DETAILED STATU-0F3840
033860-E240C3D6 C4C54040 F0F0F0F0 40404040  40404040 40404040 40404040 40404040 *S CODE  0000                    -0F3860
033880-40F0F3F0 00000000 00000000 00000000  00000000 00000000 00000000 00000000 * 030..........................-0F3880
0338A0-00000000 00000000 00000000 C6E2D4C4  C6C9D35C                           *.............FSMDFIL*           -0F38A0
 SNAP  0F37B8  TO  0F37BA
0337B8-D5E85C                                                                   *NY*                             -0F37B8
```

Figure 12–11. CALL SNAP Dump for SAMFIN Load Module (FIXSAM Action Program) (Part 2 of 2)

12.10. ONLINE FILE RECOVERY

When a transaction terminates abnormally, or requests rollback before completion, IMS rolls back user data file modifications (updates, inserts, and deletions) that occurred in the transaction and issues messages to the source terminal and system console. These messages are explained in the OS/3 system messages programmer/operator reference, UP-8076 (current version).

Automatic file rollback

On rollback, IMS returns each MIRAM, ISAM, or DAM file, modified in the terminated transaction, to its logical state before the transaction was initiated or before the last rollback point was recorded on the audit file. When abnormal termination occurs, rollback occurs automatically.

Requested file rollback

You can request rollback upon normal termination of a transaction by moving special indicator values into the LOCK-ROLLBACK-INDICATOR field of the program information block. For more information on the use of this indicator, refer to 3.11.

IMS audit file entries

Before update or deletion, IMS records in the audit file the current state of each record to be modified. In addition, before adding a new record to a file, IMS records in the audit file the keys or record numbers of records to be added. It also records data marking the initiation and termination of each transaction that modifies a file. If you specify a lock rollback indicator value to establish lock rollback points, IMS also records these rollback points in the audit file.

Table 12-2 lists the functions IMS performs to roll back file modifications.

Table 12-2. File Rollback

File Modification	Functions that Cause Modification	Functions Performed to Roll Back Modification
Update	GETUP, PUT	GETUP (current image), PUT (before-image)
Delete	GETUP, DELETE	INSERT (before-image)
Insert	INSERT	GETUP (current image), DELETE

Error Returns

Unrecoverable audit file errors

When unrecoverable I/O errors occur in the audit file, IMS notifies the source terminal operator, sends an error message to the print file, and attempts rollback of all existing transactions logged in the audit file. If you didn't configure LOCK=UP in the configurator FILE section, IMS prohibits any additional update requests and returns a status code of 3 (invalid request) and one of the detailed status codes listed in Appendix D.

Prefix Area Format

Data file I/O errors

If an I/O error occurs on a user data file during rollback of a file modification, IMS takes a snapshot dump of the prefix area of the record being rolled back. After the snapshot dump, IMS continues rolling back all modifications made to user data files for that transaction.

AUDCONF/AUDFILE errors

If an error occurs on the AUDCONF or AUDFILE during rollback of updates made by a transaction, IMS places the name, ZU#ROL, into the current action program name field of the prefix area.

Prefix area format and contents

Figure 12-12 shows the format of the prefix area and Table 12-3 describes the content of each field.

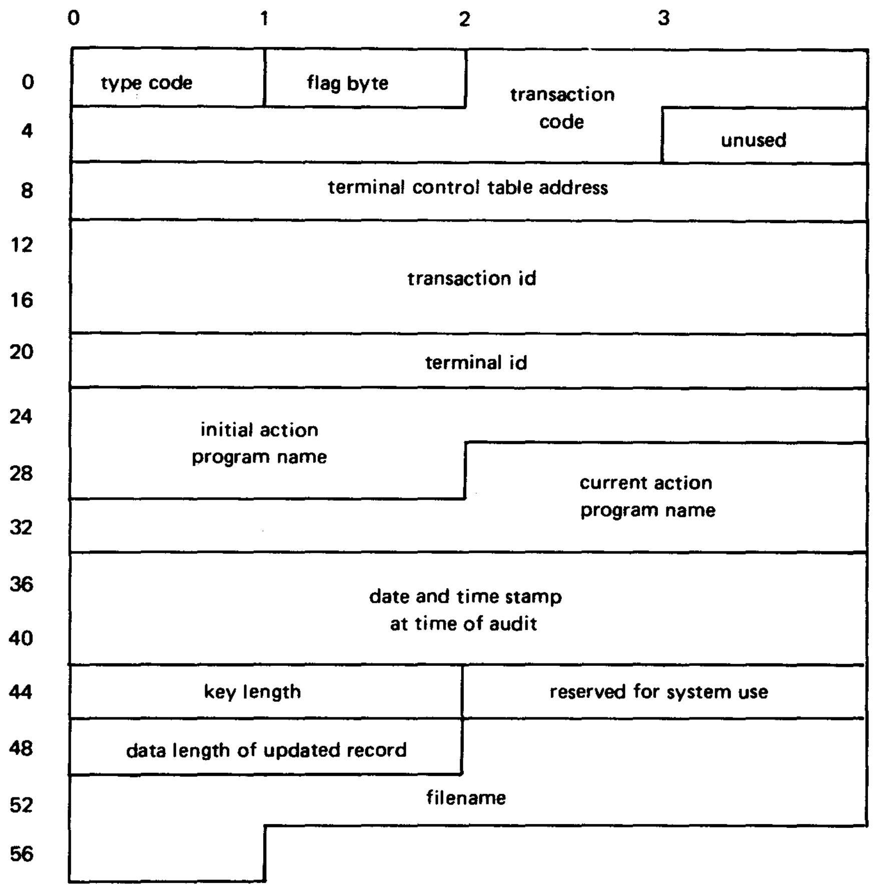

0
1
2
3
0
4
8
12
16
20
24
28
32
36
40
44
48
52
56
type code
flag byte
transaction code
unused
terminal control table address
transaction id
terminal id
initial action program name
current action program name
date and time stamp at time of audit
key length
reserved for system use
data length of updated record
filename

Figure 12-12. Format of Prefix Area of Records in the Audit File (Online Recovery)

Table 12-3. Contents of Prefix Area for Records in the Audit File (Online Recovery) (Part 1 of 2)

Label	Field Name	Bytes	Code	Description
ZF#RTC	Type code	0	Binary	Bits Set to 1 / Meaning 0 Not used 1 Not used 3 Termination 4 Not used 5 Rollback point 6 Before-image, MIRAM 6, 7 Before-image, ISAM 7 Before-image, DAM
ZF#AFB	Flag byte	1	Binary	Bits Set to 1 / Meaning 0 First before-image for transaction 1 Inserted record 2 Abnormal termination 3 Not used 4 MIRAM, indexed 5–7 Not used

Table 12–3. Contents of Prefix Area for Records in the Audit File (Online Recovery) (Part 2 of 2)

Label	Field Name	Bytes	Code	Description
ZF#ATC	Transaction code	2-6	EBCDIC	Configured code identifying the current transaction; one to five alphanumeric characters, left-justified in field
–	–	7	–	Unused
ZF#ACT	TCT address	8–11	Hexadecimal	Address of terminal control table (TCT) for terminal originating this transaction. Full-word aligned
ZF#ATRID	Transaction id	12–19	Binary	Data-time of initiation of this transaction, in the form: yy-mm-dd-hh-mm-ss
ZF#ATMID	Terminal-id	20–23	Hexadecimal	Configured identification of network termination initiating this transaction
ZF#AIAP	Initial action program	24–29	EBCDIC	Program-name of first action program initiated for this transaction; one to six alphanumeric characters, left-justified
ZF#ACAP	Current action program	30–35	EBCDIC	Program-name of currently active action program
ZF#ADT	Date-time of audit	36–43	Binary	Date-time of writing this record to the audit file, in same form as transaction-id
ZF#KLIDA	Key length	44–45	Binary	Length of key in an indexed record; set to 0 for a DAM Record
ZF#CNKN	–	46–47	–	Reserved for system use
ZF#DLIDA or ZF#NAUT	Data length	48–49	Binary	Length of data portion of updated record, or number of active update transactions.
ZF#FNM	File name	50–57	EBCDIC	Logical name of data file being accessed by current action program; one to seven alphanumeric characters, left-justified

NOTES:

1. When records are written to the audit file for a UNIQUE action program, the *transaction-code* field contains OPEN, the *initial-action-program* field contains ZU#OPEN, and the *current-action-program* field contains the name of the UNIQUE module active at the time of audit.

2. When the current action program is accessing a defined file, a prefix is written for each logical record involved. In the prefix, the *file-name* field contains the LFD-name of a conventional user data file contributing a logical record (or part of one) to the defined record. It never contains the *defined-file-name* specified with the DFILE keyword.

12.11. COBOL ACTION PROGRAM ERROR MESSAGE BUFFER

Locating the COBOL error message buffer

Error message buffer contents for 1974 COBOL

The COBOL error routines C@@MSI (1974 COBOL) and COBJERR (extended COBOL) record data in a 4-byte message buffer that corresponds to errors contained in the canned message file. To find the cause of error, locate this message buffer by checking for its address in general register 1 of the program dump listing. Table 12-4 shows the contents of the message buffer for 1974 COBOL and Table 12-5 describes the error messages.

Table 12-4. 1974 COBOL Message Buffer Contents

Byte	Hexadecimal Content	Description
0	C3	Canned message prefix
1	C5	Canned message prefix
2-3	nnnn	Hexadecimal message number

NOTE:

The hexadecimal message number in bytes 2 and 3 is one of the following and corresponds to the numbered COBOL message shown (nnnn). For the meaning of the message and suggested corrective action refer to the OS/3 system messages programmer/operator reference, UP-8076 (current version).

Table 12-5. 1974 COBOL Error Messages for Action Programs

1974 COBOL error messages

COBOL Message	Message Text
CE23	END OF PROCEDURE DIVISION EXECUTED
CE25	NEGATIVE VALUE EXPONENTIATED
CE29	FLOATING POINT ERROR

Error message buffer contents for extended COBOL

Table 12-6 shows the contents of the message buffer for extended COBOL. Table 12-7 describes the error messages.

Table 12-6. Extended COBOL Message Buffer Contents

Byte	Hexadecimal Content	Description
0	5B	Canned message indicator ($)
1-2	nnnn	Hexadecimal message number
3	40	End-of-table indicator (blank)

NOTE:

The hexadecimal message number in bytes 1 and 2 is one of the following and corresponds to the numbered COBOL message shown (nnnn). For the meaning of the message and suggested corrective action refer to the OS/3 system messages programmer/operator reference, UP-8076 (current version).

Table 12-7. Extended COBOL Error Messages for Action Programs

Bytes 1-2 Content	COBOL Message	Message Text
043A	CE03	END OF PROCEDURE DIVISION EXECUTED
043B	CE04	INVALID EXECUTION OF ENTRY POINT
043C	CE05	NEGATIVE VALUE EXPONENTIATED

PART 6

APPENDIXES

Appendix A. Statement Conventions

Throughout this document, certain conventions are observed on formats for statements and commands. General rules with examples pertaining to these conventions follow:

Capital letters

- Capital letters and punctuation marks (except braces, brackets, and ellipses) must be coded exactly as shown. For example:

  ```
  CALL 'GET' USING filename record-area record-number.
  ```

  is coded:

  ```
  CALL 'GET' USING CUSTFIL CUS-REC REC-KEY.
  ```

Lowercase letters

- Lowercase letters and words are generic terms representing information that you supply. Such terms may contain acronyms and hyphens for readability. For example:

  ```
  PROCEDURE DIVISION USING program-information-block
                           input-message-area
                           [work-area] [output-message-area]
                           [continuity-data-area].
  ```

  is coded:

  ```
  PROCEDURE DIVISION USING PIB IMA WA OMA CDA.
  ```

Braces

- Information within braces { } represents necessary entries, one of which must be chosen.

  For example:

  ```
  {CALL   }{GET  },(filename,record-area,record-number)
  {ZG#CALL}{GETUP}
  ```

is coded:

```
1       10    16
-----------------------------------------
        ZG#CALL GET,(STATE,RECORD,SNKEY)
```

or

```
        CALL   GETUP,(STATE,RECORD,SNKEY)
```

Brackets

- Information within brackets [ ], including commas and semicolons, represents optional entries that you include or omit, depending on program requirements. Braces within brackets indicate that you must choose one of the entries if you include that operand. For example:

  $\text{JUS=}\begin{Bmatrix}\text{L}\\\text{R}\end{Bmatrix}$

  is coded:

  ```
  JUS=L
  ```

Default parameters

- Default parameter specifications are indicated by shading. For example, if no TYP parameter is specified as input to the edit table generator, the M is supplied, meaning alphanumeric type data is expected.

  $\left[\text{TYP=}\begin{Bmatrix}\text{A}\\\text{B}\\\text{M}\\\text{N}\\\text{P}\end{Bmatrix}\right]$ (default value)

Periods

- A series of three periods vertically spaced, occurring in a program example, indicates that other coding not directly relating to the example is omitted.

  Example:

  ```
  PARA-1.
     CALL 'GET' USING STATE RECORD SNKEY.
     .
     .
     .
  PARA-2.
  ```

Statement conventions and coding rules specific to individual functions are described where applicable throughout this document.

Appendix B. COBOL Action Programming Examples

B.1. DESCRIPTION

Contents

Appendix B contains compiler listings of sample COBOL action programs. Parts of coding from some of these programs appear out of context in different parts of the manual where we describe specific subjects and how to handle the coding.

Summary

The nine COBOL action programs in this appendix illustrate the complete action program coding for simple and dialog transactions, external and immediate internal succession, use of screen format services, sending a message to another terminal, output-for-input queueing, and continuous output.

CSCAN series

The CSCAN action program series (Figures B-1 through B-18) consists of four action programs:

- DMSCAN
- DMDETL
- DMPYMT
- DMTOTL

Simple transactions

These programs represent a series of simple transactions that:

- page through a customer file (CSCAN transaction code);
- display a customer's account status (CDETL transaction code);
- apply payments to a customer's account (PAYMT transaction code); and
- request audit data about all payments applied to a customer's account (TOTAL transaction code).

OVERVIEW OF COBOL ACTION PROGRAMMING EXAMPLES

ACT1/ACT2 dialog transaction

Action programs ACT1 and ACT2 (Figures B-21 and B-22) illustrate a dialog transaction with ACT1 naming ACT2 as external successor.

JAMENU screen formatting

JAMENU (Figure B-23) is one of a series of action programs that make up an entitlement accounting system. By validating a password entered from the terminal, JAMENU displays either a menu screen or an error screen.

In addition to using both external and immediate internal succession, JAMENU uses the BUILD function call to construct screen formatted messages for a valid or an invalid password.

BEGIN1 output-for-input queueing

The BEGIN1 action program (Figure B-24) illustrates use of the SEND function to initiate a transaction that performs continuous output at another terminal. It also shows the output-for-input queueing feature.

PRINT continuous output

The PRINT action program (Figure B-25) creates continuous output, sends it to the source terminal, and uses delivery notice scheduling for control and recovery.

B.2. SAMPLE COBOL ACTION PROGRAMS PERFORMING SIMPLE TRANSACTIONS (CSCAN SERIES)

CSCAN program series description

The four action programs: DMSCAN, DMDETL, DMPYMT, and DMTOTL perform a series of simple transactions. The transaction code CSCAN starts the first transaction in the series.

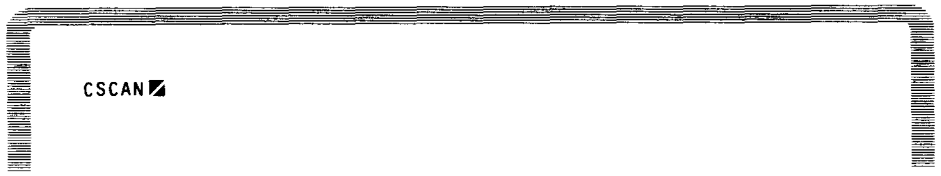
Files used

These four action programs use three indexed files that have been defined to IMS in the FILE section of the configuration:

1. DMOALT — A customer file (alternate account file), sorted on zip code, customer last name, and customer account number sequence (See Figure B-14, lines 12 and 89-96.)
2. DMOMSTR — A customer master file, containing current financial data per customer and sorted in account number sequence. (See Figure B-15, lines 11 and 98-111, and Figure B-16, lines 11 and 94-99.)
3. DMOXACT — An audit file created or updated by the PAYMT transaction and accessed for display by the TOTAL transaction. (See Figure B-16, lines 12 and 100-115, and Figure B-17, lines 11 and 91-108.)

Key in CSCAN transaction code

You begin the first transaction by keying in the transaction code, CSCAN on line 1 of the screen and pressing the **TRANSMIT** key.

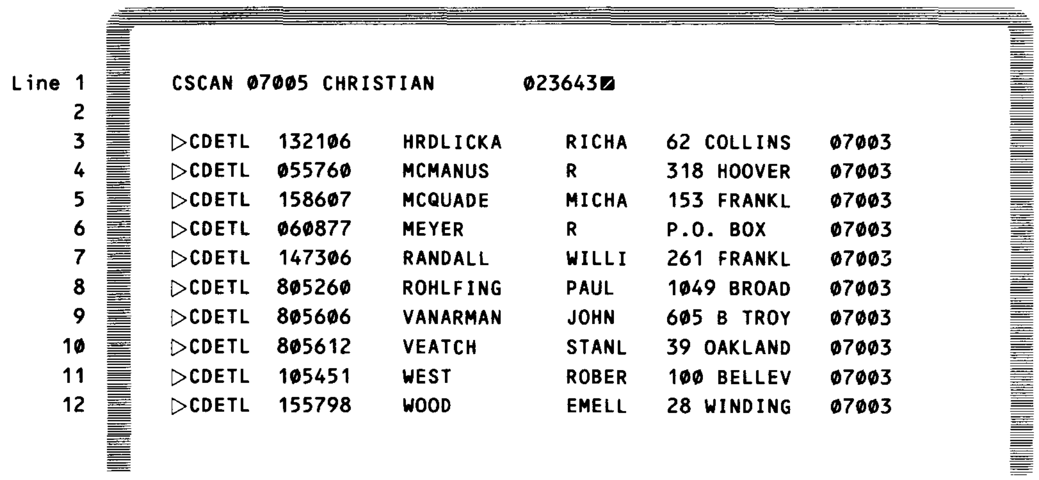


```
CSCAN▨
```

Figure B-1. Initiating the CSCAN Transaction

Resulting CSCAN output

The CSCAN transaction lists basic customer data by zip code, allowing you to scan the lists. The alternate account file, DMOALT, serves as an index to the customer master file, DMOMSTR. It is sequenced by zip code, customer last name, and customer account number. Figure B-2 shows the resulting output.



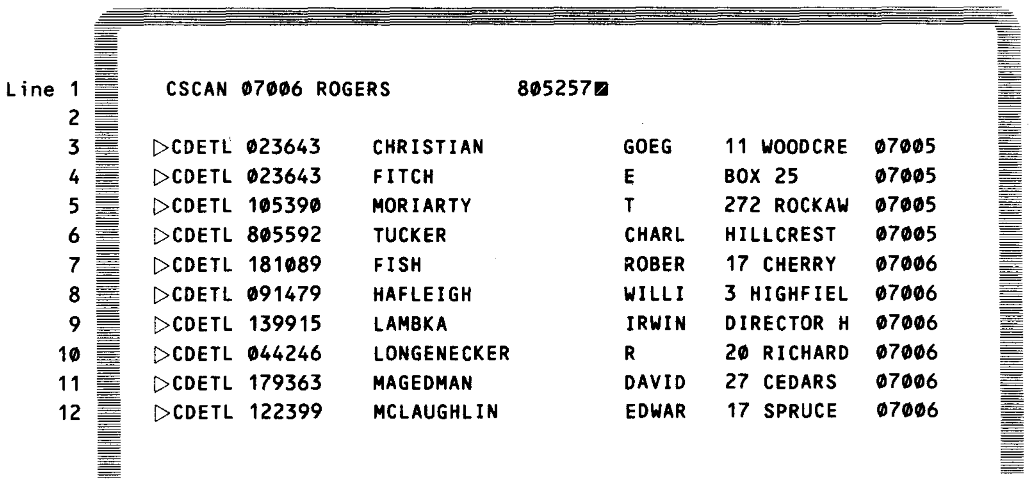
```
Line 1   CSCAN 07005 CHRISTIAN       023643▨
     2
     3   ▷CDETL  132106   HRDLICKA    RICHA   62 COLLINS   07003
     4   ▷CDETL  055760   MCMANUS     R       318 HOOVER   07003
     5   ▷CDETL  158607   MCQUADE     MICHA   153 FRANKL   07003
     6   ▷CDETL  060877   MEYER       R       P.O. BOX     07003
     7   ▷CDETL  147306   RANDALL     WILLI   261 FRANKL   07003
     8   ▷CDETL  805260   ROHLFING    PAUL    1049 BROAD   07003
     9   ▷CDETL  805606   VANARMAN    JOHN    605 B TROY   07003
    10   ▷CDETL  805612   VEATCH      STANL   39 OAKLAND   07003
    11   ▷CDETL  105451   WEST        ROBER   100 BELLEV   07003
    12   ▷CDETL  155798   WOOD        EMELL   28 WINDING   07003
```

Figure B-2. Output from CSCAN Transaction Code

The DMSCAN action program (Figure B-14, lines 111-128) displays the first ten records of the DMOALT file (Figure B-2, lines 3-12). The record displayed on line 1 of the screen is the next available record on the file.

Displaying more records

By pressing the **TRANSMIT** key, you can display the next ten records on the file as shown in Figure B-3. (See the DMSCAN action program, Figure B-14, lines 135-141.) Notice that the CSCAN transaction code is displayed on line 1 of the screen, so that when you press **TRANSMIT**, a new transaction begins and DMSCAN is recheduled.



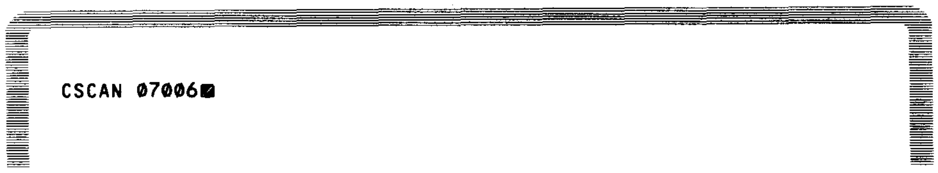
```
Line 1   CSCAN Ø7ØØ6 ROGERS          8Ø5257■
     2
     3  ▷CDETL Ø23643   CHRISTIAN           GOEG    11 WOODCRE  Ø7ØØ5
     4  ▷CDETL Ø23643   FITCH               E       BOX 25      Ø7ØØ5
     5  ▷CDETL 1Ø539Ø   MORIARTY            T       272 ROCKAW  Ø7ØØ5
     6  ▷CDETL 8Ø5592   TUCKER              CHARL   HILLCREST   Ø7ØØ5
     7  ▷CDETL 181Ø89   FISH                ROBER   17 CHERRY   Ø7ØØ6
     8  ▷CDETL Ø91479   HAFLEIGH            WILLI   3 HIGHFIEL  Ø7ØØ6
     9  ▷CDETL 139915   LAMBKA              IRWIN   DIRECTOR H  Ø7ØØ6
    1Ø  ▷CDETL Ø44246   LONGENECKER         R       2Ø RICHARD  Ø7ØØ6
    11  ▷CDETL 179363   MAGEDMAN            DAVID   27 CEDARS   Ø7ØØ6
    12  ▷CDETL 122399   MCLAUGHLIN          EDWAR   17 SPRUCE   Ø7ØØ6
```

Figure B-3. Continuation of Output from CSCAN Transaction Code

Displaying specific records

You can continue displaying customer records until you reach the end of the file (Figure B-14, lines 151-156 and 175-194).

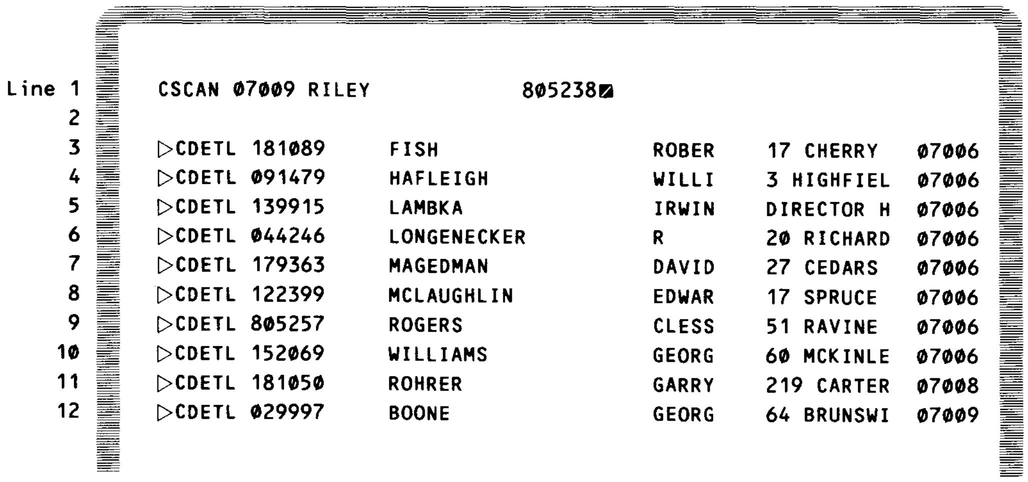
The CSCAN transaction allows you to scan in another way. Instead of displaying records at the beginning of a file and scanning until you find the customer zip code you want, you can display the first ten records with the desired zip code or higher. By entering the zip code you want after the CSCAN transaction code (see Figure B-4), the DMSCAN action program begins scanning the DMOALT file for the first record that contains that zip code (Figure B-14, lines 151-171 and 179-194).



```
CSCAN Ø7ØØ6■
```

Figure B-4. Initiating a Qualified CSCAN Transaction

Figure B-5 shows the results of this entry after you press the **TRANSMIT** key.



```
Line 1   CSCAN 07009 RILEY           805238▨
     2
     3   ▷CDETL 181089   FISH               ROBER   17 CHERRY   07006
     4   ▷CDETL 091479   HAFLEIGH           WILLI   3 HIGHFIEL  07006
     5   ▷CDETL 139915   LAMBKA             IRWIN   DIRECTOR H  07006
     6   ▷CDETL 044246   LONGENECKER        R       20 RICHARD  07006
     7   ▷CDETL 179363   MAGEDMAN           DAVID   27 CEDARS   07006
     8   ▷CDETL 122399   MCLAUGHLIN         EDWAR   17 SPRUCE   07006
     9   ▷CDETL 805257   ROGERS             CLESS   51 RAVINE   07006
    10   ▷CDETL 152069   WILLIAMS           GEORG   60 MCKINLE  07006
    11   ▷CDETL 181050   ROHRER             GARRY   219 CARTER  07008
    12   ▷CDETL 029997   BOONE              GEORG   64 BRUNSWI  07009
```

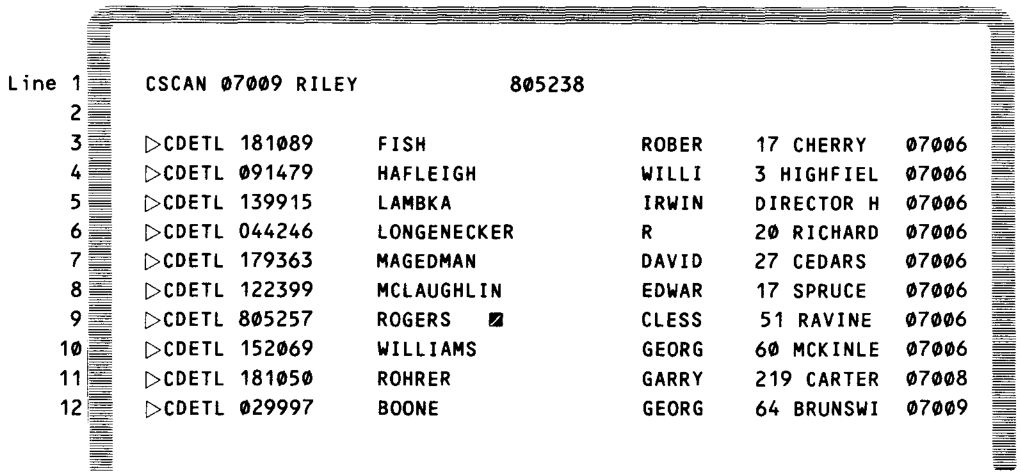
Figure B-5. Output from Qualified CSCAN Transaction Code

Initiating CDETL

When you've found the customer account for which you want detailed information, you are ready to initiate the CDETL transaction. There are two ways to do this. Let's assume ROGERS is the customer for whom you want to display detailed account information.

1. You can enter the transaction code (CDETL) and ROGERS' account number (805257) on line 1 of the screen and press **TRANSMIT**.

2. You can forward tab the cursor to a position beyond the last name of the desired customer (ROGERS) as shown in Figure B-6 and press the **TRANSMIT** key. This method is more efficient because it reduces the number of keystrokes required and the possibility of erroneous data entry.

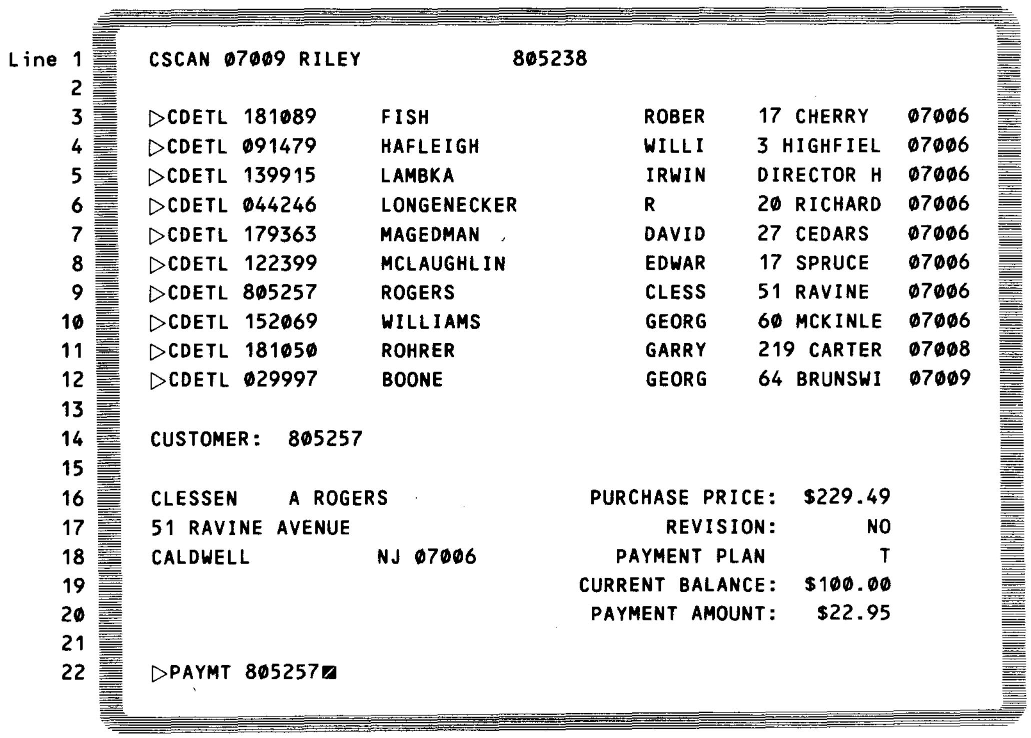


```
Line 1   CSCAN 07009 RILEY              805238
     2
     3   ▷CDETL 181089   FISH                ROBER   17 CHERRY    07006
     4   ▷CDETL 091479   HAFLEIGH            WILLI   3 HIGHFIEL   07006
     5   ▷CDETL 139915   LAMBKA              IRWIN   DIRECTOR H   07006
     6   ▷CDETL 044246   LONGENECKER         R       20 RICHARD   07006
     7   ▷CDETL 179363   MAGEDMAN            DAVID   27 CEDARS    07006
     8   ▷CDETL 122399   MCLAUGHLIN          EDWAR   17 SPRUCE    07006
     9   ▷CDETL 805257   ROGERS    ▨         CLESS   51 RAVINE    07006
    10   ▷CDETL 152069   WILLIAMS            GEORG   60 MCKINLE   07006
    11   ▷CDETL 181050   ROHRER              GARRY   219 CARTER   07008
    12   ▷CDETL 029997   BOONE               GEORG   64 BRUNSWI   07009
```

Figure B-6. Initiating the CDETL Transaction

Resulting output

Figure B-7 shows the output screen resulting from using the cursor tabbing/TRANSMIT method of initiating the CDETL transaction. The customer information on the lower part of the screen is displayed by the DMDETL action program (Figure B-15, lines 127-167.)



```
Line 1   CSCAN 07009 RILEY              805238
     2
     3   ▷CDETL 181089   FISH                ROBER   17 CHERRY    07006
     4   ▷CDETL 091479   HAFLEIGH            WILLI   3 HIGHFIEL   07006
     5   ▷CDETL 139915   LAMBKA              IRWIN   DIRECTOR H   07006
     6   ▷CDETL 044246   LONGENECKER         R       20 RICHARD   07006
     7   ▷CDETL 179363   MAGEDMAN            DAVID   27 CEDARS    07006
     8   ▷CDETL 122399   MCLAUGHLIN          EDWAR   17 SPRUCE    07006
     9   ▷CDETL 805257   ROGERS              CLESS   51 RAVINE    07006
    10   ▷CDETL 152069   WILLIAMS            GEORG   60 MCKINLE   07006
    11   ▷CDETL 181050   ROHRER              GARRY   219 CARTER   07008
    12   ▷CDETL 029997   BOONE               GEORG   64 BRUNSWI   07009
    13
    14   CUSTOMER:  805257
    15
    16   CLESSEN    A ROGERS              PURCHASE PRICE:  $229.49
    17   51 RAVINE AVENUE                       REVISION:       NO
    18   CALDWELL          NJ 07006         PAYMENT PLAN         T
    19                                  CURRENT BALANCE:  $100.00
    20                                    PAYMENT AMOUNT:   $22.95
    21
    22   ▷PAYMT 805257▨
```

Figure B-7. Output from CDETL Transaction

Processing CDETL

When the DMDETL program reads the master record successfully and it contains a Y in its last byte, the program moves the word 'YES' to the output field containing REVISION and you can make changes to the customer record you selected. (See Figure B-15, lines 199 and 200). Otherwise, the DMDETL program moves the word 'NO' to the REVISION output field and you can display another customer's account information at the bottom of the screen.

Automatic succession

Notice that the DMDETL program automatically succeeds to the PAYMT transaction when you update the customer whose detailed information you displayed. DMDETL accomplishes this by moving the transaction code, PAYMT, in the form of a constant from working storage to the output message area (Figure B-16, line 196). Then, when you move the cursor to a point beyond the PAYMT transaction code and account number, the PAYMT transaction begins.

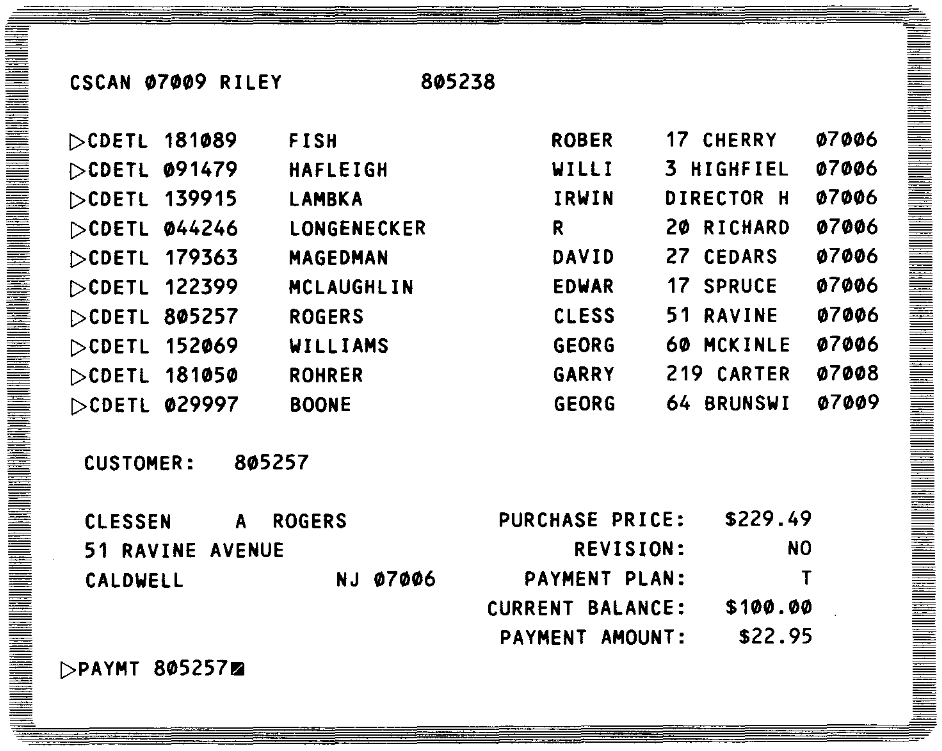
Initiating PAYMT

There are two ways to initiate the PAYMT transaction:

1. Forward tab the cursor to a position beyond the account number following the PAYMT transaction code and press **TRANSMIT**. (See Figure B-8.)

2. Enter a payment amount different than the payment plan amount. You enter the amount next to the account number following the PAYMT transaction code and press **TRANSMIT**. (See Figure B-10.)

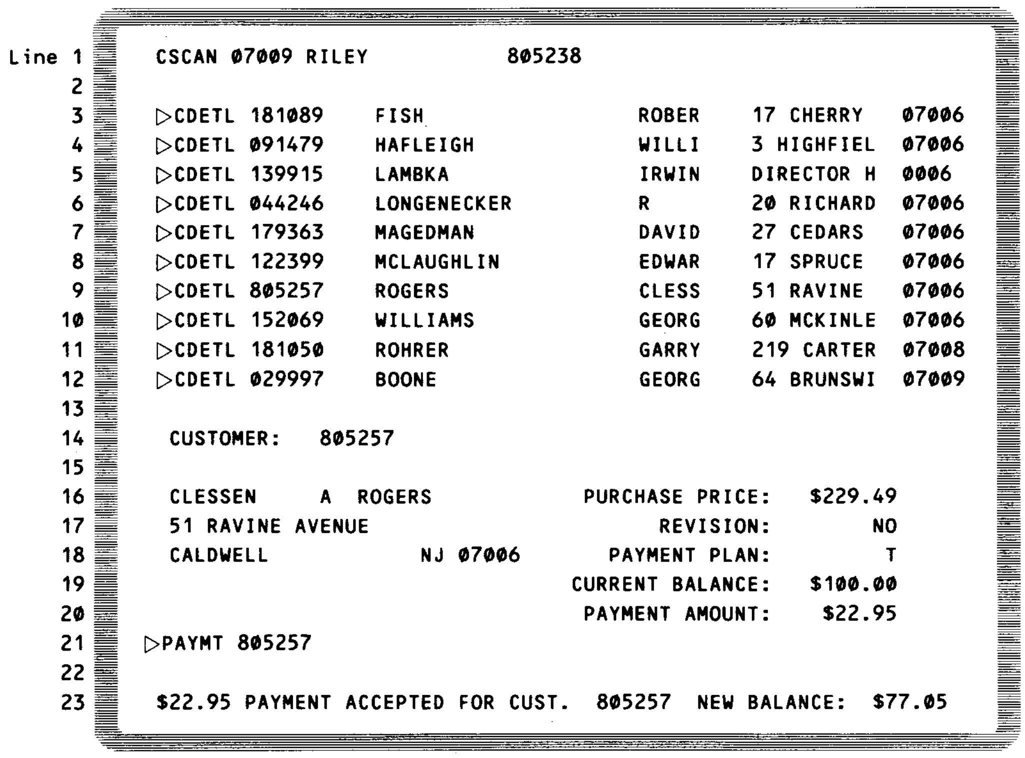
The first method instructs the DMPYMT action program to subtract the payment plan amount ($22.95 in Figure B-8) from this customer's current balance ($100.00 in Figure B-8). (See Figure B-16, line 157.)



```
CSCAN 07009 RILEY          805238

▷CDETL 181089    FISH                 ROBER    17 CHERRY   07006
▷CDETL 091479    HAFLEIGH             WILLI    3 HIGHFIEL  07006
▷CDETL 139915    LAMBKA               IRWIN    DIRECTOR H  07006
▷CDETL 044246    LONGENECKER          R        20 RICHARD  07006
▷CDETL 179363    MAGEDMAN             DAVID    27 CEDARS   07006
▷CDETL 122399    MCLAUGHLIN           EDWAR    17 SPRUCE   07006
▷CDETL 805257    ROGERS               CLESS    51 RAVINE   07006
▷CDETL 152069    WILLIAMS             GEORG    60 MCKINLE  07006
▷CDETL 181050    ROHRER               GARRY    219 CARTER  07008
▷CDETL 029997    BOONE                GEORG    64 BRUNSWI  07009

 CUSTOMER:   805257

 CLESSEN     A   ROGERS          PURCHASE PRICE:   $229.49
 51 RAVINE AVENUE                      REVISION:        NO
 CALDWELL           NJ 07006       PAYMENT PLAN:         T
                                CURRENT BALANCE:   $100.00
                                 PAYMENT AMOUNT:    $22.95

▷PAYMT 805257■
```

Figure B-8. First Method for Initiating the PAYMT Transaction

Figure B-9 shows the results of this subtraction to obtain the customer's new balance.



```
CSCAN Ø7ØØ9 RILEY          8Ø5238

▷CDETL 181Ø89    FISH                  ROBER    17 CHERRY   Ø7ØØ6
▷CDETL Ø91479    HAFLEIGH              WILLI    3 HIGHFIEL  Ø7ØØ6
▷CDETL 139915    LAMBKA                IRWIN    DIRECTOR H  ØØØ6
▷CDETL Ø44246    LONGENECKER           R        2Ø RICHARD  Ø7ØØ6
▷CDETL 179363    MAGEDMAN              DAVID    27 CEDARS   Ø7ØØ6
▷CDETL 122399    MCLAUGHLIN            EDWAR    17 SPRUCE   Ø7ØØ6
▷CDETL 8Ø5257    ROGERS                CLESS    51 RAVINE   Ø7ØØ6
▷CDETL 152Ø69    WILLIAMS              GEORG    6Ø MCKINLE  Ø7ØØ6
▷CDETL 181Ø5Ø    ROHRER                GARRY    219 CARTER  Ø7ØØ8
▷CDETL Ø29997    BOONE                 GEORG    64 BRUNSWI  Ø7ØØ9

 CUSTOMER:   8Ø5257

 CLESSEN     A  ROGERS            PURCHASE PRICE:    $229.49
 51 RAVINE AVENUE                       REVISION:         NO
 CALDWELL           NJ Ø7ØØ6        PAYMENT PLAN:          T
                                 CURRENT BALANCE:    $1ØØ.ØØ
                                  PAYMENT AMOUNT:     $22.95
▷PAYMT 8Ø5257

$22.95 PAYMENT ACCEPTED FOR CUST.  8Ø5257  NEW BALANCE:  $77.Ø5
```

Figure B-9. Output from PAYMT Transaction Using Standard Payment Amount

Processing PAYMT

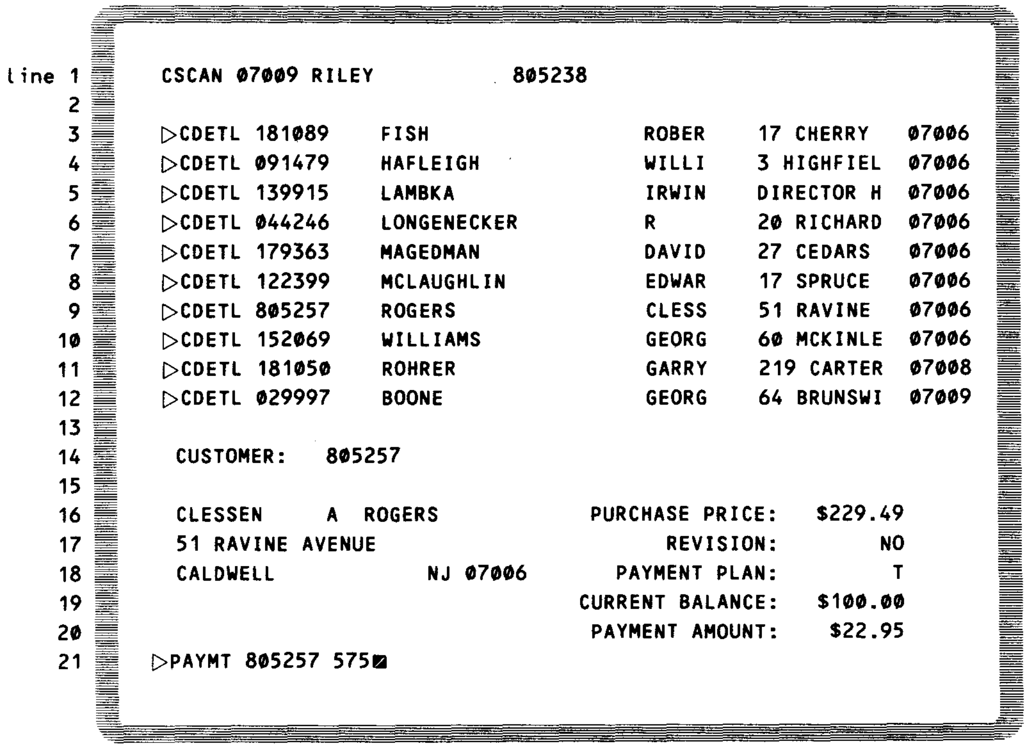
Transmitting only the transaction code and customer account number confirms the amount applied to the customer's new balance. In addition, two processing operations occur:

1. The DMPYMT action program updates customer's current balance on the customer master file (DMOMSTR). (Figure B-16, lines 158-159.)

2. The DMPYMT action program adds a payment transaction record to a daily terminal transaction file. (See Figure B-16, lines 169-200 especially lines 185-187.)

Another initiation method

With the second method of initiating the PAYMT transaction, you enter a payment amount different than the payment plan amount next to the customer number that follows the PAYMT transaction code on the screen. Position your cursor next and depress the **TRANSMIT** key as shown in Figure B-10, line 21.

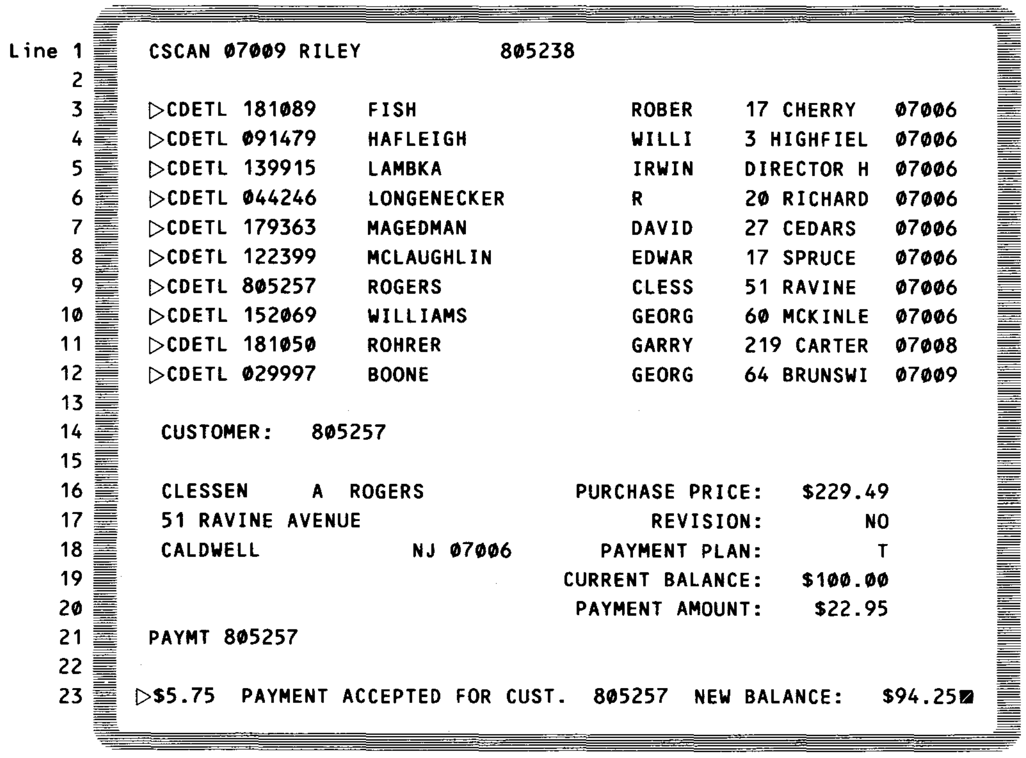


```
line 1   CSCAN Ø7ØØ9 RILEY          8Ø5238
     2
     3   ▷CDETL 181Ø89    FISH               ROBER    17 CHERRY    Ø7ØØ6
     4   ▷CDETL Ø91479    HAFLEIGH           WILLI    3 HIGHFIEL   Ø7ØØ6
     5   ▷CDETL 139915    LAMBKA             IRWIN    DIRECTOR H   Ø7ØØ6
     6   ▷CDETL Ø44246    LONGENECKER        R        2Ø RICHARD   Ø7ØØ6
     7   ▷CDETL 179363    MAGEDMAN           DAVID    27 CEDARS    Ø7ØØ6
     8   ▷CDETL 122399    MCLAUGHLIN         EDWAR    17 SPRUCE    Ø7ØØ6
     9   ▷CDETL 8Ø5257    ROGERS             CLESS    51 RAVINE    Ø7ØØ6
    1Ø   ▷CDETL 152Ø69    WILLIAMS           GEORG    6Ø MCKINLE   Ø7ØØ6
    11   ▷CDETL 181Ø5Ø    ROHRER             GARRY    219 CARTER   Ø7ØØ8
    12   ▷CDETL Ø29997    BOONE              GEORG    64 BRUNSWI   Ø7ØØ9
    13
    14    CUSTOMER:   8Ø5257
    15
    16    CLESSEN     A   ROGERS            PURCHASE PRICE:     $229.49
    17    51 RAVINE AVENUE                        REVISION:          NO
    18    CALDWELL           NJ Ø7ØØ6        PAYMENT PLAN:           T
    19                                     CURRENT BALANCE:     $1ØØ.ØØ
    2Ø                                      PAYMENT AMOUNT:      $22.95
    21   ▷PAYMT 8Ø5257 575■
```

Figure B-10. Second Method for Initiating PAYMT Transaction

Updating payment amount

Suppose you enter the value 575 ($5.75) next to the account number. When you press the **TRANSMIT** key, the result is as shown in Figure B-11.



```
CSCAN 07009 RILEY          805238

▷CDETL 181089    FISH                  ROBER    17 CHERRY   07006
▷CDETL 091479    HAFLEIGH              WILLI    3 HIGHFIEL  07006
▷CDETL 139915    LAMBKA                IRWIN    DIRECTOR H  07006
▷CDETL 044246    LONGENECKER           R        20 RICHARD  07006
▷CDETL 179363    MAGEDMAN              DAVID    27 CEDARS   07006
▷CDETL 122399    MCLAUGHLIN            EDWAR    17 SPRUCE   07006
▷CDETL 805257    ROGERS                CLESS    51 RAVINE   07006
▷CDETL 152069    WILLIAMS              GEORG    60 MCKINLE  07006
▷CDETL 181050    ROHRER                GARRY    219 CARTER  07008
▷CDETL 029997    BOONE                 GEORG    64 BRUNSWI  07009

 CUSTOMER:   805257

 CLESSEN     A  ROGERS            PURCHASE PRICE:   $229.49
 51 RAVINE AVENUE                       REVISION:        NO
 CALDWELL            NJ 07006       PAYMENT PLAN:         T
                                 CURRENT BALANCE:   $100.00
                                  PAYMENT AMOUNT:    $22.95
PAYMT 805257

▷$5.75  PAYMENT ACCEPTED FOR CUST.  805257  NEW BALANCE:   $94.25▨
```

Figure B-11. Result of Entering Different Payment Amount on PAYMT Transaction

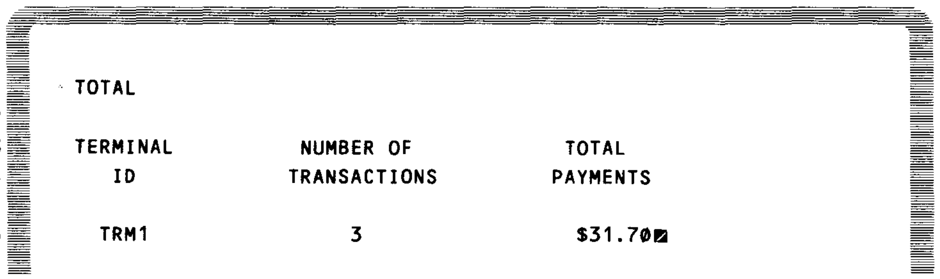
DMPYMT confirms the receipt of payment by issuing a message (Figure B-16, lines 29-32 and 194-197) and applies the entered amount to the customer's new balance (Figure B-16, line 157).

Initiating TOTAL

The last action program, DMTOTL, totals all payment amounts entered for a particular customer. To initiate this audit trail program, you enter the TOTAL transaction code.

Processing TOTAL

Let's assume that in addition to the payment plan amount of $22.95 for account number 805257, you've entered two payments for other customers, one for $5.75 and another for $3.00. You therefore entered three payments at terminal 1 totaling $31.70. By entering the TOTAL transaction code (Figure B-12, line 1), you can obtain an audit report display (Figure B-12, lines 3-6) showing the number of payments and total payment amount initiated from your terminal (TRM1).



```
 TOTAL

 TERMINAL          NUMBER OF          TOTAL
    ID           TRANSACTIONS        PAYMENTS

   TRM1                3              $31.70▨
```

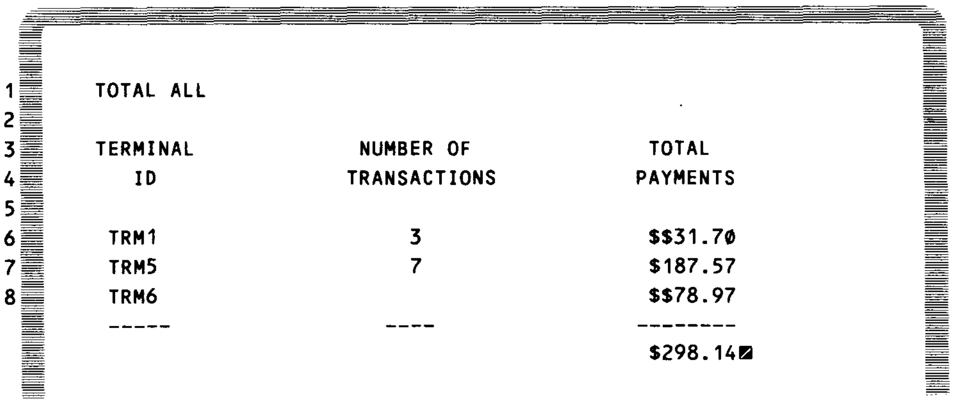
Figure B-12. Result of Initiating the TOTAL Transaction

Processing TOTAL transaction with ALL option

If you enter the option ALL following the transaction code, the DMTOTL action program also can accumulate totals for all transactions and all payments made at all terminals for an entire session.

Suppose three transactions were entered from terminal 1 with total payments of $31.70. Then seven more transactions were entered at terminal 5 totaling $187.57. Finally, four more transactions were made at terminal 6 totaling $78.97 in payments.

When you enter TOTAL ALL at the terminal the DMTOTL action program not only accumulates the total transactions and payments for each terminal but also accumulates a grand total of transactions and payments made in this session. Figure B-13 illustrates the output message generated when you enter the transaction code TOTAL and the option ALL.



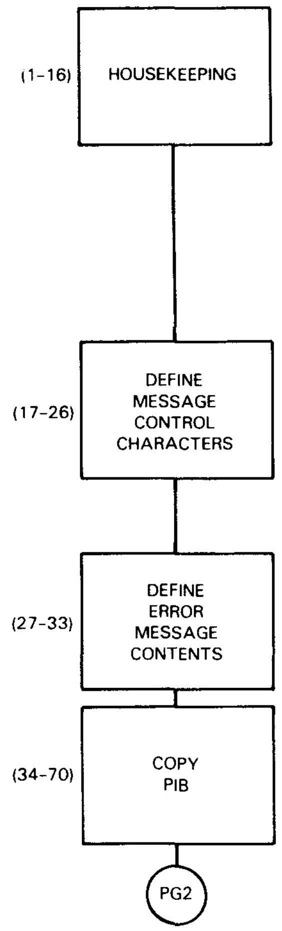
```
 TOTAL ALL

 TERMINAL            NUMBER OF            TOTAL
    ID             TRANSACTIONS          PAYMENTS

   TRM1                  3               $$31.70
   TRM5                  7               $187.57
   TRM6                                  $$78.97
   -----               ----             --------
                                         $298.14▨
```

Figure B-13. Result of Initiating the TOTAL Transaction with ALL Option

Compilations and flowcharts

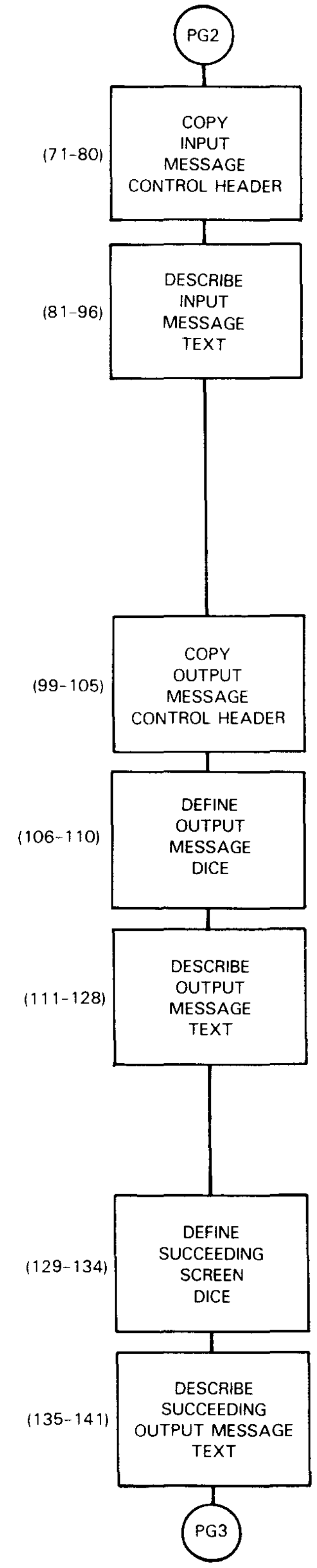
General flowcharts for the coding in DMSCAN, DMDETL, DMPYMT, and DMTOTL action programs (Figures B-14 through B-17) adjoin each program. Program line numbers in parentheses near flowchart boxes represent the lines of coding that implement the process described.

SIMPLE TRANSACTION IN COBOL: DMSCAN PROGRAM


(1-16)
HOUSEKEEPING
(17-26)
DEFINE MESSAGE CONTROL CHARACTERS
(27-33)
DEFINE ERROR MESSAGE CONTENTS
(34-70)
COPY PIB
PG2

```
LINE NO. SOURCE ENTRY

00001  IDENTIFICATION DIVISION.
00002  PROGRAM-ID.  DMSCAN.
00003  AUTHOR.  F.D.F.S.F.  (NYNA 7/76)
00004  DATE-WRITTEN.  7/13/76.
00005  DATE-COMPILED. 82/05/03.
00006  ENVIRONMENT DIVISION.
00007  CONFIGURATION SECTION.
00008  SOURCE-COMPUTER.  UNIVAC-OS3.
00009  OBJECT-COMPUTER.  UNIVAC-OS3.
00010  DATA DIVISION.
00011  WORKING-STORAGE SECTION.
00012  77  DMOALT                PIC X(7)    VALUE 'DMOALT'.
00013  77  OUT-MSG-LEN           PIC 9999    COMP-4        VALUE 768.
00014  77  GE                    PIC X       VALUE 'G'.
00015  77  CSCAN                 PIC X(5)    VALUE 'CSCAN'.
00016  77  CDETL                 PIC X(5)    VALUE 'CDETL'.
00017  01  SCOPE-CHAR.
00018      02 CR                 PIC X         VALUE ='0D'.
00019      02 DLE                PIC X         VALUE ='10'.
00020      02 ESC                PIC X         VALUE ='27'.
00021      02 HT                 PIC X         VALUE ='05'.
00022      02 STX                PIC X         VALUE ='02'.
00023      02 EXT                PIC X         VALUE ='03'.
00024      02 SOE                PIC X         VALUE ='1E'.
00025      02 ONE                PIC X         VALUE ='01'.
00026      02 THREE              PIC X         VALUE ='03'.
00027  01  ERR-MSG-LITS.
00028      02 FILLER             PIC X(19)   VALUE '**INVALID KEY**'.
00029      02 FILLER             PIC X(19)   VALUE '**END OF FILE**'.
00030      02 FILLER             PIC X(19)   VALUE '**INVALID REQUEST**'.
00031      02 FILLER             PIC X(19)   VALUE '**I/O ERROR**'.
00032  01  ERR-MSG-TAB           REDEFINES ERR-MSG-LITS.
00033      02 ERR                PIC X(19)   OCCURS 4.
00034  LINKAGE SECTION.
00035  01  P-I-B.                COPY PIB74.
00036      02  STATUS-CODE                 PIC 9(4) COMP-4.
00037      02  DETAILED-STATUS-CODE        PIC 9(4) COMP-4.
00038      02  RECORD-TYPE REDEFINES DETAILED-STATUS-CODE.
00039          03 PREDICTED-RECORD-TYPE    PIC X.
00040          03 DELIVERED-RECORD-TYPE    PIC X.
00041      02  SUCCESSOR-ID                PIC X(6).
00042      02  TERMINATION-INDICATOR       PIC X.
00043      02  LOCK-ROLLBACK-INDICATOR     PIC X.
00044      02  TRANSACTION-ID.
00045          03 YEAR                     PIC 9(4) COMP-4.
00046          03 TODAY                    PIC 9(4) COMP-4.
00047          03 HR-MIN-SEC               PIC 9(9) COMP-4.
00048      02  DATA-DEF-REC-NAME           PIC X(7).
00049      02  DEFINED-FILE-NAME           PIC X(7).
00050      02  STANDARD-MSG-LINE-LENGTH    PIC 9(4) COMP-4.
00051      02  STANDARD-MSG-NUMBER-LINES   PIC 9(4) COMP-4.
00052      02  WORK-AREA-LENGTH            PIC 9(4) COMP-4.
00053      02  CONTINUITY-DATA-INPUT-LENGTH  PIC 9(4) COMP-4.
00054      02  CONTINUITY-DATA-OUTPUT-LENGTH  PIC 9(4) COMP-4.
00055      02  WORK-AREA-INC               PIC 9(4) COMP-4.
00056      02  CONTINUITY-DATA-AREA-INC    PIC 9(4) COMP-4.
00057      02  SUCCESS-UNIT-ID.
00058          03 TRANSACTION-DATE.
00059             04 YEAR                  PIC 99.
00060             04 MONTH                 PIC 99.
00061             04 TODAY                 PIC 99.
00062          03 TIME-OF-DAY.
00063             04 HOUR                  PIC 99.
00064             04 MINUTE                PIC 99.
00065             04 SECOND                PIC 99.
00066          03 FILLER                   PIC XXX.
00067      02  SOURCE-TERMINAL-CHARS.
00068          03 SOURCE-TERMINAL-TYPE  PIC X.
00069          03 SOURCE-TERM-MSG-LINE-LENGTH      PIC 9(4) COMP-4.
00070          03 SOURCE-TERM-MSG-NUMBER-LINES    PIC 9(4) COMP-4.
```

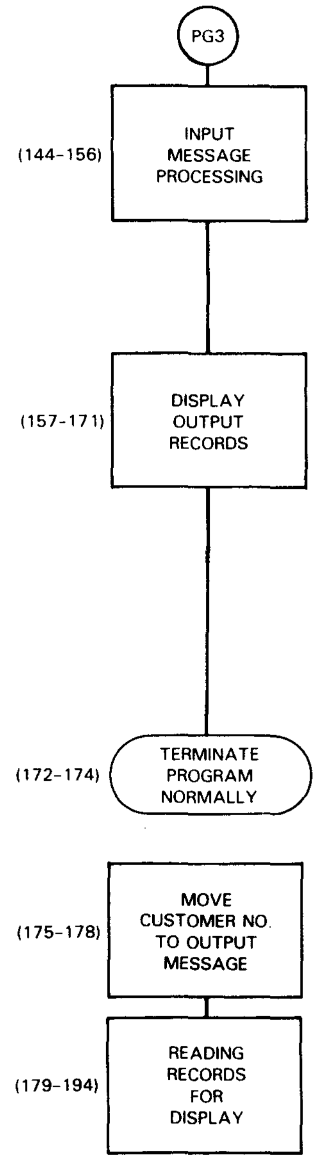
Figure B-14. Sample COBOL Action Program DMSCAN (Part 1 of 3)


PG2
(71-80)
COPY INPUT MESSAGE CONTROL HEADER
(81-96)
DESCRIBE INPUT MESSAGE TEXT
(99-105)
COPY OUTPUT MESSAGE CONTROL HEADER
(106-110)
DEFINE OUTPUT MESSAGE DICE
(111-128)
DESCRIBE OUTPUT MESSAGE TEXT
(129-134)
DEFINE SUCCEEDING SCREEN DICE
(135-141)
DESCRIBE SUCCEEDING OUTPUT MESSAGE TEXT
PG3

```
00071   01    I-M-A.                   COPY IMA74.
00072         02 SOURCE-TERMINAL-ID                  PIC X(4).
00073         02 DATE-TIME-STAMP.
00074            03 YEAR                             PIC 9(4) COMP-4.
00075            03 TODAY                            PIC 9(4) COMP-4.
00076            03 HR-MIN-SEC                       PIC 9(9) COMP-4.
00077         02 TEXT-LENGTH                         PIC 9(4) COMP-4.
00078         02 AUXILIARY-DEV-ID.
00079            03 FILLER                          PIC X.
00080            03 AUX-DEV-NO                      PIC X.
00081         02 FILLER                PIC X(6).
00082         02 ZIP-IN                PIC X(5).
00083         02 FILLER                PIC X.
00084         02 LN-CUSTID.
00085             03 LN-IN             PIC X(17).
00086             03 FILLER            PIC X(8).
00087         02 RED-LN-CUSTID         REDEFINES LN-CUSTID.
00088             03 CHAR              PIC X        OCCURS 25   INDEXED BY I.
00089   01    ALT-REC.
00090         02 ALT-KEY.
00091             03 ZIP               PIC X(5).
00092             03 LN                PIC X(17).
00093             03 CUSTOMER.
00094                 04 KEY-CHAR      PIC X        OCCURS 6    INDEXED BY J.
00095         02 FN-5                  PIC X(5).
00096         02 ADDR-ID               PIC X(10).
00097   01    O-M-A.                   COPY OMA74.
00098         02 DESTINATION-TERMINAL-ID      PIC X(4).
00099                  02 SFS-OPTIONS                       PIC X(2).
00100                  02 FILLER                            PIC X(2).
00101         02 CONTINUOUS-OUTPUT-CODE       PIC X(4).
00102         02 TEXT-LENGTH                PIC 9(4)    COMP-4.
00103         02 AUXILIARY-DEVICE-ID.
00104            03 AUX-FUNCTION              PIC X.
00105            03 AUX-DEVICE-NO             PIC X.
00106         02 DICE-1.
00107             03 TEN-1             PIC X.
00108             03 FN-1              PIC X.
00109             03 Y-1               PIC X.
00110             03 X-1               PIC X.
00111         02 DISPLAY-TABLE.
00112             03 CUST-LINE                      OCCURS 10   INDEXED BY K.
00113                 04 SOE-OUT       PIC X.
00114                 04 TRAN-OUT      PIC X(5).
00115                 04 FILLER        PIC X.
00116                 04 CUST-OUT      PIC X(6).
00117                 04 FILLER        PIC X(5).
00118                 04 LN-OUT        PIC X(17).
00119                 04 FILLER        PIC XX.
00120                 04 ESC-OUT       PIC X.
00121                 04 HT-OUT        PIC X.
00122                 04 FILLER        PIC XX.
00123                 04 FN-OUT        PIC X(5).
00124                 04 FILLER        PIC X(5).
00125                 04 ADDR-OUT      PIC X(10).
00126                 04 FILLER        PIC X(5).
00127                 04 ZIP-OUT       PIC X(5).
00128                 04 CR-OUT        PIC X.
00129         02 TOP-OF-SCREEN.
00130             03 DICE-2.
00131                 04 TEN-2         PIC X.
00132                 04 FN-2          PIC X.
00133                 04 Y-2           PIC X.
00134                 04 X-2           PIC X.
00135             03 NEXT-TRAN         PIC X(5).
00136             03 FILLER            PIC X.
00137             03 NEXT-ZIP          PIC X(5).
00138             03 FILLER            PIC X.
00139             03 NEXT-LN           PIC X(17).
00140             03 FILLER            PIC X.
00141             03 NEXT-CUST         PIC X(6).
```

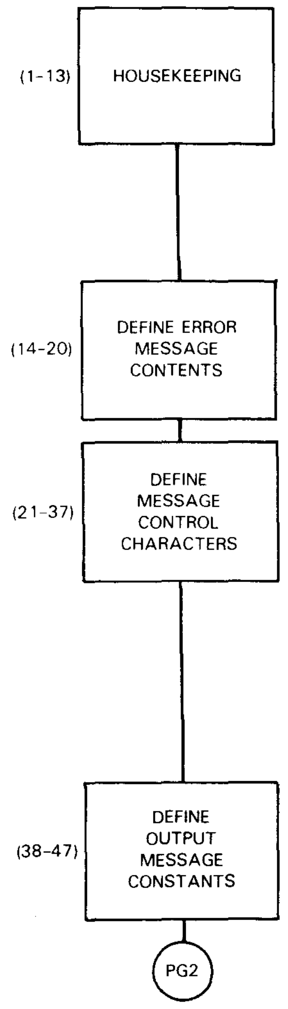
Figure B-14. Sample COBOL Action Program DMSCAN (Part 2 of 3)

SIMPLE TRANSACTION IN COBOL: DMSCAN PROGRAM



```
00142  PROCEDURE DIVISION
00143      USING P-1-B I-M-A ALT-REC O-M-A.
00144  BEGIN-PROC.
00145      MOVE SPACES TO DISPLAY-TABLE TOP-OF-SCREEN.
00146      MOVE DLE TO TEN-1 TEN-2.
00147      MOVE EXT TO FN-1.
00148      MOVE STX TO FN-2.
00149      MOVE THREE TO Y-1.
00150      MOVE ONE TO X-1 X-2 Y-2.
00151      IF TEXT-LENGTH IN I-M-A > 24
00152          THEN SET I TO TEXT-LENGTH IN I-M-A
00153                SET I DOWN BY 22
00154                IF CHAR (I) IS NUMERIC AND CHAR (I + 5) IS NUMERIC
00155                    THEN PERFORM MV-ID
00156                               VARYING J FROM 1 BY 1 UNTIL J > 6.
00157      MOVE LN-IN TO LN.
00158      MOVE ZIP-IN TO ZIP.
00159      CALL 'SETL'
00160          USING DMOALT GE ALT-KEY.
00161      IF STATUS-CODE > 0
00162          THEN MOVE ERR (STATUS-CODE) TO CUST-LINE (I)
00163                GO TO END-PROG.
00164      CALL 'GET'
00165          USING DMOALT ALT-REC.
00166      PERFORM READ-LOOP
00167          VARYING K FROM 1 BY 1 UNTIL K > 10.
00168      MOVE CSCAN TO NEXT-TRAN.
00169      MOVE ZIP TO NEXT-ZIP.
00170      MOVE LN TO NEXT-LN.
00171      MOVE CUSTOMER TO NEXT-CUST.
00172  END-PROG.
00173      MOVE OUT-MSG-LEN TO TEXT-LENGTH IN O-M-A.
00174      CALL 'RETURN'.
00175  MV-ID.
00176      MOVE CHAR (I) TO KEY-CHAR (J).

00177      MOVE SPACE TO CHAR (I).
00178      SET I UP BY 1.
00179  READ-LOOP.
00180      IF STATUS-CODE > 0
00181          THEN MOVE ERR (STATUS-CODE) TO CUST-LINE (K)

00182                GO TO END-PROG.
00183      MOVE SOE TO SOE-OUT (K).
00184      MOVE CDETL TO TRAN-OUT (K).
00185      MOVE CUSTOMER TO CUST-OUT (K).
00186      MOVE LN TO LN-OUT (K).
00187      MOVE ESC TO ESC-OUT (K).
00188      MOVE HT TO HT-OUT (K).
00189      MOVE FN-5 TO FN-OUT (K).
00190      MOVE ADDR-10 TO ADDR-OUT (K).
00191      MOVE ZIP TO ZIP-OUT (K).
00192      MOVE CR TO CR-OUT (K).
00193      CALL 'GET'
00194          USING DMOALT ALT-REC.
```

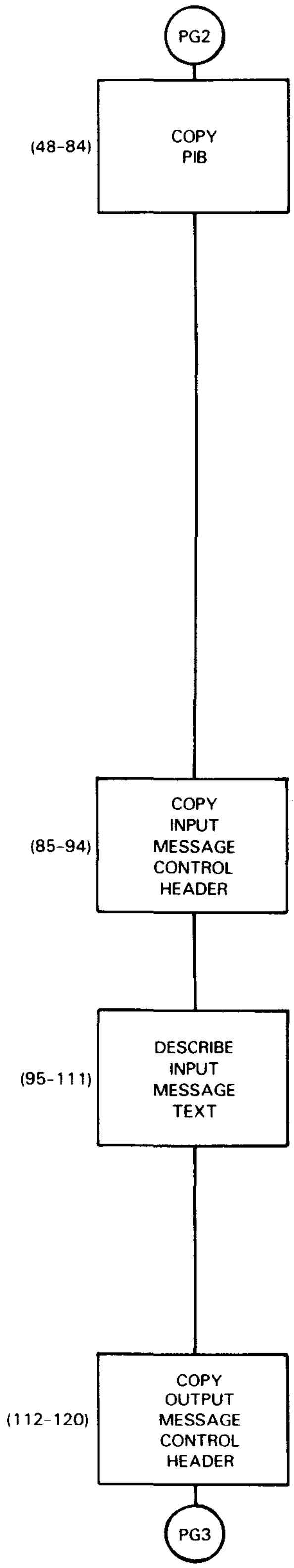
Figure B-14. Sample COBOL Action Program DMSCAN (Part 3 of 3)


(1-13)
HOUSEKEEPING
(14-20)
DEFINE ERROR MESSAGE CONTENTS
(21-37)
DEFINE MESSAGE CONTROL CHARACTERS
(38-47)
DEFINE OUTPUT MESSAGE CONSTANTS
PG2

```
LINE NO. SOURCE ENTRY

00001   IDENTIFICATION DIVISION.
00002   PROGRAM-ID.  DMDETL.
00003   AUTHOR.  E.O.F.S.F.  (NYNA 7/76)
00004   DATE-WRITTEN.  7/16/76.
00005   ENVIRONMENT DIVISION.
00006   CONFIGURATION SECTION.
00007   SOURCE-COMPUTER.  UNIVAC-OS3.
00008   OBJECT-COMPUTER.  UNIVAC-OS3.
00009   DATA DIVISION.
00010   WORKING-STORAGE SECTION.
00011   77  DMOMSTR                 PIC X(7)    VALUE 'DMOMSTR'.
00012   77  OUT-MSG-LEN             PIC 9999    COMP-4      VALUE 327.
00013   77  ERR-MSG-LEN             PIC 9999    COMP-4      VALUE  27.
00014   01  ERR-MSG-LITS.
00015       02 FILLER               PIC X(19)   VALUE '**INVALID KEY**'.
00016       02 FILLER               PIC X(19)   VALUE '**END OF FILE**'.
00017       02 FILLER               PIC X(19)   VALUE '**INVALID REQUEST**'.
00018       02 FILLER               PIC X(19)   VALUE '**I/O ERROR**'.
00019   01  ERR-MSG-TBL             REDEFINES ERR-MSG-LITS.
00020       02 ERR-MSG              PIC X(19)   OCCURS 4.
00021   01  SCOPE-CHAR.
00022       02 DLE                  PIC X          VALUE ='10'.
00023       02 STX                  PIC X          VALUE ='02'.
00024       02 EXT                  PIC X          VALUE ='03'.
00025       02 SOE                  PIC X          VALUE ='1E'.
00026       02 CR                   PIC X          VALUE ='0D'.
00027       02 ONE                  PIC X          VALUE ='01'.
00028       02 THREE                PIC X          VALUE ='03'.
00029       02 FIVE                 PIC X          VALUE ='05'.
00030       02 EIGHT                PIC X          VALUE ='08'.
00031       02 NINE                 PIC X          VALUE ='09'.
00032       02 FIFTEEN              PIC X          VALUE ='0F'.
00033       02 SIXTEEN              PIC X          VALUE ='10'.
00034       02 EIGHTEEN             PIC X          VALUE ='12'.
00035       02 TWENTY-ONE           PIC X          VALUE ='15'.
00036       02 TWENTY-TWO           PIC X          VALUE ='16'.
00037       02 FORTY-ONE            PIC X          VALUE ='29'.
00038   01  MSG-LITS.
00039       02 CUSTOMER-LIT         PIC X(11)   VALUE 'CUSTOMER #:'.
00040       02 PURCH-PR             PIC X(15)   VALUE 'PURCHASE PRICE:'.
00041       02 PAYMT-PLAN           PIC X(13)   VALUE 'PAYMENT PLAN:'.
00042       02 PAYMT-AMT            PIC X(15)   VALUE 'PAYMENT AMOUNT:'.
00043       02 BAL                  PIC X(16)   VALUE 'CURRENT BALANCE:'.
00044       02 REVISION             PIC X(9)    VALUE 'REVISION:'.
00045       02 YES                  PIC XXX     VALUE 'YES'.
00046       02 KNO                  PIC XXX     VALUE ' NO'.
00047       02 PAYMT                PIC X(5)    VALUE 'PAYMT'.
```

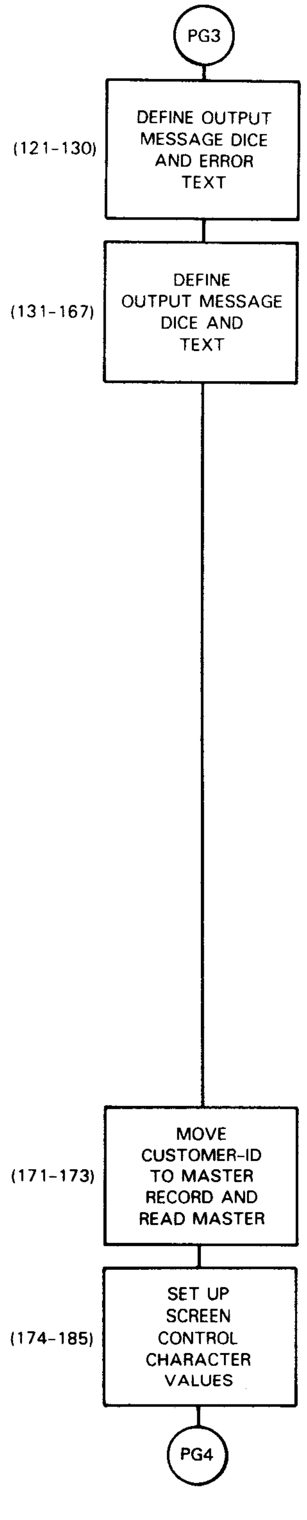
Figure B-15. Sample COBOL Action Program DMDETL (Part 1 of 4)

SIMPLE TRANSACTION IN COBOL: DMDETL PROGRAM


PG2
(48-84)
COPY PIB
(85-94)
COPY INPUT MESSAGE CONTROL HEADER
(95-111)
DESCRIBE INPUT MESSAGE TEXT
(112-120)
COPY OUTPUT MESSAGE CONTROL HEADER
PG3

```
00048   LINKAGE SECTION.
00049   01   P-I-B.                 COPY PIB74.
00050        02  STATUS-CODE                 PIC 9(4) COMP-4.
00051        02  DETAILED-STATUS-CODE        PIC 9(4) COMP-4.
00052        02  RECORD-TYPE REDEFINES DETAILED-STATUS-CODE.
00053            03 PREDICTED-RECORD-TYPE    PIC X.
00054            03 DELIVERED-RECORD-TYPE    PIC X.
00055        02  SUCCESSOR-ID                PIC X(6).
00056        02  TERMINATION-INDICATOR       PIC X.
00057        02  LOCK-ROLLBACK-INDICATOR     PIC X.
00058        02  TRANSACTION-ID.
00059            03 YEAR                     PIC 9(4) COMP-4.
00060            03 TODAY                    PIC 9(4) COMP-4.
00061            03 HR-MIN-SEC               PIC 9(9) COMP-4.
00062        02  DATA-DEF-REC-NAME           PIC X(7).
00063        02  DEFINED-FILE-NAME           PIC X(7).
00064        02  STANDARD-MSG-LINE-LENGTH    PIC 9(4) COMP-4.
00065        02  STANDARD-MSG-NUMBER-LINES   PIC 9(4) COMP-4.
00066        02  WORK-AREA-LENGTH            PIC 9(4) COMP-4.
00067        02  CONTINUITY-DATA-INPUT-LENGTH  PIC 9(4) COMP-4.
00068        02  CONTINUITY-DATA-OUTPUT-LENGTH  PIC 9(4) COMP-4.
00069        02  WORK-AREA-INC               PIC 9(4) COMP-4.
00070        02  CONTINUITY-DATA-AREA-INC    PIC 9(4) COMP-4.
00071        02  SUCCESS-UNIT-ID.
00072            03 TRANSACTION-DATE.
00073               04 YEAR                  PIC 99.
00074               04 MONTH                 PIC 99.
00075               04 TODAY                 PIC 99.
00076            03 TIME-OF-DAY.
00077               04 HOUR                  PIC 99.
00078               04 MINUTE                PIC 99.
00079               04 SECOND                PIC 99.
00080            03 FILLER                   PIC XXX.
00081        02  SOURCE-TERMINAL-CHARS.
00082            03 SOURCE-TERMINAL-TYPE  PIC X.
00083            03 SOURCE-TERM-MSG-LINE-LENGTH   PIC 9(4) COMP-4.
00084            03 SOURCE-TERM-MSG-NUMBER-LINES  PIC 9(4) COMP-4.
00085   01   I-M-A.                 COPY IMA74.
00086        02 SOURCE-TERMINAL-ID                  PIC X(4).
00087        02 DATE-TIME-STAMP.
00088           03 YEAR                             PIC 9(4) COMP-4.
00089           03 TODAY                            PIC 9(4) COMP-4.
00090           03 HR-MIN-SEC                       PIC 9(9) COMP-4.
00091        02 TEXT-LENGTH                         PIC 9(4) COMP-4.
00092        02 AUXILIARY-DEV-ID.
00093           03 FILLER                          PIC X.
00094           03 AUX-DEV-NO                      PIC X.
00095        02 FILLER              PIC X(6).
00096        02 CUSTID              PIC X(6).
00097        02 FILLER              PIC X(24).
00098   01   MSTR-REC.
00099        02 CUSTNO              PIC X(6).
00100        02 FN                  PIC X(10).
00101        02 MI                  PIC X.
00102        02 LN                  PIC X(17).
00103        02 ADDR                PIC X(23).
00104        02 CITY                PIC X(18).
00105        02 ST                  PIC XX.
00106        02 ZIP                 PIC X(5).
00107        02 PURCHASE-PRICE      PIC 999V99.
00108        02 PAY-PLAN            PIC X.
00109        02 PAY-AMT             PIC 999V99.
00110        02 BALANCE             PIC 999V99.
00111        02 UPDATE              PIC X.
00112   01   O-M-A.                 COPY OMA74.
00113        02 DESTINATION-TERMINAL-ID       PIC X(4).
00114                 02 SFS-OPTIONS                   PIC X(2).
00115                 02 FILLER                        PIC X(2).
00116        02 CONTINUOUS-OUTPUT-CODE        PIC X(4).
00117        02 TEXT-LENGTH                 PIC 9(4)    COMP-4.
00118        02 AUXILIARY-DEVICE-ID.
00119           03 AUX-FUNCTION               PIC X.
00120           03 AUX-DEVICE-NO              PIC X.
```

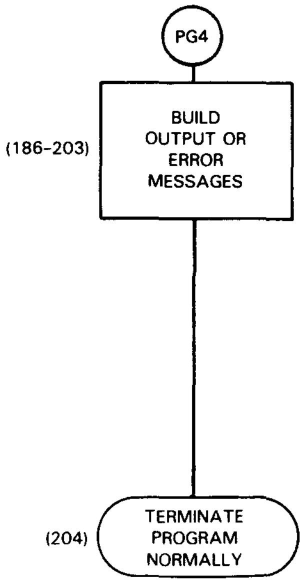
Figure B-15. Sample COBOL Action Program DMDETL (Part 2 of 4)


PG3
(121-130)
DEFINE OUTPUT MESSAGE DICE AND ERROR TEXT
(131-167)
DEFINE OUTPUT MESSAGE DICE AND TEXT
(171-173)
MOVE CUSTOMER-ID TO MASTER RECORD AND READ MASTER
(174-185)
SET UP SCREEN CONTROL CHARACTER VALUES
PG4

```
00121  02 DICE-1.
00122       03 DLE-1          PIC X.
00123       03 EXT-1          PIC X.
00124       03 Y-1            PIC X.
00125       03 X-1            PIC X.
00126  02 ERR-LINE            PIC X(19).
00127  02 CUST                REDEFINES ERR-LINE.
00128       03 CUSTOMER       PIC X(13).
00129       03 CUST-1         PIC X(6).
00130  02 DICE-2.
00131       03 DLE-2          PIC X.
00132       03 STX-2          PIC X.
00133       03 Y-2            PIC X.
00134       03 X-2            PIC X.
00135  02 FN                  PIC X(11).
00136  02 MI                  PIC XXX.
00137  02 LN                  PIC X(27).
00138  02 PURCH-PR            PIC X(17).
00139  02 PURCHASE-PRICE      PIC $$$9.99.
00140  02 CR-1                PIC X.
00141  02 ADDR                PIC X(47).
00142  02 REVISION            PIC X(15).
00143  02 UPD                 PIC XXX.
00144  02 CR-2                PIC X.
00145  02 CITY                PIC X(20).
00146  02 ST                  PIC XXXX.
00147  02 ZIP                 PIC X(19).
00148  02 PAYMT-PLAN          PIC X(21).
00149  02 PAY-PLAN            PIC X.
00150  02 DICE-3.
00151       03 DLE-3          PIC X.
00152       03 STX-3          PIC X.
00153       03 Y-3            PIC X.
00154       03 X-3            PIC X.
00155       02 BAL                PIC X(18).
00156       02 BALANCE            PIC $$$9.99.
00157       02 CR-3               PIC X.
00158       02 SOE-OUT            PIC X.
00159       02 PAYMT              PIC X(6).
00160       02 CUST-2             PIC X(34).
00161       02 PAYMT-AMT          PIC X(17).
00162       02 PAY-AMT            PIC $$$9.99.
00163       02 DICE-4.
00164            03 DLE-4         PIC X.
00165            03 STX-4         PIC X.
00166            03 Y-4           PIC X.
00167            03 X-4           PIC X.
00168  PROCEDURE DIVISION
00169       USING F-I-B I-M-A MSTR-REC O-M-A.

00170  BEGIN-PROC.
00171       MOVE CUSTID TO CUSTNO.
00172       CALL 'GET'
00173            USING DMOMSTR MSTR-REC CUSTNO.
00174       MOVE DLE TO DLE-1.
00175       MOVE EXT TO EXT-1.
00176       IF STANDARD-MSG-NUMBER-LINES NOT > 12
00177            THEN MOVE THREE TO Y-1
00178                 MOVE FIVE TO Y-2
00179                 MOVE EIGHT TO Y-3
00180                 MOVE NINE TO Y-4
00181            ELSE MOVE SIXTEEN TO Y-1
00182                 MOVE EIGHTEEN TO Y-2
00183                 MOVE TWENTY-ONE TO Y-3
00184                 MOVE TWENTY-TWO TO Y-4.
00185       MOVE ONE TO X-1.
```

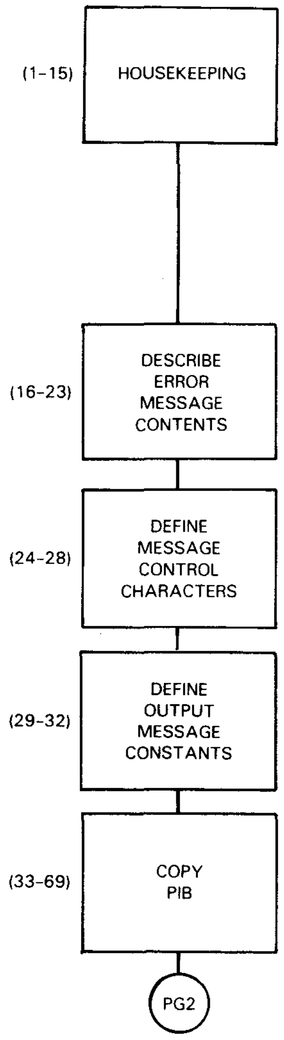
Figure B-15. Sample COBOL Action Program DMDETL (Part 3 of 4)

SIMPLE TRANSACTION IN COBOL: DMDETL PROGRAM



```
00186         IF STATUS-CODE = 0
00187             THEN MOVE CR TO CR-1 CR-2 CR-3

00188                  MOVE DLE TO DLE-2 DLE-3 DLE-4

00189                  MOVE STX TO STX-2 STX-3 STX-4

00190                  MOVE ONE TO X-2
00191                  MOVE FORTY-ONE TO X-3
00192                  MOVE FIFTEEN TO X-4
00193                  MOVE SOE TO SOE-OUT
00194                  MOVE CUSTOMER-LIT TO CUSTOMER
00195                  MOVE CUSTNO TO CUST-1 CUST-2
00196                  MOVE CORR MSG-LITS TO O-M-A
00197                  MOVE CORR MSTR-REC TO O-M-A
00198                  MOVE OUT-MSG-LEN TO TEXT-LENGTH IN O-M-A
00199                  IF UPDATE = 'Y'
00200                      THEN MOVE YES TO UPD
00201                      ELSE MOVE KNO TO UPD
00202             ELSE MOVE ERR-MSG (STATUS-CODE) TO ERR-LINE
00203                  MOVE ERR-MSG-LEN TO TEXT-LENGTH IN O-M-A.
00204         CALL 'RETURN'.
```

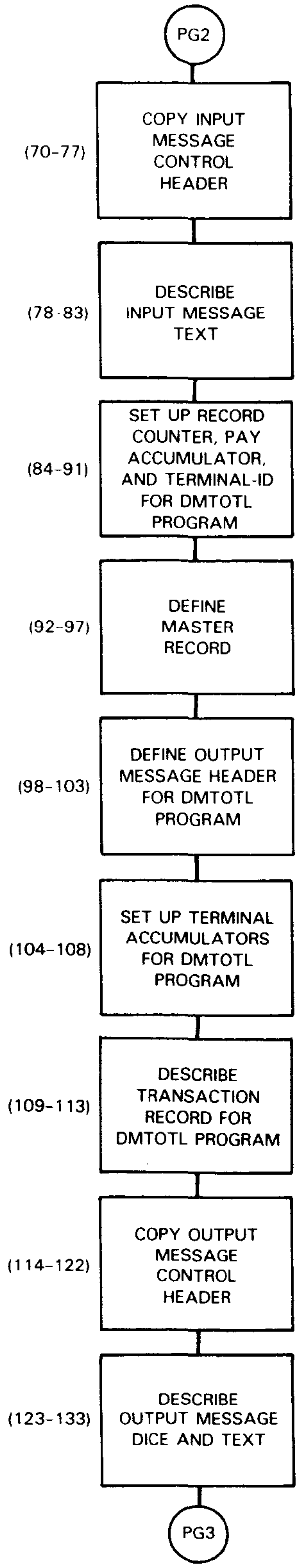
Figure B-15. Sample COBOL Action Program DMDETL (Part 4 of 4)


(1-15)
HOUSEKEEPING
(16-23)
DESCRIBE ERROR MESSAGE CONTENTS
(24-28)
DEFINE MESSAGE CONTROL CHARACTERS
(29-32)
DEFINE OUTPUT MESSAGE CONSTANTS
(33-69)
COPY PIB
PG2

```
LINE NO. SOURCE ENTRY

00001 IDENTIFICATION DIVISION.
00002 PROGRAM-ID. DMPYMT.
00003 AUTHOR. E.O.F.S.F. (NYNA 7/76)
00004 DATE-WRITTEN. 7/19/76.
00005 ENVIRONMENT DIVISION.
00006 CONFIGURATION SECTION.
00007 SOURCE-COMPUTER. UNIVAC-OS3.
00008 OBJECT-COMPUTER. UNIVAC-OS3.
00009 DATA DIVISION.
00010 WORKING-STORAGE SECTION.
00011 77  DMOMSTR                 PIC X(7)    VALUE 'DMOMSTR'.
00012 77  DMOXAC                  PIC X(7)    VALUE 'DMOXACT'.
00013 77  TWELVE                  PIC X       VALUE ='12'.
00014 77  OUT-MSG-LEN             PIC 9999    COMP-4       VALUE 78.
00015 77  ERR-MSG-LEN             PIC 9999    COMP-4       VALUE 29.
00016 01  ERR-MSG-LITS.
00017     02 FILLER               PIC X(21)   VALUE '**INVALID KEY**'.
00018     02 FILLER               PIC X(21)   VALUE '**END OF FILE**'.
00019     02 FILLER               PIC X(21)   VALUE '**INVALID REQUEST**'.
00020     02 FILLER               PIC X(21)   VALUE '**I/O ERROR**'.
00021     02 FILLER               PIC X(21)  VALUE '**PAYMENT > BALANCE**'.
00022 01  ERR-MSG-TBL             REDEFINES ERR-MSG-LITS.
00023     02 ERR-MSG              PIC X(21)   OCCURS 5.
00024 01  DICE-CODE.
00025     02 DLE                  PIC X         VALUE ='10'.
00026     02 STX                  PIC X         VALUE ='02'.
00027     02 TWENTY-FOUR          PIC X         VALUE ='18'.
00028     02 ONE                  PIC X         VALUE ='01'.
00029 01  MSG-LIT.
00030     02 FILLER               PIC X(20)   VALUE ' PAYMENT ACCEPTED FO'.
00031     02 FILLER               PIC X(22)   VALUE 'R CUSTOMER #'.
00032     02 FILLER               PIC X(14)   VALUE 'NEW BALANCE:'.
00033 LINKAGE SECTION.
00034 01  P-I-B.                  COPY PIB74.
00035     02  STATUS-CODE                   PIC 9(4) COMP-4.
00036     02  DETAILED-STATUS-CODE          PIC 9(4) COMP-4.
00037     02  RECORD-TYPE REDEFINES DETAILED-STATUS-CODE.
00038         03 PREDICTED-RECORD-TYPE  PIC X.
00039         03 DELIVERED-RECORD-TYPE  PIC X.
00040     02  SUCCESSOR-ID                  PIC X(6).
00041     02  TERMINATION-INDICATOR         PIC X.
00042     02  LOCK-ROLLBACK-INDICATOR       PIC X.
00043     02  TRANSACTION-ID.
00044         03 YEAR                       PIC 9(4) COMP-4.
00045         03 TODAY                      PIC 9(4) COMP-4.
00046         03 HR-MIN-SEC                 PIC 9(9) COMP-4.
00047     02  DATA-DEF-REC-NAME             PIC X(7).
00048     02  DEFINED-FILE-NAME             PIC X(7).
00049     02  STANDARD-MSG-LINE-LENGTH      PIC 9(4) COMP-4.
00050     02  STANDARD-MSG-NUMBER-LINES     PIC 9(4) COMP-4.
00051     02  WORK-AREA-LENGTH              PIC 9(4) COMP-4.
00052     02  CONTINUITY-DATA-INPUT-LENGTH  PIC 9(4) COMP-4.
00053     02  CONTINUITY-DATA-OUTPUT-LENGTH  PIC 9(4) COMP-4.
00054     02  WORK-AREA-INC                 PIC 9(4) COMP-4.
00055     02  CONTINUITY-DATA-AREA-INC      PIC 9(4) COMP-4.
00056     02  SUCCESS-UNIT-ID.
00057         03 TRANSACTION-DATE.
00058            04 YEAR                    PIC 99.
00059            04 MONTH                   PIC 99.
00060            04 TODAY                   PIC 99.
00061         03 TIME-OF-DAY.
00062            04 HOUR                    PIC 99.
00063            04 MINUTE                  PIC 99.
00064            04 SECOND                  PIC 99.
00065         03 FILLER                     PIC XXX.
00066     02  SOURCE-TERMINAL-CHARS.
00067         03 SOURCE-TERMINAL-TYPE  PIC X.
00068         03 SOURCE-TERM-MSG-LINE-LENGTH   PIC 9(4) COMP-4.
00069         03 SOURCE-TERM-MSG-NUMBER-LINES  PIC 9(4) COMP-4.
```

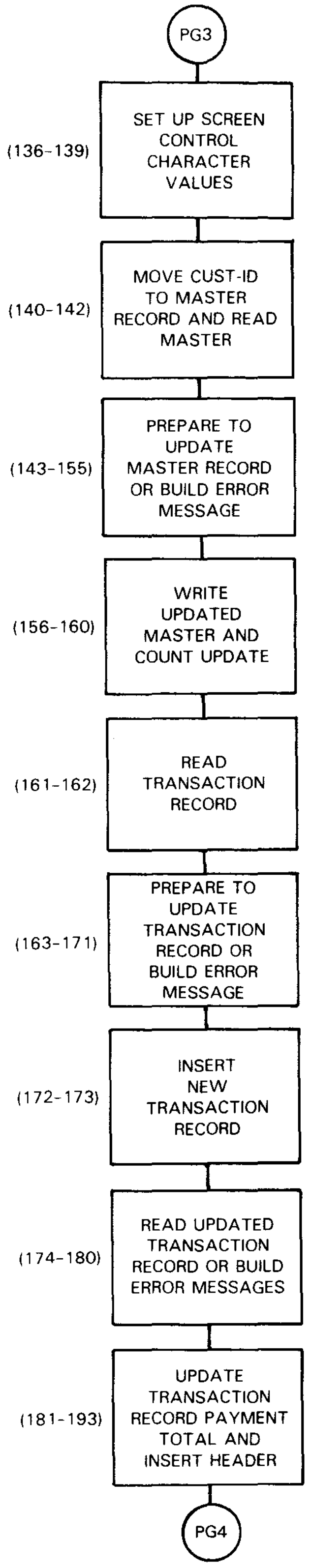
Figure B–16. Sample COBOL Action Program DMPYMT (Part 1 of 4)

SIMPLE TRANSACTION IN COBOL: DMPYMT PROGRAM


PG2
(70-77) COPY INPUT MESSAGE CONTROL HEADER
(78-83) DESCRIBE INPUT MESSAGE TEXT
(84-91) SET UP RECORD COUNTER, PAY ACCUMULATOR, AND TERMINAL-ID FOR DMTOTL PROGRAM
(92-97) DEFINE MASTER RECORD
(98-103) DEFINE OUTPUT MESSAGE HEADER FOR DMTOTL PROGRAM
(104-108) SET UP TERMINAL ACCUMULATORS FOR DMTOTL PROGRAM
(109-113) DESCRIBE TRANSACTION RECORD FOR DMTOTL PROGRAM
(114-122) COPY OUTPUT MESSAGE CONTROL HEADER
(123-133) DESCRIBE OUTPUT MESSAGE DICE AND TEXT
PG3

```
00070  01   I-M-A.                   COPY IMA74.
00071       02 SOURCE-TERMINAL-ID                   PIC X(4).
00072       02 DATE-TIME-STAMP.
00073          03 YEAR                              PIC 9(4) COMP-4.
00074          03 TODAY                             PIC 9(4) COMP-4.
00075          03 HR-MIN-SEC                        PIC 9(9) COMP-4.
00076       02 TEXT-LENGTH                          PIC 9(4) COMP-4.
00077       02 AUXILIARY-DEV-ID.
00078          03 FILLER                           PIC X.
00079          03 AUX-DEV-NO                       PIC X.
00080       02 FILLER              PIC X(6).
00081       02 CUSTID              PIC X(6).
00082       02 FILLER              PIC X.
00083       02 MSG-PAY.
00084           03 MSG-CHAR        PIC X         OCCURS 7     INDEXED BY I.
00085       02 FILLER              PIC X.
00086  01   W-A.
00087       02 REC-CTR             PIC 9(12)     COMP-4.
00088       02 PAY-AREA.
00089           03 PAY-CHAR        PIC X         OCCURS 5     INDEXED BY J.
00090       02 PAYMENT             REDEFINES PAY-AREA       PIC 999V99.
00091       02 TRM.
00092           03 FILLER          PIC XXX.
00093           03 TRM-NUM         PIC 9.
00094       02 MSTR-REC.
00095           03 CUSTNO          PIC X(6).
00096           03 FILLER          PIC X(82).
00097           03 PAY-AMT         PIC 999V99.
00098           03 BALANCE         PIC 999V99.
00099           03 FILLER          PIC X.
00100       02 XAC-HDR.
00101           03 YEAR            PIC 9999      COMP-4.
00102           03 TO-DAY          PIC 9999      COMP-4.
00103           03 PTR-FIRST       PIC 9999      COMP-4.
00104           03 PTR-NEXT        PIC 9999      COMP-4.
00105           03 FILLER          PIC X(8).
00106       02 XAC-TRM.
00107           03 TRM-ID          PIC 9.
00108           03 XAC-CTR         PIC 9999.
00109           03 TRM-TOTAL       PIC 9(5)V99.
00110           03 FILLER          PIC XXXX.
00111       02 XAC-REC             REDEFINES XAC-TRM.
00112           03 TRM-XAC         PIC 9.
00113           03 XAC-TIME        PIC 9(9)      COMP-4.
00114           03 CUSTOMER        PIC X(6).
00115           03 XAC-AMT         PIC 999V99.
00116  01   O-M-A.                 COPY OMA74.
00117       02 DESTINATION-TERMINAL-ID        PIC X(4).
00118                 02 SFS-OPTIONS                       PIC X(2).
00119                 02 FILLER                            PIC X(2).
00120       02 CONTINUOUS-OUTPUT-CODE         PIC X(4).
00121       02 TEXT-LENGTH                 PIC 9(4)     COMP-4.
00122       02 AUXILIARY-DEVICE-ID.
00123          03 AUX-FUNCTION                PIC X.
00124          03 AUX-DEVICE-NO               PIC X.
00125       02 DICE-OUT.
00126           03 FILLER          PIC XX.
00127           03 DICE-Y          PIC X.
00128           03 FILLER          PIC X.
00129       02 OUT-MSG.
00130           03 PAY-OUT         PIC $$$9.99.
00131           03 LIT-OUT.
00132               04 FILLER      PIC X(32).
00133               04 CUST-OUT    PIC X(6).
00134               04 FILLER      PIC X(18).
00135           03 NEW-BAL         PIC $$$9.99.
```

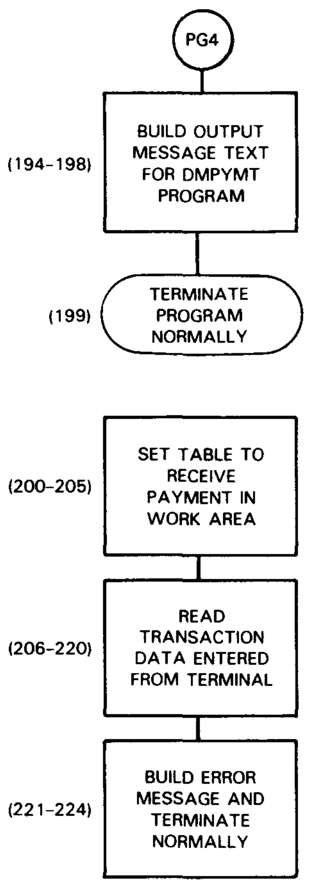
Figure B-16. Sample COBOL Action Program DMPYMT (Part 2 of 4)


PG3
(136-139)
SET UP SCREEN CONTROL CHARACTER VALUES
(140-142)
MOVE CUST-ID TO MASTER RECORD AND READ MASTER
(143-155)
PREPARE TO UPDATE MASTER RECORD OR BUILD ERROR MESSAGE
(156-160)
WRITE UPDATED MASTER AND COUNT UPDATE
(161-162)
READ TRANSACTION RECORD
(163-171)
PREPARE TO UPDATE TRANSACTION RECORD OR BUILD ERROR MESSAGE
(172-173)
INSERT NEW TRANSACTION RECORD
(174-180)
READ UPDATED TRANSACTION RECORD OR BUILD ERROR MESSAGES
(181-193)
UPDATE TRANSACTION RECORD PAYMENT TOTAL AND INSERT HEADER
PG4

```
00136  PROCEDURE DIVISION
00137       USING P-I-B I-M-A W-A O-M-A.
00138  BEGIN-PROC.
00139       MOVE DICE-CODE TO DICE-OUT.
00140       IF STANDARD-MSG-NUMBER-LINES NOT > 12
00141           THEN MOVE TWELVE TO DICE-Y.
00142       MOVE CUSTID TO CUSTNO.
00143       CALL 'GETUP'
00144           USING DMOMSTR MSTR-REC CUSTNO.
00145       IF STATUS-CODE NOT = ZERO
00146           THEN GO TO ERR-OFF.
00147       MOVE ZERO TO PAYMENT.
00148       SET J TO 5.
00149       SUBTRACT 17 FROM TEXT-LENGTH IN I-M-A.
00150       PERFORM MV-NUM
00151           VARYING I FROM TEXT-LENGTH IN I-M-A BY -1 UNTIL I < 1.
00152       IF PAYMENT NOT > ZERO
00153           THEN MOVE PAY-AMT TO PAYMENT.
00154       IF PAYMENT > BALANCE
00155           THEN MOVE 5 TO STATUS-CODE
00156                GO TO ERR-OFF.
00157       SUBTRACT PAYMENT FROM BALANCE.
00158       CALL 'PUT'
00159           USING DMOMSTR MSTR-REC.
00160       IF STATUS-CODE NOT = ZERO
00161           THEN GO TO ERR-OFF.
00162       MOVE 1 TO REC-CTR.
00163       CALL 'GET'
00164           USING DMOXAC XAC-HDR REC-CTR.
00165       IF STATUS-CODE NOT = ZERO
00166           THEN GO TO ERR-OFF.
00167       IF TODAY IN DATE-TIME-STAMP NOT = TO-DAY IN XAC-HDR
00168           THEN PERFORM INIT-RTN.
00169       MOVE SOURCE-TERMINAL-ID TO TRM.
00170       MOVE TRM-NUM TO TRM-XAC.
00171       MOVE HR-MIN-SEC IN DATE-TIME-STAMP TO XAC-TIME.
00172       MOVE CUSTNO TO CUSTOMER.
00173       MOVE PAYMENT TO XAC-AMT.
00174       CALL 'INSERT'
00175           USING DMOXAC XAC-REC REC-CTR.
00176       IF STATUS-CODE NOT = ZERO
00177           THEN GO TO ERR-OFF.
00178       ADD 1 TRM-NUM GIVING REC-CTR.
00179       CALL 'GET'
00180           USING DMOXAC XAC-TRM REC-CTR.
00181       IF STATUS-CODE NOT = ZERO
00182           THEN GO TO ERR-OFF.
00183       MOVE TRM-NUM TO TRM-ID.
00184       ADD 1 TO XAC-CTR.
00185       ADD PAYMENT TO TRM-TOTAL.
00186       CALL 'INSERT'
00187           USING DMOXAC XAC-TRM REC-CTR.
00188       IF STATUS-CODE NOT = ZERO
00189           THEN GO TO ERR-OFF.
00190       ADD 1 TO PTR-NEXT.
00191       MOVE 1 TO REC-CTR.
00192       CALL 'INSERT'
00193           USING DMOXAC XAC-HDR REC-CTR.
```

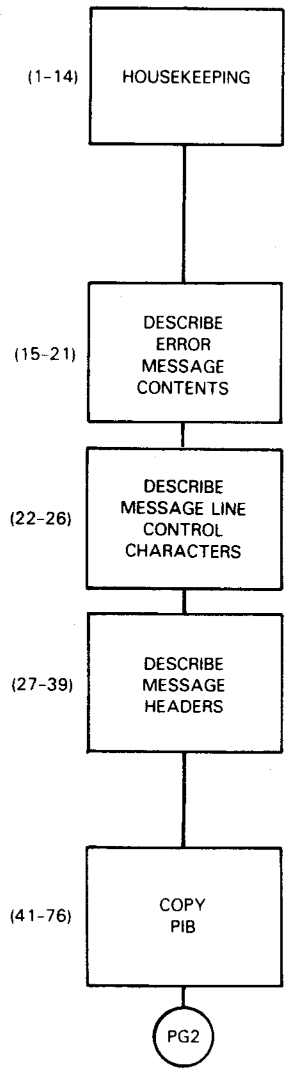
Figure B-16. Sample COBOL Action Program DMPYMT (Part 3 of 4)

SIMPLE TRANSACTION IN COBOL: DMPYMT PROGRAM


PG4
(194-198)
BUILD OUTPUT MESSAGE TEXT FOR DMPYMT PROGRAM
(199)
TERMINATE PROGRAM NORMALLY
(200-205)
SET TABLE TO RECEIVE PAYMENT IN WORK AREA
(206-220)
READ TRANSACTION DATA ENTERED FROM TERMINAL
(221-224)
BUILD ERROR MESSAGE AND TERMINATE NORMALLY

```
00194        IF  STATUS-CODE NOT  = ZERO
00195            THEN GO TO ERR-OFF.
00196        MOVE MSG-LIT TO LIT-OUT.
00197        MOVE PAYMENT TO PAY-OUT.
00198        MOVE CUSTNO TO CUST-OUT.
00199        MOVE BALANCE TO NEW-BAL.
00200        MOVE OUT-MSG-LEN TO TEXT-LENGTH IN O-M-A.
00201        CALL 'RETURN'.
00202    MV-NUM.
00203        IF MSG-CHAR (I) IS NUMERIC
00204            THEN MOVE MSG-CHAR (I) TO PAY-CHAR (J)
00205                 SET J DOWN BY 1
00206                 IF J < 1
00207                     THEN SET I TO 1.
00208    INIT-RTN.
00209        MOVE CORR DATE-TIME-STAMP TO XAC-HDR.
00210        MOVE PTR-FIRST TO PTR-NEXT.
00211        PERFORM TRM-INIT
00212            VARYING REC-CTR FROM 2 BY 1 UNTIL REC-CTR NOT < PTR-FIRST.
00213    TRM-INIT.
00214        CALL 'GET'
00215            USING DMOXAC XAC-TRM REC-CTR.
00216        IF  STATUS-CODE NOT  = ZERO
00217            THEN GO TO ERR-OFF.
00218        MOVE ZEROS TO XAC-CTR TRM-TOTAL.
00219        IF  STATUS-CODE NOT  = ZERO
00220            THEN GO TO ERR-OFF.
00221    ERR-OFF.
00222        MOVE ERR-MSG (STATUS-CODE) TO OUT-MSG.
00223        MOVE ERR-MSG-LEN TO TEXT-LENGTH IN O-M-A.
00224        CALL 'RETURN'.
```

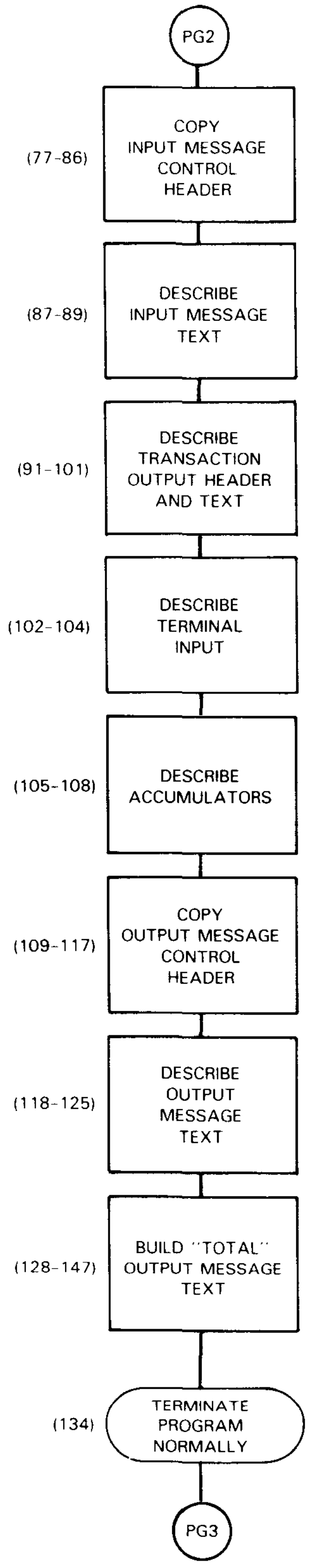
Figure B-16. Sample COBOL Action Program DMPYMT (Part 4 of 4)



```
LINE NO. SOURCE ENTRY

00001 IDENTIFICATION DIVISION.
00002 PROGRAM-ID. DMTOTL.
00003 AUTHOR. E.O.F.S.F. (NYNA 7/76)
00004 DATE-WRITTEN. 9/13/76.
00005 ENVIRONMENT DIVISION.
00006 CONFIGURATION SECTION.
00007 SOURCE-COMPUTER. UNIVAC-OS3.
00008 OBJECT-COMPUTER. UNIVAC-OS3.
00009 DATA DIVISION.
00010 WORKING-STORAGE SECTION.
00011 77  DMOXAC                 PIC X(7)    VALUE 'DMOXACT'.
00012 77  MIN-MSG-LEN            PIC 9999    COMP-4  VALUE 126.
00013 77  ERR-MSG-LEN            PIC 9999    COMP-4  VALUE 29.
00014 77  MSG-LINE-LEN           PIC 9999    COMP-4  VALUE 40.
00015 01  ERR-MSG-LITS.
00016     02 FILLER              PIC X(21)   VALUE '**INVALID KEY*.'.
00017     02 FILLER              PIC X(21)   VALUE '**END OF FILE**'.
00018     02 FILLER              PIC X(21)   VALUE '**INVALID REQUEST**'.
00019     02 FILLER              PIC X(21)   VALUE '**I/O ERROR**'.
00020 01  ERR-MSG-TBL            REDEFINES ERR-MSG-LITS.
00021     02 ERR-MSG             PIC X(21)   OCCURS 4.
00022 01  DICE-CODE.
00023     02 DLE                 PIC X         VALUE ='10'.
00024     02 ETX                 PIC X         VALUE ='03'.
00025     02 THREE               PIC X         VALUE ='03'.
00026     02 ONE                 PIC X         VALUE ='01'.
00027 01  MSG-HDR.
00028     02 FILLER              PIC X(14)   VALUE 'TERMINAL'.
00029     02 FILLER              PIC X(18)   VALUE 'NUMBER OF'.
00030     02 FILLER              PIC X(5)    VALUE 'TOTAL'.
00031     02 FILLER              PIC XXXX    VALUE ' '.
00032     02 FILLER              PIC X(10)   VALUE 'ID'.
00033     02 FILLER              PIC X(18)   VALUE 'TRANSACTIONS'.
00034     02 FILLER              PIC X(8)    VALUE 'PAYMENTS'.
00035     02 CR                  PIC X       VALUE ' '.
00036 01  DASH-LINE.
00037     02 FILLER              PIC X(18)   VALUE '   -----'.
00038     02 FILLER              PIC X(14)   VALUE '----'.
00039     02 FILLER              PIC X(8)    VALUE '--------'.
00040 LINKAGE SECTION.
00041 01  P-I-B.                 COPY PIB74.
00042     02  STATUS-CODE                  PIC 9(4) COMP-4.
00043     02  DETAILED-STATUS-CODE         PIC 9(4) COMP-4.
00044     02  RECORD-TYPE REDEFINES DETAILED-STATUS-CODE.
00045         03 PREDICTED-RECORD-TYPE     PIC X.
00046         03 DELIVERED-RECORD-TYPE     PIC X.
00047     02  SUCCESSOR-ID                 PIC X(6).
00048     02  TERMINATION-INDICATOR        PIC X.
00049     02  LOCK-ROLLBACK-INDICATOR      PIC X.
00050     02  TRANSACTION-ID.
00051         03 YEAR                      PIC 9(4) COMP-4.
00052         03 TODAY                     PIC 9(4) COMP-4.
00053         03 HR-MIN-SEC                PIC 9(9) COMP-4.
00054     02  DATA-DEF-REC-NAME            PIC X(7).
00055     02  DEFINED-FILE-NAME            PIC X(7).
00056     02  STANDARD-MSG-LINE-LENGTH     PIC 9(4) COMP-4.
00057     02  STANDARD-MSG-NUMBER-LINES    PIC 9(4) COMP-4.
00058     02  WORK-AREA-LENGTH             PIC 9(4) COMP-4.
00059     02  CONTINUITY-DATA-INPUT-LENGTH  PIC 9(4) COMP-4.
00060     02  CONTINUITY-DATA-OUTPUT-LENGTH  PIC 9(4) COMP-4.
00061     02  WORK-AREA-INC                PIC 9(4) COMP-4.
00062     02  CONTINUITY-DATA-AREA-INC     PIC 9(4) COMP-4.
00063     02  SUCCESS-UNIT-ID.
00064         03 TRANSACTION-DATE.
00065            04 YEAR                   PIC 99.
00066            04 MONTH                  PIC 99.
00067            04 TODAY                  PIC 99.
00068         03 TIME-OF-DAY.
00069            04 HOUR                   PIC 99.
00070            04 MINUTE                 PIC 99.
00071            04 SECOND                 PIC 99.
00072         03 FILLER                    PIC XXX.
00073     02  SOURCE-TERMINAL-CHARS.
00074         03 SOURCE-TERMINAL-TYPE  PIC X.
00075         03 SOURCE-TERM-MSG-LINE-LENGTH    PIC 9(4) COMP-4.
00076         03 SOURCE-TERM-MSG-NUMBER-LINES   PIC 9(4) COMP-4.
```

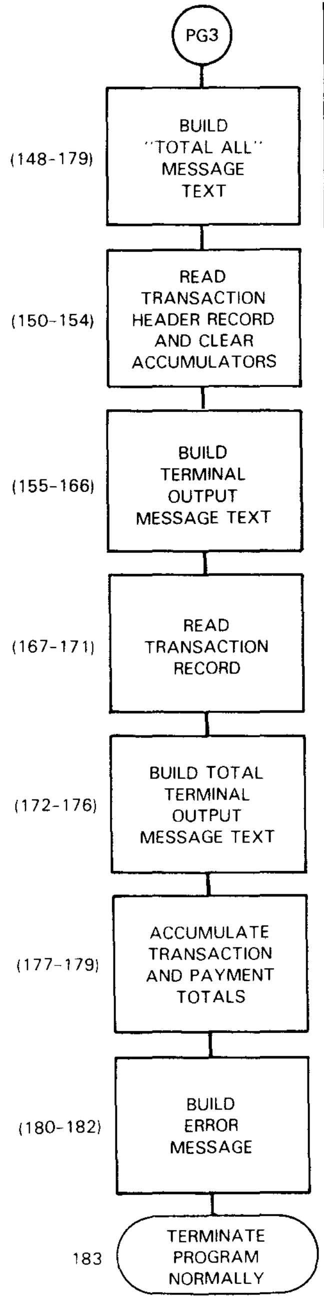
Figure B-17. Sample COBOL Action Program DMTOTL (Part 1 of 3)

SIMPLE TRANSACTION IN COBOL: DMTOTL PROGRAM


PG2
(77-86)
COPY INPUT MESSAGE CONTROL HEADER
(87-89)
DESCRIBE INPUT MESSAGE TEXT
(91-101)
DESCRIBE TRANSACTION OUTPUT HEADER AND TEXT
(102-104)
DESCRIBE TERMINAL INPUT
(105-108)
DESCRIBE ACCUMULATORS
(109-117)
COPY OUTPUT MESSAGE CONTROL HEADER
(118-125)
DESCRIBE OUTPUT MESSAGE TEXT
(128-147)
BUILD "TOTAL" OUTPUT MESSAGE TEXT
(134)
TERMINATE PROGRAM NORMALLY
PG3

```
00077  01   I-M-A.                    COPY IMA74.
00078       02 SOURCE-TERMINAL-ID                      PIC X(4).
00079       02 DATE-TIME-STAMP.
00080          03 YEAR                                 PIC 9(4) COMP-4.
00081          03 TODAY                                PIC 9(4) COMP-4.
00082          03 HR-MIN-SEC                           PIC 9(9) COMP-4.
00083       02 TEXT-LENGTH                             PIC 9(4) COMP-4.
00084       02 AUXILIARY-DEV-ID.
00085          03 FILLER                              PIC X.
00086          03 AUX-DEV-NO                          PIC X.
00087       02 FILLER                 PIC X(6).
00088       02 OPT                    PIC XXX.
00089       02 FILLER                 PIC XXX.
00090  01   W-A.
00091       02 XAC-HUR.
00092           03 FILLER             PIC XXXX.
00093           03 XAC-FIRST          PIC 9999    COMP-4.
00094           03 FILLER             PIC X(10).
00095       02 J-FILL-1               PIC X(256).
00096       02 XAC-TRM.
00097           03 TRM-ID             PIC 9.
00098           03 XAC-CTR            PIC 9999.
00099           03 TRM-TOTAL          PIC 9(5)V99.
00100           03 FILLER             PIC XXXX.
00101       02 J-FILL-2               PIC X(256).
00102       02 TRM.
00103           03 TRM-LIT            PIC XXX.
00104           03 TRM-NUM            PIC 9.
00105       02 PAY-TOT                PIC 9(5)V99.
00106       02 XAC-TOT                PIC 9999.
00107       02 REC-PTR                PIC 9(12)   COMP-4.
00108       02 FILLER                 PIC X.
00109  01   O-M-A.                    COPY OMA74.
00110       02 DESTINATION-TERMINAL-ID           PIC X(4).
00111                 02 SFS-OPTIONS                          PIC X(2).
00112                 02 FILLER                               PIC X(2).
00113       02 CONTINUOUS-OUTPUT-CODE            PIC X(4).
00114       02 TEXT-LENGTH                  PIC 9(4)   COMP-4.
00115       02 AUXILIARY-DEVICE-ID.
00116          03 AUX-FUNCTION                   PIC X.
00117          03 AUX-DEVICE-NO                  PIC X.
00118       02 DICE-OUT               PIC XXXX.
00119       02 HDR-OUT                PIC X(78).
00120       02 TOT-LINE               OCCURS 7    INDEXED BY I.
00121           03 CR-OUT             PIC XXX.
00122           03 TRM-OUT            PIC X(15).
00123           03 CTR-OUT            PIC ZZZ9.
00124           03 FILLER             PIC X(9).
00125           03 TOT-OUT            PIC $(5)9.99.
00126  PROCEDURE DIVISION
00127       USING P-I-B I-M-A W-A O-M-A.
00128  BEGIN-PROC.
00129       IF OPT = 'ALL'
00130           THEN PERFORM RTN-ALL
00131           ELSE PERFORM RTN-ONE.
00132       MOVE DICE-CODE TO DICE-OUT.
00133       MOVE MSG-HDR TO HDR-OUT.
00134       CALL 'RETURN'.
00135  RTN-ONE.
00136       MOVE SOURCE-TERMINAL-ID TO TRM.
00137       MOVE TRM-NUM TO REC-PTR.
00138       ADD 1 TO REC-PTR.
00139       CALL 'GET'
00140           USING DMOXAC XAC-TRM REC-PTR.
00141       IF STATUS-CODE NOT = ZERO
00142           THEN GO TO ERR-OFF.
00143       MOVE CR TO CR-OUT (I).
00144       MOVE TRM TO TRM-OUT (I).
00145       MOVE XAC-CTR TO CTR-OUT (I).
00146       MOVE TRM-TOTAL TO TOT-OUT (I).
00147       MOVE MIN-MSG-LEN TO TEXT-LENGTH IN O-M-A.
```

Figure B-17. Sample COBOL Action Program DMTOTL (Part 2 of 3)


PG3
(148-179)
BUILD "TOTAL ALL" MESSAGE TEXT
(150-154)
READ TRANSACTION HEADER RECORD AND CLEAR ACCUMULATORS
(155-166)
BUILD TERMINAL OUTPUT MESSAGE TEXT
(167-171)
READ TRANSACTION RECORD
(172-176)
BUILD TOTAL TERMINAL OUTPUT MESSAGE TEXT
(177-179)
ACCUMULATE TRANSACTION AND PAYMENT TOTALS
(180-182)
BUILD ERROR MESSAGE
183
TERMINATE PROGRAM NORMALLY

```
00148  RTN-ALL.
00149      MOVE 1 TO REC-PTR.
00150      CALL 'GET'
00151          USING DMOXAC XAC-HDR REC-PTR.
00152      IF STATUS-CODE NOT = ZERO
00153          THEN GO TO ERR-OFF.
00154      MOVE ZEROS TO XAC-TOT PAY-TOT.
00155      MOVE 'TRM' TO TRM-LIT.
00156      ADD 1 TO REC-PTR.
00157      PERFORM SUM-TRM
00158          VARYING I FROM 1 BY 1 UNTIL REC-PTR NOT < XAC-FIRST.
00159      MOVE DASH-LINE TO TOT-LINE (I).
00160      SET I UP BY 1.
00161      MOVE CR TO CR-OUT (I).
00162      MOVE 'TOTAL' TO TRM-OUT (I).
00163      MOVE XAC-TOT TO CTR-OUT (I).
00164      MOVE PAY-TOT TO TOT-OUT (I).
00165      COMPUTE TEXT-LENGTH IN O-M-A = MIN-MSG-LEN +
00166                                     MSG-LINE-LEN * REC-PTR.
00167  SUM-TRM.
00168      CALL 'GET'
00169          USING DMOXAC XAC-TRM REC-PTR.
00170      IF STATUS-CODE NOT = ZERO
00171          THEN GO TO ERR-OFF.
00172      MOVE CR TO CR-OUT (I).
00173      MOVE TRM-ID TO TRM-NUM.
00174      MOVE TRM TO TRM-OUT (I).
00175      MOVE XAC-CTR TO CTR-OUT (I).
00176      MOVE TRM-TOTAL TO TOT-OUT (I).
00177      ADD XAC-CTR TO XAC-TOT.
00178      ADD TRM-TOTAL TO PAY-TOT.
00179      ADD 1 TO REC-PTR.
00180  ERR-OFF.
00181      MOVE ERR-MSG (STATUS-CODE) TO HDR-OUT.
00182      MOVE ERR-MSG-LEN TO TEXT-LENGTH IN O-M-A.
00183      CALL 'RETURN'.
```

Figure B-17. Sample COBOL Action Program DMTOTL (Part 3 of 3)

CSCAN series analysis

You may have noticed that in this series of action programs consisting of five separate transactions, each transaction contained only one action program. In other words, one action program received one input message and issued one output message for each transaction.

These action programs were chained together by placing the succeeding action program's transaction code itself into the output message issued by the current action program. In this way, control passed from one action program to another, establishing a sense of succession between the programs without actually moving values into the SUCCESSOR-ID and TERMINATION-INDICATOR fields of the PIB. This technique is effective for processing simple transactions in a series. However, there are situations that require more than one program to process a transaction. We call these *dialog* transactions.

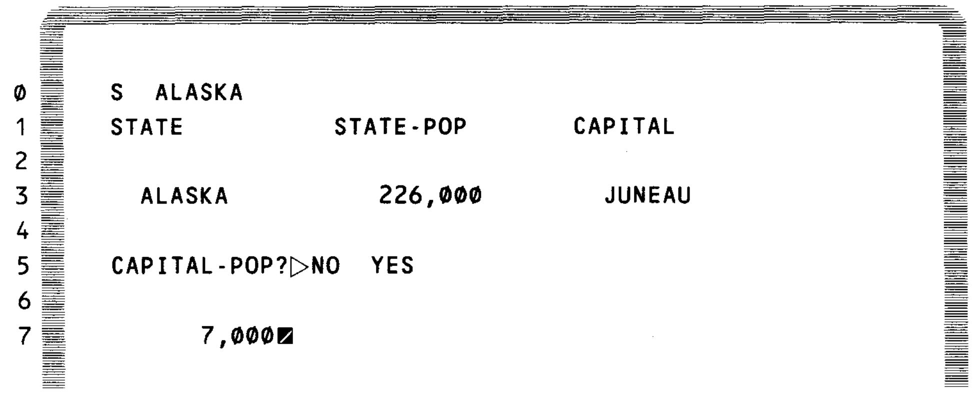
B.3. SAMPLE COBOL ACTION PROGRAMS PERFORMING A DIALOG TRANSACTION WITH EXTERNAL SUCCESSION (ACT 1 AND ACT 2)

ACT1/ACT2 description

The two action programs, ACT1 and ACT2, perform a dialog transaction. This transaction references two indexed files named STATE and CITY. The STATE file contains a record for each state. Each state record consists of a state name, state population, and capital city name. The CITY file contains a record for each city. In each city record is the city name, population, and state name. Assume for the purposes of this example that all city names in the CITY file are unique.

Processing ACT1

The purpose of this transaction is to provide information about a state. Each time you enter the transaction code S, IMS associates it with the action program ACT1. In addition to the transaction code, you include a state name (Figure B-18, line 0). ACT1 uses the state name you give to obtain a record from the STATE file.



```
0   S  ALASKA
1   STATE            STATE-POP        CAPITAL
2
3     ALASKA            226,000          JUNEAU
4
5   CAPITAL-POP?▷NO  YES
6
7       7,000▨
```

NOTE:

The cursor (▨) may appear at only one location on the screen at any one time. In this example, it also would have appeared after ALASKA when the operator entered the initial input message (line 0) and after NO upon transmission of the first output response built by ACT 1 (line 5). The start-of-entry character (▷) may appear at multiple locations.

Figure B-18. Sample Dialog Transaction with YES Option Taken

Resulting output

If the record exists, ACT1 responds by sending an output message to the terminal. The output message contains headers, the state name, population, and capital name plus a question asking if you want the capital's population (Figure B-18, lines 1-5). ACT1 moves output message headings (Figure B-21, lines 16 and 17) and control characters (lines 12-15) from the working-storage section to the output message area.

You can request capital city population or terminate the transaction. Start-of-entry (▷) and cursor (◪) characters are positioned in the output message area so that:

1. If you want to terminate the transaction without seeing capital population, press TRANSMIT.

2. If you want to see capital population, press TAB followed by TRANSMIT.

External succession

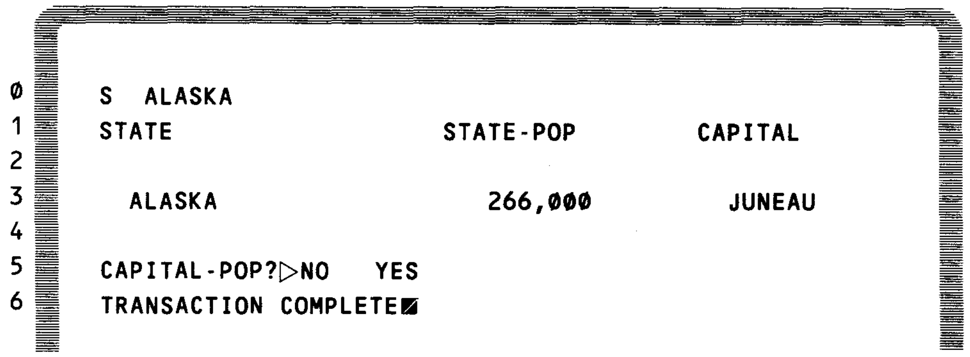
Before succeeding externally to ACT2, ACT1 saves the capital city name in the continuity data area (lines 108 and 109). When ACT1 succeeds to ACT2, IMS passes the contents of this area to ACT2 (lines 124 and 125). To succeed to ACT2, ACT1 moves a termination code of E for external succession to the TERMINATION-INDICATOR field (line 127). It also moves the name, ACT2, to the SUCCESSOR-ID field (line 128).

'rocessing ACT2

When you choose the YES option, ACT2 obtains the CITY record for capital city named in the continuity data area (Figure B-22, line 92), builds an output message containing the capital population (Figure B-18, line 7 and Figure B-22, lines 97-99), and terminates normally with the CALL RETURN function.

Choosing NO option

When you choose the NO option, ACT2 moves zero to the TEXT-LENGTH field in the output message area control header before terminating normally (Figure B-22, lines 93 and 94). Because ACT2 doesn't provide an output message, IMS returns the standard transaction termination message to the source terminal as shown in Figure B-19, line 6.

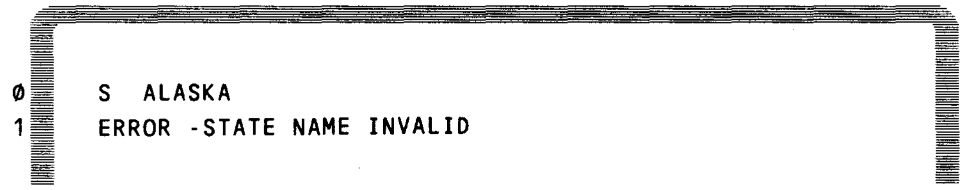


```
0  S  ALASKA
1  STATE                        STATE-POP        CAPITAL
2
3   ALASKA                        266,000          JUNEAU
4
5  CAPITAL-POP?▷NO   YES
6  TRANSACTION COMPLETE■
```

Figure B-19. Sample Dialog Transaction with NO Option Taken

Error handling

Suppose you enter a state name that cannot be found in the STATE file. ACT1 builds an error message in the OMA (Figure B-21, lines 28 and 29) and moves the length of this error message to the TEXT-LENGTH field of the output message area control header to override the previous text length value (lines 115, 130–133). The transaction terminates normally with a CALL RETURN function and IMS sends the error output message to the terminal as shown in Figure B-20, line 1.



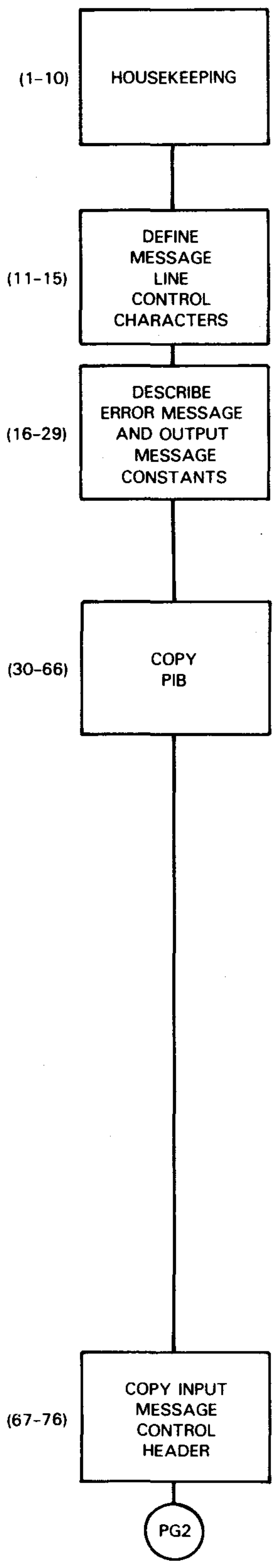
```
0  S  ALASKA
1  ERROR -STATE NAME INVALID
```

Figure B-20. Sample Transaction with Error Message

Compilations and flowcharts

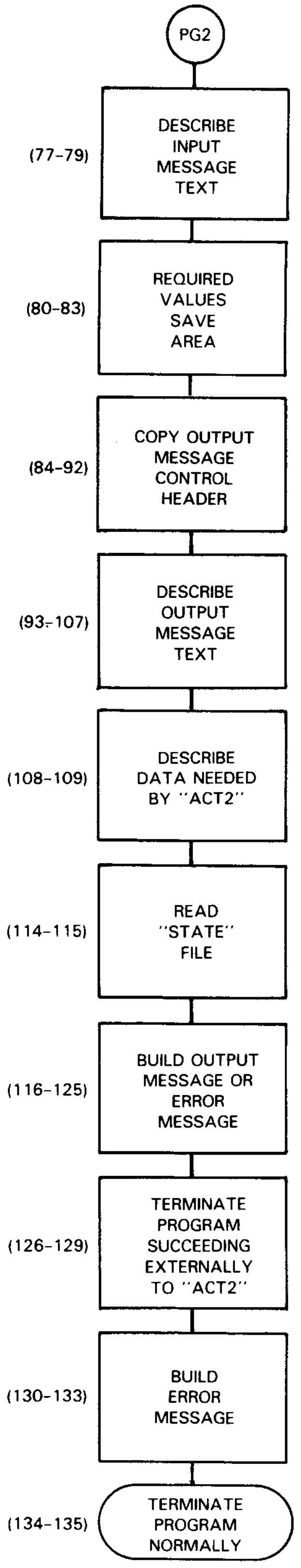
General flowcharts for the coding in ACT1 and ACT2 action programs (Figures B-21 and B-22) appear to the left of the program code in these figures. Program line numbers in parentheses to the side of the flowchart boxes represent the lines of coding that implement the process described.

DIALOG TRANSACTION IN COBOL: ACT 1 PROGRAM


(1-10)
HOUSEKEEPING
(11-15)
DEFINE MESSAGE LINE CONTROL CHARACTERS
(16-29)
DESCRIBE ERROR MESSAGE AND OUTPUT MESSAGE CONSTANTS
(30-66)
COPY PIB
(67-76)
COPY INPUT MESSAGE CONTROL HEADER
PG2

```
LINE NO.    SOURCE ENTRY

00001  IDENTIFICATION DIVISION.
00002  PROGRAM-ID. ACT1.
00003  ENVIRONMENT DIVISION.
00004  CONFIGURATION SECTION.
00005  SOURCE-COMPUTER. UNIVAC-OS3.
00006  OBJECT-COMPUTER. UNIVAC-OS3.
00007  DATA DIVISION.
00008  WORKING-STORAGE SECTION.
00009  77  STATE PIC A(7) VALUE 'STATE'.
00010  77  ERROR-TEXT-LENGTH PIC 9(4) COMP-4 VALUE 34.
00011  01  LINE-0.
00012      02  DLE              PIC X      VALUE ='10'.
00013      02  PCAC             PIC X      VALUE ='05'.
00014      02  ROW-0            PIC X      VALUE ='00'.
00015      02  COLUMN-0         PIC X      VALUE ='00'.
00016  01  LINE-1-MSG-1A        PIC X(39) VALUE 'STATE          STATE-POP
00017      '         CAPITAL'.
00018  01  LINE-5-MSG-1A.
00019      02  E-1              PIC X(13) VALUE 'CAPITAL-POP? '.
00020      02  SOE              PIC X      VALUE ='1E'.
00021      02  E-2              PIC X(7)  VALUE 'NO  YES'.
00022      02  ESC-1            PIC X      VALUE ='27'.
00023      02  HT               PIC X      VALUE ='05'.
00024      02  DLE              PIC X      VALUE ='10'.
00025      02  FC               PIC X      VALUE ='02'.
00026      02  ROW-5            PIC X      VALUE ='10'.
00027      02  COLUMN-16        PIC X      VALUE ='05'.
00028  01  LINE-1-MSG-1B.
00029      02 E-1 PIC X(26) VALUE 'ERROR - STATE NAME INVALID'.
00030  LINKAGE SECTION.
00031  01  PROGRAM-INFORMATION-BLOCK. COPY PIB74.
00032      02  STATUS-CODE                 PIC 9(4) COMP-4.
00033      02  DETAILED-STATUS-CODE        PIC 9(4) COMP-4.
00034      02  RECORD-TYPE REDEFINES DETAILED-STATUS-CODE.
00035          03 PREDICTED-RECORD-TYPE  PIC X.
00036          03 DELIVERED-RECORD-TYPE  PIC X.
00037      02  SUCCESSOR-ID                PIC X(6).
00038      02  TERMINATION-INDICATOR       PIC X.
00039      02  LOCK-ROLLBACK-INDICATOR     PIC X.
00040      02  TRANSACTION-ID.
00041          03 YEAR                     PIC 9(4) COMP-4.
00042          03 TODAY                    PIC 9(4) COMP-4.
00043          03 HR-MIN-SEC               PIC 9(9) COMP-4.
00044      02  DATA-DEF-REC-NAME           PIC X(7).
00045      02  DEFINED-FILE-NAME           PIC X(7).
00046      02  STANDARD-MSG-LINE-LENGTH    PIC 9(4) COMP-4.
00047      02  STANDARD-MSG-NUMBER-LINES   PIC 9(4) COMP-4.
00048      02  WORK-AREA-LENGTH            PIC 9(4) COMP-4.
00049      02  CONTINUITY-DATA-INPUT-LENGTH  PIC 9(4) COMP-4.
00050      02  CONTINUITY-DATA-OUTPUT-LENGTH  PIC 9(4) COMP-4.
00051      02  WORK-AREA-INC               PIC 9(4) COMP-4.
00052      02  CONTINUITY-DATA-AREA-INC    PIC 9(4) COMP-4.
00053          02  SUCCESS-UNIT-ID.
00054              03 TRANSACTION-DATE.
00055                 04 YEAR                    PIC 99.
00056                 04 MONTH                   PIC 99.
00057                 04 TODAY                   PIC 99.
00058              03 TIME-OF-DAY.
00059                 04 HOUR                    PIC 99.
00060                 04 MINUTE                  PIC 99.
00061                 04 SECOND                  PIC 99.
00062          03 UNIQUE-SUFFIX            PIC 999.
00063      02  SOURCE-TERMINAL-CHARS.
00064          03 SOURCE-TERMINAL-TYPE  PIC X.
00065          03 SOURCE-TERM-MSG-LINE-LENGTH    PIC 9(4) COMP-4.
00066          03 SOURCE-TERM-MSG-NUMBER-LINES   PIC 9(4) COMP-4.
00067  01  INPUT-MESSAGE-AREA. COPY IMA74.
00068      02 SOURCE-TERMINAL-ID                 PIC X(4).
00069      02 DATE-TIME-STAMP.
00070         03 YEAR                            PIC 9(4) COMP-4.
00071         03 TODAY                           PIC 9(4) COMP-4.
00072         03 HR-MIN-SEC                      PIC 9(9) COMP-4.
00073      02 TEXT-LENGTH                        PIC 9(4) COMP-4.
00074      02 AUXILIARY-DEV-ID.
00075         03 FILLER                        PIC X.
00076         03 AUX-DEV-NO                    PIC X.
```

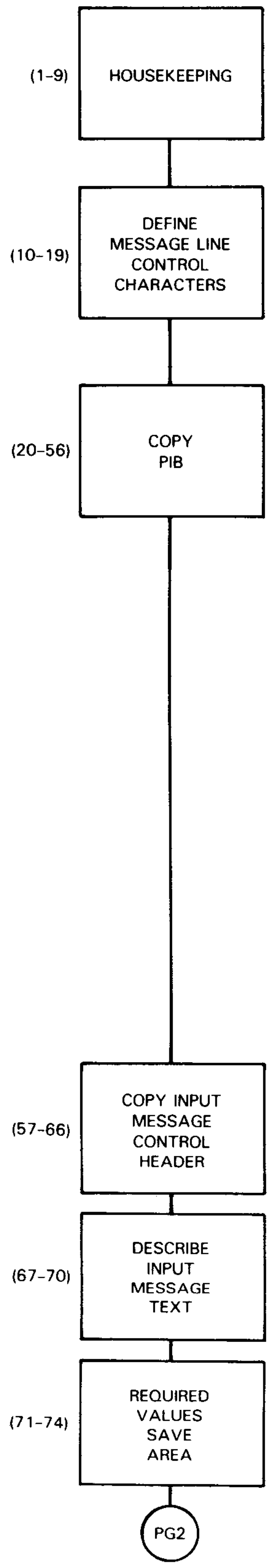
Figure B-21. Sample COBOL Action Program ACT1 (Part 1 of 2)


PG2
(77-79)
DESCRIBE INPUT MESSAGE TEXT
(80-83)
REQUIRED VALUES SAVE AREA
(84-92)
COPY OUTPUT MESSAGE CONTROL HEADER
(93-107)
DESCRIBE OUTPUT MESSAGE TEXT
(108-109)
DESCRIBE DATA NEEDED BY "ACT2"
(114-115)
READ "STATE" FILE
(116-125)
BUILD OUTPUT MESSAGE OR ERROR MESSAGE
(126-129)
TERMINATE PROGRAM SUCCEEDING EXTERNALLY TO "ACT2"
(130-133)
BUILD ERROR MESSAGE
(134-135)
TERMINATE PROGRAM NORMALLY

```
00077        02 TRANSACTION-CODE   PIC X.
00078        02 FILLER             PIC X.
00079        02 STATE-NAME-IN      PIC A(14).
00080    01  WORK-AREA.
00081        02 STATE-NAME PIC A(14).
00082        02 STATE-POP  PIC 9(8).
00083        02 CAPITAL    PIC A(25).
00084    01  OUTPUT-MESSAGE-AREA.       COPY OMA74.
00085        02 DESTINATION-TERMINAL-ID       PIC X(4).
00086                 02 SFS-OPTIONS                      PIC X(2).
00087                  02 FILLER                          PIC X(2).
00088        02 CONTINUOUS-OUTPUT-CODE        PIC X(4).
00089        02 TEXT-LENGTH                PIC 9(4)    COMP-4.
00090        02 AUXILIARY-DEVICE-ID.
00091           03 AUX-FUNCTION               PIC X.
00092           03 AUX-DEVICE-NO              PIC X.
00093        02 LINE-0-OUT PIC X(4).
00094        02 LINE-1-OUT.
00095          03 E1-OUT           PIC X(39).
00096          03 CONTROL-1        PIC X(4).
00097          03 CONTROL-2        PIC X(4).
00098        02 LINE-3-OUT.
00099          03 FILLER           PIC XX.
00100          03 STATE-NAME       PIC A(14).
00101          03 FILLER           PIC X(4).
00102          03 STATE-POP        PIC 99,999,999.
00103          03 FILLER           PIC X(4).
00104          03 CAPITAL          PIC A(25).
00105          03 CONTROL-3        PIC X(4).
00106          03 CONTROL-4        PIC X(4).
00107        02 LINE-5-OUT PIC X(27).
00108    01  CONTINUITY-DATA-AREA.
00109        02 CAPITAL            PIC A(25).
00110    PROCEDURE DIVISION USING PROGRAM-INFORMATION-BLOCK
00111        INPUT-MESSAGE-AREA WORK-AREA OUTPUT-MESSAGE-AREA
00112        CONTINUITY-DATA-AREA.
00113    GET-STATE-RECORD.
00114        CALL 'GET' USING STATE WORK-AREA STATE-NAME-IN.
00115        IF STATUS-CODE EQUAL  1 GO TO PROCESS-ERROR.
00116    BUILD-OUTPUT-MESSAGE.
00117        MOVE LINE-0 TO LINE-0-OUT.
00118        MOVE LINE-1-MSG-1A TO E1-OUT.
00119        MOVE LINE-0 TO CONTROL-1 CONTROL-2.
00120        MOVE SPACES TO LINE-3-OUT.
00121        MOVE CORRESPONDING WORK-AREA TO LINE-3-OUT.
00122        MOVE LINE-0 TO CONTROL-3 CONTROL-4.
00123        MOVE LINE-5-MSG-1A TO LINE-5-OUT.
00124    SAVE-CONTINUITY-DATA.
00125        MOVE CAPITAL OF WORK-AREA TO CAPITAL OF CONTINUITY-DATA-AREA.
00126    TERM-WITH-EXTERNAL-SUCCESSOR.
00127        MOVE 'E' TO TERMINATION-INDICATOR.
00128        MOVE 'ACT200' TO SUCCESSOR-ID.
00129        CALL 'RETURN'.
00130    PROCESS-ERROR.
00131        MOVE LINE-0 TO LINE-0-OUT.
00132        MOVE LINE-1-MSG-1B TO LINE-1-OUT.
00133        MOVE ERROR-TEXT-LENGTH TO TEXT-LENGTH OF OUTPUT-MESSAGE-AREA.
00134    TERMINATE-NORMALLY.
00135        CALL 'RETURN'.
```

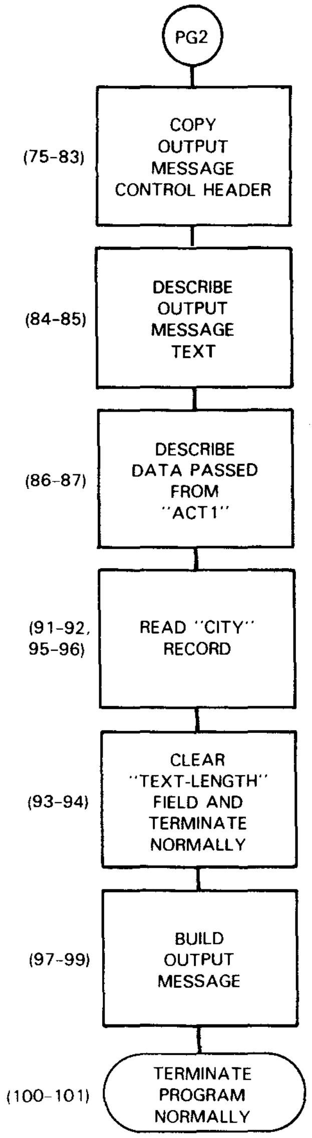
Figure B-21. Sample COBOL Action Program ACT1 (Part 2 of 2)

DIALOG TRANSACTION IN COBOL: ACT 2 PROGRAM


(1-9)
HOUSEKEEPING
(10-19)
DEFINE MESSAGE LINE CONTROL CHARACTERS
(20-56)
COPY PIB
(57-66)
COPY INPUT MESSAGE CONTROL HEADER
(67-70)
DESCRIBE INPUT MESSAGE TEXT
(71-74)
REQUIRED VALUES SAVE AREA
PG2

```
LINE NO. SOURCE ENTRY

00001  IDENTIFICATION DIVISION.
00002  PROGRAM-ID. ACT2.
00003  ENVIRONMENT DIVISION.
00004  CONFIGURATION SECTION.
00005  SOURCE-COMPUTER. UNIVAC-OS3.
00006  OBJECT-COMPUTER. UNIVAC-OS3.
00007  DATA DIVISION.
00008  WORKING-STORAGE SECTION.
00009  77  CITY PIC A(7) VALUE 'CITY'.
00010  01  LINE-1.
00011      02  DLE-1          PIC X      VALUE ='10'.
00012      02  PCAC-1         PIC X      VALUE ='05'.
00013      02  ROW-O-1        PIC X      VALUE ='00'.
00014      02  COLUMN-O-1     PIC X      VALUE ='00'.
00015      02  DLE-2          PIC X      VALUE ='10'.
00016      02  PCAC-2         PIC X      VALUE ='05'.
00017      02  ROW-O-2        PIC X      VALUE ='00'.
00018      02  COLUMN-O-2     PIC X      VALUE ='00'.
00019      02  FILLER         PIC XX     VALUE SPACES.
00020  LINKAGE SECTION.
00021  01  PROGRAM-INFORMATION-BLOCK. COPY PIB74.
00022      02  STATUS-CODE                PIC 9(4) COMP-4.
00023      02  DETAILED-STATUS-CODE       PIC 9(4) COMP-4.
00024      02  RECORD-TYPE REDEFINES DETAILED-STATUS-CODE.
00025          03 PREDICTED-RECORD-TYPE   PIC X.
00026          03 DELIVERED-RECORD-TYPE   PIC X.
00027      02  SUCCESSOR-ID               PIC X(6).
00028      02  TERMINATION-INDICATOR      PIC X.
00029      02  LOCK-ROLLBACK-INDICATOR    PIC X.
00030      02  TRANSACTION-ID.
00031          03 YEAR                    PIC 9(4) COMP-4.
00032          03 TODAY                   PIC 9(4) COMP-4.
00033          03 HR-MIN-SEC              PIC 9(9) COMP-4.
00034      02  DATA-DEF-REC-NAME          PIC X(7).
00035      02  DEFINED-FILE-NAME          PIC X(7).
00036      02  STANDARD-MSG-LINE-LENGTH   PIC 9(4) COMP-4.
00037      02  STANDARD-MSG-NUMBER-LINES  PIC 9(4) COMP-4.
00038      02  WORK-AREA-LENGTH           PIC 9(4) COMP-4.
00039      02  CONTINUITY-DATA-INPUT-LENGTH  PIC 9(4) COMP-4.
00040      02  CONTINUITY-DATA-OUTPUT-LENGTH  PIC 9(4) COMP-4.
00041      02  WORK-AREA-INC              PIC 9(4) COMP-4.
00042      02  CONTINUITY-DATA-AREA-INC   PIC 9(4) COMP-4.
00043          02  SUCCESS-UNIT-ID.
00044              03 TRANSACTION-DATE.
00045                 04 YEAR                     PIC 99.
00046                 04 MONTH                    PIC 99.
00047                 04 TODAY                    PIC 99.
00048              03 TIME-OF-DAY.
00049                 04 HOUR                     PIC 99.
00050                 04 MINUTE                   PIC 99.
00051                 04 SECOND                   PIC 99.
00052          03 UNIQUE-SUFFIX           PIC 999.
00053      02  SOURCE-TERMINAL-CHARS.
00054          03 SOURCE-TERMINAL-TYPE  PIC X.
00055          03 SOURCE-TERM-MSG-LINE-LENGTH   PIC 9(4) COMP-4.
00056          03 SOURCE-TERM-MSG-NUMBER-LINES  PIC 9(4) COMP-4.
00057  01  INPUT-MESSAGE-AREA. COPY IMA74.
00058      02 SOURCE-TERMINAL-ID                  PIC X(4).
00059      02 DATE-TIME-STAMP.
00060         03 YEAR                             PIC 9(4) COMP-4.
00061         03 TODAY                            PIC 9(4) COMP-4.
00062         03 HR-MIN-SEC                       PIC 9(9) COMP-4.
00063      02 TEXT-LENGTH                         PIC 9(4) COMP-4.
00064      02 AUXILIARY-DEV-ID.
00065         03 FILLER                         PIC X.
00066         03 AUX-DEV-NO                     PIC X.
00067      02 FILLER              PIC X.
00068      02 NO-POP              PIC XX.
00069      02 FILLER              PIC XX.
00070      02 YES                 PIC XXX.
00071  01  WORK-AREA.
00072      02 CITIES              PIC A(25).
00073      02 CITY-POP            PIC 9(7).
00074      02 STATE               PIC A(14).
```

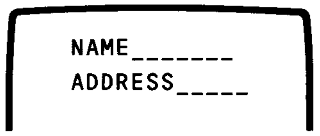
Figure B-22. Sample COBOL Action Program ACT2 (Part 1 of 2)


PG2
(75-83)
COPY OUTPUT MESSAGE CONTROL HEADER
(84-85)
DESCRIBE OUTPUT MESSAGE TEXT
(86-87)
DESCRIBE DATA PASSED FROM "ACT1"
(91-92, 95-96)
READ "CITY" RECORD
(93-94)
CLEAR "TEXT-LENGTH" FIELD AND TERMINATE NORMALLY
(97-99)
BUILD OUTPUT MESSAGE
(100-101)
TERMINATE PROGRAM NORMALLY

```
00075  01   OUTPUT-MESSAGE-AREA.      COPY OMA74.
00076       02 DESTINATION-TERMINAL-ID      PIC X(4).
00077             02 SFS-OPTIONS                      PIC X(2).
00078             02 FILLER                           PIC X(2).
00079       02 CONTINUOUS-OUTPUT-CODE       PIC X(4).
00080       02 TEXT-LENGTH               PIC 9(4)    COMP-4.
00081       02 AUXILIARY-DEVICE-ID.
00082          03 AUX-FUNCTION              PIC X.
00083          03 AUX-DEVICE-NO             PIC X.
00084       02 E-1                 PIC X(10).
00085       02 CAPITAL-POP         PIC 9,999,999.
00086  01   CONTINUITY-DATA-AREA.
00087       02 CAPITAL             PIC A(25).
00088  PROCEDURE DIVISION USING PROGRAM-INFORMATION-BLOCK
00089       INPUT-MESSAGE-AREA WORK-AREA OUTPUT-MESSAGE-AREA
00090       CONTINUITY-DATA-AREA.
00091  CHECK-RESPONSE.
00092       IF YES EQUAL  'YES' GO TO GET-CITY-RECORD.
00093       MOVE ZERO TO TEXT-LENGTH OF OUTPUT-MESSAGE-AREA.
00094       GO TO TERMINATE-NORMALLY.
00095  GET-CITY-RECORD.
00096       CALL 'GET' USING CITY WORK-AREA CAPITAL.
00097  BUILD-OUTPUT-MESSAGE.
00098       MOVE LINE-1 TO E-1.
00099       MOVE CITY-POP TO CAPITAL-POP.
00100  TERMINATE-NORMALLY.
00101       CALL 'RETURN'.
```

Figure B-22. Sample COBOL Action Program ACT2 (Part 2 of 2)

B.4. SAMPLE COBOL ACTION PROGRAM USING SCREEN FORMAT SERVICES (JAMENU)



JAMENU description

The JAMENU action program is the first of a series of programs that make up an entitlement accounting system. JAMENU processes a password entered as input from the terminal. If the password is valid, JAMENU displays a menu screen using screen format services.

The operator then chooses the menu number of the action program he needs to perform the next operation on his file. If the password he enters is invalid, JAMENU displays an error screen and terminates.

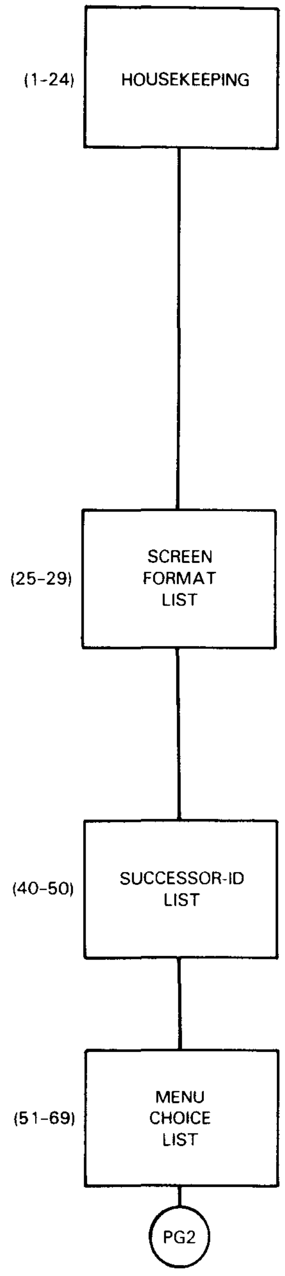
JAMENU analysis

Figure B-23 is a compiler listing of the JAMENU action program. Because this program is one in a series of interrelated action programs, note that a special function call section (lines 269-363) includes many more calls than JAMENU uses. Including a repertoire of these calls in each action program makes them available for any logic used in each procedure division of programs in the series.

Also, in the working-storage section, all screen formats and successor-ids are identified enabling the program to reference any one of them, though it does not use all of them. This programming technique saves time particularly when a series of action programs can succeed differently to each other.

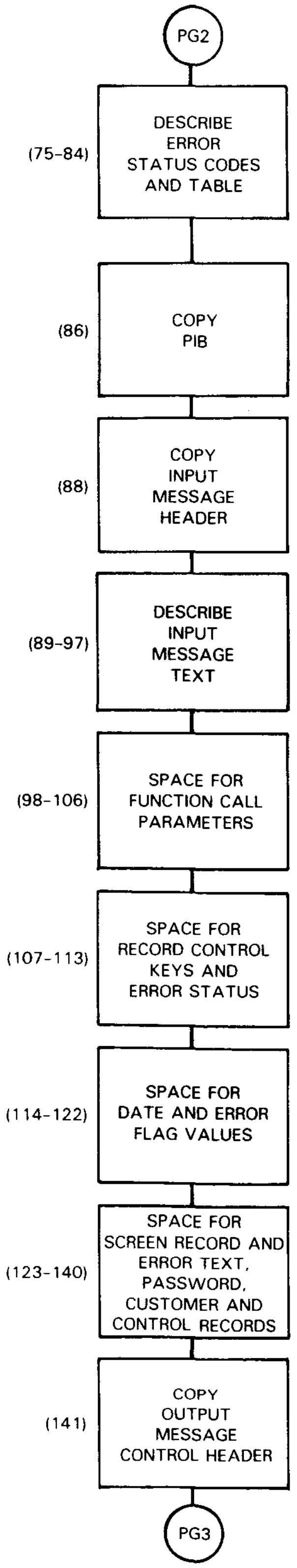
Compilation and flowchart

A flowchart corresponding to the JAMENU action program appears to the left of the coding in Figure B-23. Program line numbers in parentheses near the flowchart boxes represent the lines of coding that implement the process described.



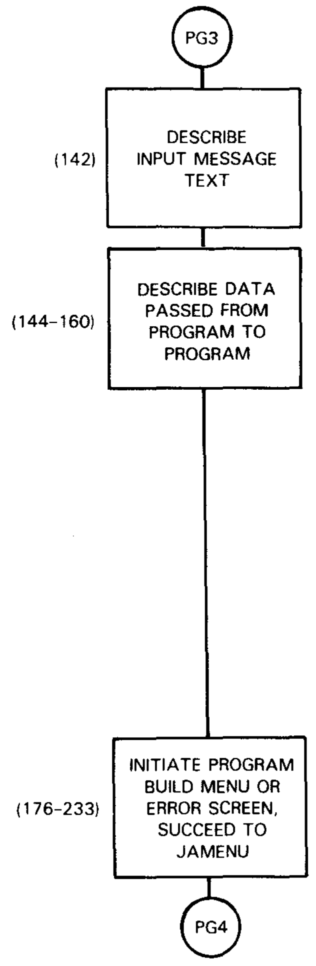
```
LINE NO. SOURCE ENTRY

000001 IDENTIFICATION DIVISION.
000002
000003 PROGRAM-ID.                      JAMENU.
000004*REMARKS.                         PROCESS SIGNON + MENU.
000005*-----------------------------------------------------------------*
000006*     THIS PROGRAM PROCESSES THE SIGNON AND SYSTEM/80 MENU       *
000007*     SCREEN FOR THE ENTITLEMENT ACCOUNTING SYSTEM.              *
000008*     IF THE SIGNON IS FOUND TO BE VALID, THE MENU WILL BE       *
000009*     DISPLAYED. OTHERWISE, THE ERROR OVERLAY SCREEN WILL BE     *
000010*     DISPLAYED AND THE TRANSACTION TERMINATED.                  *
000011*-----------------------------------------------------------------*
000012
000013 ENVIRONMENT DIVISION.
000014
000015 CONFIGURATION SECTION.
000016 SOURCE-COMPUTER.                 UNIVAC-OS3.
000017 OBJECT-COMPUTER.                 UNIVAC-OS3.
000018
000019 DATA DIVISION.
000020
000021 WORKING-STORAGE SECTION.
000022 77  CUST-FILENAME                PIC X(7)       VALUE 'CUSTMST'.
000023 77  SCTL-FILENAME                PIC X(7)       VALUE 'SYSCTL'.
000024
000025 01  SCREEN-FORMAT-IDS.
000026     05  SF-MENU                  PIC X(8)      VALUE 'JA$MENU '.
000027     05  SF-ADD1                  PIC X(8)      VALUE 'JA$ADD1 '.
000028     05  SF-ADD2                  PIC X(8)      VALUE 'JA$ADD2 '.
000029     05  SF-ADD3                  PIC X(8)      VALUE 'JA$ADD3 '.
000030     05  SF-CHG1                  PIC X(8)      VALUE 'JA$CHG1 '.
000031     05  SF-CHG2                  PIC X(8)      VALUE 'JA$CHG2 '.
000032     05  SF-CHG3                  PIC X(8)      VALUE 'JA$CHG3 '.
000033     05  SF-DEL1                  PIC X(8)      VALUE 'JA$DEL1 '.
000034     05  SF-DIS1                  PIC X(8)      VALUE 'JA$DIS1 '.
000035     05  SF-LST1                  PIC X(8)      VALUE 'JA$LST1 '.
000036     05  SF-WAR1                  PIC X(8)      VALUE 'JA$WAR1 '.
000037     05  SF-ERR1                  PIC X(8)      VALUE 'JA$ERR  '.
000038     05  SF-TERM                  PIC X(8)      VALUE 'JA$TERM '.
000039
000040 01  VALID-SUCCESSOR-IDS.
000041     05  MENU                     PIC X(6)       VALUE 'JAMENU'.
000042     05  CUST-ADD                 PIC X(6)       VALUE 'JAADD1'.
000043     05  CUST-CHG-1               PIC X(6)       VALUE 'JACHG1'.
000044     05  CUST-CHG-2               PIC X(6)       VALUE 'JACHG2'.
000045     05  CUST-CHG-3               PIC X(6)       VALUE 'JACHG3'.
000046     05  CUST-DEL                 PIC X(6)       VALUE 'JADEL1'.
000047     05  CUST-DISPLAY             PIC X(6)       VALUE 'JADIS1'.
000048     05  CUST-LIST                PIC X(6)       VALUE 'JALST1'.
000049     05  WS-ACTIVITY              PIC X(6)       VALUE 'JAWAR1'.
000050
000051 01  WS-TABLES.
000052     05  MENU-TABLE.
000053         10  FILLER       PIC X(9)  VALUE '01JAADD1I'.
000054         10  FILLER       PIC X(9)  VALUE '02JACHG1I'.
000055         10  FILLER       PIC X(9)  VALUE '03JACHG2I'.
000056         10  FILLER       PIC X(9)  VALUE '04JACHG3I'.
000057         10  FILLER       PIC X(9)  VALUE '05JADEL1I'.
000058         10  FILLER       PIC X(9)  VALUE '06JADIS1I'.
000059         10  FILLER       PIC X(9)  VALUE '07JALST1I'.
000060         10  FILLER       PIC X(9)  VALUE '08JAWAR1I'.
000061         10  FILLER       PIC X(9)  VALUE '09JAMENUN'.
000062         10  FILLER       PIC X(9)  VALUE '10JAMENUI'.
000063
000064     05  MENU-TBL REDEFINES MENU-TABLE OCCURS 10 TIMES
000065         INDEXED BY MENU-INDX.
000066         10  MENU-SEL                 PIC 9(2).
000067         10  MENU-NAME                PIC X(6).
000068         10  MENU-IND                 PIC X.
000069
000070*****************************************************************
000071*  THIS IS THE ERR PROCESSING TABLE FOR RETRIEVING ERR
000072*  MESSAGES FROM THE SYSTEM CONTROL FILE FOR DISPLAY
000073*  WITH THE OVERLAY.
000074*****************************************************************
```

Figure B-23. Sample Action Program JAMENU Using Screen Formats (Part 1 of 6)


PG2
(75-84)
DESCRIBE ERROR STATUS CODES AND TABLE
(86)
COPY PIB
(88)
COPY INPUT MESSAGE HEADER
(89-97)
DESCRIBE INPUT MESSAGE TEXT
(98-106)
SPACE FOR FUNCTION CALL PARAMETERS
(107-113)
SPACE FOR RECORD CONTROL KEYS AND ERROR STATUS
(114-122)
SPACE FOR DATE AND ERROR FLAG VALUES
(123-140)
SPACE FOR SCREEN RECORD AND ERROR TEXT, PASSWORD, CUSTOMER AND CONTROL RECORDS
(141)
COPY OUTPUT MESSAGE CONTROL HEADER
PG3

```
000075       05  ERR-STATUS-TABLE.
000076           10  FILLER                   PIC X(40)
000077               VALUE  '0104021203130414051506160717080109021020'.
000078
000079       05  ERR-TABLE REDEFINES ERR-STATUS-TABLE OCCURS 10 TIMES
000080                     INDEXED BY ERR-INDX.
000081           10  ERR-CODE                 PIC 99.
000082           10  ERR-KEY                  PIC XX.
000083       05  NO-ERR                       PIC 99 VALUE 25.
000084/
000085 LINKAGE SECTION.
000086 01   PROGRAM-INFORMATION-BLOCK.  COPY PIB.
000087
000088 01   INPUT-MESSAGE-AREA.         COPY IMA.
000089       05  IMA-PASS-1.
000090           10  IMA-DICE               PIC X(4).
000091           10  IMA-SIGNON             PIC X(5).
000092           10  IMA-PASSWRD            PIC X(4).
000093       05  IMA-SCREEN-REC REDEFINES IMA-PASS-1.
000094           10  SR-CUST-NBR                          PIC 9(6).
000095           10  SR-MENU                              PIC 99.
000096           10  SR-TRSMIT                            PIC X.
000097           10  FILLER                               PIC X(4).
000098 01   WORK-AREA.
000099       05  IMS-PARAMETER-LIST.
000100           10  IMS-FILENAME           PIC X(7).
000101           10  IMS-RECORD-AREA        PIC X(256).
000102           10  IMS-KEY                PIC X(14).
000103           10  IMS-FILE-POSITION      PIC X.
000104           10  IMS-ALGN                             COMP-4 SYNC.
000105           10  IMS-SCREEN-ID          PIC X(8).
000106           10  SCREEN-SIZE                 PIC 9(4) COMP SYNC.
000107       05  WA-CONTROL-KEY.
000108           10  CNTL1                  PIC X(2).
000109           10  CNTL2-3                PIC X(4).
000110           10  CNTL23 REDEFINES CNTL2-3.
000111               15  CNTL2              PIC 9(2).
000112               15  CNTL3              PIC X(2).
000113       05  ERR-STATUS                 PIC 99.
000114       05  REFORMAT-DATE.
000115           10  P-MONTH                PIC 99.
000116           10  P-DAY                  PIC 99.
000117           10  P-YEAR                 PIC 99.
000118       05  ERR-FLAG                        PIC X.
000119           88  ERR                          VALUE '1' '2' '3' '4'.
000120           88  SEL-ERR                        VALUE '2'.
000121           88  BLD-ERR                        VALUE '3'.
000122           88  PASSWRD-ERR                    VALUE '4'.
000123       05  SCREEN-RECORD.
000124           10  SR-DATE                     PIC 9(6).
000125           10  SR-TIME                     PIC 9(6).
000126       05  SR-ERR-TEXT                     PIC X(50).
000127       05   SG-STAT                        PIC X(5).
000128       05  PASSWRD-REC.
000129           10  FILLER                      PIC X(2).
000130           10  PASSWRD                     PIC X(4).
000131           10  PASSWRD-SEL                 PIC X(25).
000132           10  PASSWRD-MENU-SEL REDEFINES PASSWRD-SEL
000133                 PIC X OCCURS 25 TIMES INDEXED BY PASSWRD-INDX.
000134           10  FILLER                      PIC X(33).
000135/
000136       05  CUST-RECORD.           COPY CUSTMST.
000137
000138       05  SCTL-RECORD.           COPY SYSCTL.
000139
000140
000141 01   OUTPUT-MESSAGE-AREA.       COPY OMA.
```

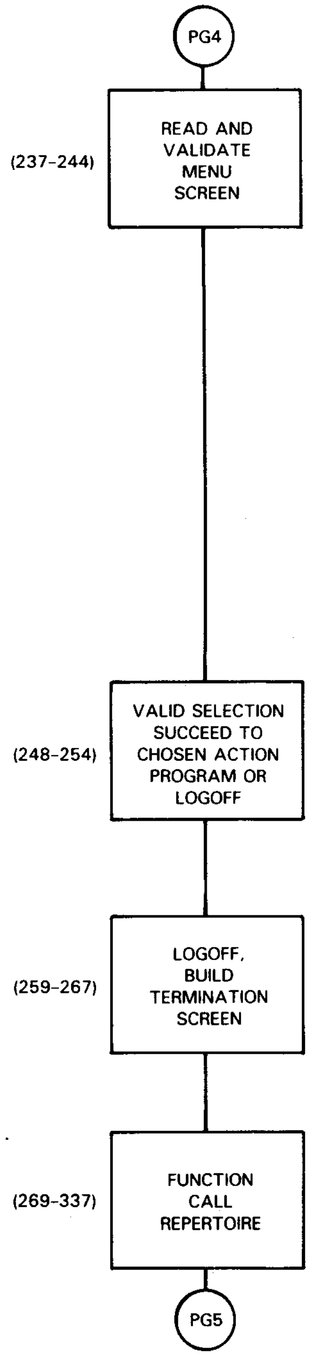
Figure B-23. Sample Action Program JAMENU Using Screen Formats (Part 2 of 6)


PG3
(142)
DESCRIBE INPUT MESSAGE TEXT
(144–160)
DESCRIBE DATA PASSED FROM PROGRAM TO PROGRAM
(176–233)
INITIATE PROGRAM BUILD MENU OR ERROR SCREEN, SUCCEED TO JAMENU
PG4

```
000142      05    OMA-TEXT                         PIC X(3000).
000143
000144 01   CONTINUITY-DATA-AREA.
000145      05  CDA-PASSWRD                        PIC X(4).
000146      05  CDA-MENU-SEL                       PIC X(25).
000147      05  CDA-PASSWRD-MENU-SEL  REDEFINES CDA-MENU-SEL
000148              PIC X OCCURS 25 TIMES.
000149      05  CDA-CUST-KEY.
000150          10  CDA-CUST-NBR               PIC 9(6).
000151      05  CDA-ACCT-CODE                  PIC X(4).
000152      05  PASS-FLAG                          PIC X.
000153          88  PASS-THRU                   VALUE '1''2''3''4''5'.
000154          88  PASS1                       VALUE '1'.
000155          88  PASS2                       VALUE '2'.
000156          88  PASS3                       VALUE '3'.
000157          88  PASS4                       VALUE '4'.
000158          88  PASS5                       VALUE '5'.
000159      05  CDA-STATUS-BYTE                    PIC X.
000160      05 CDA-PROGRAM-NAME                    PIC X(6).
000161/
000162 PROCEDURE DIVISION                USING PROGRAM-INFORMATION-BLOCK
000163                                         INPUT-MESSAGE-AREA
000164                                         WORK-AREA
000165                                         OUTPUT-MESSAGE-AREA
000166                                         CONTINUITY-DATA-AREA.
000167 MAIN-LOOP SECTION.
000168*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
000169*  THE PASS FLAG IN THIS SECTION TELLS THE PROGRAM AT WHICH
000170*  POINT IMS HAS RETURNED CONTROL OF THE PROCESSING TO THE
000171*  PROGRAM A PASS 2 FLAG MEANS THE PROGRAM HAS ALREADY PUT
000172*  OUT THE SCREEN AND IS NOW READY TO ACCEPT THE DATA FROM
000173*  THAT SCREEN TO PROCESS. OTHERWISE THE PROGRAM WANTS TO
000174*  DO THE INITIAL PROCESSING TO PUT OUT A SCREEN.
000175*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
000176 000-BEGIN.
000177     IF  NOT PASS2
000178             PERFORM 100-INITIALIZE
000179             IF  NOT ERR
000180                     PERFORM 200-BUILD-SCREEN
000181             ELSE
000182                     PERFORM 900-ERR-MESSAGE
000183     ELSE
000184             PERFORM 250-READ-SCREEN.
000185     PERFORM 507-RETURN.
000186
000187*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
000188*  INITIALIZATION OF FIELDS AND FLAGS IS DONE HERE.
000189*  ALSO CHECKING IS DONE TO SEE IF THE PROGRAM ENTERED
000190* FROM A SIGN ON OR WAS CALLED FROM ANOTHER PROGRAM.
000191*  ONLY IF ENTER FROM A SIGN ON DOES THE PROGRAM RETRIEVE
000192* THE PASSWORD RECORD. OTHER WISE IT IS CARRIED TO CHECK
000193*  VALIDITY OF MENU SELECTION.
000194*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
000195 100-INITIALIZE.
000196     MOVE SPACE                                TO IMS-KEY
000197                                                  IMS-FILENAME
000198                                                  SR-ERR-TEXT.
000199     MOVE '2'                                  TO PASS-FLAG.
000200     MOVE '0'                                  TO ERR-FLAG.
000201     MOVE CORR P-TRANSACTION-DATE              TO REFORMAT-DATE.
000202     IF  CDA-PROGRAM-NAME  EQUAL LOW-VALUES
000203         MOVE IMA-PASSWRD                      TO CNTL2-3
000204         MOVE 'PW'                             TO CNTL1
000205         MOVE SPACE                            TO IMS-RECORD-AREA
000206         MOVE SCTL-FILENAME                    TO IMS-FILENAME
000207         MOVE WA-CONTROL-KEY                   TO IMS-KEY
000208         PERFORM 502-GET
000209         IF  ERR
000210             MOVE '4'                          TO ERR-FLAG
000211         ELSE
000212             MOVE IMS-RECORD-AREA              TO PASSWRD-REC
000213             MOVE PASSWRD-SEL                  TO CDA-MENU-SEL
000214             MOVE PASSWRD                      TO CDA-PASSWRD.
000215     MOVE MENU                                 TO CDA-PROGRAM-NAME.
000216
```

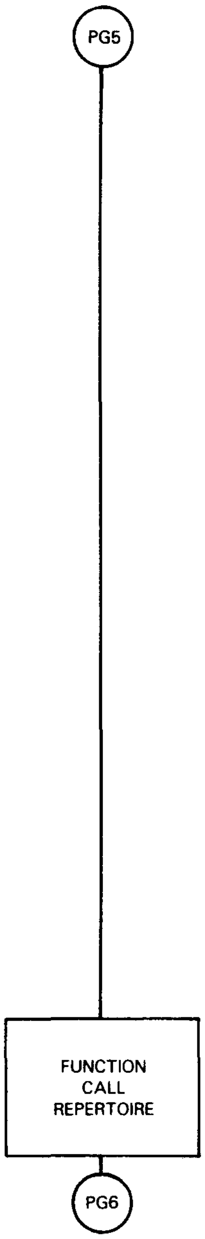
Figure B–23. Sample Action Program JAMENU Using Screen Formats (Part 3 of 6)

SCREEN FORMAT SERVICES IN COBOL: JAMENU PROGRAM


PG4
READ AND VALIDATE MENU SCREEN
(237-244)
VALID SELECTION SUCCEED TO CHOSEN ACTION PROGRAM OR LOGOFF
(248-254)
LOGOFF, BUILD TERMINATION SCREEN
(259-267)
FUNCTION CALL REPERTOIRE
(269-337)
PG5

```
000217*************************************************************
000218*  THE MAIN PROCESSING AND BUILDING OF THE SCREEN DATA IS DONE
000219* IN HERE.
000220*************************************************************
000221 200-BUILD-SCREEN.
000222     MOVE IMA-SOURCE-TERMINAL-ID        TO OMA-DESTINATION-TERM-ID.
000223     MOVE SF-MENU                       TO IMS-SCREEN-ID.
000224     MOVE ALL '0'                       TO SCREEN-RECORD.
000225     MOVE REFORMAT-DATE                 TO SR-DATE.
000226     MOVE P-TIME-OF-DAY                 TO SR-TIME.
000227     MOVE 12                            TO SCREEN-SIZE.
000228     PERFORM 505-BUILD.
000229     IF ERR
000230        PERFORM 900-ERR-MESSAGE.
000231     MOVE MENU                          TO PIB-SUCCESSOR-ID.
000232     MOVE 'E'                           TO PIB-TERMINATION-IND.
000233
000234*************************************************************
000235*  THE MENU SELECTION VALIDITY IS CHECKED HERE
000236*************************************************************
000237 250-READ-SCREEN.
000238     IF CDA-PASSWRD-MENU-SEL (SR-MENU) NOT EQUAL TO '1'
000239              MOVE '2'                  TO ERR-FLAG.
000240     IF  ERR
000241              PERFORM 900-ERR-MESSAGE
000242     ELSE
000243              PERFORM 260-SET-MENU.
000244
000245*************************************************************
000246* IF MENU SELECTION IS VALID CONTROL IS SET FOR NEXT STAGE.
000247*************************************************************
000248 260-SET-MENU.
000249     MOVE MENU-NAME (SR-MENU)           TO PIB-SUCCESSOR-ID.
000250     MOVE MENU-IND (SR-MENU)            TO PIB-TERMINATION-IND.
000251     MOVE SR-CUST-NBR                   TO CDA-CUST-NBR.
000252     MOVE '0'                           TO PASS-FLAG.
000253     IF SR-MENU = 9
000254        PERFORM 270-LOGOFF.
000255*
000256*************************************************************
000257*  BUILD TERMINATION SCREEN
000258*************************************************************
000259 270-LOGOFF.
000260     MOVE IMA-SOURCE-TERMINAL-ID        TO OMA-DESTINATION-TERM-ID.
000261     MOVE SF-TERM                       TO IMS-SCREEN-ID.
000262     MOVE ALL '0'                       TO SCREEN-RECORD.
000263     MOVE CORR P-TRANSACTION-DATE       TO REFORMAT-DATE.
000264     MOVE REFORMAT-DATE                 TO SR-DATE.
000265     MOVE P-TIME-OF-DAY                 TO SR-TIME.
000266     MOVE 12                            TO SCREEN-SIZE.
000267     PERFORM 505-BUILD.
000268/
000269 IMS-CALLS SECTION.
000270
000271 500-SETL.
000272     CALL 'SETL'                 USING IMS-FILENAME
000273                                       IMS-FILE-POSITION.
000274 500-EXIT.
000275     EXIT.
000276
000277 501-ESETL.
000278     CALL 'ESETL'                USING IMS-FILENAME.
000279     IF  PIB-STATUS-CODE IS GREATER THAN 0
000280              MOVE '1' TO ERR-FLAG.
000281 501-EXIT.
000282     EXIT.
000283
000284 502-GET.
000285     CALL 'GET'                  USING IMS-FILENAME
000286                                       IMS-RECORD-AREA
000287                                       IMS-KEY.
000288     IF  PIB-STATUS-CODE IS GREATER THAN 0
000289              MOVE '1' TO ERR-FLAG.
000290 502-EXIT.
```

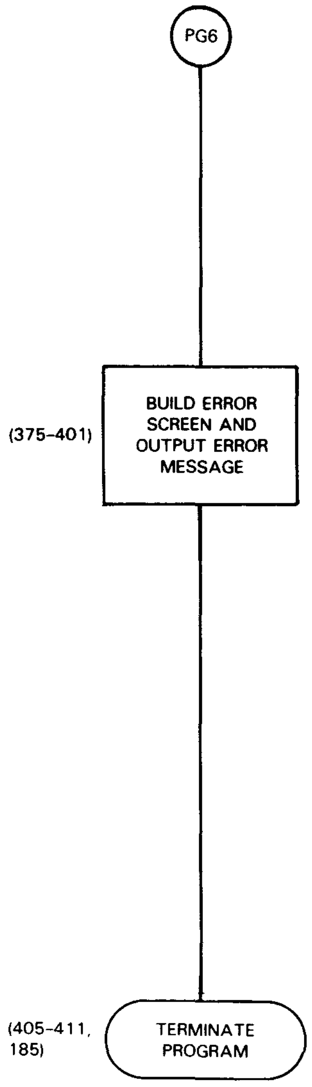
Figure B-23. Sample Action Program JAMENU Using Screen Formats (Part 4 of 6)



```
000291      EXIT.
000292
000293 503-GETUP.
000294      CALL 'GETUP'                  USING IMS-FILENAME
000295                                          IMS-RECORD-AREA
000296                                          IMS-KEY.
000297      IF  PIB-STATUS-CODE IS GREATER THAN 0
000298              MOVE '1' TO ERR-FLAG.
000299 503-EXIT.
000300      EXIT.
000301
000302 504-PUT.
000303      CALL 'PUT'                    USING IMS-FILENAME
000304                                          IMS-RECORD-AREA.
000305      IF  PIB-STATUS-CODE IS GREATER THAN 0
000306              MOVE '1' TO ERR-FLAG.
000307 504-EXIT.
000308      EXIT.
000309
000310*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
000311*  CALL FOR MAIN SCREEN FOR PROGRAM
000312*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
000313 505-BUILD.
000314      CALL 'BUILD'                  USING OUTPUT-MESSAGE-AREA
000315                                          IMS-SCREEN-ID
000316                                          SCREEN-RECORD
000317                                          SCREEN-SIZE
000318                                          SG-STAT.
000319      IF PIB-STATUS-CODE IS GREATER THAN 0
000320         MOVE '3' TO ERR-FLAG.
000321 505-EXIT.
000322      EXIT.
000323
000324*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
000325*  CALL FOR ERR OVERLAY SCREEN
000326*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
000327 505-BUILD-ERR.
000328      CALL 'BUILD'                  USING OUTPUT-MESSAGE-AREA
000329                                          IMS-SCREEN-ID
000330                                          SR-ERR-TEXT
000331                                          SCREEN-SIZE
000332                                          SG-STAT.
000333      IF PIB-STATUS-CODE IS GREATER THAN 0
000334         MOVE '3' TO ERR-FLAG.
000335 505-BLD-ERR-EXIT.
000336      EXIT.
000337
000338 506-REBUILD.
000339      CALL 'REBUILD'                USING IMS-SCREEN-ID
000340                                          IMS-RECORD-AREA.
000341 506-EXIT.
000342      EXIT.
000343
000344 507-RETURN.
000345      CALL 'RETURN'.
000346 507-EXIT.
000347      EXIT.
000348
000349 508-INSERT.
000350      CALL 'INSERT'                 USING IMS-FILENAME
000351                                          IMS-RECORD-AREA
000352                                          IMS-KEY.
000353      IF  PIB-STATUS-CODE IS GREATER THAN 0
000354              MOVE '1'                    TO ERR-FLAG.
000355 508-EXIT.
000356      EXIT.
000357
000358 599-SNAP.
```

Figure B–23. Sample Action Program JAMENU Using Screen Formats (Part 5 of 6)

SCREEN FORMAT SERVICES IN COBOL: JAMENU PROGRAM



```
000359      MOVE 'S' TO PIB-TERMINATION-IND.
000360      CALL 'SNAP' USING PROGRAM-INFORMATION-BLOCK CDA-STATUS-BYTE.
000361 599-EXIT.
000362      EXIT.
000363/
000364 ERROR-PROCESSING SECTION.
000365*
000366*************************************************************
000367*  ERR PROCESSING IS DONE HERE.  THE TYPE OF ERR IS
000368*  DETERMINED AND A SEARCH OF THE ERR TABLE IS MADE
000369*  THE APPROPRIATE ERR MESSAGE IS RETRIEVED FROM THE
000370*  SYSTEM CONTROL FILE AND DISPLAY IT ON THE OVERLAY
000371*  ERR 8 IS ILLEGAL PASS WORD ERR 9 IS ILLEGAL MENU
000372*  SELECTION FOR PASSWORD  OTHER ERRS CONFORM TO IMS
000373*  STATUS ERRORS.
000374**************************************************************
000375 900-ERR-MESSAGE.
000376      MOVE SPACE                              TO IMS-RECORD-AREA
000377                                                 IMS-FILENAME
000378                                                 IMS-KEY.
000379      MOVE SCTL-FILENAME                      TO IMS-FILENAME.
000380      IF PASSWRD-ERR
000381         MOVE '5'                             TO PASS-FLAG
000382         MOVE 8                               TO ERR-STATUS
000383      ELSE
000384      IF  SEL-ERR
000385              MOVE 9                          TO ERR-STATUS
000386      ELSE
000387          MOVE PIB-STATUS-CODE                TO ERR-STATUS.
000388      MOVE 'EM'                               TO CNTL1.
000389      MOVE SPACE                              TO CNTL3.
000390      SET ERR-INDX                            TO 1.
000391      SEARCH ERR-TABLE
000392          AT END
000393              MOVE NO-ERR                     TO CNTL2
000394          WHEN ERR-CODE (ERR-INDX) IS EQUAL TO ERR-STATUS
000395              MOVE ERR-KEY (ERR-INDX)         TO CNTL2.
000396      MOVE WA-CONTROL-KEY                     TO IMS-KEY.
000397      PERFORM 502-GET.
000398      MOVE IMS-RECORD-AREA                    TO SCTL-RECORD.
000399      MOVE ALL '0' TO SR-ERR-TEXT.
000400      MOVE SCERR-TEXT                         TO SR-ERR-TEXT.
000401      MOVE SF-ERR1                            TO IMS-SCREEN-ID.
000402      MOVE 50            TO SCREEN-SIZE.
000403      MOVE IMA-SOURCE-TERMINAL-ID             TO OMA-DESTINATION-TERM-ID.
000404      PERFORM 505-BUILD-ERR.
000405      IF PASS5
000406         MOVE MENU                            TO PIB-SUCCESSOR-ID
000407         MOVE 'N'                             TO PIB-TERMINATION-IND
000408      ELSE
000409         MOVE MENU                           TO PIB-SUCCESSOR-ID
000410         MOVE 'E'                            TO PIB-TERMINATION-IND
000411         MOVE '0'                            TO PASS-FLAG.
```

Figure B-23. Sample Action Program JAMENU Using Screen Formats (Part 6 of 6)

JAMENU discussion

The following discussion of the JAMENU action program assumes that you have already created a menu screen format called JA$MENU and filed it in the screen format file. Any line numbers referenced in this discussion refer to the code in the JAMENU action program, Figure B-23. Also, expansions of the program information block, input message area, and output message area cannot be seen in this listing; however, their fields may be referenced in the code (e.g., lines 406 and 407) and are available to JAMENU.

Files used

JAMENU uses two files (lines 22 and 23):

1. CUSTMST file
2. SYSCTL file

Record types in CUSTMST file

The CUSTMST file contains customer information. The SYSCTL file contains four types of records:

1. Account access records (AA)
2. Branch records (BR)
3. Error message text records (EM)
4. Password records (PW)

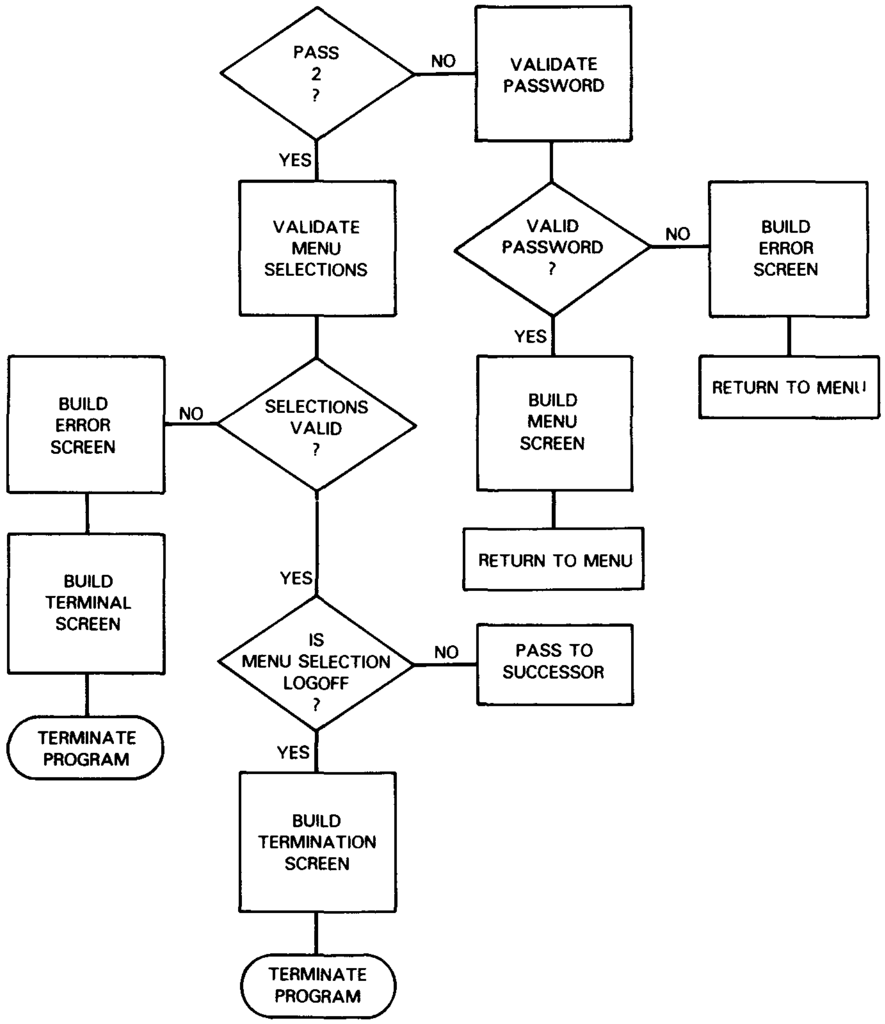
Each type record is identified by a 2-byte control key field. (See lines 108-112 and 129.) JAMENU accesses the SYSCTL file to validate passwords and retrieve error messages for display in the error message screen format.

JAMENU routines

JAMENU performs five types of routines. It:

1. validates passwords;
2. builds menu screen;
3. validates menu selections;
4. builds error screen; and
5. builds termination screen

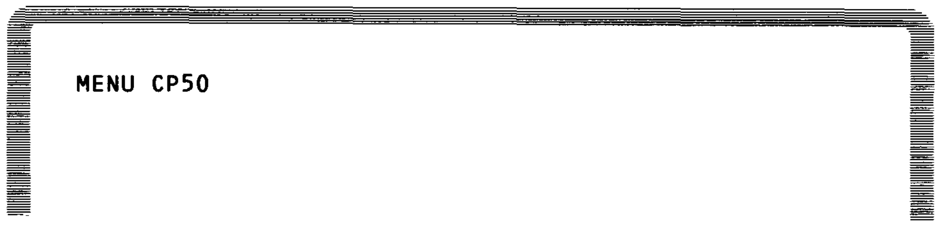
The following general flowchart shows these main routines in the JAMENU program.



JAMENU GENERAL FLOWCHART

Processing JAMENU

Begin executing the JAMENU program by entering the transaction code, MENU, followed by the password. This is considered the sign-on or first pass through JAMENU.

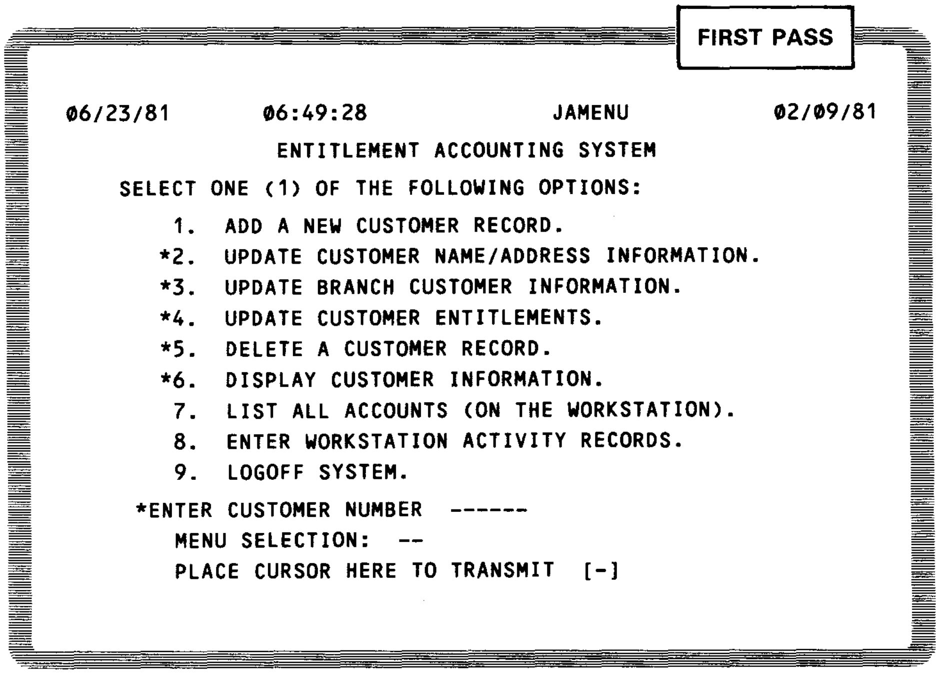


```
MENU CP50
```

Processing password

On the first pass, JAMENU accesses the SYSCTL file to validate the password entered at the terminal. If the password is valid, JAMENU saves all data pertinent to that password in the continuity data area (line 211–216), builds the menu screen (lines 221–232), and terminates in external succession to itself (JAMENU). Menu screen JA$MENU follows.

Menu screen



```
ø6/23/81        ø6:49:28              JAMENU           ø2/ø9/81
                  ENTITLEMENT ACCOUNTING SYSTEM
     SELECT ONE (1) OF THE FOLLOWING OPTIONS:
          1.  ADD A NEW CUSTOMER RECORD.
         *2.  UPDATE CUSTOMER NAME/ADDRESS INFORMATION.
         *3.  UPDATE BRANCH CUSTOMER INFORMATION.
         *4.  UPDATE CUSTOMER ENTITLEMENTS.
         *5.  DELETE A CUSTOMER RECORD.
         *6.  DISPLAY CUSTOMER INFORMATION.
          7.  LIST ALL ACCOUNTS (ON THE WORKSTATION).
          8.  ENTER WORKSTATION ACTIVITY RECORDS.
          9.  LOGOFF SYSTEM.
      *ENTER CUSTOMER NUMBER  ------
         MENU SELECTION:  --
         PLACE CURSOR HERE TO TRANSMIT  [-]
```

Building menu screen

In the menu screen build routine (lines 221–232), the BUILD function call that actually calls the menu screen identifies the buffer address where IMS receives the screen format as the output message area (line 314); the format name as IMS-SCREEN-ID (line 315, defined on line 105); the variable data as SCREEN-RECORD (line 316, defined on lines 123–125); the data size as SCREEN-SIZE (line 317, defined on line 106); and, the output status as SG-STAT (line 318, defined on line 127).

Notice, all the parameters you specify on the BUILD function must be defined in the work area.

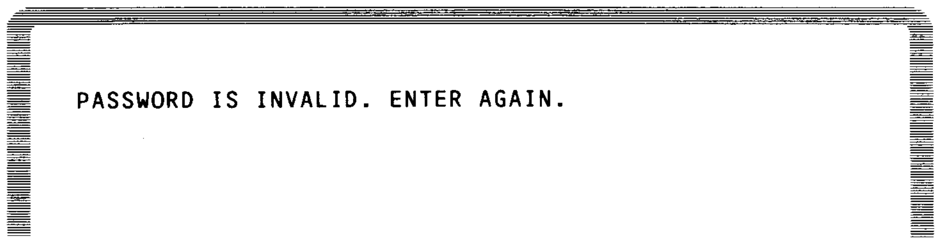
Unsuccessful BUILD

If the BUILD function is unsuccessful (lines 319 and 320), JAMENU moves an error code of 3 to the ERR-FLAG lines 118 and 121) indicating a build error.

Invalid password

If the password is invalid on the first pass, JAMENU accesses the SYSCTL file via the EM record key for the error message record (lines 380–388), searches an error table to find the appropriate error message (lines 390–395), retrieves that error message (lines 396–398), builds the error message screen (lines 399–404), and terminates in external succession to itself (lines 408–411). The password error screen follows:

Password error screen



```
PASSWORD IS INVALID. ENTER AGAIN.
```

Menu selection validation

On the second pass through JAMENU, the program tests the menu selection made, to see if it is accessible to the password specified in the first pass. If the menu selection is valid for that password, JAMENU performs 260-SET-MENU (lines 248–255). This moves the correct program name to process the menu selection to the successor-id and an I to the termination indicator.

Notice here that the programmer has set up a menu table (lines 52–62) containing not only the menu selection numbers and their corresponding action programs but also the termination indicators used to end each action program. The menu is redefined with selection numbers (MENU-SEL) in the first two bytes of each table field, the action program names are in the next 6 bytes (MENU-NAME), and, finally, the termination indicators are in the last byte of each field (MENU-IND).

Succession and termination from table

When the program moves the successor-id and termination indicator to the program information block (lines 248–250), it moves the menu name indexed by the menu number entered at the terminal. JAMENU picks up the correct program name for the successor-id by using this index value to reference the first two bytes of the menu table entry. Likewise, JAMENU moves the termination indicator value to the program information block by using the index value to reference the last byte of the menu table entry chosen.

Redefining the menu table (lines 52–68) saves coding by making three types of data accessible in one table: the menu selection numbers, action program names for successor-ids, and termination indicators.

Process invalid menu selection

If the menu selection is invalid, JAMENU moves code 2 indicating selection error to ERR-FLAG (lines 237–241), builds the menu selection error message screen (lines 375–411), and succeeds externally to itself.

Several tests occur in the beginning error message building routine. The first separates password errors from menu selection errors and function call errors (lines 380–387).

Password error

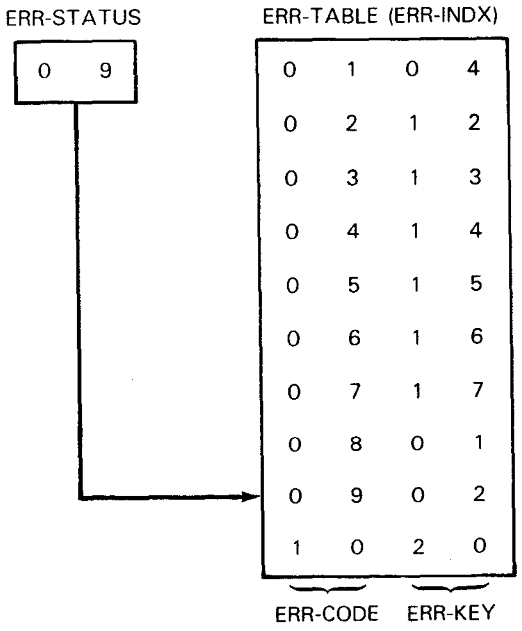
For a password error, JAMENU places code 5 in the pass flag to force the normal termination of the transaction and moves 8 to the work area location, ERR-STATUS (lines 380–382).

Menu selection error

For a menu selection error, JAMENU moves a 9 to ERR-STATUS in the work area (lines 113, 384, and 385). This code corresponds to one of the values 01 through 10 contained in the first two bytes of each table entry in the ERR-TABLE. These leading two bytes in each table entry also correspond to the index value being used to search ERR-TABLE (lines 75–83). Thus, when the value in ERR-STATUS equals the value in the first two bytes of an ERR-TABLE entry, JAMENU moves the contents of ERR-KEY (the last two bytes in the corresponding ERR-TABLE entry) to the record key area used to retrieve that error message record from the SYSCLT file (lines 394 and 395).

Obtaining error message record

The following diagram illustrates the ERR-TABLE, its index (ERR-INDX), and the way JAMENU uses the value in ERR-STATUS to find the ERR-KEY value in the table by searching ERR-TABLE for the error code (ERR-CODE) that matches the value in ERR-STATUS.



ERR-CODE	ERR-KEY
0 1	0 4
0 2	1 2
0 3	1 3
0 4	1 4
0 5	1 5
0 6	1 6
0 7	1 7
0 8	0 1
0 9	0 2
1 0	2 0

JAMENU clears the work-area locations (lines 376–378). It moves the SYSCTL file name to the work area file name to prepare for retrieval of the SYSCTL record. This record contains the 'EM' prefix, the error message number to be sent to the screen, and the error message text (line 379).

To find the appropriate error message corresponding to the password error menu selection error, or other function call error, JAMENU searches the table, ERR-TABLE (lines 390–395). If it finds no corresponding error code, it moves a message number of 25 (line 83) to the key field (CNTL-2, line 395) used to call the corresponding record from the SYSCTL error message file (lines 396 and 397 and 284–289).

Example of ERR-TABLE search

If, for example, JAMENU finds an 09 error code (lines 394 and 395), JAMENU uses error message number 02 from the ERR-TABLE (see ERR-TABLE diagram and coding line 77) as a key to locate the corresponding error message text in the SYSCTL file (lines 102, 107–112, and 396 and 397).

When JAMENU retrieves the SYSCTL error message (EM) record, it uses this message number to locate the error message text immediately following the 02 error number on the SYSCTL record. JAMENU then uses this message text in building the error message screen.

Notice in lines 398–404, including lines 327–334, that JAMENU clears the screen error text area to receive the error message text from the SYSCTL file; identifies the terminal to receive the error message; transmits the message; and terminates in external succession to itself. If a build error occurs, JAMENU sets the error flag to 3 and succeeds externally to itself.

Process valid menu selection

If the menu selection including customer number is valid, JAMENU executes another short routine (260-SET-MENU, lines 248–254) that passes control to the appropriate action program to process the menu selection. This routine also checks for a logoff menu selection (9) that builds the termination screen similarly to the way JAMENU built the error message screen (lines 259–267). Successor programs selected from the menu perform file operations required. When processing is complete, control returns to the JAMENU program via immediate internal succession and the terminal operator again receives the menu screen to enter another selection.

B.5. SAMPLE COBOL ACTION PROGRAM PERFORMING OUTPUT-FOR-INPUT QUEUEING (BEGIN1)

BEGIN1 menu selection

The BEGIN1 action program (Figure B-24) initiates a continuous output print transaction at a terminal other than the source terminal. (See Figure B-25 for an action program performing continuous output.) To do this, BEGIN1 uses output-for-input queueing. By placing the output-for-input queueing function code into the AUX-FUNCTION field of the output message area header, BEGIN1 queues its output message as input to a different terminal.

The program also issues messages to the source terminal operator telling him whether the output message was successfully or unsuccessfully delivered to the destination terminal.

Processing BEGIN1

When activated at the source terminal, BEGIN1 expects an input message in the following format (lines 61-65):

```
BEGIN dest-terminal text
```

where:

`BEGIN`
: Is the 5-character transaction code the terminal operator enters to activate BEGIN1. (BEGIN should also appear in the configurator TRANSACT section).

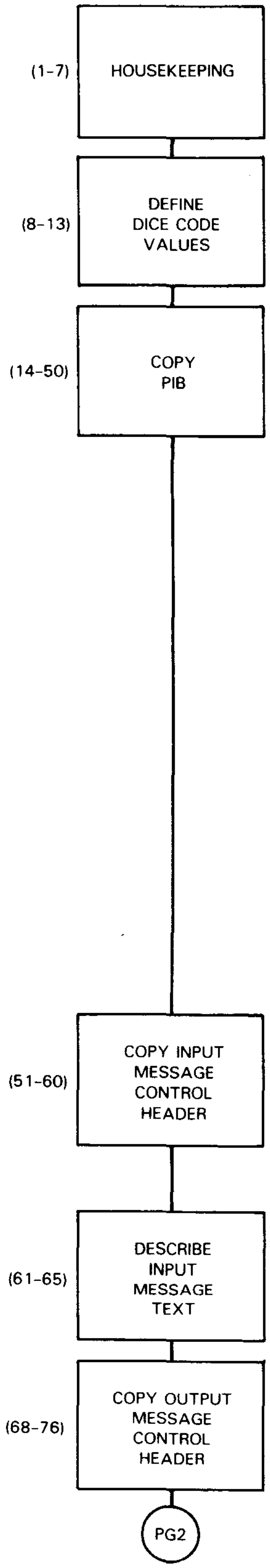
`dest-terminal`
: Is the 4-character *terminal-id* of the destination terminal where the continuous output print transaction is initiated. (Assign this same terminal-id in the ICAM network definition.)

`text`
: Is the alphanumeric text entered by the source terminal operator. This text is the input message expected by the print transaction that performs continuous output at the destination terminal. It must begin with the transaction code that causes scheduling to initiate the transaction.

Compilation and flowchart

A flowchart describing the corresponding lines of BEGIN1 code is to the left of Figure B-24.

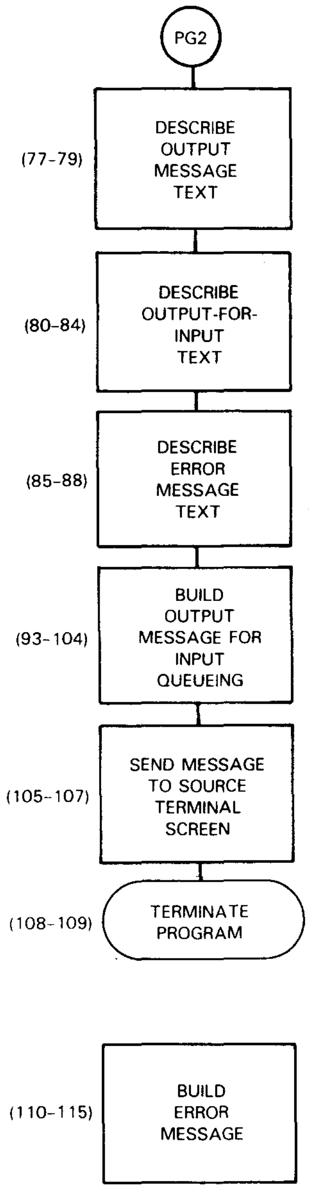
OUTPUT-FOR-INPUT QUEUEING IN COBOL: BEGIN 1 PROGRAM



(1-7)
HOUSEKEEPING
(8-13)
DEFINE DICE CODE VALUES
(14-50)
COPY PIB
(51-60)
COPY INPUT MESSAGE CONTROL HEADER
(61-65)
DESCRIBE INPUT MESSAGE TEXT
(68-76)
COPY OUTPUT MESSAGE CONTROL HEADER
PG2

```
LINE NO.SOURCE ENTRY

00001 IDENTIFICATION DIVISION.
00002 PROGRAM-ID. BEGIN1.
00003 ENVIRONMENT DIVISION.
00004 CONFIGURATION SECTION.
00005 SOURCE-COMPUTER. UNIVAC-OS3.
00006 OBJECT-COMPUTER. UNIVAC-OS3.
00007 DATA DIVISION.
00008 WORKING-STORAGE SECTION.
00009 01  DICE-SEQ.
00010     02 DICE-CODE                PIC X       VALUE ='10'.
00011     02 FUNC-CODE                PIC X       VALUE ='01'.
00012     02 Y-COORD                  PIC X       VALUE ='0C'.
00013     02 X-COORD                  PIC X       VALUE ='00'.
00014 LINKAGE SECTION.
00015 01  PROGRAM-INFORMATION-BLOCK. COPY PIB74.
00016     02  STATUS-CODE               PIC 9(4) COMP-4.
00017     02  DETAILED-STATUS-CODE      PIC 9(4) COMP-4.
00018     02  RECORD-TYPE REDEFINES DETAILED-STATUS-CODE.
00019         03 PREDICTED-RECORD-TYPE  PIC X.
00020         03 DELIVERED-RECORD-TYPE  PIC X.
00021     02  SUCCESSOR-ID              PIC X(6).
00022     02  TERMINATION-INDICATOR     PIC X.
00023     02  LOCK-ROLLBACK-INDICATOR   PIC X.
00024     02  TRANSACTION-ID.
00025         03 YEAR                   PIC 9(4) COMP-4.
00026         03 TODAY                  PIC 9(4) COMP-4.
00027         03 HR-MIN-SEC             PIC 9(9) COMP-4.
00028     02  DATA-DEF-REC-NAME         PIC X(7).
00029     02  DEFINED-FILE-NAME         PIC X(7).
00030     02  STANDARD-MSG-LINE-LENGTH  PIC 9(4) COMP-4.
00031     02  STANDARD-MSG-NUMBER-LINES PIC 9(4) COMP-4.
00032     02  WORK-AREA-LENGTH          PIC 9(4) COMP-4.
00033     02  CONTINUITY-DATA-INPUT-LENGTH   PIC 9(4) COMP-4.
00034     02  CONTINUITY-DATA-OUTPUT-LENGTH   PIC 9(4) COMP-4.
00035     02  WORK-AREA-INC             PIC 9(4) COMP-4.
00036     02  CONTINUITY-DATA-AREA-INC  PIC 9(4) COMP-4.
00037         02  SUCCESS-UNIT-ID.
00038             03 TRANSACTION-DATE.
00039                04 YEAR                  PIC 99.
00040                04 MONTH                 PIC 99.
00041                04 TODAY                 PIC 99.
00042             03 TIME-OF-DAY.
00043                04 HOUR                  PIC 99.
00044                04 MINUTE                PIC 99.
00045                04 SECOND                PIC 99.
00046         03 UNIQUE-SUFFIX          PIC 999.
00047     02  SOURCE-TERMINAL-CHARS.
00048         03 SOURCE-TERMINAL-TYPE  PIC X.
00049         03 SOURCE-TERM-MSG-LINE-LENGTH   PIC 9(4) COMP-4.
00050         03 SOURCE-TERM-MSG-NUMBER-LINES  PIC 9(4) COMP-4.
00051 01  INPUT-MESSAGE-AREA. COPY IMA74.
00052     02 SOURCE-TERMINAL-ID               PIC X(4).
00053     02 DATE-TIME-STAMP.
00054        03 YEAR                          PIC 9(4) COMP-4.
00055        03 TODAY                         PIC 9(4) COMP-4.
00056        03 HR-MIN-SEC                    PIC 9(9) COMP-4.
00057     02 TEXT-LENGTH                      PIC 9(4) COMP-4.
00058     02 AUXILIARY-DEV-ID.
00059        03 FILLER                       PIC X.
00060        03 AUX-DEV-NO                   PIC X.
00061     02 TRANS-CODE                PIC X(5).
00062     02 FILLER                    PIC X.
00063     02 DEST-TERM                 PIC X(4).
00064     02 FILLER                    PIC X.
00065     02 TEXT-AREA                 PIC X(29).
00066 01  WORK-AREA.
00067     02 DUMMY                     PIC X.
00068 01  OUTPUT-MESSAGE-AREA.      COPY OMA74.
00069     02 DESTINATION-TERMINAL-ID        PIC X(4).
00070              02 SFS-OPTIONS                      PIC X(2).
00071              02 FILLER                           PIC X(2).
00072     02 CONTINUOUS-OUTPUT-CODE         PIC X(4).
```

Figure B-24. Sample Action Program BEGIN1 Using Output-for-Input Queueing (Part 1 of 2)


PG2
(77-79)
DESCRIBE OUTPUT MESSAGE TEXT
(80-84)
DESCRIBE OUTPUT-FOR-INPUT TEXT
(85-88)
DESCRIBE ERROR MESSAGE TEXT
(93-104)
BUILD OUTPUT MESSAGE FOR INPUT QUEUEING
(105-107)
SEND MESSAGE TO SOURCE TERMINAL SCREEN
(108-109)
TERMINATE PROGRAM
(110-115)
BUILD ERROR MESSAGE

```
00073         02 TEXT-LENGTH                PIC 9(4)   COMP-4.
00074         02 AUXILIARY-DEVICE-ID.
00075            03 AUX-FUNCTION               PIC X.
00076            03 AUX-DEVICE-NO              PIC X.
00077         02 SEND-MSG.
00078            03 OUTPUT-TEXT             PIC X(29).
00079            03 FILLER                  PIC X(14).
00080         02 BEGIN-MSG REDEFINES SEND-MSG.
00081            03 CURSOR-1                PIC X(4).
00082            03 MSG-1                   PIC X(30).
00083            03 TERM-NAME               PIC X(4).
00084            03 FILLER                  PIC X(5).
00085         02 ERROR-MSG REDEFINES SEND-MSG.
00086            03 CURSOR-2                PIC X(4).
00087            03 MSG-2                   PIC X(35).
00088            03 ERROR-CODE              PIC ZZZZ.
00089     PROCEDURE DIVISION USING PROGRAM-INFORMATION-BLOCK
00090                              INPUT-MESSAGE-AREA
00091                              WORK-AREA
00092                              OUTPUT-MESSAGE-AREA.
00093     MOVE-MESSAGE.
00094         MOVE DEST-TERM TO DESTINATION-TERMINAL-ID.
00095         SUBTRACT 11 FROM TEXT-LENGTH IN INPUT-MESSAGE-AREA
00096                   GIVING TEXT-LENGTH IN OUTPUT-MESSAGE-AREA
00097         MOVE 'I' TO AUX-FUNCTION.
00098         MOVE TEXT-AREA TO OUTPUT-TEXT.
00099         CALL 'SEND' USING OUTPUT-MESSAGE-AREA.
00100         IF STATUS-CODE NOT EQUAL TO 0 GO TO ERROR-PROC.
00101         MOVE DICE-SEQ TO CURSOR-1.
00102         MOVE 'TRANSACTION BEGUN AT TERMINAL ' TO MSG-1
00103         MOVE DEST-TERM TO TERM-NAME.
00104         MOVE 42 TO TEXT-LENGTH IN OUTPUT-MESSAGE-AREA.
00105     TERMINATE-ROUTINE.
00106         MOVE LOW-VALUES TO DESTINATION-TERMINAL-ID.
00107         MOVE LOW-VALUE TO AUX-FUNCTION.
00108         MOVE 'N' TO TERMINATION-INDICATOR.
00109         CALL 'RETURN'.
00110     ERROR-PROC.
00111         MOVE DICE-SEQ TO CURSOR-2.
00112         MOVE 'TRANSACTION NOT BEGUN DUE TO ERROR ' TO MSG-2.
00113         MOVE DETAILED-STATUS-CODE TO ERROR-CODE.
00114         MOVE 47 TO TEXT-LENGTH IN OUTPUT-MESSAGE-AREA.
00115         GO TO TERMINATE-ROUTINE.
```

Figure B-24. Sample Action Program BEGIN1 Using Output-for-Input Queueing (Part 2 of 2)

Setting output-for-input queueing

When BEGIN1 is activated, the MOVE-MESSAGE routine forms an output message that is queued as input for the destination terminal. Line 94 places the destintation-terminal named in the input message into the output message header. Lines 95 and 96 specify the length of the output message, including four bytes for the TEXT-LENGTH field. Line 97 sets the AUXILIARY-FUNCTION field of the output message area header to the value (X'C9' or C'I') that directs IMS to queue the output message as input for the destination terminal. In line 99, the SEND function transmits the output message to the destination terminal.

Successful SEND

If IMS encounters no errors in executing the SEND function, the operator of the originating terminal receives a message indicating that the print transaction was successfully queued at the destination terminal. Lines 101 and 102 provide the screen positioning and text of the message sent to the operator of the originating terminal. Line 106 sets the DESTINATION- TERMINAL-ID field of the output message area header to binary 0 and thus ensures that this message is sent to the source terminal. Line 107 ensures that this message is sent to the UNISCOPE screen instead of to the communications output printer (COP).

BEGIN1 terminates normally without succession (lines 108 and 109) and the source terminal is freed for other interactive use.

Queueing error

On the other hand, if IMS encounters an error in queueing the message output by BEGIN1 as input to the destination terminal, the ERROR-PROC routine (line 100 and 110–115) formats an error message for output to the originating operator, and BEGIN1 terminates normally (lines 108 and 109). The output message is dequeued. The operator, depending on the nature of the error, may reenter the original input message.

Successful SEND message

Although the text of the message sent to the source terminal on successful return from the SEND function (line 102) states 'TRANSACTION BEGUN AT TERMINAL', this may not be true. All that actually occurred was that the output message was successfully queued as input from the destination terminal. If the transaction code it contains is invalid, however, or some other error intervenes, the print transaction does not begin. IMS does not report such occurrences to the originating action program, but to the destination terminal.

BEGIN1 analysis

Remember, the purpose of BEGIN1 is to initiate a transaction at another terminal by sending a transaction code in the output message it queues as input to the destination terminal. Suppose the terminal operator enters this input:

```
BEGIN TRM5 PRINT ORDFILE 5732468 TRM1 COP
```

Initiating transaction at another terminal

The MOVE statement on line 98 places this input into the output text area. The message entered by the terminal operator contains the transaction code needed to start the transaction at the destination terminal.

BEGIN1 redefines the output message text area to handle both a successful and an unsuccessful SEND operation.

Unsuccessful SEND function

If the SEND function is unsuccessful, BEGIN1 positions the cursor and moves the unsuccessful SEND message text to the output text. In this case, the source terminal operator receives the message,

```
TRANSACTION NOT BEGUN DUE TO  ERROR Ø6Ø4
```

By examining the status and detailed status codes in Table D-4, you discover the reason for the error: the destination terminal or auxiliary device was invalid.

Successful SEND function

If the SEND function is successful, BEGIN1 positions the cursor and moves the successful SEND message text to the output text. The source terminal operator then receives the message,

```
TRANSACTION BEGUN AT TERMINAL TRM5
```

at his terminal (lines 101-104) and BEGIN1 terminates normally.

When the TRM1 operator receives the successful SEND message, the program, PRINT, begins processing the ORDFILE order number 5732468 at TRM5 and sends continuous output from the PRINT program to a communications output printer attached to TRM5.

Initiating continuous output

Most output-for-input queueing applications initiate a continuous output transaction at another terminal, to free the source terminal for further interactive processing. The continouous output program initiated by the source terminal operator in the message entered on the BEGIN transaction was PRINT.

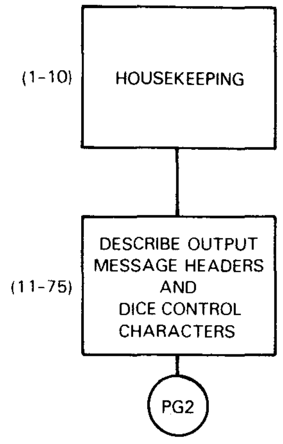
The PRINT action program showing how continuous output is handled follows in B.6.

B.6. SAMPLE COBOL ACTION PROGRAM PERFORMING CONTINUOUS OUTPUT WITH DELIVERY NOTICE SCHEDULING (PRINT)

PRINT description

Figure B-25 illustrates a compiler listing of a sample COBOL action program, PRINT with corresponding flowchart. The PRINT program:

- Prepares three types of output messages by processing customer order information entered at the terminal against an indexed file.
- Lists these messages as continuous output at the originating terminal. (If the parameter, COP, is included in the initial input message, the output from PRINT is sent to a communications output printer.)

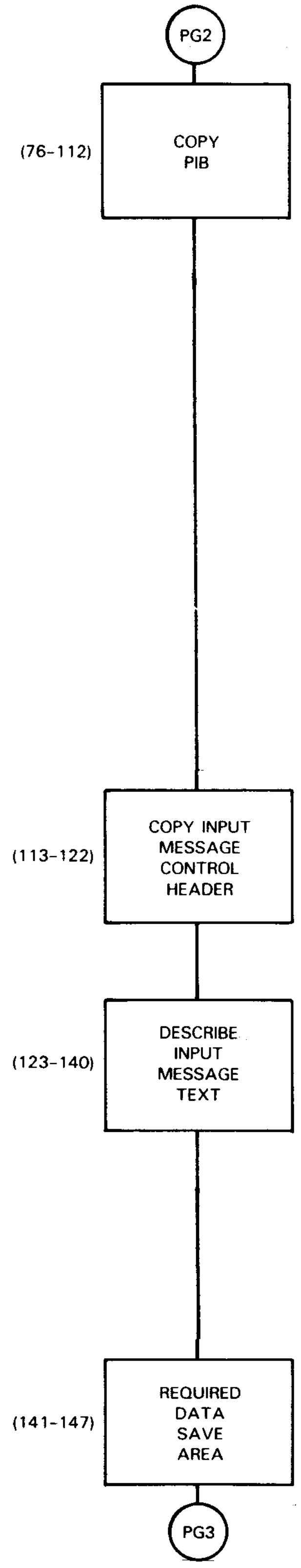


```
LINE NO.    SOURCE ENTRY

00001 IDENTIFICATION DIVISION.
00002 PROGRAM-ID. PRINT.
00003 ENVIRONMENT DIVISION.
00004 CONFIGURATION SECTION.
00005 SOURCE-COMPUTER. UNIVAC-OS3.
00006 OBJECT-COMPUTER. UNIVAC-OS3.
00007 DATA DIVISION.
00008 WORKING-STORAGE SECTION.
00009 77  POS-GE                       PIC X   VALUE 'G'.
00010 77  SUCCESSFUL-DEL-NOTICE        PIC X   VALUE ='C1'.
00011 01  TOTAL-POS.
00012     02 DICE-TP                   PIC X     VALUE ='10'.
00013     02 FUNC-TP                   PIC X      VALUE ='04'.
00014     02 Y-TP                      PIC X      VALUE ='00'.
00015     02 X-TP                      PIC X      VALUE ='33'.
00016 01  HEADER-LINES.
00017     02 ORDER-LINE.
00018        03 HOME-POS-CLEAR.
00019           05 DICE-HPC            PIC X       VALUE ='10'.
00020           05 FUNC-HPC            PIC X       VALUE ='03'.
00021           05 Y-HPC               PIC X       VALUE ='00'.
00022           05 X-HPC               PIC X       VALUE ='00'.
00023        03 MIDDLE-COL-POS.
00024           05 DICE-MCP            PIC X       VALUE ='10'.
00025           05 FUNC-MCP            PIC X       VALUE ='02'.
00026           05 Y-MCP               PIC X       VALUE ='00'.
00027           05 X-MCP               PIC X       VALUE ='37'.
00028        03 P-ORDER-HEAD           PIC X(10) VALUE 'ORDER #
00029        03 P-ORDER-NO             PIC 9(7).
00030        03 NEWLINE-3.
00031           05 DICE-N3             PIC X       VALUE ='10'.
00032           05 FUNC-N3             PIC X       VALUE ='04'.
00033           05 Y-N3                PIC X       VALUE ='02'.
00034           05 X-N3                PIC X       VALUE ='00'.
00035     02 MAIL-LINES.
00036        03 P-NAME                 PIC X(20).
00037        03 NEWLINE-A.
00038           05 DICE-N1A            PIC X       VALUE ='10'.
00039           05 FUNC-N1A            PIC X       VALUE ='04'.
00040           05 Y-N1A               PIC X       VALUE ='00'.
00041           05 X-N1A               PIC X       VALUE ='00'.
00042        03 P-ADDR                 PIC X(15).
00043        03 NEWLINE-B.
00044           05 DICE-N1B            PIC X       VALUE ='10'.
00045           05 FUNC-N1B            PIC X       VALUE ='04'.
00046           05 Y-N1B               PIC X       VALUE ='00'.
00047           05 X-N1B               PIC X       VALUE ='00'.
00048        03 P-CITY                 PIC X(15).
00049        03 P-ZIP                  PIC X(5).
00050        03 NEWLINE-2.
00051           05 DICE-N2             PIC X       VALUE ='10'.
00052           05 FUNC-N2             PIC X       VALUE ='04'.
00053           05 Y-N2                PIC X       VALUE ='01'.
00054           05 X-N2                PIC X       VALUE ='00'.
00055     02 HEADING-LINE.
00056        03 PRODUCT-HEADING        PIC X(19)
00057                                  VALUE '      PRODUCT
00058        03 UNIT-COST-HEADING      PIC X(11)
00059                                  VALUE 'UNIT-COST  '.
00060        03 AMOUNT-HEADING         PIC X(8)
00061                                  VALUE 'AMOUNT  '.
00062        03 SUBTOTAL-HEADING       PIC X(10)
00063                                  VALUE 'SUBTOTAL  '.
00064        03 SPACING                PIC X(3)  VALUE '   '.
00065        03 TOTAL-HEADING          PIC X(8)   VALUE ' TOTAL
00066        03 NEWLINE-C.
00067           05 DICE-N1C            PIC X        VALUE ='10'.
00068           05 FUNC-N1C            PIC X        VALUE ='04'.
00069           05 Y-N1C               PIC X        VALUE ='00'.
00070           05 X-N1C               PIC X        VALUE ='00'.
00071 01  ERROR-POSITION.
00072     03 DICE-EP                   PIC X        VALUE ='10'.
00073     03 FUNC-EP                   PIC X        VALUE ='01'.
00074     03 Y-EP                      PIC X        VALUE ='00'.
00075     03 X-EP                      PIC X        VALUE ='00'.
```

Compilation and flowchart

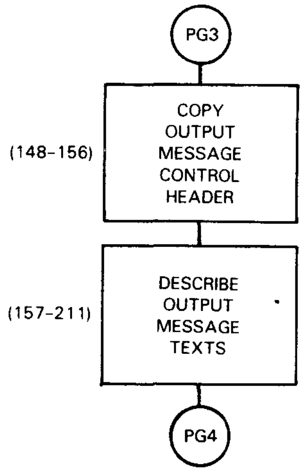
Figure B–25. Sample Action Program PRINT Performing Continuous Output (Part 1 of 6)

CONTINUOUS OUTPUT IN COBOL: PRINT PROGRAM



```
00076 LINKAGE SECTION.
00077 01   PROGRAM-INFORMATION-BLOCK. COPY PIB74.
00078          02  STATUS-CODE                  PIC 9(4) COMP-4.
00079          02  DETAILED-STATUS-CODE         PIC 9(4) COMP-4.
00080          02  RECORD-TYPE REDEFINES DETAILED-STATUS-CODE.
00081              03 PREDICTED-RECORD-TYPE   PIC X.
00082              03 DELIVERED-RECORD-TYPE   PIC X.
00083          02  SUCCESSOR-ID                 PIC X(6).
00084          02  TERMINATION-INDICATOR        PIC X.
00085          02  LOCK-ROLLBACK-INDICATOR      PIC X.
00086          02  TRANSACTION-ID.
00087              03 YEAR                      PIC 9(4) COMP-4.
00088              03 TODAY                     PIC 9(4) COMP-4.
00089              03 HR-MIN-SEC                PIC 9(9) COMP-4.
00090          02  DATA-DEF-REC-NAME            PIC X(7).
00091          02  DEFINED-FILE-NAME            PIC X(7).
00092          02  STANDARD-MSG-LINE-LENGTH     PIC 9(4) COMP-4.
00093          02  STANDARD-MSG-NUMBER-LINES    PIC 9(4) COMP-4.
00094          02  WORK-AREA-LENGTH             PIC 9(4) COMP-4.
00095          02  CONTINUITY-DATA-INPUT-LENGTH    PIC 9(4) COMP-4.
00096          02  CONTINUITY-DATA-OUTPUT-LENGTH    PIC 9(4) COMP-4.
00097          02  WORK-AREA-INC                PIC 9(4) COMP-4.
00098          02  CONTINUITY-DATA-AREA-INC     PIC 9(4) COMP-4.
00099              02  SUCCESS-UNIT-ID.
00100                  03 TRANSACTION-DATE.
00101                     04 YEAR                       PIC 99.
00102                     04 MONTH                      PIC 99.
00103                     04 TODAY                      PIC 99.
00104                  03 TIME-OF-DAY.
00105                     04 HOUR                       PIC 99.
00106                     04 MINUTE                     PIC 99.
00107                     04 SECOND                     PIC 99.
00108              03 UNIQUE-SUFFIX             PIC 999.
00109          02  SOURCE-TERMINAL-CHARS.
00110              03 SOURCE-TERMINAL-TYPE   PIC X.
00111              03 SOURCE-TERM-MSG-LINE-LENGTH    PIC 9(4) COMP-4.
00112              03 SOURCE-TERM-MSG-NUMBER-LINES   PIC 9(4) COMP-4.
00113 01   INPUT-MESSAGE-AREA. COPY IMA74.
00114          02 SOURCE-TERMINAL-ID                    PIC X(4).
00115          02 DATE-TIME-STAMP.
00116             03 YEAR                               PIC 9(4) COMP-4.
00117             03 TODAY                              PIC 9(4) COMP-4.
00118             03 HR-MIN-SEC                         PIC 9(9) COMP-4.
00119          02 TEXT-LENGTH                           PIC 9(4) COMP-4.
00120          02 AUXILIARY-DEV-ID.
00121             03 FILLER                           PIC X.
00122             03 AUX-DEV-NO                       PIC X.
00123          02 TRANS-TEXT.
00124             05 TRANS-CODE                 PIC X(5).
00125             05 FILLER                     PIC X.
00126             05 T-FILE-NAME                PIC X(7).
00127             05 T-ORDER-NO                 PIC 9(7).
00128             05 FILLER                     PIC X(1).
00129             05 INIT-TERMINAL              PIC X(4).
00130             05 FILLER                     PIC X(1).
00131             05 AT-COP                     PIC X(3).
00132          02 ACTION-TEXT REDEFINES TRANS-TEXT.
00133             05 COMMAND-CODE               PIC X(5).
00134             05 FILLER                     PIC X(2).
00135             05 A-ORDER-NO                 PIC 9(7).
00136             05 FILLER                     PIC X(15).
00137          02 DEL-NOTICE-TEXT REDEFINES TRANS-TEXT.
00138             05 DEL-NOTICE-CODE            PIC X(4).
00139             05 DEL-NOTICE-STATUS          PIC X.
00140             05 FILLER                     PIC X(24).
00141 01   WORK-AREA.
00142          03 RECORD-AREA.
00143             05 RECORD-KEY.
00144                07 K-ORDER-NO              PIC S9(7)  COMP-3.
00145                07 K-ORDER-ENTRY           PIC S999   COMP-3.
00146             05 FILLER                     PIC X(74).
00147          03 ERROR-IND                     PIC X.
```

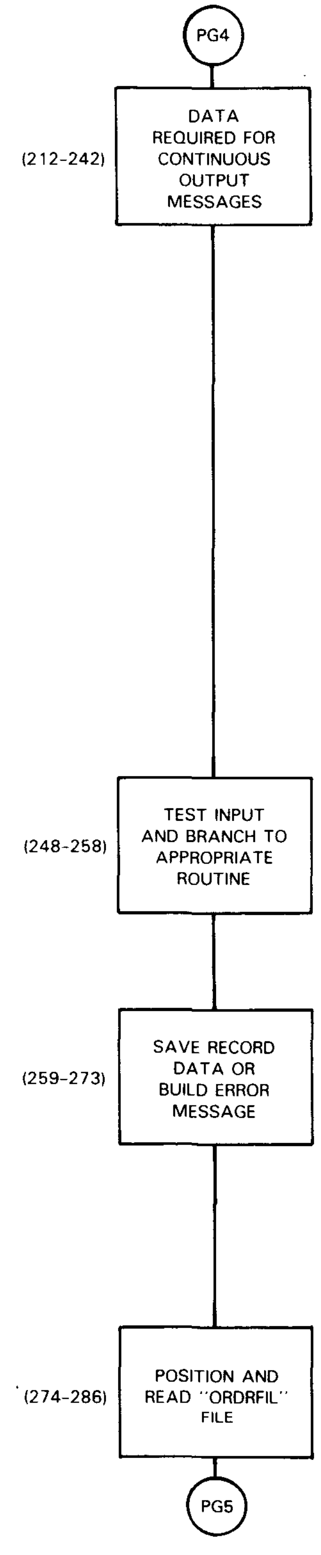
Figure B-25. Sample Action Program PRINT Performing Continuous Output (Part 2 of 6)


PG3
(148-156)
COPY OUTPUT MESSAGE CONTROL HEADER
(157-211)
DESCRIBE OUTPUT MESSAGE TEXTS
PG4

```
00148 01  OUTPUT-MESSAGE-AREA.      COPY OMA74.
00149     02 DESTINATION-TERMINAL-ID      PIC X(4).
00150              02 SFS-OPTIONS                 PIC X(2).
00151              02 FILLER                      PIC X(2).
00152     02 CONTINUOUS-OUTPUT-CODE       PIC X(4).
00153     02 TEXT-LENGTH                PIC 9(4)    COMP-4.
00154     02 AUXILIARY-DEVICE-ID.
00155        03 AUX-FUNCTION              PIC X.
00156        03 AUX-DEVICE-NO             PIC X.
00157     03 MESSAGE-1.
00158        05 FILLER                 PIC X(18).
00159        05 M-ORDER                PIC 9(7).
00160        05 FILLER                 PIC X(4).
00161        05 M-NAME                 PIC X(20).
00162        05 FILLER                 PIC X(4).
00163        05 M-ADDR                 PIC X(15).
00164        05 FILLER                 PIC X(4).
00165        05 M-CITY                 PIC X(15).
00166        05 M-ZIP                  PIC X(5).
00167        05 FILLER                 PIC X(114).
00168     03 MESSAGE-2 REDEFINES MESSAGE-1.
00169        05 P-BRAND                PIC X(17).
00170        05 COST                   PIC $$,$$$.99.
00171        05 FILLER                 PIC X(2).
00172        05 NUM                    PIC ZZZ.
00173        05 FILLER                 PIC X(2).
00174        05 P-SUBTOTAL             PIC $$,$$$,$$$.99.
00175        05 NEXTLINE-2             PIC X(4).
00176        05 FILLER                 PIC X(156).
00177     03 MESSAGE-3 REDEFINES MESSAGE-1.
00178        05 CURSOR-POS             PIC X(4).
00179        05 M-TOTAL                PIC $$,$$$,$$$.99.
00180        05 VALIDITY-CHAR          PIC X.
00181        05 FILLER                 PIC X(188).
00182     03 MESSAGE-4 REDEFINES MESSAGE-1.
00183        05 POSITION-4             PIC X(4).
00184        05 HEADER-4               PIC X(42).
00185        05 ORDER-4                PIC 9(7).
00186        05 FILLER                 PIC X(153).
00187     03 MESSAGE-5 REDEFINES MESSAGE-1.
00188        05 POSITION-5             PIC X(4).
00189        05 HEADER-5               PIC X(19).
00190        05 TERM-NAME              PIC X(4).
00191        05 FILLER                 PIC X(179).
00192     03 MESSAGE-6 REDEFINES MESSAGE-1.
00193        05 POSITION-6             PIC X(4).
00194        05 BREAK-OUTPUT           PIC X(53).
00195        05 FILLER                 PIC X(149).
00196     03 MESSAGE-7 REDEFINES MESSAGE-1.
00197        05 POSITION-7             PIC X(4).
00198        05 RESUME-ERROR-OUTPUT    PIC X(24).
00199        05 FILLER                   PIC X(178).
00200     03 MESSAGE-8 REDEFINES MESSAGE-1.
00201        05 POSITION-8             PIC X(4).
00202        05 END-OUTPUT             PIC X(23).
00203        05 FILLER                 PIC X(179).
00204     03 MESSAGE-9 REDEFINES MESSAGE-1.
00205        05 POSITION-9             PIC X(4).
00206        05 INPUT-ERROR-OUTPUT     PIC X(32).
00207        05 FILLER                 PIC X(170).
00208     03 MESSAGE-10 REDEFINES MESSAGE-1.
00209        05 POSITION-10            PIC X(4).
00210        05 FILE-ERROR-OUTPUT      PIC X(42).
00211        05 FILLER                 PIC X(160).
```

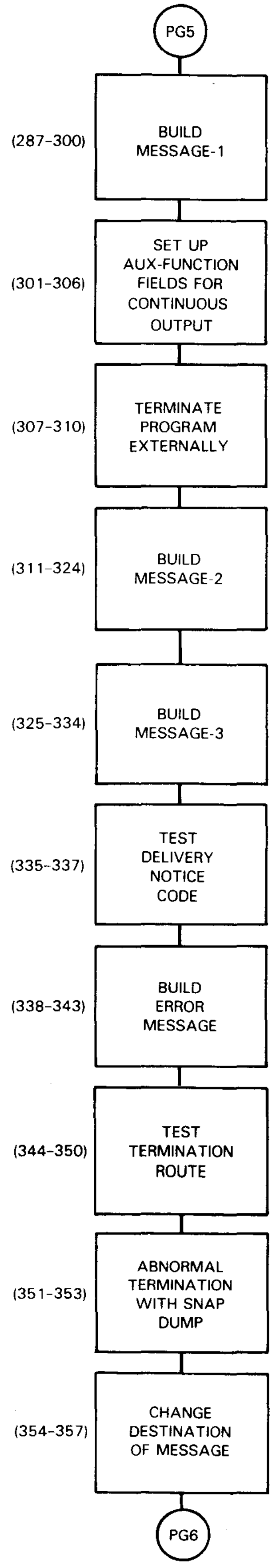
Figure B–25. Sample Action Program PRINT Performing Continuous Output (Part 3 of 6)

CONTINUOUS OUTPUT IN COBOL: PRINT PROGRAM


PG4
(212-242)
DATA REQUIRED FOR CONTINUOUS OUTPUT MESSAGES
(248-258)
TEST INPUT AND BRANCH TO APPROPRIATE ROUTINE
(259-273)
SAVE RECORD DATA OR BUILD ERROR MESSAGE
(274-286)
POSITION AND READ "ORDRFIL" FILE
PG5

```
00212 01  CONTINUITY-DATA-AREA.
00213     03 CUSTOMER-RECORD.
00214        05 C-KEY                   PIC X(6).
00215        05 C-ID                    PIC X(5).
00216        05 C-NAME                  PIC X(20).
00217        05 C-ADDR                  PIC X(15).
00218        05 C-CITY                  PIC X(15).
00219        05 C-ZIP                   PIC X(5).
00220        05 C-TOTAL                 PIC S9(7)V99    COMP-3.
00221        05 FILLER                   PIC X(14).
00222     03 PRODUCT-RECORD.
00223        05 P-KEY                   PIC X(6).
00224        05 PRODUCT                 PIC X(17).
00225        05 UNIT-COST               PIC S9(3)V99    COMP-3.
00226        05 AMOUNT                  PIC S999        COMP-3.
00227        05 SUBTOTAL                PIC S9(7)V99    COMP-3.
00228        05 FILLER                   PIC X(47).
00229     03 CURRENT-ORDER-NO           PIC S9(7)  COMP-3.
00230     03 CURRENT-CONT-CODE REDEFINES CURRENT-ORDER-NO  PIC X(4).
00231     03 CURRENT-ENTRY-NO           PIC S999   COMP-3.
00232     03 CURRENT-TOTAL              PIC S9(7)V99  COMP-3.
00233     03 INIT-TERM                  PIC X(4).
00234     03 DEST-TERM                   PIC X(4).
00235     03 COP-OUTPUT                 PIC X(3).
00236     03 FILE-KEY.
00237        05 FILE-KEY-1               PIC S9(7) COMP-3.
00238        05 FILE-KEY-2               PIC S9(3) COMP-3.
00239     03 FILE-NAME                   PIC X(7).
00240     03 INPUT-TOTAL                PIC S9(7)V99 COMP-3.
00241     03 BREAK-MODE                 PIC X.
00242     03 PRINT-DEST                 PIC X(4).
00243 PROCEDURE DIVISION USING PROGRAM-INFORMATION-BLOCK
00244                            INPUT-MESSAGE-AREA
00245                            WORK-AREA
00246                            OUTPUT-MESSAGE-AREA
00247                            CONTINUITY-DATA-AREA.
00248 EXAMINE-INPUT.
00249     IF TRANS-CODE EQUAL TO 'PRINT' GO TO BEGIN-TRANS.
00250     IF COMMAND-CODE EQUAL TO 'END '   GO TO END-TRANS.
00251     IF COMMAND-CODE EQUAL TO 'BREAK' GO TO BREAK-TRANS.
00252     IF COMMAND-CODE EQUAL TO 'RESUM' GO TO RESUME-TRANS.
00253     IF DEL-NOTICE-CODE EQUAL TO 'END '  GO TO END-OF-FILE.
00254     IF DEL-NOTICE-CODE EQUAL TO CURRENT-CONT-CODE
00255                                        GO TO DEL-NOTICE.
00256     MOVE 'INVALID DELIVERY NOTICE CODE     '
00257         TO INPUT-ERROR-OUTPUT.
00258     GO TO DEL-NOTICE-ERROR.
00259 BEGIN-TRANS.
00260     MOVE 0 TO CURRENT-ORDER-NO
00261     MOVE 0 TO CURRENT-ENTRY-NO
00262     MOVE 0 TO FILE-KEY-1  FILE-KEY-2.
00263     MOVE 0 TO CURRENT-TOTAL
00264     MOVE 0 TO BREAK-MODE
00265     MOVE SPACES TO INIT-TERM
00266     MOVE SPACES TO DEST-TERM
00267     MOVE AT-COP TO COP-OUTPUT.
00268     IF T-FILE-NAME NOT EQUAL TO 'ORDRFIL' GO TO INPUT-ERROR.
00269     MOVE INIT-TERMINAL TO INIT-TERM.
00270     MOVE SOURCE-TERMINAL-ID TO PRINT-DEST.
00271     IF T-ORDER-NO NOT EQUAL TO LOW-VALUES AND SPACES
00272         MOVE T-ORDER-NO TO FILE-KEY-1.
00273     MOVE T-FILE-NAME TO FILE-NAME.
00274 POSITION-FILE.
00275     CALL 'SETL' USING FILE-NAME POS-GE FILE-KEY.
00276     IF STATUS-CODE EQUAL TO 0 GO TO READ-RECORD.
00277     MOVE 'END ' TO CURRENT-CONT-CODE.
00278     GO TO TOTAL-PROC.
00279 READ-RECORD.
00280     CALL 'GET' USING FILE-NAME RECORD-AREA.
00281     IF STATUS-CODE EQUAL TO 1  GO TO FILE-ERROR.
00282     IF STATUS-CODE GREATER THAN 2 GO TO FILE-ERROR.
00283     IF STATUS-CODE EQUAL TO 2 GO TO END-OF-FILE.
00284     IF K-ORDER-ENTRY NOT EQUAL TO 0 GO TO PRODUCT-PROC.
00285     MOVE RECORD-AREA TO CUSTOMER-RECORD.
00286     IF CURRENT-TOTAL NOT EQUAL TO 0 GO TO TOTAL-PROC.
```

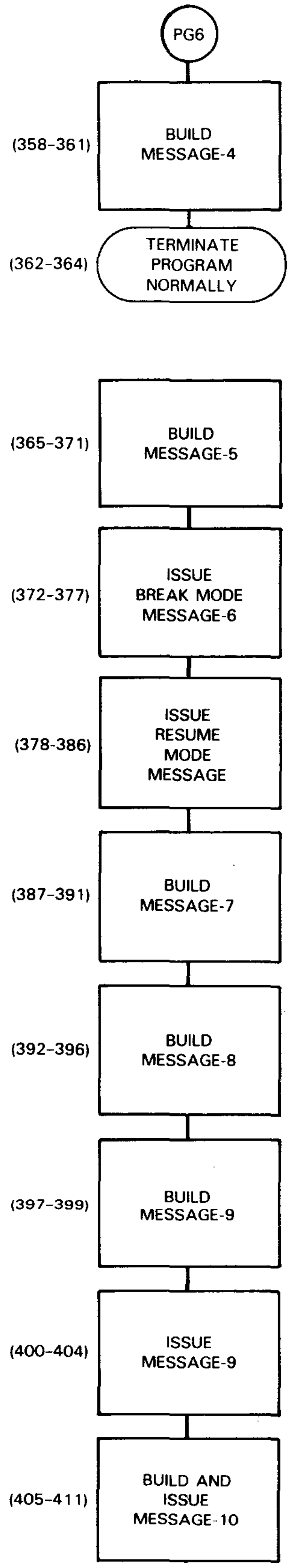
Figure B-25. Sample Action Program PRINT Performing Continuous Output (Part 4 of 6)



```
00287 CUSTOMER-PROC.
00288     MOVE C-TOTAL TO INPUT-TOTAL.
00289     MOVE K-ORDER-NO TO CURRENT-ORDER-NO.
00290     MOVE K-ORDER-NO TO FILE-KEY-1.
00291     MOVE K-ORDER-ENTRY TO CURRENT-ENTRY-NO.
00292     MOVE K-ORDER-ENTRY TO FILE-KEY-2.
00293     ADD 1 TO FILE-KEY-2.
00294     MOVE HEADER-LINES TO MESSAGE-1.
00295     MOVE CURRENT-ORDER-NO TO M-ORDER.
00296     MOVE C-NAME TO M-NAME.
00297     MOVE C-ADDR TO M-ADDR.
00298     MOVE C-CITY TO M-CITY.
00299     MOVE C-ZIP TO M-ZIP.
00300     MOVE 163 TO TEXT-LENGTH IN OUTPUT-MESSAGE-AREA.
00301 CREATE-CONTINUOUS-OUTPUT.
00302     IF COP-OUTPUT NOT EQUAL TO 'COP'
00303                           MOVE 'C' TO AUX-FUNCTION
00304               ELSE MOVE '7' TO AUX-FUNCTION
00305                    MOVE 1 TO AUX-DEVICE-NO.
00306     MOVE CURRENT-CONT-CODE TO CONTINUOUS-OUTPUT-CODE.
00307 EXTERNAL-TERMINATION.
00308     MOVE 'E' TO TERMINATION-INDICATOR.
00309     MOVE 'PRINTO' TO SUCCESSOR-ID.
00310     CALL 'RETURN'.
00311 PRODUCT-PROC.
00312     MOVE K-ORDER-ENTRY TO CURRENT-ENTRY-NO.
00313     MOVE K-ORDER-ENTRY TO FILE-KEY-2.
00314     ADD 1 TO FILE-KEY-2.
00315     MOVE RECORD-AREA TO PRODUCT-RECORD.
00316     ADD SUBTOTAL TO CURRENT-TOTAL.
00317     MOVE PRODUCT TO P-BRAND.
00318     MOVE UNIT-COST TO COST.
00319     MOVE AMOUNT TO NUM.
00320     MOVE SUBTOTAL TO P-SUBTOTAL.
00321     MOVE NEWLINE-A TO NEXTLINE-2.
00322     MOVE CURRENT-CONT-CODE TO CONTINUOUS-OUTPUT-CODE.
00323     MOVE 54 TO TEXT-LENGTH IN OUTPUT-MESSAGE-AREA.
00324     GO TO CREATE-CONTINUOUS-OUTPUT.
00325 TOTAL-PROC.
00326     IF INPUT-TOTAL EQUAL TO 0 MOVE CURRENT-TOTAL TO INPUT-TOTAL.
00327     MOVE SPACE TO VALIDITY-CHAR.
00328     MOVE INPUT-TOTAL TO M-TOTAL.
00329     IF INPUT-TOTAL NOT EQUAL TO CURRENT-TOTAL
00330                           MOVE '*' TO VALIDITY-CHAR.
00331     MOVE TOTAL-POS TO CURSOR-POS.
00332     MOVE 22 TO TEXT-LENGTH IN OUTPUT-MESSAGE-AREA.
00333     MOVE 0 TO CURRENT-TOTAL.
00334     GO TO CREATE-CONTINUOUS-OUTPUT.
00335 DEL-NOTICE.
00336     IF DEL-NOTICE-STATUS     EQUAL TO SUCCESSFUL-DEL-NOTICE
00337         GO TO POSITION-FILE.
00338 OUTPUT-ERROR.
00339     MOVE ERROR-POSITION TO POSITION-4.
00340     MOVE 'OUTPUT ERROR WHILE TRYING TO PRINT ORDER# ' TO
00341                                                HEADER-4.
00342     MOVE CURRENT-ORDER-NO TO ORDER-4.
00343     MOVE 57 TO TEXT-LENGTH IN OUTPUT-MESSAGE-AREA.
00344 DESTINATION-DETERMINATION.
00345     IF INIT-TERM NOT EQUAL TO LOW-VALUES AND SPACES AND
00346         SOURCE-TERMINAL-ID
00347         GO TO SWITCH-ERROR-MSG.
00348     MOVE 1 TO BREAK-MODE.
00349     IF ERROR-IND EQUAL TO LOW-VALUE GO TO EXTERNAL-TERMINATION.
00350     IF ERROR-IND EQUAL TO 1 GO  TO NORMAL-TERMINATION.
00351 ABNORMAL-TERMINATION.
00352     MOVE 'S' TO TERMINATION-INDICATOR.
00353     CALL 'RETURN'.
00354 SWITCH-ERROR-MSG.
00355     MOVE INIT-TERM TO DESTINATION-TERMINAL-ID.
00356     CALL 'SEND' USING OUTPUT-MESSAGE-AREA.
00357     IF STATUS-CODE NOT EQUAL TO 0 GO TO ABNORMAL-TERMINATION.
```

Figure B-25. Sample Action Program PRINT Performing Continuous Output (Part 5 of 6)

CONTINUOUS OUTPUT IN COBOL: PRINT PROGRAM


PG6
(358-361) BUILD MESSAGE-4
(362-364) TERMINATE PROGRAM NORMALLY
(365-371) BUILD MESSAGE-5
(372-377) ISSUE BREAK MODE MESSAGE-6
(378-386) ISSUE RESUME MODE MESSAGE
(387-391) BUILD MESSAGE-7
(392-396) BUILD MESSAGE-8
(397-399) BUILD MESSAGE-9
(400-404) ISSUE MESSAGE-9
(405-411) BUILD AND ISSUE MESSAGE-10

```
00358  CREATE-NULL-MSG.
00359      MOVE LOW-VALUES TO DESTINATION-TERMINAL-ID.
00360      MOVE NEWLINE-A TO POSITION-4.
00361      MOVE 8 TO TEXT-LENGTH IN OUTPUT-MESSAGE-AREA.
00362  NORMAL-TERMINATION.
00363      MOVE 'N' TO TERMINATION-INDICATOR.
00364      CALL 'RETURN'.
00365  END-OF-FILE.
00366      MOVE 1 TO ERROR-IND.
00367      MOVE ERROR-POSITION TO POSITION-5.
00368      MOVE 'PRINT COMPLETED AT ' TO HEADER-5.
00369      MOVE PRINT-DEST TO TERM-NAME.
00370      MOVE 31 TO TEXT-LENGTH IN OUTPUT-MESSAGE-AREA.
00371      GO TO DESTINATION-DETERMINATION.
00372  BREAK-TRANS.
00373      MOVE 1 TO BREAK-MODE.
00374      MOVE 'BREAK ENFORCED - RESUME REQUIRED TO CONTINUE PRINTING'
00375            TO BREAK-OUTPUT.
00376      MOVE 61 TO TEXT-LENGTH IN OUTPUT-MESSAGE-AREA.
00377      GO TO EXTERNAL-TERMINATION.
00378  RESUME-TRANS.
00379      IF BREAK-MODE EQUAL TO 0 GO TO RESUME-ERROR.
00380      MOVE 0 TO BREAK-MODE.
00381      IF A-ORDER-NO EQUAL TO LOW-VALUES OR SPACES
00382          GO TO POSITION-FILE.
00383      MOVE A-ORDER-NO TO FILE-KEY-1.
00384      MOVE 0 TO FILE-KEY-2.
00385      MOVE 0 TO CURRENT-TOTAL.
00386      GO TO POSITION-FILE.
00387  RESUME-ERROR.
00388      MOVE ERROR-POSITION TO POSITION-7.
00389      MOVE 'RESUME INVALID - IGNORED' TO RESUME-ERROR-OUTPUT.
00390      MOVE 32 TO TEXT-LENGTH IN OUTPUT-MESSAGE-AREA.
00391      GO TO NORMAL-TERMINATION.
00392  END-TRANS.
00393      MOVE ERROR-POSITION TO POSITION-8.
00394      MOVE 'PRINT TRANSACTION ENDED' TO END-OUTPUT.
00395      MOVE 31 TO TEXT-LENGTH IN OUTPUT-MESSAGE-AREA.
00396      GO TO NORMAL-TERMINATION.
00397  INPUT-ERROR.
00398      MOVE 'INPUT IN ERROR - PRINT NOT BEGUN'
00399                                   TO INPUT-ERROR-OUTPUT.
00400  DEL-NOTICE-ERROR.
00401      MOVE 2 TO ERROR-IND.
00402      MOVE ERROR-POSITION TO POSITION-9.
00403      MOVE 40 TO TEXT-LENGTH IN OUTPUT-MESSAGE-AREA.
00404      GO TO DESTINATION-DETERMINATION.
00405  FILE-ERROR.
00406      MOVE 2 TO ERROR-IND.
00407      MOVE ERROR-POSITION TO POSITION-10.
00408      MOVE 'FILE ACCESS ERROR - TRANSACTION TERMINATED'
00409                                   TO FILE-ERROR-OUTPUT.
00410      MOVE 50 TO TEXT-LENGTH IN OUTPUT-MESSAGE-AREA.
00411      GO TO DESTINATION-DETERMINATION.
```

Figure B-25. Sample Action Program PRINT Performing Continuous Output (Part 6 of 6)

Delivery notice scheduling

After delivery notice of each message is received from IMS, PRINT uses delivery notice scheduling to determine whether output should continue or error processing should occur. If output continues successfully, PRINT terminates in external succession, naming itself as successor to create the next output message to be printed. When end-of-file is reached, PRINT terminates normally, with an output message to the operator that printing is completed.

Unsuccessful delivery notice

If the PRINT program receives an unsuccessful delivery notice, it does not terminate immediately but first reports an output error to the terminal operator and allows him to control further output, terminating in external succession to await his response. He may respond by breaking off, resuming, or terminating the transaction normally.

PRINT input message

When it is first activated by action scheduling, PRINT expects to process an input message in the following form:

```
PRINT filename order-number init-terminal[COP]
```

where:

`PRINT`
: Is the transaction code that schedules the PRINT action program.

`filename`
: Is the name of the data file to be accessed. In this example, the file is an indexed file; PRINT expects to process a file named ORDRFIL and validates the filename keyed in (line 268, Figure B-25).

`order-number`
: Is an order number used as a key search argument in positioning the file for retrieval (lines 271 and 272).

`init-terminal`
: Is the terminal-id of the originating terminal, used in the switching of output error messages to the operator (line 355).

`COP`
: Is the 3-character code entered by the terminal operator to designate that output should be printed on the COP. Notice its use in line 302.

The input message received by the PRINT program in this example was sent from another terminal via the BEGIN1 action program as output-for-input queueing. The input message received by PRINT from TRM1 contains the transaction code that initiates the PRINT transaction at TRM5.

If the terminal operator at TRM1 entered the sample message shown in B.5, the message received by the PRINT action program is:

```
PRINT ORDFILE 5732468 TRM1 COP
```

Processing PRINT

On initial activation, PRINT passes control to the BEGIN-TRANS routine, which initializes certain fields of the continuity data area and work area and validates the name of the file to be processed (lines 259-268). BEGIN-TRANS positions the file for sequential processing and, retrieving a record (lines 269-275), processes it and the input message (lines 279-286). It forms a customer record, (lines 287-300), a product record, (lines 311-324) or a total record (lines 325-334), in the output message area; control then passes to the CREATE-CONTINUOUS-OUTPUT routine (lines 301-306).

Input message without COP

Here, if the terminal operator did not key in COP to direct the output message to a communications output printer, the routine moves the hexadecimal value C3 to the AUX-FUNCTION byte of the AUXILIARY-DEVICE-ID field in the OMA header (line 303). This causes the output message to be written as continuous output on the screen of the originating terminal. Otherwise, line 304 moves the hexadecimal value F7 to this byte, to cause print-transparent continuous output on a communications output printer, and line 305 moves a 1 to the AUX-DEVICE-NO byte of the AUXILIARY-DEVICE-ID to specify the COP relative number as defined in the ICAM generation.

Receiving CONTINUOUS-OUTPUT-CODE

Line 306 moves into the CONTINUOUS-OUTPUT-CODE field of the OMA header a 4-character value (represented by the current order number). After an attempt is made to deliver the message as specified, this 4-character value identifies this output message when received in the 5-byte input message that IMS creates for the next activation of PRINT.

After specifying external succession (line 308) and moving its own program name into the SUCCESSOR-ID field of the program information block (line 309), PRINT terminates to await reactivation by action scheduling.

Verifying DELIVERY-NOTICE-CODE and STATUS

On receiving the 5-byte input message from IMS, the PRINT program is reactivated. PRINT examines the input message, DEL-NOTICE-CODE, (first four bytes) to ensure that it is processing the expected input (line 348) and then proceeds to verify that the delivery attempt was successful. It does this at line 336 by comparing the fifth byte of the input message (DEL-NOTICE-STATUS) against the value 'A'. This value, which it has established for the constant SUCCESSFUL-DEL-NOTICE in a 77-level entry in the working-storage section (line 10), is the translated value for a successful delivery notice status (hexadecimal 81) reported to IMS by ICAM. On successful delivery, it resumes processing. If delivery was unsuccessful, PRINT does not attempt to determine the reason but sends an error message to the terminal operator. If an initiating terminal is specifed in the input message, PRINT sends error messages to that terminal.

Successful delivery

Unsuccessful delivery

RESUM/END commands

PRINT terminates in external succession after it sends an output message to the operator informing him of unsuccessful delivery of the last continuous output message (line 349). It expects him to enter either the command RESUM (line 252) or the command END (line 250) and is prepared to process one of these as its next reactivation. If he enters the command END (line 396), the program terminates with normal termination. If he enters the command RESUM, the program allows him to continue printing from where he left off, or from an earlier order number specified as an optional parameter of the RESUM command (line 135).

Abnormal PRINT termination

PRINT voluntarily terminates abnormally, with a SNAP dump, when:

- it receives an unexpected input message on activation (line 258);
- the terminal operator attempts to access some file other than ORDRFIL (line 268);
- an unsuccessful return was made to the STATUS-CODE field of the program information block after issuing the GET function to ORDRFIL (lines 280-283);
- any of its error or warning messages switched to the terminal operator were not successfully sent (line 357).

PRINT sends a message to the terminal operator before terminating when the operator enters the wrong file name (line 397) or there is an error on the GET function (line 405).

Appendix C. Basic Assembly Language (BAL) Action Programming Examples

C.1. DESCRIPTION

Appendix C contains compiler listings of three action programs. These examples illustrate complete action program coding for simple and dialog transactions including the use of delayed internal succession. In addition, an IMS configuration supplies the parameters needed to run these action programs.

ACT3 action program

The ACT3 action program processes a simple inquiry transaction to retrieve the capital city name of the state entered at a terminal. The program terminates normally by default.

SUPPLY action program

The SUPPLY action program, a more complex application, can terminate normally by default or abnormally by moving an 'S' to the TERMINATION-INDICATOR after determining that an S was entered as input. SUPPLY processes two successive simple transactions.

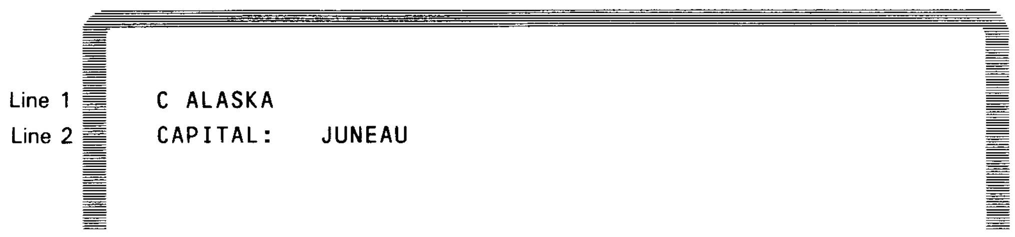
APCHKS action program

The APCHKS action program inserts or changes records entered at the terminal and uses delayed internal succession to call the APITMS action program. The APITMS action program uses delayed internal succession for error processing to return to the APCHKS action program for changes or corrections to records.

C.2. SAMPLE BAL ACTION PROGRAM PERFORMING A SIMPLE TRANSACTION (ACT3)

Processing a simple transaction

Action program, ACT3 (Figure C-2), processes a simple transaction. After receiving a transaction code of 'C' and the state name in its input message area (see Figure C-1, line 1), ACT3 issues the ZG#CALL GET macroinstruction to retrieve the capital name from the STATE file (Figure C-2, line 31).



```
C ALASKA
CAPITAL:   JUNEAU
```

Figure C-1. Terminal Entry and Output Message for ACT3 Simple Inquiry Transaction

Terminal entry used as record key

Here, ACT3 uses the state name entered at the terminal as a key to retrieve that state record from the STATE file.

Successful status code

If IMS returns a successful status code of 0, ACT3 then builds the output message (Figure C-2, lines 32 and 36-44) by setting the 4-byte DICE sequence (line 36) and moving the MSGCON1 constant (line 40) and state capital name (line 76) into the output message area (line 43). Finally, after terminating normally by default (line 58), ACT3 sends the message to the terminal. See Figure C-1, line 2.

Unsuccessful status code

If there is an I/O error (a status code other than 0 or 1 in this action program) after ACT3 issues the ZG#CALL GET macroinstruction, ACT3 moves MSGCON3 to the output area (line 55), and sends the message,'I/O ERROR', to the terminal on normal termination (line 63).

Invalid record key

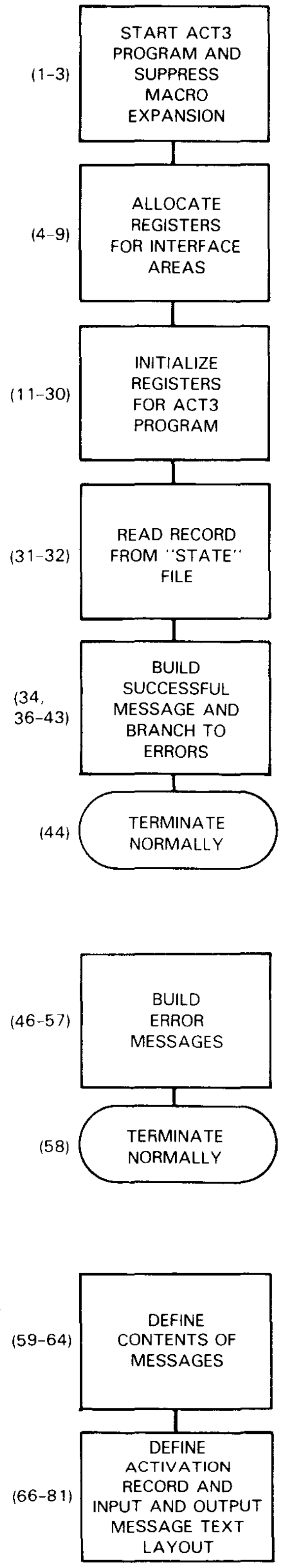
If IMS returns a status code of 1 (line 50), ACT3 moves MSGCON2 to the output message area and terminates normally, sending the error message 'INVALID STATE NAME' to the terminal (line 52).

Default for termination indicator

Notice that because N is the default value for the TERMINATION-INDICATOR field (ZA#PSIND) in the program information block, it is unnecessary to move the value 'N' to ZA#PSIND to terminate this transaction normally.

Default for destination terminal identification

Because a specific value is not moved to the DESTINATION-TERMINAL-ID field (ZA#ODTID) of the output message area, the output message is sent to the source terminal. Also, because ACT3 doesn't move a specific length to the text-length field (ZA#OTL) in the output message area, the text length of the output message is taken from the value configured on the OUTSIZE parameter for this action.


(1-3)
START ACT3 PROGRAM AND SUPPRESS MACRO EXPANSION
(4-9)
ALLOCATE REGISTERS FOR INTERFACE AREAS
(11-30)
INITIALIZE REGISTERS FOR ACT3 PROGRAM
(31-32)
READ RECORD FROM "STATE" FILE
(34, 36-43)
BUILD SUCCESSFUL MESSAGE AND BRANCH TO ERRORS
(44)
TERMINATE NORMALLY
(46-57)
BUILD ERROR MESSAGES
(58)
TERMINATE NORMALLY
(59-64)
DEFINE CONTENTS OF MESSAGES
(66-81)
DEFINE ACTIVATION RECORD AND INPUT AND OUTPUT MESSAGE TEXT LAYOUT

```
LINE   SOURCE STATEMENT                                                  OS/3 ASM

  1            PRINT NOGEN
  2 ACT3       START 0                      ACTION PROGRAM ENTRY POINT
  3 * ALLOCATE REGISTERS TO COVER:
  4            USING *,2                    THIS ACTION PROGRAM
  5            USING ZA#DPIB,3              THE PROGRAM INFORMATION BLOCK
  6            USING ZA#IMH,4               THE INPUT MESSAGE AREA
  7            USING WORK,5                 THE WORK AREA
  8            USING ZA#OMH,6               THE OUTPUT MESSAGE AREA
  9 * INITIALIZE
 10            STM    14,12,12(13)          SAVE ACTION SCHEDULING REGISTERS
 11 *                                       RD$ CONTAINS SAVE AREA ADDRESS WHEN
 12 *                                       CONTROL IS RECEIVED BY
 13 *                                       ACTION PROGRAM.
 14            LR     2,15                  ESTABLISH ADDRESSABILITY FOR ACTION
 15 *                                       PROGRAM. RF$ CONTAINS ENTRY POINT
 16 *                                       ADDRESS WHEN CONTROL IS RECEIVED
 17 *                                       BY ACTION PROGRAM.
 18            LM     3,6,0(1)              ESTABLISH ADDRESSABILITY FOR
 19 *                                       ACTIVATION RECORD. R1$ CONTAINS
 20 *                                       PARAMETER LIST ADDRESS WHEN CONTROL
 21 *                                       IS RECEIVED BY
 22 *                                       ACTION PROGRAM.
 23            LA     10,SAVEAREA               GET ADDRESS OF ACT3 SAVE AREA
 24            ST     10,8(,13)                 SET FORWARD POINTER FROM ACTION
 25 *                                           SCHEDULING SAVE AREA
 26            ST     13,4(,10)                 SET BACKWARD POINTER TO ACTION
 27 *                                           SCHEDULING SAVE AREA
 28            LR     13,10                       MAKE ACT3 SAVE AREA THE CURRENT
 29 *                                             SAVE AREA
 30 * GET STATE RECORD FROM FILE USING STATE NAME KEY IN INPUT MESSAGE
 31            ZG#CALL GET,(STATE,RECORD,SNKEY)     ISSUE CALL TO IMS
 32            CLI    ZA#PSC+1,0            TEST STATUS CODE RETURNED IN PROGRAM
 33 *                                       INFORMATION BLOCK BY IMS
 34            BNE    ERROR                 NON-ZERO MEANS ERROR
 35 * BUILD OUTPUT MESSAGE
 36            MVC    OUTTEXT(4),NEWLINE PUT DEVICE INDEPENDENT CONTROL
 37 *                                       CHARACTERS INTO MESSAGE TO CLEAR
 38 *                                       TO END OF LINE AND POSITION TO
 39 *                                       BEGINNING OF NEXT LINE
 40            MVC    OUTTEXT+4(L'MSGCON1),MSGCON1 PUT TEXT CONSTANT INTO
 41 *                                                  MESSAGE
 42 *          PUT CAPITAL NAME INTO MESSAGE
 43            MVC    OUTTEXT+4+L'MSGCON1(L'SCAPITAL),SCAPITAL
 44            B      TERM
 45 * PROCESS ERROR
 46 ERROR      MVC    OUTTEXT(4),NEWLINE                  CLEAR TO END OF LINE AND
 47 *                                                     POSITION TO BEGINNING
 48 *                                                     OF NEXT LINE
 49            CLI    ZA#PSC+1,1                           TEST STATUS CODE
 50            BNE    IOERROR
 51 *                                       ONE MEANS INVALID KEY
 52            MVC    OUTTEXT+4(L'MSGCON2),MSGCON2     PUT'INVALID STATE NAME'
 53 *                                                     INTO MESSAGE
 54            B      TERM
 55 IOERROR    MVC    OUTTEXT+4(L'MSGCON3),MSGCON3     PUT 'IO ERROR' INTO
 56 *                                                     MESSAGE
 57 * TERMINATE EXECUTION OF ACTION PROGRAM BY RETURN CALL TO IMS.
 58 TERM       ZG#CALL  RETURN
 59 * CONSTANTS
 60 STATE      DC     CL7'STATE'                ISAM FILE NAME
 61 MSGCON1    DC     C'CAPITAL'
 62 MSGCON2    DC     C'INVALID STATE NAME'
 63 MSGCON3    DC     C'I/O ERROR'
 64 NEWLINE    ZO#POSC 0,0             ICAM PROCEDURE TO GENERATE DICE SEQUENCE
 65 *                                  FOR NEW LINE CONTROL WITH CLEAR
 66 * ACTIVATION RECORD DEFINITION
 67            ZM#DPIB
 68            ZM#DIMH
 69 ICODE      DS     X             TRANSACTION CODE
 70            DS     X             SPACE
 71 SNKEY      DS     XL4          STATE NAME KEY
 72 WORK       DSECT               WORK AREA
 73 RECORD     EQU    *
 74 SNAME      DS     XL14         STATE NAME
 75 SPOP       DS     XL8          STATE POPULATION
 76 SCAPITAL   DS     XL25         STATE CAPITAL
 77 SAVEAREA   DS     18A          REGISTER SAVE AREA
 78 PLIST      DS     4A           PARAMETER LIST REFERENCE BY ZG#CALL MACRO
 79            ZM#DOMH             OMA CONTROL HEADER
 80 OUTTEXT    DS     XL42         OUTPUT MESSAGE TEXT AREA
 81            END
```

Figure C–2. Sample BAL Action Program ACT3 Processing a Simple Transaction

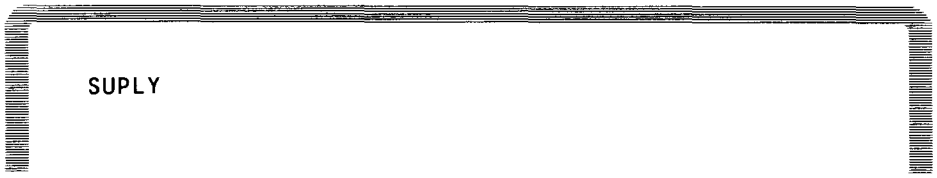
C.3. SAMPLE BAL ACTION PROGRAM PROCESSING SUCCESSIVE TRANSACTIONS (SUPPLY)

Simple transaction with screen format

The SUPPLY action program (Figure C-7) processes successive simple transactions that display a screen format for the terminal operator to enter supply charges, verify the data entered, create or change a record, and display results.

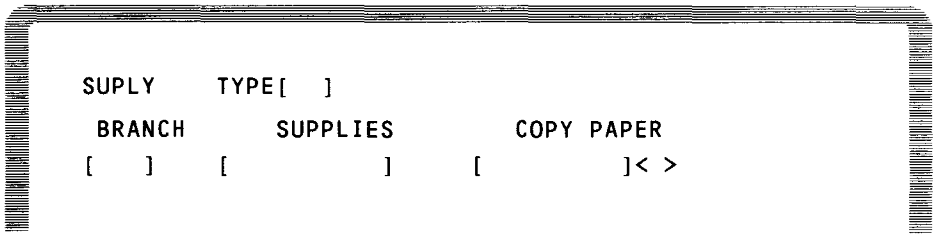
Processing SUPPLY action program

When the terminal operator enters the transaction code, SUPLY (Figure C-3), the SUPPLY action program returns the screen format (Figure C-4). The operator enters a TYPE code of I or G indicating the type of changes made, a branch number for the branch company being charged, and the amount (SUPPLIES) charged for supplies (Figure C-5).



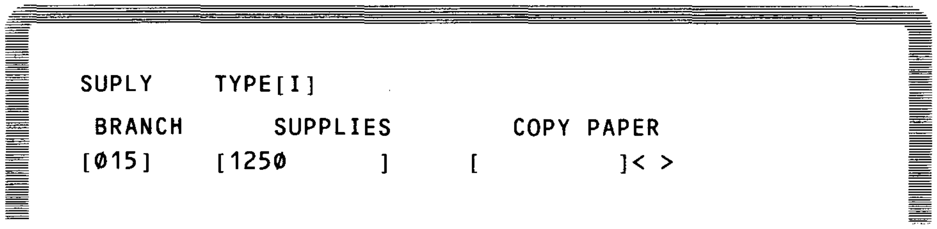
```
SUPLY
```

Figure C-3. Initiating the SUPLY Transaction



```
SUPLY    TYPE[  ]
 BRANCH       SUPPLIES        COPY PAPER
[    ]   [            ]     [          ]< >
```

Figure C-4. SUPPLY Action Program Screen Format Return

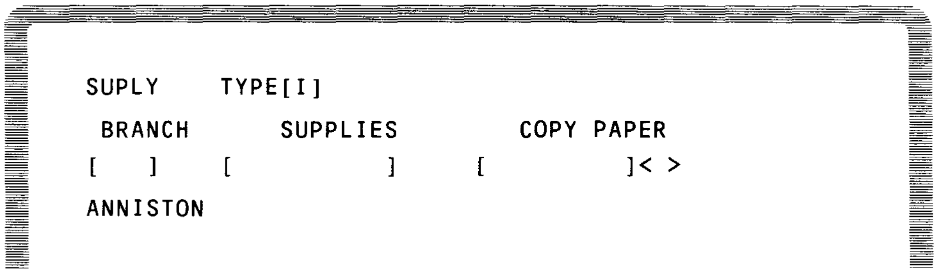


```
SUPLY    TYPE[I]
 BRANCH       SUPPLIES        COPY PAPER
[Ø15]    [125Ø        ]     [          ]< >
```

Figure C-5. Reinitiating the SUPLY Transaction with Input Data

Verifying data and creating a record

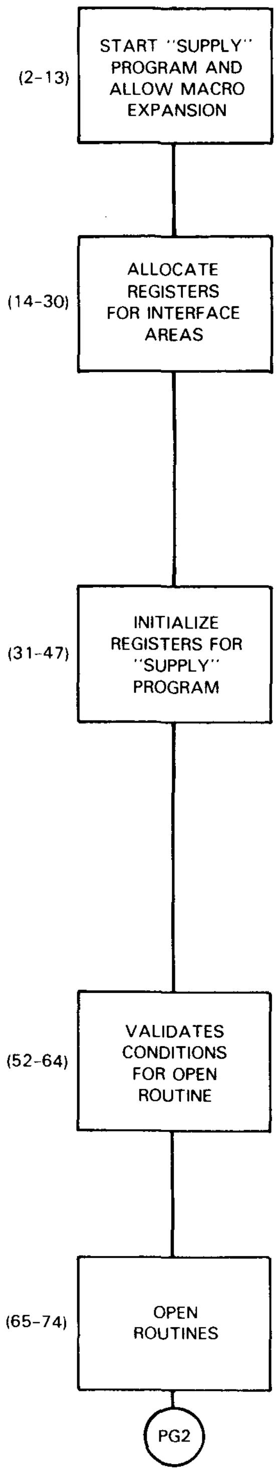
Next, he places the cursor and presses the **TRANSMIT** key. This reinitiates the SUPLY transaction, and the SUPPLY action program is scheduled again to verify the data and create the record. When the record is successfully changed or created, SUPPLY returns the name of the branch company and the type charges made to it (Figure C-6).



```
SUPLY    TYPE[I]
 BRANCH      SUPPLIES        COPY PAPER
[   ]   [           ]      [         ]< >
ANNISTON
```

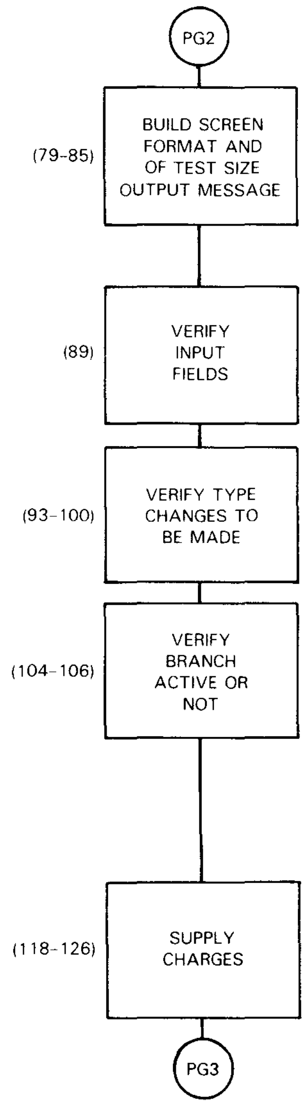
Figure C-6. Output From Second SUPLY Transaction

SUCCESSIVE TRANSACTIONS IN BAL: SUPPLY PROGRAM


(2-13)
START "SUPPLY" PROGRAM AND ALLOW MACRO EXPANSION
(14-30)
ALLOCATE REGISTERS FOR INTERFACE AREAS
(31-47)
INITIALIZE REGISTERS FOR "SUPPLY" PROGRAM
(52-64)
VALIDATES CONDITIONS FOR OPEN ROUTINE
(65-74)
OPEN ROUTINES
PG2

```
LINE   SOURCE STATEMENT                                              OS/3 ASM

  2 SUPPLY    START 0
  3 ***********************************************************************
  4 *                                                                     *
  5 *                  AUTHOR      : PAM BUIE                             *
  6 *                  DATE        : 05/05/76                             *
  7 *                  SITE        : GAY & TAYLOR, INC                    *
  8 *                  FUNCTION    : THIS PROGRAM WILL BE USED TO ENTER SUPPLY *
  9 *                                CHARGES AND VERIFY THE INFORMATION.  *
 10 *                  ***** C H A N G E   L O G *****                    *
 11 ***********************************************************************
 13           PRINT GEN
 14 *                  ESTABLISH REGISTER EQUATES
 15 R0        EQU    0
 16 R1        EQU    1
 17 R2        EQU    2                       PROGRAM COVER #1
 18 R3        EQU    3                       PROGRAM COVER #2
 19 R4        EQU    4
 20 R5        EQU    5                       INTERNAL SUBROUTINE LINKAGE
 21 R6        EQU    6                       I/O NORMAL RETURN
 22 R7        EQU    7                       I/O ERROR RETURN
 23 R8        EQU    8                       CDA COVER (OPTIONAL)
 24 R9        EQU    9                       PIB COVER
 25 R10       EQU    10                      IMA COVER
 26 R11       EQU    11                      WORK COVER
 27 R12       EQU    12                      OMA COVER
 28 R13       EQU    13
 29 R14       EQU    14
 30 R15       EQU    15
 31 *                  PROGRAM COVERING
 32           USING  *,R2,R3                 PROGRAM CODE
 33           USING  ZA#DPIB,R9              PIB
 34           USING  ZA#IMH,R10              IMA
 35           USING  WORK,R11                WORK
 36           USING  ZA#OMH,R12              OMA
 37 *                  IMS INTERFACE
 38           STM    R14,R12,12(R13)         STORE REG IN CALLS' SAVE AREA
 39           LR     R2,R15                  THIS PROGRAM ADDRESS
 40           LM     R9,R12,0(R1)             COVER IMS AREAS
 41           LR     R3,R2                   PROGRAM SECOND COVER
 42           LA     R3,1(R3)
 43           LA     R3,4095(,R3)            ADD 4096
 44           LA     R7,SAVE                 THIS PROGRAM SAVE AREA
 45           ST     R7,8(,R13)              PUT ADDR THIS SAVE IN CALLS' SAVE
 46           ST     R13,4(,R7)              PUT CALLS' SAVE ADDR IN THIS SAVE
 47           LR     R13,R7                  SET REG 13 FOR THIS SAVE AREA
 49 ***********************************************************************
 50 *                  CHECK SECURITY FOR OPEN                            *
 51 ***********************************************************************
 52           MVC    TABKEY(8),BLANK
 53           MVC    TABKEY(3),=C'T80'
 54           MVC    TABKEY+3(2),=C'SC'
 55           LA     R7,OPN020
 56           BAL    R6,TABGET
 57           CLI    TABSTS,C' '
 58           BNE    OPN020
 59           MVI    WORK1,X'00'
 60           MVC    WORK1+1(7),WORK1
 61           MVC    WORK1+8(2),ZA#ISTID+2
 62           PACK   WORK1+6(2),WORK1+8(2)
 63           CVB    R6,WORK1                TERMINAL # TO BINARY
 64           LA     R7,TERMTAB-4            INIT TERMINAL FIELDS ADDRESS
 65 OPN010    LA     R7,4(0,R7)              NEXT TERMINAL FIELDS
 66           BCT    R6,OPN010               NOT  TERMINAL FIELD-GET NEXT
 67           CLC    0(3,R7),=C'   '
 68           BE     OPN020
 69           MVC    WHO(3),0(R7)            SAVE USER INITIALS
 70           CLC    3(1,R7),LIMIT           OVER LIMIT?
 71           BNH    START000
 72 OPN020    MVC    MSGOUT(LMSG2),MSG2 "APPLICATION NOT OPEN"
 73           MVC    ZA#OTL(2),=Y(0+LMSG2+4)
 74           B      TERM
```

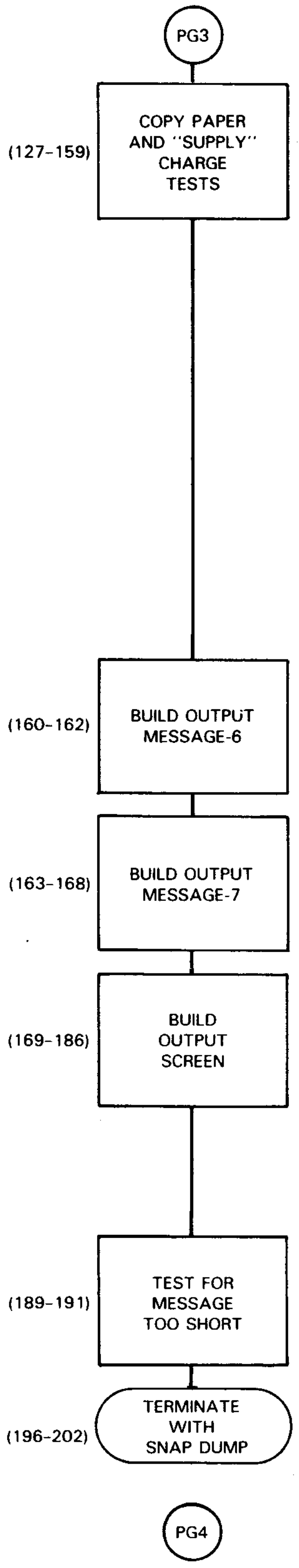
Figure C-7. Sample BAL Action Program SUPPLY Processing Successive Transactions (Part 1 of 9)


PG2
(79-85)
BUILD SCREEN FORMAT AND OF TEST SIZE OUTPUT MESSAGE
(89)
VERIFY INPUT FIELDS
(93-100)
VERIFY TYPE CHANGES TO BE MADE
(104-106)
VERIFY BRANCH ACTIVE OR NOT
(118-126)
SUPPLY CHARGES
PG3

```
 76 ***************************************************************************
 77 *                   DETERMINE ACTION
 78 ***************************************************************************
 79 STARTCOD EQU    *                          PROCESS MESSAGE
 80          BAL    R6,SCREENO                 MOVE SCREEN FORMAT TO OMA
 81          CLC    ZA#ITL(2),=Y(TRANLEN-MSGIN+4+4)
 82          BNH    TERM
 83          CLC    ZA#ITL(2),=Y(TLEN-MSGIN+4+4)
 84          BNL    VERIFYC0
 85          B      TOOSHORT
 86 *
 87 ************** VERIFY INPUT FIELDS
 88 *
 89 VERIFY00 EQU    *
 90 *
 91 ************** VERIFY TYPE
 92 *
 93          MVC    OTYPE,ITYPE
 94          CLI    ITYPE,C'G'
 95          BE     VERIFYC5
 96          CLI    ITYPE,C'I'
 97          BE     VERIFY05
 98          MVI    LBC11+2,X'1C'
 99          MVI    RBD12+1,X'1D'
100          MVI    ERRST,C'Y'
101 *
102 ************** VERIFY BRANCH
103 *
104 VERIFY05 EQU    *
105          MVC    OBRC(3),IBRC
106          MVC    BRKEY(3),IBRC
107          LA     R7,NOGOT
108          BAL    R6,BRCGET
109          MVC    OBRNAME,BRNAME
110          CLI    BRGL,C' '                  IS BRANCH ACTIVE?
111          BNE    NOGOT
112          B      VERIFY10
113 NOGOT    MVI    DICE041+2,X'1C'
114          MVI    DICE042+1,X'1D'
115          MVI    ERRST,C'Y'                 SHOW ERROR
116 ************** ARE SUPPLY CHARGES NUMERIC?
117 *
118 VERIFY10 EQU    *
119          MVC    OSUPPLY,ISUPPLY
120          LA     R1,ISUPPLY
121          LH     R0,=H'10'
122          BAL    R5,RJ00
123          BZ     VERIFY25
124          MVI    DICE043+2,X'1C'
125          MVI    DICE044+1,X'1D'
126          B      TERM
```

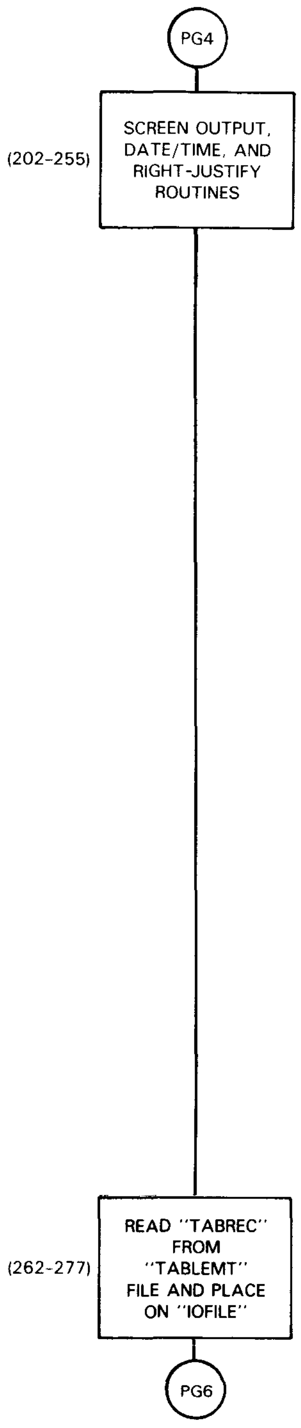
Figure C-7. Sample BAL Action Program SUPPLY Processing Successive Transactions (Part 2 of 9)

SUCCESSIVE TRANSACTIONS IN BAL: SUPPLY PROGRAM



```
127 *
128 *******,****** IS FILM NUMERIC?
129 *
130 VERIFY25 EQU   *
131          MVC   OCP,ICP
132          LA    R1,ICP
133          LH    R0,=H'10'
134          BAL   R5,RJOO
135          BZ    VERIFY30
136          MVI   DICE045+2,X'1C'
137          MVI   DICE046+1,X'1D'
138          B     TERM
139 *
140 *******,****** ARE SUPPLY CHARGES ZERO?
141 *
142 VERIFY30 EQU   *
143          MVI   TEST,X'00'
144          CLC   ISUPPLY+1(9),=C'000000000'
145          BNE   VERIFY35
146          OI    TEST,X'80'                 SET BIT FOR SUPPLY=0
147 *
148 *******,****** IS COPY PAPER ZERO?
149 *
150 VERIFY35 EQU   *
151          CLC   ICP+1(9),=C'000000000'
152          BNE   ENDVERD0
153          OI    TEST,X'40'
154 *
155 ************** TEST CONDITIONS FOR ERROR
156 *
157 ENDVERD0 EQU   *
158          CLI   TEST,X'C0'                 ARE BOTH ZERO?
159          BNE   NEXTESTO                   NO-SEE IF NEITHER ARE ZERO
160          MVC   MSGOUT(LMSG6),MSG6         BOTH ZERO
161          MVC   ZA#OTL(2),=Y(0+LMSG6+4)
162          B     TERM
163 NEXTESTO EQU   *
164          CLI   TEST,X'00'
165          BNE   INSERTSC
166          MVC   MSGOUT(LMSG7),MSG7
167          MVC   ZA#OTL(2),=Y(0+LMSG7+4)
168          B     TERM
169 INSERTSC EQU   *
170          CLI   ERRST,C'Y'
171          BE    TERM
172          MVC   WTYPE,ITYPE
173          MVI   WSTATUS,C' '
174          MVI   WRECID,C'B'
175          MVC   WBRC,IBRC
176          PACK  WSUPPLY(5),ISUPPLY+1(9)
177          PACK  WCP(5),ICP+1(9)
178          BAL   R6,DAYTIME
179          MVC   WTRTIME,HHMM
180          MVC   WTRDATE,YYMMDD
181          LA    R7,ABTERM
182          BAL   R6,DTFINSRT
183          MVC   OBRC(3),BLANK
184          MVC   OSUPPLY(10),BLANK
185          MVC   OCP(10),BLANK
186          B     TERM
187 *********************************
188 *              MESSAGE TOO SHORT
189 TOOSHORT MVC   MSGOUT(LMSG3),MSG3        CURSOR REPOSITIONED
190          MVC   ZA#OTL(2),=Y(0+LMSG3+4)
191          B     TERM
192 *
193 *
194 *******,****** TERMINATE PROGRAM
195 *
196 TERM     CLI   ISNAP,C'S'           ASK FOR SNAP?
197          BNE   GOBACK               NO
198 ABTERM   MVI   ZA#PSIND,C'S'        TERMINATE WITH SNAP
199 GOBACK   ZG#CALL RETURN
200+GOBACK   DS    0H
201+         L     15,=V(RETURN)
202+         BALR  14,15
```

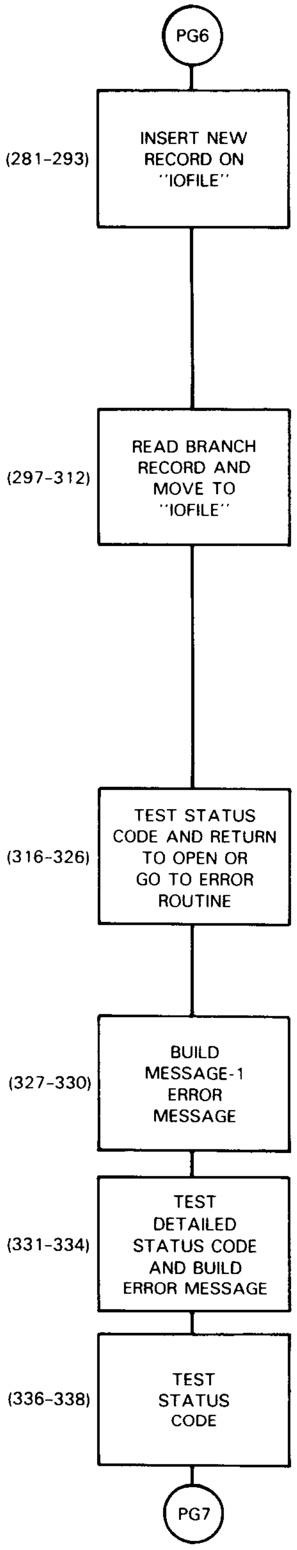
Figure C-7. Sample BAL Action Program SUPPLY Processing Successive Transactions (Part 3 of 9)


PG4
(202-255)
SCREEN OUTPUT, DATE/TIME, AND RIGHT-JUSTIFY ROUTINES
(262-277)
READ "TABREC" FROM "TABLEMT" FILE AND PLACE ON "IOFILE"
PG6

```
204 ***********************************************************************
205 *                                                                     *
206 *                  INTERNAL ROUTINES                                  *
207 *                                                                     *
208 ***********************************************************************
209 *
210 ************** MOVE SCREEN TO OMA
211 *
212 SCREENO  EQU    *
213          ST     R6,R6SAVE1
214          MVC    MSGOUT(LHEADER),SCREEN
215          MVC    OLINE1(LLINE1),LINE1
216          MVC    OLINE2(LLINE2),LINE2
217          MVC    OLINE3(LLINE3),LINE3
218          MVC    OLINE4(LLINE4),LINE4
219          MVC    OLINE5(LLINE5),LINE5
220          MVC    OLINE6(LLINE6),LINE6
221          MVC    ZA#OTL(4),=Y(ENDSCR-SCREEN+4)
222          L      R6,R6SAVE1
223          BR     R6                    RETURN
224 *
225 ************** DATE/TIME STAMP ****************************************
226 *
227 DAYTIME  EQU    *
228          ST     R6,R6SAVE2
229          GETIME S
230+         DS     0H
231+         SR     1,1
232+         SVC    7
233          ST     R0,WORK1              DATE-0YYMMDD+
234          UNPK   WORK1+4(7),WORK1(4)
235          MVC    YYMMDD(6),WORK1+5
236          OI     YYMMDD+5,X'F0'
237          ST     R1,WORK1              TIME-CHHMMSS+
238          UNPK   WORK1+4(7),WORK1(4)
239          MVC    HHMM(4),WORK1+5
240          OI     HHMM+3,X'F0'            FIX ZONE
241          L      R6,R6SAVE2
242          BR     R6
243 ************** RIGHT JUSTIFY ******************************************
244 *
245 *                REGISTER 0  = FIELD LENGTH ( LA R0,=H'XXXX' )
246 *                REGISTER 1  = ADDRESS OF FIELD
247 *                REGISTER 15 = RETURN STATUS CODE ( ZEROES IF NO ERROR )
248 RJ00     ORG    *
249          LA     R13,SAVE        LOAD ADDRESS OF PROGRAM SAVE AREA
250          DC     0Y(0)
251          EXTRN  MODRJ1
252          L      R15,=A(MODRJ1)
253          BALR R14,R15           BRANCH TO RJ
254          LTR    R15,R15         SET CONDITION CODE FOR ERRORS
255          BR     R5
256 *
257 ************** FILE I/O ***********************************************
258 *
259 *
260 *                TABLE MASTER I/O
261 *
262 TABGET   MVC    IOKEY(8),TABKEY
263          MVI    IOKEY+8,C'G'
264          ZG#CALL GET,(TABLEMT,TABREC,TABKEY)
265+         DS     0H
266+         LA     15,TABLEMT
267+         ST     15,PLIST+4*(1-1)
268+         LA     15,TABREC
269+         ST     15,PLIST+4*(2-1)
270+         LA     15,TABKEY
271+         ST     15,PLIST+4*(3-1)
272+         OI     PLIST+4*(3-1),X'80'
273+         LA     1,PLIST
274+         L      15,=V(GET)
275+         BALR   14,15
276          MVC    IOFILE(7),TABLEMT
277          B      IOSTATUS
```

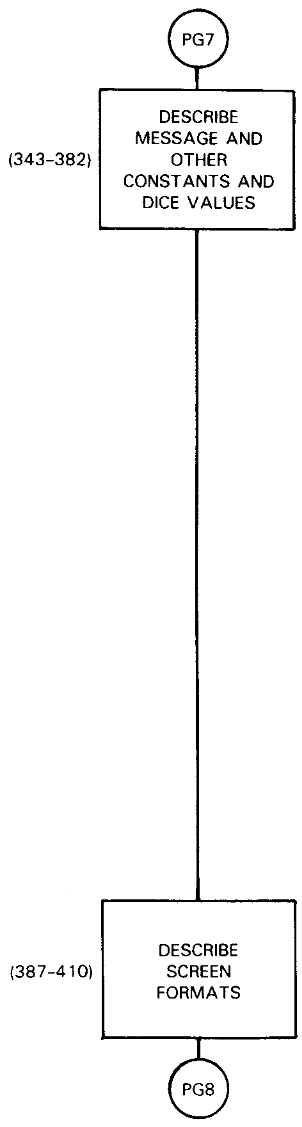
Figure C-7. Sample BAL Action Program SUPPLY Processing Successive Transactions (Part 4 of 9)

SUCCESSIVE TRANSACTIONS IN BAL: SUPPLY PROGRAM


PG6
(281-293)
INSERT NEW RECORD ON "IOFILE"
(297-312)
READ BRANCH RECORD AND MOVE TO "IOFILE"
(316-326)
TEST STATUS CODE AND RETURN TO OPEN OR GO TO ERROR ROUTINE
(327-330)
BUILD MESSAGE-1 ERROR MESSAGE
(331-334)
TEST DETAILED STATUS CODE AND BUILD ERROR MESSAGE
(336-338)
TEST STATUS CODE
PG7

```
278 *
279 ************** DAILY TRANSACTION FILE
280 *
281 DTFINSRT MVI    IOKEY+15,C'I'
282          ZG#CALL INSERT,(TRANACT,DTFREC)
283+         DS     0H
284+         LA     15,TRANACT
285+         ST     15,PLIST+4*(1-1)
286+         LA     15,DTFREC
287+         ST     15,PLIST+4*(2-1)
288+         OI     PLIST+4*(2-1),X'80'
289+         LA     1,PLIST
290+         L      15,=V(INSERT)
291+         BALR   14,15
292          MVC    IOFILE(7),TRANACT
293          B      IOSTATUS
294 *
295 ************** BRANCH MASTER
296 *
297 BRCGET   MVC    IOKEY(3),BRKEY
298          MVI    IOKEY+3,C'C'
299 GETBRC   ZG#CALL GET,(BRANCHM,BRREC,BRKEY)
300+GETBRC   DS     0H
301+         LA     15,BRANCHM
302+         ST     15,PLIST+4*(1-1)
303+         LA     15,BRREC
304+         ST     15,PLIST+4*(2-1)
305+         LA     15,BRKEY
306+         ST     15,PLIST+4*(3-1)
307+         OI     PLIST+4*(3-1),X'80'
308+         LA     1,PLIST
309+         L      15,=V(GET)
310+         BALR   14,15
311          MVC    IOFILE(7),BRANCHM
312          B      IOSTATUS
313 *
314 ************** I/O STATUS
315 *
316 IOSTATUS EQU    *
317          CLI    ZA#PSC+1,0          SUCCESSFUL?
318          BER    R6                  YES
319          CLI    ZA#PSC+1,1          NO-INVALID KEY?
320          BER    R7                  YES
321          CLI    ZA#PDSC+1,5         FILE NOT DEFINED?
322          BNE    IO1                 NO
323 IO0      CLI    IOERROR,C'R'        RETURN?
324          BNE    IOA                 NO
325          SR     R10,R10             CLEAR RC
326          BR     R7
327 IOA      MVC    MSGOUT(LMSG1),MSG1
328          MVC    MSGOUT+DM1FILE(7),IOFILE
329          MVC    ZA#OTL(2),=Y(0+LMSG1+4)
330          B      TERM
331 IO1      CLI    ZA#PDSC+1,6         FILE CLOSED?
332          BNE    IO2
333          MVC    MSGOUT+DM1HOW(10),M1AVAL
334          B      IO0
335 STSTRN   DC     C'0123456789ABCDEFX'     STATUS CODE TRANSLATE TABLE
336 IO2      MVC    IOSTS(4),ZA#PSC
337          TR     IOSTS,STSTRN
338          B      ABTERM
340 ************************************************************************
341 *               PROGRAM CONSTANTS                                      *
342 ************************************************************************
```

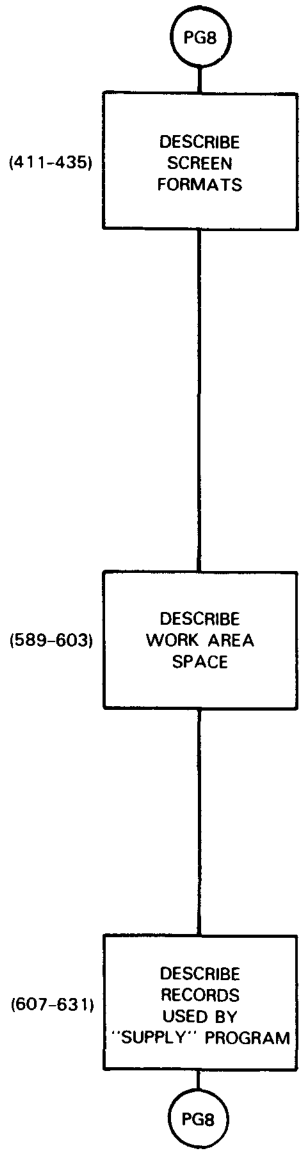
Figure C-7. Sample BAL Action Program SUPPLY Processing Successive Transactions (Part 5 of 9)



```
343 TABLEMT   DC     C'TABLEMT'
344 TRANACT   DC     C'TRANACT'
345 BRANCHM   DC     C'BRANCHM'
346 BLANK     DC     CL64' '
347 MSG1      DC     X'100218C11C40'
348 DM1FILE   EQU    *-MSG1
349           DC     CL7' '
350           DC     C'-FILE NOT '
351 DM1HOW    EQU    *-MSG1
352           DC     C'DEFINED    '
353           DC     C'TO IMS '
354           DC     X'1D'
355 LMSG1     EQU    *-MSG1
356 M1AVAL    DC     C'AVAILABLE '
357 MSG2      DC     X'100218011C40'
358           DC     C'APPLICATION NOT OPEN'
359           DC     X'1D'
360 LMSG2     EQU    *-MSG2
361 MSG3      DC     X'100218C11C'
362           DC     C'CURSOR REPOSITIONED-RETRANSMIT'
363           DC     X'1D'
364 DM3DICE   EQU    *-MSG3
365           DC     X'1002042C'
366 LMSG3     EQU    *-MSG3
367 MSG4      DC     C'      MESSAGE FROM TERMINAL-'
368 M4TERM    DC     CL5' '
369           DC     CL28'*'
370 MSG5      DC     CL60' '
371 MSG6      DC     X'100218011C'
372           DC     C'NO DATA ENTERED - REENTER
373           DC     X'1D'
374 DM6DICE   EQU    *-MSG6
375           DC     X'1002040D'
376 LMSG6     EQU    *-MSG6
377 MSG7      DC     X'100218C11C'
378           DC     C'ONLY SUPPLIES OR COPY PAPER VALID - REENTER'
379           DC     X'1D'
380 DM7DICE   EQU    *-MSG7
381           DC     X'1002040D'
382 LMSG7     EQU    *-MSG7
384 *
385 ************** SCREEN FORMAT ********************************************
386 *
387 SCREEN    EQU    *
388 DDICED    DC     X'10020001'
389           DC     X'2704'
390           DC     XL18'00'
391 LHEADER   EQU    *-SCREEN
392 LINE1     ORG    * 1111111111111111111111111111111111111111111111111111
393           DC     C'SUPLY'              TRANSACTION CODE
394           DC     CL4' '
395           DC     X'0E'                 START PROTECT FIELD
396           DC     C'TYPE'
397 DLB011    DC     X'27054ACF'           TAB-LB-END PROTECT
398 DOTYPE    DC     CL1' '
399 DRB012    DC     X'0E4F'                ST PROTECT-RB
400 LLINE1    EQU    *-LINE1
401 LINE2     ORG    * 2222222222222222222222222222222222222222222222222222
402           DC     CL143' '
403 LLINE2    EQU    *-LINE2
404 LINE3     ORG    * 3333333333333333333333333333333333333333333333333333
405           DC     C' BRANCH'
406           DC     CL6' '
407           DC     C'SUPPLIES'
408           DC     CL9' '
409           DC     C'COPY PAPER'
410           DC     CL40' '
```

Figure C-7. Sample BAL Action Program SUPPLY Processing Successive Transactions (Part 6 of 9)

SUCCESSIVE TRANSACTIONS IN BAL: SUPPLY PROGRAM



```
411 LLINE3   EQU   *-LINE3
412 LINE4    ORG   * 4444444444444444444444444444444444444444444444444444444
413 DDICE041 DC    X'27054A0F'              TAB-LB-END-PROTECT
414 DOBRC    DC    CL3' '
415 DDICE042 DC    X'0E4F'                  RB-START PROTECT
416          DC    CL4' '
417 DDICE043 DC    X'27054A0F'
418 DOSUPPLY DC    CL10' '
419 DDICE044 DC    X'0E4F'
420          DC    CL5' '
421 DDICE045 DC    X'27054A0F'
422 DOCP     DC    CL10' '
423 DDICE046 DC    X'0E4F'
424          DC    X'27054C0F406E'          < >
425          DC    X'10040000'
426 LLINE4   EQU   *-LINE4
427 LINE5    ORG   * 5555555555555555555555555555555555555555555555555555555
428 DDICE51  DC    X'10040000'
429 LLINE5   EQU   *-LINE5
430 LINE6    ORG   * 6666666666666666666666666666666666666666666666666666666
431          DC    CL1' '
432 DOBRNAME DC    CL25' '                  BRANCH NAME
433 DICE241  DC    X'10020110'
434 LLINE6   EQU   *-LINE6
586 *****************************************************************
587 *              WORK
588 *****************************************************************
589 WORK     DSECT
590 WORK1    DS    2D
591 YYMMDD   DS    CL6                    YEAR,MONTH,DAY
592 HHMM     DS    CL4                    HOURS,MINUTES,
593 R6SAVE1  DS    F                      SCREENO
594 R6SAVE2  DS    F                      DAYTIME
595 PLIST    DS    4A                     PARAMETER LIST FOR "CALLS"
596 SAVL     DS    18F                    IMS SAVE AREA
597 IOFILE   DS    CL8                    I/O FILE
598 IOKEY    DS    CL15                   I/O KEY
599 IOSTS    DS    CL4                    I/O STATUS
600 IOERROR  DS    CL1                    I/O ERROR RETURN ON NOT DEFINED/AVAIL
601 WHO      DS    CL3                    USER INITIALS
602 TEST     DS    CL1
603 ERRST    DS    CL1
604 *
605 ************** TABLE REC
606 *
607 TABKEY   DS    CL8
608 TABREC   DS    CL80
609 TABSTS   EQU   TABREC+8,1
610 LIMIT    EQU   TABREC+15,1
611 TERMTAB  EQU   TABREC+16              START OF TERMINAL FIELDS
612 *
613 ************** SUPPLY CHARGE RECORD
614 *
615 DTFKEY   DS    CL15
616 DTFREC   DS    CL165
617 WRECID   EQU   DTFREC,1               RECORD ID
618 WBRC     EQU   DTFREC+1,3             BRANCH
619 WTRDATE  EQU   DTFREC+4,6             TRANSACTION DATE
620 WTRTIME  EQU   DTFREC+10,4            TRANSACTION TIME
621 WTYPE    EQU   DTFREC+14,1            TYPE
622 WSTATUS  EQU   DTFREC+15,1            STATUS
623 WSUPPLY  EQU   DTFREC+16,5            SUPPLY CHARGES-AMOUNT-PD
624 WCP      EQU   DTFREC+21,5            COPY PAPER-PD2
625 *
626 ************** BRANCH RECORD
627 *
628 BRKEY    DS    CL3
629 BRREC    DS    CL250
630 BRNAME   EQU   BRREC+13,25            BRANCH NAME
631 BRGL     EQU   BRREC+195,1            GENERAL LEDGER STATUS.
633 *****************************************************************
634 *              OMA                                              *
635 *****************************************************************
636          ZM#DOMH
```

Figure C-7. Sample BAL Action Program SUPPLY Processing Successive Transactions (Part 7 of 9)

PG8

(700-721) DESCRIBE RECORDS USED BY "SUPPLY" PROGRAM

```
700 MSGOUT   DS    CL2048
701 DICEO    EQU   MSGOUT+DDICEO-SCREEN,4
702 OLINE1   EQU   MSGOUT+LINE1-SCREEN 11111111111111111111111111111111111
703 LBO11    EQU   OLINE1+DLBC11-LINE1
704 OTYPE    EQU   OLINE1+DOTYPE-LINE1,1
705 RBO12    EQU   OLINE1+DRBO12-LINE1
706 OLINE2   EQU   MSGOUT+LINE2-SCREEN 22222222222222222222222222222222222
707 OLINE3   EQU   MSGOUT+LINE3-SCREEN 33333333333333333333333333333333333
708 OLINE4   EQU   MSGOUT+LINE4-SCREEN 44444444444444444444444444444444444
709 DICEO41  EQU   OLINE4+DDICEO41-LINE4
710 OBRC     EQU   OLINE4+DOBRC-LINE4,3
711 DICEO42  EQU   OLINE4+DDICEO42-LINE4
712 DICEO43  EQU   OLINE4+DDICEO43-LINE4
713 OSUPPLY  EQU   OLINE4+DOSUPPLY-LINE4,10
714 DICEO44  EQU   OLINE4+DDICEO44-LINE4
715 DICEO45  EQU   OLINE4+DDICEO45-LINE4
716 OCP      EQU   OLINE4+DOCP-LINE4,10
717 DICEO46  EQU   OLINE4+DDICEO46-LINE4
718 OLINE5   EQU   MSGOUT+LINE5-SCREEN 55555555555555555555555555555555555
719 OLINE6   EQU   MSGOUT+LINE6-SCREEN 66666666666666666666666666666666666
720 OBRNAME  EQU   OLINE6+DOBRNAME-LINE6,25
721          END
```

```
UNIVAC SYSTEM OS/3  LINKAGE EDITOR                                                              VERC00000
DATE- 81/04/16 TIME- 13.03

CONTROL STREAM ENCOUNTERED AND PROCESSED AS FOLLOWS-

             /$
                       LOADM SUPPLY
                       LINKOP ALIB=$Y$OBJ
                       LINKOP OUT=IMSLOD
                       LINKOP CMT='GAY.AND.TAYLOR'          DESCRIPTION
SUPPLY   *RUN LIBE MODULE*
GET      *AUTO-INCLUDED*
MODRJI   *AUTO-INCLUDED*


                                         *DEFINITIONS DICTIONARY*

SYMBOL.   TYPE.  PHASE. ADDRESS.         SYMBOL.   TYPE.  PHASE. ADDRESS.         SYMBOL.   TYPE.  PHASE. ADDRESS.

ADDKY     ENTRY  ROOT   00000018         ARETURN   ENTRY  ROOT   0000005C         BUILD     ENTRY  ROOT   00000000
CLOSE     ENTRY  ROOT   00000090         CMDRB     ENTRY  ROOT   00000054         DELETE    ENTRY  ROOT   00000070
DELKY     ENTRY  ROOT   0000001C         DLADR     ENTRY  ROOT   00000070         DLKCP     ENTRY  ROOT   00000024
ENDCRL    ENTRY  ROOT   00000050         ESETL     ENTRY  ROOT   0000007C         ESLMT     ENTRY  ROOT   0000007C
FIND      ENTRY  ROOT   00000094         FREE      ENTRY  ROOT   00000080         GET       ENTRY  ROOT   00000064
GETLOAD   ENTRY  ROOT   0000000C         GETUP     ENTRY  ROOT   00000068         GTADR     ENTRY  ROOT   00000074
INSERT    ENTRY  ROOT   00000074         KE$ALP    ENTRY  ABS    00000683         KE$RES    ENTRY  ABS    00000883
LNKCP     ENTRY  ROOT   00000020         MODRJI    CSECT  ROOT   000000F0         OPEN      ENTRY  ROOT   0000008C
OPENF     ENTRY  ROOT   0000008C         PUT       ENTRY  ROOT   0000006C         RDID      ENTRY  ROOT   00000064
RDIDL     ENTRY  ROOT   00000068         RDKEY     ENTRY  ROOT   00000028         RDKEYL    ENTRY  ROOT   0000002C
RDKYI     ENTRY  ROOT   00000030         RDSQ      ENTRY  ROOT   0000003C         RDSQI     ENTRY  ROOT   00000040
RDSQL     ENTRY  ROOT   00000080         RDSR      ENTRY  ROOT   00000034         RDSRL     ENTRY  ROOT   00000038
REBUILD   ENTRY  ROOT   00000004         RELREC    ENTRY  ROOT   00000084         RETURN    ENTRY  ROOT   0000009C
SEND      ENTRY  ROOT   00000098         SETL      ENTRY  ROOT   00000078         SETLOAD   ENTRY  ROOT   00000010
SNAP      ENTRY  ROOT   00000058         SSLOCK    ENTRY  ROOT   00000044         SSUNLK    ENTRY  ROOT   00000048
STCRL     ENTRY  ROOT   0000004C         STLMT     ENTRY  ROOT   00000078         SUB       ENTRY  ROOT   00000014
SUBPROG   ENTRY  ROOT   00000014         SUPPLY    CSECT  ROOT   00000100         UNLOCK    ENTRY  ROOT   00000086
WRID      ENTRY  ROOT   0000006C         XR70MS    ENTRY  ROOT   00000008         ZF#LINK   CSECT  ROOT   00000000
```

Figure C-7. Sample BAL Action Program SUPPLY Processing Successive Transactions (Part 8 of 9)

SUCCESSIVE TRANSACTIONS IN BAL: SUPPLY PROGRAM

```
                                              ** ALLOCATION MAP **

                       LOAD MODULE -    SUPPLY          SIZE -    00000883

PHASE NAME  TRANS ADDR     FLAG        LABEL      TYPE       ESID          LNK ORG      HIADDR     LENGTH       OBJ ORG
SUPPLYCO       NODE - ROOT                                                 00000000     00000882   00000883
*** START OF AUTO-INCLUDED ELEMENTS -
       - 79/06/28 18.40 -             ZF#LINK     OBJ
                                      ZF#LINK    CSECT        01          00000000     000000EB   000000EC     00000000
                                      XR7DMS      ENTRY       01          00000008                              00000008
                                      BUILD       ENTRY       01          00000000                              00000000
                                      REBUILD     ENTRY       01          00000004                              00000004
                                      GET         ENTRY       01          00000064                              00000064
                                      GETUP       ENTRY       01          00000068                              00000068
                                      PUT         ENTRY       01          0000006C                              0000006C
                                      DELETE      ENTRY       01          00000070                              00000070
                                      INSERT      ENTRY       01          00000074                              00000074
                                      SETL        ENTRY       01          00000078                              00000078
                                      ESETL       ENTRY       01          0000007C                              0000007C
                                      FREE        ENTRY       01          00000080                              0000008C
                                      RELREC      ENTRY       01          00000084                              00000084
                                      UNLOCK      ENTRY       01          00000088                              00000088
                                      OPEN        ENTRY       01          0000008C                              0000008C
                                      CLOSE       ENTRY       01          00000090                              00000090
                                      FIND        ENTRY       01          00000094                              00000094
                                      SEND        ENTRY       01          00000098                              00000098
                                      RETURN      ENTRY       01          0000009C                              0000009C
                                      ARETURN     ENTRY       01          0000005C                              0000005C
                                      SNAP        ENTRY       01          00000058                              00000058
                                      SUB         ENTRY       01          00000014                              00000014
                                      RDSUL       ENTRY       01          00000080                              00000080
                                      GTADP       ENTRY       01          00000074                              00000074
                                      DLADR       ENTRY       01          00000070                              00000070
                                      ADDKY       ENTRY       01          00000018                              00000018
                                      DELKY       ENTRY       01          0000001C                              0000001C
                                      LNKCP       ENTRY       01          00000020                              00000020
                                      DLKCP       ENTRY       01          00000024                              00000024
                                      WRID        ENTRY       01          0000006C                              0000006C
                                      RDID        ENTRY       01          00000064                              00000064
                                      RDIDL       ENTRY       01          00000068                              00000068
                                      RDKEY       ENTRY       01          00000028                              00000028
                                      RDKEYL      ENTRY       01          0000002C                              0000002C
                                      RDKYI       ENTRY       01          00000030                              00000030
                                      RDSR        ENTRY       01          00000034                              00000034
                                      RDSRL       ENTRY       01          00000038                              00000038
                                      RDSG        ENTRY       01          0000003C                              0000003C
                                      RDSGI       ENTRY       01          00000040                              00000040
                                      STLMT       ENTRY       01          00000078                              00000076
                                      ESLMT       ENTRY       01          0000007C                              0000007C
                                      SSLOCK      ENTRY       01          00000044                              00000044
                                      SSUNLK      ENTRY       01          00000048                              00000048
                                      STCRL       ENTRY       01          0000004C                              0000004C
                                      ENDCRL      ENTRY       01          00000050                              00000050
                                      CHDRB       ENTRY       01          00000054                              00000054
                                      OPENF       ENTRY       01          0000008C                              0000008C
                                      SUBPROG     ENTRY       01          00000014                              00000014
                                      SETLOAD     ENTRY       01          00000010                              00000010
                                      GETLOAD     ENTRY       01          0000000C                              0000000C
       - 76/12/20 09.48 -             MODRJ1      OBJ
                                      MODRJ1     CSECT        01          000000F0     000001C9   000000DA     00000000
*** END OF AUTO-INCLUDED ELEMENTS -
       - 81/04/16 12.58 -             SUPPLY      OBJ
                                      SUPPLY     CSECT        01          00000100     00000882   000006E3     00000000
          00000100


                                               FLAG CODES -
B - BLK DATA CSECT      D - AUTO-DELETED       E - EXCLUSIVE 'A' REF    G - GENERATED EXTRN       I - INCLUSIVE 'V' REF
L - DEFERRED LENGTH     M - MULTIPLY DEFINED   N - NOT INCLUDED         P - PROMOTED COMMON       R - SHARED REC PRODUCED







S - SHARED ITEM          U - UNDEFINED REF     V - VCON ITEM
*ANY OTHER CODES REPRESENT PROCESS ERRORS*


LINK EDIT OF 'SUPPLY'   COMPLETED
DATE- 81/04/16 TIME- 13.05
ERRORS ENCOUNTERED- 0000  UPSI- X'00'
```

Figure C-7. Sample BAL Action Program SUPPLY Processing Successive Transactions (Part 9 of 9)

C.4. SAMPLE BAL ACTION PROGRAMS PERFORMING DIALOG TRANSACTIONS (APCHKS SERIES)

APCHKS and APITMS action programs

The APCHKS action program uses delayed internal succession to call the APITMS action program (Figure C-11). The APITMS action program uses delayed internal succession for error processing to return to the APCHKS action program for changes or corrections to records.

The APCHKS Action Program

APCHKS description

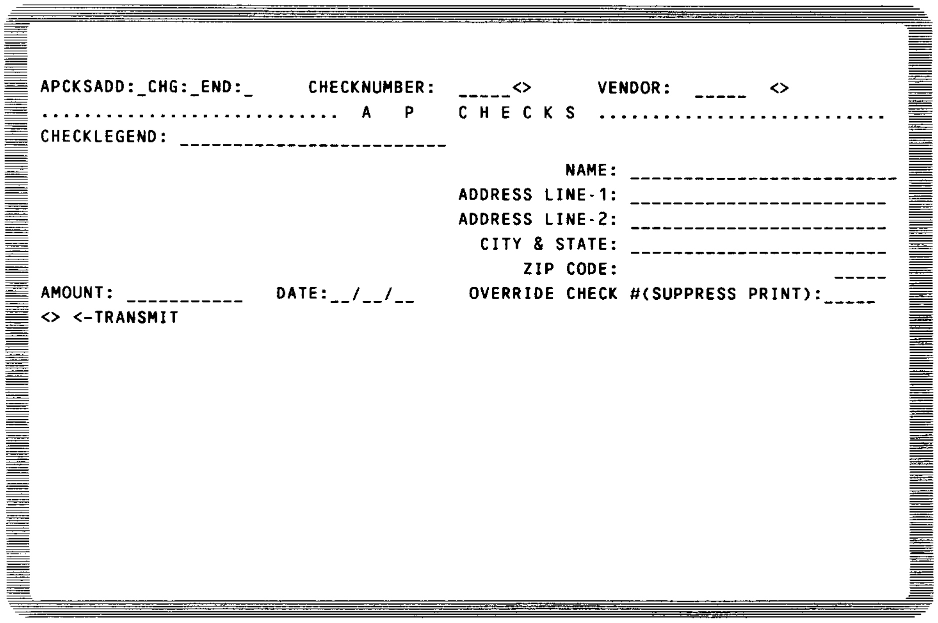
The APCHKS action program (Figure C-10) either adds new records to the master vendor file or updates and corrects records on that file. It also ends by accumulating a batch total of all checks paid.

Output screen formats input

When the terminal operator enters the transaction code, APCKS, the APCHKS action program builds a screen format as output, which is queued as input to the APITMS action program.

Delayed internal succession

Here, APCHKS uses delayed internal succession (Figure C-10, lines 647-652) to call the APITMS action program (Figure C-11), which in turn sends out the screen format shown in Figure C-8.



```
APCKSADD:_CHG:_END:_      CHECKNUMBER:  _____<>      VENDOR:  _____  <>
...........................  A   P   C H E C K S  ...........................
CHECKLEGEND: ________________________
                                                 NAME: ________________________
                                       ADDRESS LINE-1: _______________________
                                       ADDRESS LINE-2: _______________________
                                         CITY & STATE: _______________________
                                             ZIP CODE:                   _____
AMOUNT: ___________   DATE:__/__/__    OVERRIDE CHECK #(SUPPRESS PRINT):_____
<> <-TRANSMIT
```

Figure C-8. Screen Format 1 Generated by APITMS Action Program

Processing APCHKS program

The operator can add or change a record on the vendor master file, VENDORM, or end the work session and obtain a checks total. When adding or changing a record, he must supply a check number and vendor number followed by the name and address of the new vendor or vendor for update. In addition, he must supply the amount of the check for that vendor and the date, place the cursor, and transmit.

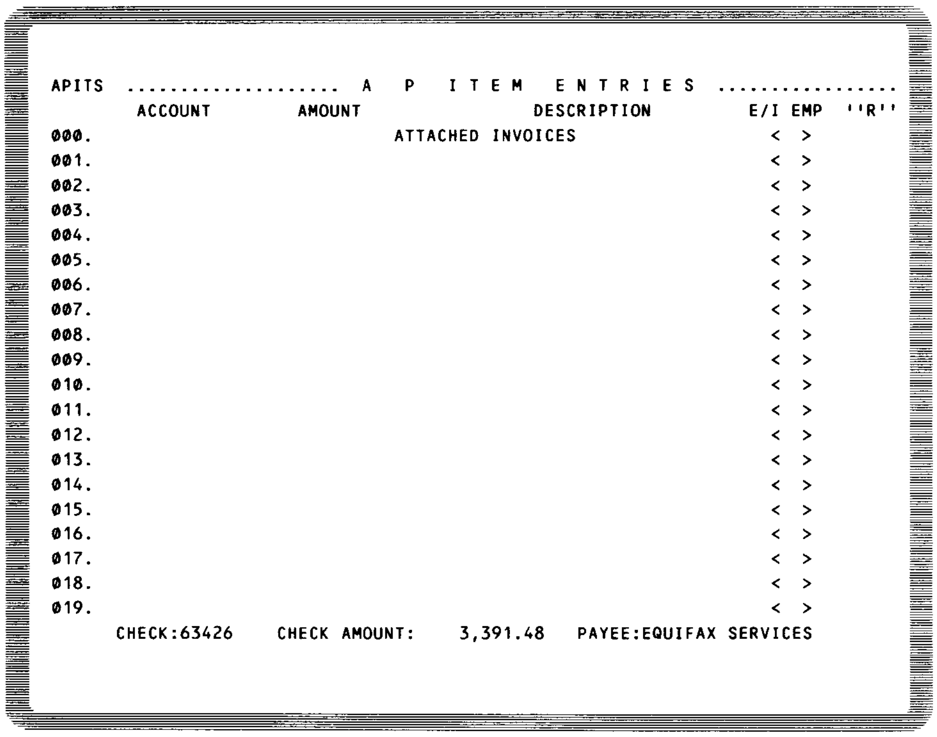
File updating and succession

This transmit reschedules the APCHKS action program which in turn validates the new or updated vendor record data, adds it to or changes it in the vendor master file, and uses delayed internal succession to pass control to the APITMS action program.

The APITMS Action Program

Operator entries

This program (Figure C-11) receives control from the APCHKS action program and generates a screen (Figure C-9) for the operator to enter the item invoices designating account number, amount of check, description, and whether the check is for an employee or for income.



```
APITS  .................... A   P   I T E M   E N T R I E S  .................
           ACCOUNT        AMOUNT                DESCRIPTION           E/I EMP  ''R''
ØØØ.                              ATTACHED INVOICES                    <  >
ØØ1.                                                                   <  >
ØØ2.                                                                   <  >
ØØ3.                                                                   <  >
ØØ4.                                                                   <  >
ØØ5.                                                                   <  >
ØØ6.                                                                   <  >
ØØ7.                                                                   <  >
ØØ8.                                                                   <  >
ØØ9.                                                                   <  >
Ø1Ø.                                                                   <  >
Ø11.                                                                   <  >
Ø12.                                                                   <  >
Ø13.                                                                   <  >
Ø14.                                                                   <  >
Ø15.                                                                   <  >
Ø16.                                                                   <  >
Ø17.                                                                   <  >
Ø18.                                                                   <  >
Ø19.                                                                   <  >
          CHECK:63426     CHECK AMOUNT:     3,391.48    PAYEE:EQUIFAX SERVICES
```

Figure C-9. Screen Format 2 Generated by APITMS Action Program

Operator entries

After the terminal operator enters all item invoices, he can place a cursor in position and transmit or place an 'R' in the cursor position and transmit.

Placing cursor in position

If he places a cursor in the cursor position, APITMS:

- verifies all invoice entries by calling itself for each screen of 20 invoices until a blank line is reached;
- accumulates all amount fields for comparison with the check amount for that account;
- writes an APITMS record for each invoice line entered on the screen; and
- creates a format on the screen with a prompting message to tell the operator how to print a check from the terminal. This format is not shown here.

Validating changes

Correcting errors

If the check amount is not equal to the item invoice total, APITMS returns control to APCHKS and displays the erroneous record for the operator to make changes to the item or add new items. Again, it verifies the changes and when correct, either creates a format for checks to be printed or allows for an account review.

Entering 'R'

If the terminal operator enters 'R', APITMS passes control to APAUDT, which returns a screen containing invoice entries. APAUDT is not illustrated here.

Obtaining batch totals

At the end of a session, when the operator chooses the END option on the APITMS screen format 1 (Figure C-8), check totals have been accumulated in the AP header record of the APCHKS file. APCHKS then returns to the screen the batch total of all checks entered for that session.

```
LINE    SOURCE STATEMENT                                                    OS/3 ASM

   2 APCHKS   START 0
   3 ***********************************************************************************
   4 *                  AUTHOR  : R L LEONARD
   5 *                  DATE    : 12 MARCH 1980
   6 *                  SITE    : GAY & TAYLOR INC, WINSTON-SALEM, NC, 27102
   7 *                  PURPOSE: TO ADD AND CORRECT RECORDS FOR ACCOUNTS PAYABLE
   8 *                           CHECKS
   9 *                  CHANGE LOG
  10 ***********************************************************************************
  11          Y$$START                          .STARTING CONVENTIONS
  13+Y$$B     EQU    * .START OF PROGRAM
  14+*
  15+*************** REGISTER EQUATES
  16+*
  17+R0       EQU    0
  18+R1       EQU    1
  19+R2       EQU    2 .PIB COVER
  20+R3       EQU    3 .IMA COVER
  21+R4       EQU    4 .WORK COVER
  22+R5       EQU    5 .OMA COVER
  23+R6       EQU    6 .CDA COVER
  24+R7       EQU    7 .INTERNAL ROUTINE LINKAGE
  25+R8       EQU    8 .I/O - NORMAL RETURN ADDRESS
  26+R9       EQU    9 .I/O - ERROR RETURN ADDRESS
  27+R10      EQU    10 .PROGRAM COVER #3
  28+R11      EQU    11 .PROGRAM COVER #2
  29+R12      EQU    12 .PROGRAM COVER #1
  30+R13      EQU    13
  31+R14      EQU    14
  32+R15      EQU    15
  33+*
  34+*************** ESTABLISH PROGRAM COVERING
  35+*
  36+         USING  *,R12,R11,R10 .PROGRAM CODE
  37+         USING  ZA#DPIB,R2 .PIB
  38+         USING  ZA#IMH,R3 .IMA
  39+         USING  WORK,R4 .WORK
  40+         USING  ZA#OMH,R5 .OMA
  41+         USING  CDA,R6 .CDA
  42+*
  43+*************** ESTABLISH IMS INTERFACE
  44+*
  45+         STM    R14,R12,12(R13) .STORE REG IN CALLS' SAVE AREA
  46+         LR     R12,R15 .ADDRESS OF THIS PROGRAM
  47+         LM     R2,R6,0(R1) .ACTIVATION AREAS FROM PARAM
  48+         LA     R11,SAVE .THIS PROGRAM SAVE AREA
  49+         ST     R11,8(,R13) .PUT THIS SAVE INTO CALLS' SAVE
  50+         ST     R13,4(,R11) .PUT CALLS' SAVE INTO THIS SAVE
  51+         LR     R13,R11 .REG 13 = THIS SAVE AREA
  52+         LR     R11,R12 .SECOND PROGRAM COVER
  53+         LA     R11,1(R11)
  54+         LA     R11,4095(R11)
  55+         LR     R10,R11 .THIRD PROGRAM COVER
  56+         LA     R10,1(R10)
  57+         LA     R10,4095(R10)
  58+         GETIME M
  59+         DS     0H
  60+         LA     1,1
  61+         SVC    7
  62+         ST     R1,STIM$ .STARTUP TIME
```

Figure C-10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 1 of 22)

```
 64          DROP   R6                        .NO CDA
 65          PRINT  GEN
 66          BAL    R7,DAYTIME                .GET DATE-TIME
 67 *
 68 ******** OPERATOR CANCEL
 69 *
 70          CLI    IMA+4,C'C'
 71          BE     EMSG8                     CANCEL
 72 ***********************************************************************
 73 *                CHECK SECURITY
 74 ***********************************************************************
 75          MVC    PASSKEY(5),=C'APCHK'
 76          Y$$SECUR                         .CHECK FOR OPEN-VALID
 77+*************************************************************************
 78+*                CHECK SECURITY FOR OPEN APPLICATION                   *
 79+*
 80+*                         ASSUMES KEY IN FIELD "PASSKEY"               *
 81+*************************************************************************
 82+         MVC    KTABLEMT(3),=C'T60'
 83+         MVC    KTABLEMT+3(5),PASSKEY
 84+         LA     R9,Y$$0020 .NO FIND ADDRESS
 85+         BAL    R8,GTABLEMT .GET SECURITY RECORD
 86+         CLI    TABSTS,C' ' .RECORD ACTIVE?
 87+         BNE    Y$$0020 .NO
 88+         MVI    WORK1,X'00' .SETUP TO CVB
 89+         MVC    WORK1+1(7),WORK1
 90+         MVC    WORK1+8(2),ZA#ISTID+2 .TERMINAL ID
 91+         PACK   WORK1+6(2),WORK1+8(2)
 92+         CVB    R1,WORK1 .TERMINAL FIELD COUNTER
 93+         LA     R7,TERMTAB-4 .BEGINNING OF TERMINAL FIELDS
 94+Y$$0010  LA     R7,4(R7) .NEXT TERMINAL FIELDS
 95+         BCT    R1,Y$$0010 .COUNT DOWN TO THIS TERMINAL
 96+         CLC    0(3,R7),=C'   ' .OPEN?
 97+         BE     Y$$0020 .NO
 98+         MVC    WHO(3),0(R7) .SAVE USER INITIALS
 99+         CLC    3(1,R7),LIMIT .OPEN BUT OVER LIMIT (SET DOWN)
100+         BNH    Y$$0030 .NO
101+Y$$0020  MVC    OMA(LY$$M1),Y$$M1 .APPLICATION NOT OPEN
102+         MVC    ZA#OTL(2),=Y(0+LY$$M1+4) .MESSAGE LENGTH
103+         B      TERM
104+Y$$M1    DC     X'100A18011C'
105+         DC     C'APPLICATION NOT OPEN'
106+         DC     X'1D10020000'
107+LY$$M1   EQU    *-Y$$M1
110                                                         Y$$TRAIL A
111+         PRINT  OFF
121+         PRINT  ON
122          CLC    IMA+11(3),=C'ADD'          TRANSMIT PROTECT?
123          BE     EMSG1                      YES
124 ***********************************************************************
125 *                INITIALIZATIONS
126 ***********************************************************************
127          Y$$IN 11                          EXTRACT SCREEN DATA
128+         LA     R0,11 .SCREEN NUMBER
129+         BAL    R8,MOVEIN .GO TO INPUT SCREEN ROUTINE
130          MVI    FILL,C'_'                  SETUP PROTECTED REPLACEMENT
131          MVI    PSTART,C':'
132          MVC    PSTART+1(LPDATA-1),PSTART
133          MVC    PMSG1(80),BLANKS
134          MVI    USTOP,X'FF'
135          MVI    PSTOP,X'FF'
```

Figure C-10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 2 of 22)

```
*
************** GET AP HEADER
*
         MVC    KACCTPAY,BLANKS
         MVC    KACCTPAY(2),=C'AP'
         LA     R9,EMSG2                    "NOT FOUND"
         BAL    R8,GACCTPAY                 GET HEADER
         MVC    HACCTPAY(165),RACCTPAY      SAVE RECORD
         CLI    UCHG,C' '                 CHANGE?
         BE     L0034                     NO
*
*        ERRORS FROM ITEM ENTRIES?
*
         LA     R9,PMSG1                  ERROR MESSAGE
         TM     APHERR,X'01'             ITEMS=CHECK?
         BZ     L0030                     YES
         MVC    0(LMSG9,R9),MSG9          NOT=
         ED     DM9A(12,R9),APHITMT       ITEM TOTAL
         LA     R9,LMSG9(,R9)             NEXT POSITION
L0030    TM     APHERR,X'02'             CASH=0?
         BZ     L0032                     YES
         MVC    0(LMSG10,R9),MSG10        CASH NOT=0
         LA     R9,LMSG10(,R9)             NEXT POSITION
L0032    TM     APHERR,X'04'             ACCRUAL=0?
         BZ     L0034                     YES
         MVC    0(LMSG11,R9),MSG11        ACCRUAL NOT=0
         LA     R9,LMSG11(,R9)            NEXT POSITION
L0034    CLC    ZA#ITL(2),=Y(0+IMA1)      INITIAL SCREEN?
         BH     L0050                      NO
         CLC    APHCHKCT(5),BLANKS
         BE     FORMAT                     FORMAT SCREEN
         MVC    UCHECK(5),APHCHKCT
         B      FORMAT                     FORMAT SCREEN
L0050    EQU    *
         CLI    UEND,C' '                  END OF BATCH?
         BNE    L0100                      NO
*
************** END OF BATCH ******************************************
*
                                                              Y$$TRAIL B
         PRINT  OFF
         PRINT  ON
         AP     APHREPT(5),APHBATCH(5)
         MVC    RACCTPAY(165),HACCTPAY
         MVC    RACCTPAY+2(6),=C'ZBATCH'
         MVC    RACCTPAY+8(3),APHBATHN
         MVC    RACCTPAY+41(2),YYMMDD
         MVC    RACCTPAY+37(4),YYMMDD
         CLI    UEND,C'N'                  NO OUTPUT RECORD?
         BE     L0060                      YES
         LA     R9,Y$$IOS30                ERROR
         BAL    R8,IACCTPAY                INSERT BATCH RECORD
L0060    MVC    OMA(LMSG3),MSG3            "TOTALS"
         MVC    OMA+DM3A(3),APHBATHN       BATCH #
         MVC    WORK1+4(2),YYMMDD
         MVC    WORK1(4),YYMMDD+2
         PACK   WORK1+6(4),WORK1(6)
         ED     OMA+DM3B(10),WORK1+6       DATE
         MVC    OMA+DM3C(3),APHCHKS        # OF CHECKS
         ED     OMA+DM3D(14),APHBATCH      AMOUNT
         PACK   WORK1(2),APHBATHN(3)       ADD 1 TO BATCH #
```

Figure C–10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 3 of 22)

```
         AP     WORK1(2),=P'1'
         UNPK   APHBATHN(3),WORK1(2)
         OI     APHBATHN+2,X'F0'
         MVC    APHCHKCT(5),BLANKS        CLEAR COUNTERS
         SP     APHBATCH(5),APHBATCH(5)   BATCH TOTAL
         MVC    APHCHKS(3),=C'000'
         MVC    APHVODS(3),=C'000'
         MVC    APHITMS(3),=C'000'
         MVC    APHERRS(3),=C'000'
         MVC    APHITMC(3),=C'000'        ITEM COUNT
         MVC    ZA#OTL(2),=Y(0+LMSG3+4)
         MVC    KACCTPAY(15),BLANKS
         MVC    KACCTPAY(2),=C'AP'
         CLI    UEND,C'N'                 NO OUTPUT RECORD?
         BE     TERM                      YES
         LA     R9,Y$$IOS30
         BAL    R8,UACCTPAY
         MVC    RACCTPAY(165),HACCTPAY
         BAL    R8,PACCTPAY
         B      TERM
*****************************************************************************
*                       VALIDATE LINE 1
*****************************************************************************
*
**************  CHECK FOR ADD/CHANGE
*
L0100    EQU    *                         CHECK ADD-CHG
                                                             Y$$TRAIL C
+        PRINT  OFF
+        PRINT  ON
         MVI    APHPRNT,C' '             CLEAR CHECK PRINT
         CLI    UADD,C' '
         BNE    L0140
         CLI    UCHG,C' '
         BNE    L0140
L0120    MVI    PADD,X'1C'
         MVI    PCHG,X'1C'
         MVI    ERR,C'Y'
         B      L0360
L0140    CLI    UADD,C' '
         BE     L0160
         CLI    UCHG,C' '
         BNE    L0120
L0160    EQU    *
         MVI    APHAOC,C'A'               ADD
         CLI    UCHG,C' '
         BE     L0165
         MVI    APHAOC,C'C'               CHANGE
*
**************  TRANSMIT POSITION 2
*
*
**************  TYPE
*
L0165    CLI    UTYPE,C' '                     TYPE ENTERED?
         BE     L0170
         MVC    APHTYPE(1),UTYPE
         B      L0175
L0170    MVI    APHTYPE,C'N'              NEW CHECK
*
**************  CHECK NUMBER
```

Figure C–10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 4 of 22)

```
*
L0175       CLC     UCHECK(5),BLANKS                    CHECK ENTERED?
            BNE     L0180                       YES
            MVC     UCHECK(5),APHCHKCT          USE NEXT CHECK
L0180       LA      R1,UCHECK
            BAL     R7,RJ5                      RIGHT JUSTIFY
            BZ      L0200
L0190       MVI     ERR,C'Y'                    SET ERRORS
            MVI     PCHECK,X'1C'
            B       L0230
L0200       CLC     UCHECK(5),=C'00000'         ZERO CHECK?
            BE      L0190
            CLI     UADD,C' '                   ADD?
            BE      L0210                       NO-CHANGE
            MVC     APHCHKCT(5),UCHECK          RESET NEXT CHECK
            MVC     APHCHECK(5),UCHECK          SET CURRENT CHECK
            B       L0230
L0210       MVC     APHCHECK(5),UCHECK          SET CURRENT CHECK
*
*                   GET UPDATE CHECK DATA FOR SCREEN
*
            CLC     ZA#ITL(2),=Y(0+IMA3)        FULL SCREEN?
            BH      L0230                       YES
                                                                       YSSTRAIL F
            PRINT   OFF
            PRINT   ON
            MVC     KACCTPAY(15),BLANKS          GET CHECK
            MVC     KACCTPAY(2),=C'AC'
            MVC     KACCTPAY+2(6),APHTYPE
            LA      R9,EMSG4
            BAL     R8,GACCTPAY
            MVC     CACCTPAY(165),RACCTPAY       MOVE DATA TO CHECK WORK AREA
            MVC     UVENDOR(5),APCVENDR
            MVC     ULEGEND(25),APCLEGND         MOVE DATA TO SCREEN
            MVC     UNAME(26),APCNAME
            MVC     UADDR1(25),APCADDR1
            MVC     UADDR2(25),APCADDR2
            MVC     UCITY(25),APCCITY
            UNPK    UZIP(5),APCZIP(3)
            UNPK    UAMOUNT+1(9),APCAMT(5)
            MVI     UAMOUNT,C'0'
            CP      APCAMT(5),=P'0'
            BNL     L0212
            MVI     UAMOUNT,C'-'
            OI      UAMOUNT+9,X'F0'
L0212       UNPK    WORK1(7),APCDATE(4)
            MVC     UDATE(6),WORK1+1
            OI      UDATE+5,X'F0'
            CLI     APCPRNT,C' '
            BE      L0230
            MVC     UOVERIDE(5),APCCHECK
*
*************** TRANSMIT POSITION 3
*
L0230       CLC     ZA#ITL(2),=Y(0+IMA3)         POSITION 3?
            BL      L0360
*
*************** GET VENDOR
*
L0280       CLI     UVENDOR,C'E'                 EMPLOYEE?
            BNE     L0300                        NO
```

Figure C–10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 5 of 22)

```
         MVC   KPAYROLL(4),UVENDOR+1     GET EMPLOYEE
         MVI   KPAYROLL+4,C'0'
         LA    R9,L0320
         BAL   R8,GPAYROLL
         MVC   VMNAME(26),PMNAME
         MVC   VMADDR1(3),PMBRW          BRANCH OF WORK
         B     L0330
L0300    MVC   KVENDORM(5),UVENDOR       GET VENDOR
         LA    R8,L0330
         BAL   R9,GVENDORM
L0320    MVI   PVENDOR,X'1C'
         MVI   ERR,C'Y'
         B     L0360
L0330    CLC   ZA#ITL(2),=Y(0+IMA3)     FULL SCREEN?
         BH    L0360
*
*        MOVE VENDOR TO SCREEN
*
         CLI   UVENDOR,C'E'             EMPLOYEE
         BE    L0335
         MVC   UNAME(26),VMNAME         NAME
         MVC   UADDR1(25),VMADDR1       LINE 1
         MVC   UADDR2(25),VMADDR2       LINE 2
         MVC   UCITY(25),VMCITY         CITY AND STATE
         MVC   UZIP(5),VMZIP            ZIP CODE
         B     L0340
L0335    MVC   UNAME(26),PMNAME         NAME
         MVC   UADDR1(3),PMBRW          BRANCH OF WORK
*
************** SYSTEM DATE
*
L0340    MVC   UDATE(4),YYMMDD+2
         MVC   UDATE+4(2),YYMMDD
*
************** ANY ERRORS ON LINE 1
*
L0360    CLI   ERR,C'Y'                  ERRORS?
         BE    FORMAT
                                                          YSSTRAIL E
         PRINT OFF
         PRINT ON
         CLC   ZA#ITL(2),=Y(0+IMA3)      FULL SCREEN?
         BH    L0500                     VERIFY FIELDS
         MVI   UTRAN2,C'.'              FLAG TO EXPECT FULL SCREEN
         B     FORMAT
***********************************************************************
*              VALIDATE SCREEN DATA
***********************************************************************
L0500    EQU   *
         CLI   UTRAN2,C'.'              SHOULD BE FULL SCREEN?
         BNE   EMSG7                    NO
*
************** CHECK NAME
*
         CLC   UNAME(26),BLANKS
         BNE   L0504
         MVI   ERR,C'Y'
         MVI   PNAME,X'1C'
*
************** CHECK CITY
*
```

Figure C-10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 6 of 22)

```
L0504     CLC    UADDR2(25),BLANKS
          BE     L0507
          CLC    UCITY+20(5),BLANKS        ROOM FOR ZIP?
          BE     L0507
          MVI    PCITY,X'1C'
          MVI    ERR,C'Y'
L0507     EQU    *
*
************** CHECK ZIP CODE
*
                                                              Y$STRAIL G
          PRINT  OFF
          PRINT  ON
          LA     R1,UZIP                    VALIDATE ZIP CODE
          BAL    R7,RJ5
          BZ     L0510
          MVI    PZIP,X'1C'
          MVI    ERR,C'Y'
*
************** CHECK AMOUNT
*
L0510     EQU    *
          MVC    PMSG1(10),UAMOUNT         SAVE INPUT
          LA     R1,UAMOUNT                 VALIDATE AMOUNT
          BAL    R7,RJ10
          BZ     L0520
L0515     MVI    PAMOUNT,X'1C'
          MVI    ERR,C'Y'
          B      L0540
L0520     CLI    UAMOUNT,C'0'              AMOUNT TOO LARGE?
          BNE    L0515                     YES
*         IS THIS A VOID CHECK? (NEGATIVE AMOUNT)
          CLI    UTYPE,C' '                TYPE ENTERED?
          BNE    L0540                     YES-SKIP
          CLI    UCHG,C' '                 CHANGE?
          BNE    L0540                     YES-SKIP
          PACK   WORK1+11(5),UAMOUNT+1(9)
          CP     WORK1+11(5),=P'0'         NEGATIVE?
          BNL    L0540                     NO
          MVI    APHTYPE,C'V'              VOID CHECK
*
************** CHECK DATE
*
L0540     MVC    WORK1(2),UDATE+4           VALIDATE DATE
          MVC    WORK1+2(4),UDATE
          BAL    R7,DATCHK
          BZ     L0560
          MVI    PDATE,X'1C'
          MVI    ERR,C'Y'
*
************** CHECK OVERRIDE CHECK NUMBER
L0560     EQU    *
          CLI    APHTYPE,C'V'              VOID CHECK?
          BNE    L0565
          CLC    UOVERIDE(5),BLANKS
          BNE    L0565                     OVERRIDE
          MVC    UAMOUNT(10),PMSG1         RESTORE INPUT AMOUNT FIELD
          MVC    PMSG1(LMSG12),MSG12
          B      L0575
L0565     CLC    UOVERIDE(5),BLANKS
          BE     L0600
```

Figure C-10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 7 of 22)

```
486          LA      R1,UOVERIDE
487          BAL     R7,RJ5
488          BZ      L0580
489 LC575    MVI     POVERIDE,X'1C'
490          MVI     ERR,C'Y'
491          B       L0600
492 L0560    MVI     APHPRNT,C'N'                    SUPPRESS PRINT FLAG
493          MVC     APHCHECK(5),UOVERIDE      OVERRIDE CHECK NUMBER
494 *
495 ******************************************************************
496 *                ANY SCREEN DATA ERRORS
497 *
498 L0600    EQU     *
499                                                             Y$$TRAIL H
500+         PRINT   OFF
510+         PRINT   ON
511          CLI     ERR,C'Y'                  ERRORS
512          BE      FORMAT                    YES
513 ******************************************************************
514 *                ADD/UPDATE CHECK RECORD
515 ******************************************************************
516                                                            Y$$TRAIL L
517+         PRINT   OFF
527+         PRINT   ON
528          MVI     CACCTPAY,C' '
529          MVC     CACCTPAY+1(164),CACCTPAY MOVE DATA TO CHECK
530          MVC     APCRID(2),=C'AC'
531          MVC     APCTYPE(1),APHTYPE
532          MVC     APCCHECK(5),APHCHECK
533          PACK    APCTDATE(4),YYMMDD(6)
534          PACK    APCDATE(4),UDATE
535          MVC     APCVENDR(5),UVENDOR
536          PACK    APCAMT(5),UAMOUNT+1(9)
537          MVC     APCNAME(26),UNAME
538          MVC     APCADDR1(25),UADDR1
539          MVC     APCADDR2(25),UADDR2
540          MVC     APCCITY(25),UCITY
541          PACK    APCZIP(3),UZIP(5)
542          MVC     APCLEGND(25),ULEGEND
543          MVC     APCPRNT(1),APHPRNT
544          SP      APHOLD(5),APHOLD(5)
545          CLI     UCHG,C' '
546          BNE     L0700
547          MVC     RACCTPAY(165),CACCTPAY    ADD CHECK
548          LA      R9,EMSG6
549          BAL     R8,IACCTPAY
550          CLI     APCPRNT,C'N'              WAS CHECK TO PRINT?
551          BE      L0720                     NO
552          PACK    WORK1(3),APHCHKCT(5)      UPDATE NEXT CHECK NUMBER
553          AP      WORK1(3),=P'1'
554          UNPK    APHCHKCT(5),WORK1(3)
555          OI      APHCHKCT+4,X'F0'
556          B       L0720
557 L0700    MVC     KACCTPAY(15),CACCTPAY     UPDATE CHECK
558                                                             Y$$TRAIL I
559+         PRINT   OFF
569+         PRINT   ON
570          LA      R9,EMSG4
571          BAL     R8,UACCTPAY
572                                                            Y$$TRAIL M
573+         PRINT   OFF
```

Figure C–10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 8 of 22)

```
         PRINT ON
         MVC   APHOLD(5),APAMT
         MVC   RACCTPAY(165),CACCTPAY
         BAL   R8,PACCTPAY
*        SETUP HEADER WITH CHECK INFORMATION
LC720    MVC   APHITMC(3),=C'001'
         MVC   APHDATE(6),UDATE
         MVC   APHVENDR(5),UVENDOR
         ZAP   APHAMT(5),APCAMT(5)
         ZAP   APHITMT(5),=P'0'
         MVC   APHNAME(26),UNAME
         MVC   APHLEGND(25),ULEGEND
         SP    APHACCR(5),APHACCR(5)
         ZAP   APHCASH(5),=P'0'
         SP    APHCASH(5),APHAMT(5)
         MVI   APHERR,C' '
         MVI   APHDONE,C' '
         MVC   ZA#PSID(6),=C'APITMS'
************************************************************
*              UPDATE AP HEADER
************************************************************
UPHEADER EQU   *
                                                    Y$$TRAIL J
         PRINT OFF
         PRINT ON
         MVC   KACCTPAY(15),BLANKS
         MVC   KACCTPAY(2),=C'AP'
         LA    R9,Y$$IOS30
         BAL   R8,UACCTPAY
         MVC   RACCTPAY(165),HACCTPAY
         BAL   R8,PACCTPAY
************************************************************
*              FORMAT OMA
*
************************************************************
FORMAT   EQU   *
                                                    Y$$TRAIL K
         PRINT OFF
         PRINT ON
         MVI   USNAP,C' '              CLEAR SNAP CODE
         Y$$OUT 11
         LA    R0,11 .SCREEN NUMBER
         BAL   R8,MOVEOUT .SCREEN AND DATA
************************************************************
*              SETUP NEXT TRANSACTION
************************************************************
         CLC   ZA#PSID(6),=C'APITMS'
         BNE   TERM
         MVC   ZA#OIL(2),=H'14'
         MVC   OMA+4(6),=C'APITS '
         MVI   ZA#PSIND,C'D'         DELAYED INTERNAL SUCCESSION
         B     TERM
         Y$$IOSTS                        .INPUT/OUTPUT STATUS
*************************************************************
**             INTERNAL ROUTINES                            *
*************************************************************
**
*************** CHECK FILE I/O STATUS
**
```

Figure C-10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 9 of 22)

```
661+IOSTATUS ORG    *
662+         CLI    ZA#PSC+1,0 .SUCCESSFUL?
663+         BNE    Y$$IOS05 .NO
664+         MVI    IOKEY,C' ' .CLEAR KEY
665+         MVC    IOKEY+1(14),IOKEY
666+         BR     R8
667+Y$$IOS05 CLI    ZA#PSC+1,1 .INVALID KEY?
668+         BER    R9
669+         CLI    ZA#PDSC+1,5 .FILE NOT DEFINED?
670+         BE     Y$$IOS10
671+         CLI    ZA#PDSC+1,6 .FILE CLOSED?
672+         BNE    Y$$IOS30
673+Y$$IOS10 CLI    IORET,C'Y' .RETURN ON FILE NOT AVAILABLE?
674+         BNE    Y$$IOS20
675+         SR     R8,R8 .FLAG FOR FILE NOT AVAILABLE
676+         BR     R9
677+Y$$IOS20 MVC    OMA(LIOM2),IOM2 .FILE NOT AVAILABLE
678+         MVC    ZA#OTL(2),=Y(0+LIOM2+4)
679+         MVC    OMA+DIOM2-IOM2(20),IOFILE
680+         B      TERM
681+Y$$IOSTR DC     C'0123456789ABCDEFX'
682+IOM1     DC     X'100A18011C'
683+         DC     C'INVALID FILE I/O '
684+DIOM1C   DC     CL5' ' .PIB STATUS
685+DIOM1A   DC     CL21' ' .FILE NAME
686+DIOM1B   DC     CL17' ' .FILE KEY
687+         DC     C'CALL ISD'
688+         DC     X'1D10020000'
689+LIOM1    EQU    *-IOM1
690+IOM2     DC     X'100A18011C'
691+DIOM2    DC     CL21' ' .FILE NAME
692+         DC     C'FILE NOT AVAILABLE'
693+         DC     X'1D10020000'
694+LIOM2    EQU    *-IOM2
695+Y$$IOS30 MVC    IOSTS,ZA#PSC
696+         TR     IOSTS,Y$$IOSTR .TRANSLATE TO PRINTABLE CHAR
697+         MVC    OMA(LIOM1),IOM1 .FILE NOT AVAILABLE
698+         MVC    OMA+DIOM1A-IOM1(21),IOFILE
699+         MVC    OMA+DIOM1B-IOM1(16),IOKEY
700+         MVC    OMA+DIOM1C-IOM1(4),IOSTS
701+         MVC    ZA#OTL(2),=Y(0+LIOM1+4)
702+         B      SNAP
703 *
704 ************** TABLE MASTER I/O
705 *
706 TABLEMT  Y$$GET 8
707+*
708+*               GET
709+*
710+GTABLEMT MVC    IOKEY(8),KTABLEMT .SAVE KEY
711+         MVI    IOKEY+8,C'G' .TYPE OF I/O
712+         ZG#CALL GET,(&FIL.,R&FIL.,K&FIL.)
713+         DS     0H
714+         LA     15,TABLEMT
715+         ST     15,PLIST+4*(1-1)
716+         LA     15,RTABLEMT
717+         ST     15,PLIST+4*(2-1)
718+         LA     15,KTABLEMT
719+         ST     15,PLIST+4*(3-1)
720+         OI     PLIST+4*(3-1),X'80'
```

Figure C-10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 10 of 22)

```
+         LA    1,PLIST
+         L     15,=V(GET)
+         BALR  14,15
+         AI    #GET,1 .INCREMENT IO COUNT
+         MVC   IOFILE(20),TABLEMT+8 .SAVE FILE
+         B     IOSTATUS .CHECK I/O STATUS
 *
 ************** VENDOR MASTER I/O
 *
 VENDORM  Y$$GET 5
+*
+*              GET
+*
+GVENDORM MVC   IOKEY(5),KVENDORM .SAVE KEY
+         MVI   IOKEY+5,C'G' .TYPE OF I/O
+         ZG#CALL GET,(&FIL.,R&FIL.,K&FIL.)
+         DS    0H
+         LA    15,VENDORM
+         ST    15,PLIST+4*(1-1)
+         LA    15,RVENDORM
+         ST    15,PLIST+4*(2-1)
+         LA    15,KVENDORM
+         ST    15,PLIST+4*(3-1)
+         OI    PLIST+4*(3-1),X'80'
+         LA    1,PLIST
+         L     15,=V(GET)
+         BALR  14,15
+         AI    #GET,1 .INCREMENT IO COUNT
+         MVC   IOFILE(20),VENDORM+8 .SAVE FILE
+         B     IOSTATUS .CHECK I/O STATUS
 *
 ************** PERSONNEL MASTER I/O
 *
 PAYROLL  Y$$GET 4
+*
+*              GET
+*
+GPAYROLL MVC   IOKEY(4),KPAYROLL .SAVE KEY
+         MVI   IOKEY+4,C'G' .TYPE OF I/O
+         ZG#CALL GET,(&FIL.,R&FIL.,K&FIL.)
+         DS    0H
+         LA    15,PAYROLL
+         ST    15,PLIST+4*(1-1)
+         LA    15,RPAYROLL
+         ST    15,PLIST+4*(2-1)
+         LA    15,KPAYROLL
+         ST    15,PLIST+4*(3-1)
+         OI    PLIST+4*(3-1),X'80'
+         LA    1,PLIST
+         L     15,=V(GET)
+         BALR  14,15
+         AI    #GET,1 .INCREMENT IO COUNT
+         MVC   IOFILE(20),PAYROLL+8 .SAVE FILE
+         B     IOSTATUS .CHECK I/O STATUS
 *
 ************** ACCOUNTS PAYABLE MASTER I/O
 *
 ACCTPAY  Y$$GET 15
+*
+*              GET
```

Figure C-10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 11 of 22)

```
781+*
782+GACCTPAY MVC    IOKEY(15),KACCTPAY .SAVE KEY
783+              MVI    IOKEY+15,C'G' .TYPE OF I/O
784+              ZG#CALL GET,(&FIL.,R&FIL.,K&FIL.)
785+              DS     0H
786+              LA     15,ACCTPAY
787+              ST     15,PLIST+4*(1-1)
788+              LA     15,RACCTPAY
789+              ST     15,PLIST+4*(2-1)
790+              LA     15,KACCTPAY
791+              ST     15,PLIST+4*(3-1)
792+              OI     PLIST+4*(3-1),X'80'
793+              LA     1,PLIST
794+              L      15,=V(GET)
795+              BALR   14,15
796+              AI     #GET,1 .INCREMENT IO COUNT
797+              MVC    IOFILE(20),ACCTPAY+8 .SAVE FILE
798+              B      IOSTATUS .CHECK I/O STATUS
799 ACCTPAY     YS$GETUP 15
800+*
801+*                    GETUP
802+*
803+UACCTPAY MVC    IOKEY(15),KACCTPAY .SAVE KEY
804+              MVI    IOKEY+15,C'U' .TYPE OF I/O
805+              ZG#CALL GETUP,(&FIL.,R&FIL.,K&FIL.)
806+              DS     0H
807+              LA     15,ACCTPAY
808+              ST     15,PLIST+4*(1-1)
809+              LA     15,RACCTPAY
810+              ST     15,PLIST+4*(2-1)
811+              LA     15,KACCTPAY
812+              ST     15,PLIST+4*(3-1)
813+              OI     PLIST+4*(3-1),X'80'
814+              LA     1,PLIST
815+              L      15,=V(GETUP)
816+              BALR   14,15
817+              AI     #GETUP,1 .INCREMENT IO COUNT
818+              MVC    IOFILE(20),ACCTPAY+8 .SAVE FILE
819+              B      IOSTATUS .CHECK I/O STATUS
820 ACCTPAY     YS$PUT 15
821+*
822+*                    PUT
823+*
824+PACCTPAY MVC    IOKEY(15),KACCTPAY .SAVE KEY
825+              MVI    IOKEY+15,C'P' .TYPE OF I/O
826+              ZG#CALL PUT,(&FIL.,R&FIL.)
827+              DS     0H
828+              LA     15,ACCTPAY
829+              ST     15,PLIST+4*(1-1)
830+              LA     15,RACCTPAY
831+              ST     15,PLIST+4*(2-1)
832+              OI     PLIST+4*(2-1),X'80'
833+              LA     1,PLIST
834+              L      15,=V(PUT)
835+              BALR   14,15
836+              AI     #PUT,1 .INCREMENT IO COUNT
837+              MVC    IOFILE(20),ACCTPAY+8 .SAVE FILE
838+              B      IOSTATUS .CHECK I/O STATUS
839 ACCTPAY     YS$INSRT 15
840+*
```

Figure C-10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 12 of 22)

```
841+*               INSERT
842+*
843+IACCTPAY MVC    IOKEY(15),KACCTPAY .SAVE KEY
844+         MVI    IOKEY+15,C'I' .TYPE OF I/O
845+         ZG#CALL INSERT,(EFIL.,REFIL.)
846+         DS     0H
847+         LA     15,ACCTPAY
848+         ST     15,PLIST+4*(1-1)
849+         LA     15,RACCTPAY
850+         ST     15,PLIST+4*(2-1)
851+         OI     PLIST+4*(2-1),X'80'
852+         LA     1,PLIST
853+         L      15,=V(INSERT)
854+         BALR   14,15
855+         AI     #INSERT,1 .INCREMENT IO COUNT
856+         MVC    IOFILE(20),ACCTPAY+8 .SAVE FILE
857+         B      IOSTATUS .CHECK I/O STATUS
858          Y$$NOW                          DATE-TIME
859+*
860+*************** DATE AND TIME STAMP ********************************
861+*
862+DAYTIME  ORG    *
863+         GETIME S
864+         DS     0H
865+         SR     1,1
866+         SVC    7
867+         ST     R0,WORK1 .DATE-0YYMMDD+
868+         UNPK   WORK1+4(7),WORK1(4)
869+         MVC    YYMMDD(6),WORK1+5
870+         OI     YYMMDD+5,X'F0' .FIX SIGN
871+         ST     R1,WORK1 .TIME-0HHMMSS+
872+         UNPK   WORK1+4(7),WORK1(4)
873+         MVC    HHMMSS(6),WORK1+5
874+         OI     HHMMSS+5,X'F0' .FIX SIGN
875+         BR     R7 .RETURN REGISTER
876          Y$$RJ                         RIGHT JUSTIFY
877+*
878+*************** RIGHT JUSTIFY ***************************************
879+*
880+*
881+*               R0  = FIELD LENGTH
882+*               R1  = FIELD ADDRESS
883+*               R15 = RETURN STATUS
884+*
885+RJ1      LA     R0,1 .SET LENGTH
886+         B      RJ
887+RJ2      LA     R0,2 .SET LENGTH
888+         B      RJ
889+RJ3      LA     R0,3 .SET LENGTH
890+         B      RJ
891+RJ4      LA     R0,4 .SET LENGTH
892+         B      RJ
893+RJ5      LA     R0,5 .SET LENGTH
894+         B      RJ
895+RJ6      LA     R0,6 .SET LENGTH
896+         B      RJ
897+RJ7      LA     R0,7 .SET LENGTH
898+         B      RJ
899+RJ8      LA     R0,8 .SET LENGTH
900+         B      RJ
```

Figure C-10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 13 of 22)

```
901+RJ9        LA     R0,9 .SET LENGTH
902+           B      RJ
903+RJ10       LA     R0,10 .SET LENGTH
904+           B      RJ
905+RJ11       LA     R0,11 .SET LENGTH
906+           B      RJ
907+RJ         ST     R7,RJSAVE .SAVE RETURN ADDRESS
908+           LA     R13,SAVE .PROGRAM SAVE AREA
909+           DC     CY(0)
910+           EXTRN  MODRJ1 .RIGHT JUSTIFY MODULE
911+           L      R15,=A(MODRJ1)
912+           BALR   R14,R15 .BRANCH TO RJ
913+           L      R7,RJSAVE .RESTORE RETURN ADDRESS
914+           LTR    R15,R15 .SET CONDITION CODE FOR ERRORS
915+           BR     R7 .RETURN TO CALL
916            Y$$DATE                          DATE VALIDATION
917+*
918+************** DATE VALIDATION *****************************************
919+*
920+DATCHKYM MVI    WORK1+4,C'0' .PLUG DAY = 1
921+           MVI    WORK1+5,C'1'
922+DATCHK     ST     R7,DVSAVE .SAVE RETURN ADDRESS
923+           LA     R1,WORK1
924+           BAL    R7,RJ6 .TEST FOR NUMERIC
925+           BNZ    DVOUT
926+           LTR    R7,R7 .SET CONDITION CODE
927+           CLC    WORK1(2),=C'70' .UNDER LOW YEAR?
928+           BL     DVOUT
929+           CLC    WORK1(2),=C'99' .OVER HIGH YEAR?
930+           BH     DVOUT
931+           CLC    WORK1+2(2),=C'01' .UNDER LOW MONTH?
932+           BL     DVOUT
933+           CLC    WORK1+2(2),=C'12' .OVER HIGH MONTH
934+           BH     DVOUT
935+           CLC    WORK1+4(2),=C'01' .UNDER LOW DAY?
936+           BL     DVOUT
937+           CLC    WORK1+4(2),=C'31' .OVER HIGH DAY?
938+           BH     DVOUT
939+           SR     R7,R7 .DATE OK
940+DVOUT      LTR    R7,R7 .SET CONDITION CODE
941+           L      R7,DVSAVE .RESTORE RETURN ADDRESS
942+           BR     R7
943            Y$$MVIN                            INPUT SCREEN FORMATING
944+*
945+************** MOVE IMA DATA TO SCREEN WORK AREA
946+*
947+MOVEIN     ST     R0,SCREEN# .SCREEN NUMBER
948+           MVC    IOKEY(4),SCREEN# .SCREEN NUMBER
949+           MVI    IOKEY+4,C'G' .GET
950+           MVI    IOFILE,C' '
951+           MVC    IOFILE+1(19),IOFILE .CLEAR TO SPACES
952+           MVC    IOFILE(13),=C'SCREEN FORMAT' .FILE NAME
953+           ZG#CALL MSGIN,(SCRNUM,IN$MSG)
954+           DS     0H
955+           LA     15,SCRNUM
956+           ST     15,PLIST+4*(1-1)
957+           LA     15,IN$MSG
958+           ST     15,PLIST+4*(2-1)
959+           OI     PLIST+4*(2-1),X'80'
960+           LA     1,PLIST
```

Figure C-10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 14 of 22)

```
+         L      15,=V(MSGIN)
+         BALR   14,15
+         LA     R9,ABTERM .I/O ERROR ADDRESS
+         B      IOSTATUS .CHECK I/O STATUS
          YS$MVOUT                         OUTPUT SCREEN FORMAT
+*
+*************** MOVE DATA FROM SCREEN WORK AREA TO OMA
+*
+MOVEOUT  ST     R0,SCREEN# .SCREEN NUMBER
+         MVC    IOKEY(4),SCREEN# .SCREEN NUMBER
+         MVI    IOKEY+4,C'P' .PUT
+         MVI    IOFILE,C' '
+         MVC    IOFILE+1(19),IOFILE .CLEAR TO SPACES
+         MVC    IOFILE(13),=C'SCREEN FORMAT' .FILE NAME
+         ZG#CALL MSGOUT,(SCRNUM,OUT$MSG,PDATA) .SCREEN AND DATA
+         DS     0H
+         LA     15,SCRNUM
+         ST     15,PLIST+4*(1-1)
+         LA     15,OUT$MSG
+         ST     15,PLIST+4*(2-1)
+         LA     15,PDATA
+         ST     15,PLIST+4*(3-1)
+         OI     PLIST+4*(3-1),X'80'
+         LA     1,PLIST
+         L      15,=V(MSGOUT)
+         BALR   14,15
+         B      YS$M0010
+MOVEOUTS ST     R0,SCREEN#
+         MVC    IOKEY(4),SCREEN# .SCREEN NUMBER
+         MVI    IOKEY+4,C'P' .PUT
+         MVI    IOFILE,C' '
+         MVC    IOFILE+1(19),IOFILE .CLEAR TO SPACES
+         MVC    IOFILE(13),=C'SCREEN FORMAT' .FILE NAME
+         ZG#CALL MSGOUT,(SCRNUM)         .SCREEN ONLY (NO DATA)
+         DS     0H
+         LA     15,SCRNUM
+         ST     15,PLIST+4*(1-1)
+         OI     PLIST+4*(1-1),X'80'
+         LA     1,PLIST
+         L      15,=V(MSGOUT)
+         BALR   14,15
+YS$M0010 LA     R9,ABTERM .I/O ERROR ADDRESS
+         B      IOSTATUS .CHECK I/O STATUS
 APCHKS   YS$SNAP                        SNAP DUMP
+*
+*************** SNAP DUMP OF ACTION PROGRAM ****************************
+*
+SNAPIT   ORG    *
+     ZG#CALL SNAP,(ZA#DPIB,EP,ZA#IMH,EI,WORK,EW,ZA#OMH,EO,ENAM.,YS$E)
+         DS     0H
+         LA     15,ZA#DPIB
+         ST     15,PLIST+4*(1-1)
+         LA     15,EP
+         ST     15,PLIST+4*(2-1)
+         LA     15,ZA#IMH
+         ST     15,PLIST+4*(3-1)
+         LA     15,EI
+         ST     15,PLIST+4*(4-1)
+         LA     15,WORK
+         ST     15,PLIST+4*(5-1)
```

Figure C-10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 15 of 22)

```
+          LA     15,EW
+          ST     15,PLIST+4*(6-1)
+          LA     15,ZA#OMH
+          ST     15,PLIST+4*(7-1)
+          LA     15,EO
+          ST     15,PLIST+4*(8-1)
+          LA     15,APCHKS
+          ST     15,PLIST+4*(9-1)
+          LA     15,Y$$E
+          ST     15,PLIST+4*(10-1)
+          OI     PLIST+4*(10-1),X'80'
+          LA     1,PLIST
+          L      15,=V(SNAP)
+          BALR   14,15
+          BR     R7 .RETURN REGISTER
 *******************************************************************************
 *                TERMINATION
 *******************************************************************************
           Y$$TERM
+*******************************************************************************
+*               PROGRAM TERMINATION                                           *
+*******************************************************************************
+TERM      CLI    ISNAP,C'N' .REQUEST NORMAL TERMINATION WITH SNAP?
+          BE     SNAP .YES
+          CLI    ISNAP,C'S' .REQUEST ABNORMAL TERMINATION WITH SNAP?
+          BNE    FINISH .NO-NORMAL TERMINATION
+ABTERM    MVI    ZA#PSIND,C'S' .TERMINATE WITH SNAP DUMP
+          B      FINISH
+SNAP      GETIME M
+SNAP      DS     0H
+          LA     1,1
+          SVC    7
+          ST     R1,ETIMS
+         ZG#CALL SNAP,(ZA#DPIB,EP,ZA#IMH,EI,WORK,EW,ZA#OMH,EO,Y$$B,Y$$E)
+          DS     0H
+          LA     15,ZA#DPIB
+          ST     15,PLIST+4*(1-1)
+          LA     15,EP
+          ST     15,PLIST+4*(2-1)
+          LA     15,ZA#IMH
+          ST     15,PLIST+4*(3-1)
+          LA     15,EI
+          ST     15,PLIST+4*(4-1)
+          LA     15,WORK
+          ST     15,PLIST+4*(5-1)
+          LA     15,EW
+          ST     15,PLIST+4*(6-1)
+          LA     15,ZA#OMH
+          ST     15,PLIST+4*(7-1)
+          LA     15,EO
+          ST     15,PLIST+4*(8-1)
+          LA     15,Y$$B
+          ST     15,PLIST+4*(9-1)
+          LA     15,Y$$E
+          ST     15,PLIST+4*(10-1)
+          OI     PLIST+4*(10-1),X'80'
+          LA     1,PLIST
+          L      15,=V(SNAP)
+          BALR   14,15
+FINISH    GETIME M
```

Figure C–10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 16 of 22)

```
+FINISH   DS     0H
+         LA     1,1
+         SVC    7
+         ST     R1,ETIMS .ENDING TIME
+         ZG#CALL RETURN        .RETURN CONTROL TO IMS
+         DS     0H
+         L      15,=V(RETURN)
+         BALR   14,15
          YS$MSG 1
+EMSG1    MVC    OMA(LMSG1),MSG1
+         MVC    ZA#OTL(2),=Y(0+LMSG1+4)
+         B      TERM
          YS$MSG 2
+EMSG2    MVC    OMA(LMSG2),MSG2
+         MVC    ZA#OTL(2),=Y(0+LMSG2+4)
+         B      TERM
          YS$MSG 3
+EMSG3    MVC    OMA(LMSG3),MSG3
+         MVC    ZA#OTL(2),=Y(0+LMSG3+4)
+         B      TERM
          YS$MSG 4,N
+EMSG4    MVC    OMA(LMSG4),MSG4
+         MVC    ZA#OTL(2),=Y(0+LMSG4+4)
          MVC     OMA+M4A-MSG4(15),KACCTPAY
          B       TERM
          YS$MSG 5
+EMSG5    MVC    OMA(LMSG5),MSG5
+         MVC    ZA#OTL(2),=Y(0+LMSG5+4)
+         B      TERM
          YS$MSG 6
+EMSG6    MVC    OMA(LMSG6),MSG6
+         MVC    ZA#OTL(2),=Y(0+LMSG6+4)
+         B      TERM
          YS$MSG 7
+EMSG7    MVC    OMA(LMSG7),MSG7
+         MVC    ZA#OTL(2),=Y(0+LMSG7+4)
+         B      TERM
          YS$MSG 8
+EMSG8    MVC    OMA(LMSG8),MSG8
+         MVC    ZA#OTL(2),=Y(0+LMSG8+4)
+         B      TERM
*****************************************************************************
*                CONSTANTS
*****************************************************************************
ACCTPAY  DC      C'ACCTPAY ACCOUNTS PAYABLE    '
TABLEMT  DC      C'TABLEMT SECURITY/CODES      '
VENDORM  DC      C'VENDORM VENDOR MASTER       '
PAYROLL  DC      C'PAYROLL PAYROLL MASTER      '
BLANKS   DC      CL80' '
*
************** MESSAGES
*
MSG1     DC      X'100A18011C'
         DC      C'PLEASE USE "TRANSMIT UNPROT DISPL" KEY TO RETRANSMIT'
         DC      X'1D10020000'
LMSG1    EQU     *-MSG1
MSG2     DC      X'100A18011C'
         DC      C'THE ACCOUNTS PAYABLE CONTROL RECORD CANNOT BE FOUND. '
         DC      C'PLEASE CONTACT ISD'
```

Figure C-10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 17 of 22)

```
                DC      X'1D10020000'
       LMSG2    EQU     *-MSG2
       MSG3     DC      X'100A0000'
                DC      C'A/P BATCH # '
       M3A      DC      CL4' '                          BATCH #
       M3B      DC      X'40212020612020612020'  DATE
                DC      CL5' '
       M3C      DC      CL3' '                          # OF CHECKS
                DC      C' CHECKS TOTALING $'
       M3D      DC      X'40206B2020206B2021204B202060'
                DC      X'10020000'
       LMSG3    EQU     *-MSG3
       DM3A     EQU     M3A-MSG3
       DM3B     EQU     M3B-MSG3
       DM3C     EQU     M3C-MSG3
       DM3D     EQU     M3D-MSG3
       MSG4     DC      X'100A18011C'
       M4A      DC      CL15' '
                DC      C'=THIS CHECK CANNOT BE FOUND. PLEASE CORRECT AND RETRY'
                DC      X'1D10020000'
       LMSG4    EQU     *-MSG4
       MSG5     DC      X'100A18011C'
                DC      C'ACTIVITY FOR THE PREVIOUS CHECK IS NOT COMPLETE'
                DC      X'1D10020000'
       LMSG5    EQU     *-MSG5
       MSG6     DC      X'100A18011C'
                DC      C'THIS CHECK IS ALREADY IN OUR FILE. '
                DC      C'PLEASE CORRECT AND RETRY'
                DC      X'1D10020000'
       LMSG6    EQU     *-MSG6
       MSG7     DC      X'100A18011C'
                DC      C'THE CURSOR WAS NOT IN THE EXPECTED POSITION. '
                DC      C'PLEASE CORRECT AND RETRY'
                DC      X'1D10020000'
       LMSG7    EQU     *-MSG7
       MSG8     DC      X'100A00001C'
                DC      C'THIS ACTION HAS BEEN TERMINATED BY OPERATOR REQUEST'
                DC      X'1D10020000'
       LMSG8    EQU     *-MSG8
       MSG9     DC      X'1C'
                DC      C'ITEMS TOTAL ='
       M9A      DC      X'40202020202021204B202060'
                DC      X'1D'
       LMSG9    EQU     *-MSG9
       DM9A     EQU     M9A-MSG9
       MSG10    DC      X'1C'
                DC      C'CASH NOT = 0'
                DC      X'1D'
       LMSG10   EQU     *-MSG10
       MSG11    DC      X'1C'
                DC      C'ACCRUAL NOT = 0'
                DC      X'1D'
       LMSG11   EQU     *-MSG11
       MSG12    DC      X'1C'
                DC      C'VOID CHECK REQUIRES OVERRIDE CHECK NUMBER'
                DC      X'1D'
       LMSG12   EQU     *-MSG12
                PRINT   GEN
                Y$$PIB                          .PROGRAM INFORMAION BLOCK
```

Figure C-10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 18 of 22)

```
1200+*********************************************************************************
1201+*                   LITERAL POOL                                                *
1202+*********************************************************************************
1203+          LTORG
1204+                 =V(GET)
1205+                 =V(GETUP)
1206+                 =V(PUT)
1207+                 =V(INSERT)
1208+                 =A(MODRJ1)
1209+                 =V(MSGIN)
1210+                 =V(MSGOUT)
1211+                 =V(SNAP)
1212+                 =V(RETURN)
1213+                 =Y(0+LYS$M1+4)
1214+                 =C'AP'
1215+                 =Y(0+IMA1)
1216+                 =C'ZBATCH'
1217+                 =Y(0+LMSG3+4)
1218+                 =Y(0+IMA3)
1219+                 =C'AC'
1220+                 =C'APITMS'
1221+                 =H'14'
1222+                 =C'APITS '
1223+                 =Y(0+LIOM2+4)
1224+                 =Y(0+LIOM1+4)
1225+                 =C'70'
1226+                 =C'99'
1227+                 =C'01'
1228+                 =C'12'
1229+                 =C'31'
1230+                 =Y(0+LMSG1+4)
1231+                 =Y(0+LMSG2+4)
1232+                 =Y(0+LMSG4+4)
1233+                 =Y(0+LMSG5+4)
1234+                 =Y(0+LMSG6+4)
1235+                 =Y(0+LMSG7+4)
1236+                 =Y(0+LMSG8+4)
1237+                 =C'APCHK'
1238+                 =C'T80'
1239+                 =C'   '
1240+                 =C'ADD'
1241+                 =P'1'
1242+                 =C'000'
1243+                 =C'00000'
1244+                 =P'0'
1245+                 =C'001'
1246+                 =C'SCREEN FORMAT'
1247+YSSE      EQU    * .END OF PROGRAM
1369 WORK      YSSWORK                         .WORK AREA
1370+*********************************************************************************
1371+*                   WORK AREA                                                   *
1372+*********************************************************************************
1373+WORK      DSECT
1374+STIMS     DS     A .START TIME (MILLISECONDS)
1375+ETIMS     DS     A .END TIME (MILLISECONDS)
1376+#GET      DS     H .NUMBER OF GET
1377+#GETUP    DS     H .          GETUP
1378+#PUT      DS     H .          PUT
1379+#INSERT   DS     H .          INSERT
1380+SAVE      DS     18F .PROGRAM SAVE AREA
```

Figure C–10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 19 of 22)

```
1381+PLIST     DS    4A .PARAMETER LIST FOR "CALLS"
1382+WHO       DS    CL3 .USER INITIALS
1383+WORK1     DS    2D .WORK FIELD
1384+PASSKEY   EQU   WORK1,5 .SECURITY RECORD FILE KEY
1385+IOFILE    DS    CL20 .LAST FILE I/O
1386+IOKEY     DS    CL20 .LAST FILE I/O KEY
1387+IOSTS     DS    CL4 .LAST FILE I/O STATUS
1388+IORET     DS    CL1 .FILE NOT AVAILABLE-RETURN
1389+ERR       DS    CL1 .ERROR FLAG
1390+YYMMDD    DS    CL6 .DATE
1391+HHMMSS    DS    CL6 .TIME
1392 RJSAVE    DS    A
1393 DVSAVE    DS    A
1394 TRAIL$    DS    CL26
1395 TRAIL$1   DS    A
1396 TRAIL$2   DS    A
1397           Y$$SWORK                          .SDMPS WORK AREA
1398+*
1399+************** SDMPS WORK AREA ************************************
1400+*
1401+SCRNUM    DS    D .SCREEN NUMBER
1402+SCREEN#   EQU   SCRNUM+4,4
1403+SCREENW   DS    CL180 .SCREEN WORK AREA
1404+MAXITL    EQU   SCREENW,2 .MAXIMUM INPUT TEXT LENGTH
1405+*
1406+************** SDMPS I/O AREAS
1407+*
1408+UDATA     EQU   *
1409+OUT$MSG   EQU   * .OUTPUT MESSAGE DATA
1410+FILL      DS    CL1 .OUTPUT FILL CHARACTER
1411+IN$MSG    EQU   * .INPUT MESSAGE DATA
1412 *
1413 ************** UNPROTECTED DATA
1414 *
1415 USTART    EQU   *
1416 UTRAN     DS    CL5                          TRANSACTION CODE
1417 USNAP     DS    CL1                          SNAP CODE
1418 IMA1      EQU   *-USTART
1419 UADD      DS    CL1                          ADD
1420 UCHG      DS    CL1                          CHANGE
1421 UEND      DS    CL1                          END
1422 UTYPE     DS    CL1                          CHECK TYPE
1423 UCHECK    DS    CL5                          CHECK NUMBER
1424 UTRAN1    DS    CL1
1425 IMA2      EQU   *-USTART
1426 UVENDOR   DS    CL5                          VENDOR CODE
1427 UTRAN2    DS    CL1
1428 IMA3      EQU   *-USTART
1429 ULEGEND   DS    CL25                         CHECK LEGEND
1430 UNAME     DS    CL26                         PAYEE NAME
1431 UADDR1    DS    CL25                         PAYEE ADDRESS LINE 1
1432 UADDR2    DS    CL25                         PAYEE ADDRESS LINE 2
1433 UCITY     DS    CL25                         PAYEE CITY AND STATE
1434 UZIP      DS    CL5                          PAYEE ZIP CODE
1435 UAMOUNT   DS    CL10                         CHECK AMOUNT
1436 UDATE     DS    CL6                          CHECK DATE (MMDDYY)
1437 UOVERIDE  DS    CL5                          OVERRIDE CHECK NUMBER
1438 UTRAN3    DS    CL1
1439 LUDATA    EQU   *-UDATA-1
1440 USTOP     DS    CL1
```

Figure C-10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 20 of 22)

```
 *
 ************** PROTECTED REPLACEMENT DATA
 *
 PDATA     EQU   *
 PSTART    EQU   *
 PADD      DS    CL1
 PCHG      DS    CL1
 PEND      DS    CL1
 PCHECK    DS    CL1
 PVENDOR   DS    CL1
 PLEGEND   DS    CL1
 PNAME     DS    CL1
 PADDR1    DS    CL1
 PADDR2    DS    CL1
 PCITY     DS    CL1
 PZIP      DS    CL1
 PAMOUNT   DS    CL1
 PDATE     DS    CL1
 POVERIDE  DS    CL1
 PMSG1     DS    CL80
 LPDATA    EQU   *-PSTART
 PSTOP     DS    CL1
 **********************************************************************
 *               RECORD AREAS
 **********************************************************************
           Y$$SY104                         .SECURITY RECORD
+*
+*************** TABLE MASTER RECORD
+*
+KTABLEMT  DS    CL6
+RTABLEMT  DS    CL80
+TABSTS    EQU   RTABLEMT+08,1 STATUS
+LIMIT     EQU   RTABLEMT+15,1 PASSWORD LIMIT
+TERMTAB   EQU   RTABLEMT+16 TERMINAL FIELDS
 *
 ******** AP002 VENDOR MASTER
 *
 KVENDORM  DS    CL5
 RVENDORM  DS    CL199
 VMNAME    EQU   RVENDORM+5,26              NAME
 VMADDR1   EQU   RVENDORM+31,25             ADDRESS 1
 VMADDR2   EQU   RVENDORM+57,25             ADDRESS 2
 VMCITY    EQU   RVENDORM+83,25             CITY
 VMZIP     EQU   RVENDORM+109,5             ZIP CODE
 *
 ******** PE010 PERSONNEL MASTER
 *
 KPAYROLL  DS    CL5
 RPAYROLL  DS    CL421
 PMNAME    EQU   RPAYROLL+12,26             NAME
 PMADDR1   EQU   RPAYROLL+41,25             ADDRESS
 PMCITY    EQU   RPAYROLL+70,25             CITY
 PMZIP     EQU   RPAYROLL+99,5              ZIP CODE
 PMBRW     EQU   RPAYROLL+200,3             BRANCH OF WORK
 *
 ************** ACCOUNTS PAYABLE
 *
 KACCTPAY  DS    CL15
 RACCTPAY  DS    CL165
```

Figure C-10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 21 of 22)

```
*
*         AP100 HEADER
*
APAMT     EQU   RACCTPAY+29,5
HACCTPAY  DS    CL165
APHRID    EQU   HACCTPAY,2
APHREPT   EQU   HACCTPAY+16,5        PD2  REPORT TOTAL
APHBATCH  EQU   HACCTPAY+21,5        PD2  BATCH TOTAL
APHCHKCT  EQU   HACCTPAY+26,5             CHECK COUNTER
APHTYPE   EQU   HACCTPAY+31,1             CHECK TYPE
APHCHECK  EQU   HACCTPAY+32,5             CHECK NUMBER
APHDATE   EQU   HACCTPAY+37,6             CHECK DATE
APHVENDR  EQU   HACCTPAY+43,5             CHECK VENDOR
APHITMT   EQU   HACCTPAY+53,5        PD2  ITEM TOTAL
APHITMC   EQU   HACCTPAY+58,3             ITEM COUNT
APHAMT    EQU   HACCTPAY+48,5        PD2  CHECK AMT
APHNAME   EQU   HACCTPAY+61,26            NAME
APHLEGND  EQU   HACCTPAY+87,26            LEGEND
APHPRNT   EQU   HACCTPAY+113,1            PRINT
APHBATHN  EQU   HACCTPAY+114,3            BATCH NUMBER
APHCHKS   EQU   HACCTPAY+117,3            NUMBER OF CHECKS
APHVODS   EQU   HACCTPAY+120,3            NUMBER OF VOIDS
APHERRS   EQU   HACCTPAY+123,3            NUMBER OF ERROR PASSES
APHITMS   EQU   HACCTPAY+126,4            NUMBER OF ITEMS
APHOLD    EQU   HACCTPAY+130,5       PD2  OLD CHECK AMOUNT
APHCASH   EQU   HACCTPAY+135,5            CASH TOTAL
APHACCR   EQU   HACCTPAY+140,5            ACCRUAL TOTAL
APHERR    EQU   HACCTPAY+145,1            ERROR CODE
APHAOC    EQU   HACCTPAY+146,1            ADD OR CHANGE
APHDONE   EQU   HACCTPAY+147,1            COMPLETION
*
********  AP103 CHECK
*
CACCTPAY  DS    CL165
APCRID    EQU   CACCTPAY,2                "AC"
APCTYPE   EQU   CACCTPAY+2,1              TYPE
APCCHECK  EQU   CACCTPAY+3,5              CHECK NUMBER
APCTDATE  EQU   CACCTPAY+16,4        PD0  TRANSACTION DATE
APCDATE   EQU   CACCTPAY+20,4        PD0  DATE
APCVENDR  EQU   CACCTPAY+24,5             VENDOR
APCAMT    EQU   CACCTPAY+29,5        PD2  AMOUNT
APCNAME   EQU   CACCTPAY+34,26            NAME
APCADDR1  EQU   CACCTPAY+60,25            ADDRESS 1
APCADDR2  EQU   CACCTPAY+85,25            ADDRESS 2
APCCITY   EQU   CACCTPAY+110,26           CITY
APCZIP    EQU   CACCTPAY+136,3       PD0  ZIP CODE
APCLEGND  EQU   CACCTPAY+139,25           LEGEND
APCPRNT   EQU   CACCTPAY+164,1            PRINT
OMA       Y$$OMA 2568                     .OUTPUT MESSGE AREA
+EW       EQU   * .END OF WORK AREA
CDA       Y$$CDA                          .CONTINUITY DATA AREA
+*************************************************************************
+*              CONTINUITY DATA AREA                                      *
+*************************************************************************
+CDA      DSECT
+         DS    0H
          END
```

Figure C-10. APCHKS Action Program Processing a Dialog Transaction with Delayed Internal Succession (Part 22 of 22)

DIALOG TRANSACTION IN BAL: APTIMS PROGRAM

```
LINE   SOURCE STATEMENT                                              OS/3 ASM

APITMS    START 0
*************************************************************************
*                 AUTHOR : R L LEONARD
*                 DATE   : 28 MARCH 1980
*                 SITE   : GAY & TAYLOR INC.,WINSTON-SALEM,NC,27102
*                 PURPOSE: TO ENTER AND VERIFY ITEM CHARGES FROM AP CHECKS
*                 CHANGE LOG:
*************************************************************************
          Y$$START                              .STARTING CONVENTIONS
+Y$$B     EQU    * .START OF PROGRAM
+*
+**************** REGISTER EQUATES
+*
+R0       EQU    0
+R1       EQU    1
+R2       EQU    2 .PIB COVER
+R3       EQU    3 .IMA COVER
+R4       EQU    4 .WORK COVER
+R5       EQU    5 .OMA COVER
+R6       EQU    6 .CDA COVER
+R7       EQU    7 .INTERNAL ROUTINE LINKAGE
+R8       EQU    8 .I/O - NORMAL RETURN ADDRESS
+R9       EQU    9 .I/O - ERROR RETURN ADDRESS
+R10      EQU    10 .PROGRAM COVER #3
+R11      EQU    11 .PROGRAM COVER #2
+R12      EQU    12 .PROGRAM COVER #1
+R13      EQU    13
+R14      EQU    14
+R15      EQU    15
+*
+**************** ESTABLISH PROGRAM COVERING
+*
+         USING *,R12,R11,R10 .PROGRAM CODE
+         USING ZA#DPIB,R2 .PIB
+         USING ZA#IMH,R3 .IMA
+         USING WORK,R4 .WORK
+         USING ZA#OMH,R5 .OMA
+         USING CDA,R6 .CDA
+*
+**************** ESTABLISH IMS INTERFACE
+*
+         STM    R14,R12,12(R13) .STORE REG IN CALLS' SAVE AREA
+         LR     R12,R15 .ADDRESS OF THIS PROGRAM
+         LM     R2,R6,0(R1) .ACTIVATION AREAS FROM PARAM
+         LA     R11,SAVE .THIS PROGRAM SAVE AREA
+         ST     R11,8(,R13) .PUT THIS SAVE INTO CALLS' SAVE
+         ST     R13,4(,R11) .PUT CALLS' SAVE INTO THIS SAVE
+         LR     R13,R11 .REG 13 = THIS SAVE AREA
+         LR     R11,R12 .SECOND PROGRAM COVER
+         LA     R11,1(R11)
+         LA     R11,4095(R11)
+         LR     R10,R11 .THIRD PROGRAM COVER
+         LA     R10,1(R10)
+         LA     R10,4095(R10)
+         GETIME M
+         DS     0H
+         LA     1,1
+         SVC    7
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 1 of 29)

```
 61+          ST     R1,STIM$ .STARTUP TIME
 63           DROP   R6                      .NO CDA
 64           PRINT  GEN
 65           BAL    R7,DAYTIME              .GET DATE-TIME
 66                                                    Y$$TRAIL A
 67+          PRINT  OFF
 77+          PRINT  ON
 78           MVC    PASSKEY(5),=C'APCHK'
 79           Y$$SECUR                       .PASSWORD SECURITY
 80+*************************************************************************
 81+*                CHECK SECURITY FOR OPEN APPLICATION                     *
 82+*
 83+*                        ASSUMES KEY IN FIELD "PASSKEY"                  *
 84+*************************************************************************
 85+          MVC    KTABLEMT(3),=C'T80'
 86+          MVC    KTABLEMT+3(5),PASSKEY
 87+          LA     R9,Y$$0020 .NO FIND ADDRESS
 88+          BAL    R8,GTABLEMT .GET SECURITY RECORD
 89+          CLI    TABSTS,C' ' .RECORD ACTIVE?
 90+          BNE    Y$$0020 .NO
 91+          MVI    WORK1,X'00' .SETUP TO CVB
 92+          MVC    WORK1+1(7),WORK1
 93+          MVC    WORK1+8(2),ZA#ISTID+2 .TERMINAL ID
 94+          PACK   WORK1+6(2),WORK1+8(2)
 95+          CVB    R1,WORK1 .TERMINAL FIELD COUNTER
 96+          LA     R7,TERMTAB-4 .BEGINNING OF TERMINAL FIELDS
 97+Y$$C010   LA     R7,4(R7) .NEXT TERMINAL FIELDS
 98+          BCT    R1,Y$$C010 .COUNT DOWN TO THIS TERMINAL
 99+          CLC    0(3,R7),=C'   ' .OPEN?
100+          BE     Y$$0020 .NO
101+          MVC    WHO(3),0(R7) .SAVE USER INITIALS
102+          CLC    3(1,R7),LIMIT .OPEN BUT OVER LIMIT (SET DOWN)
103+          BNH    Y$$0030 .NO
104+Y$$C020   MVC    OMA(LY$$M1),Y$$M1 .APPLICATION NOT OPEN
105+          MVC    ZA#OTL(2),=Y(0+LY$$M1+4) .MESSAGE LENGTH
106+          B      TERM
107+Y$$M1     DC     X'100A18011C'
108+          DC     C'APPLICATION NOT OPEN'
109+          DC     X'1010020003'
110+LY$$M1    EQU    *-Y$$M1
111+Y$$0030   ORG    *
113           MVC    KACCTPAY(15),BLANKS
114           CLC    ZA#ITL(2),=Y(IMA1-USTART)
115           BNH    L0020
116           CLC    IPROT(5),=C'A   P'
117           BE     EMSG1                    USE UNPROT
118 L0020     EQU    *
119           CLC    ZA#ITL(2),=Y(UACCT1-USTART+1) DATA ENTERED?
120           BNH    L0030                    NO
121           Y$$IN  12                      .GET INPUT DATA
122+          LA     R0,12 .SCREEN NUMBER
123+          BAL    R8,MOVEIN .GO TO INPUT SCREEN ROUTINE
124 L0030     MVI    FILL,C' '               UNPROTECTED FILL CHARACTER
125           MVI    PSTART,C' '
126           MVC    PSTART+1(PSTOP-PSTART-1),PSTART  CLEAR PROT REPLACE
127           MVI    USTOP,X'FF'
128           MVI    PSTOP,X'FF'
129           CLC    IMA+4(5),=C'APRNT'      .PRINT?
130           BNE    L0040                   .YES PRINT CHECK
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 2 of 29)

DIALOG TRANSACTION IN BAL: APTIMS PROGRAM

```
*****                                                         Y$$TRAIL B
          MVC   KACCTPAY(2),=C'AC'
          MVI   KACCTPAY+2,C'N'
          MVC   KACCTPAY+3(5),IMA+10      CHECK NUMBER
          CLI   IMA+15,C'V'               VOID CHECK?
          BNE   L0035                     NO
          MVI   KACCTPAY+2,C'V'
L0035     B     L0610
L0040     MVC   KACCTPAY(2),=C'AP'        GET HEADER
          LA    R9,EMSG2
          BAL   R8,GACCTPAY
          MVC   ACCTPAYH(165),RACCTPAY    STORE HEADER
**********************************************************************
*               BUILD BASE SCREEN
**********************************************************************
*
*         CHECK DATA
*
          CLI   HTYPE,C'N'              NEW CHECK?
          BE    L0050                   YES
          MVC   PTYPE(1),HTYPE
L0050     MVC   PCHECK(5),HCHECK
          MVC   PCAMT(14),=X'40206B2020206B2021204B202060'
          ED    PCAMT(14),HAMOUNT
          MVC   PCNAME(25),HNAME
*
*         LINE NUMBERS
*
          LA    R10,PLIN#1              FIRST LINE # POSITION
          LA    R6,20                  COUNTER
          PACK  WORK1(2),HITMCNT(3)
L0060     UNPK  0(3,R10),WORK1(2)       MOVE INTO LINE # POSITION
          OI    2(R10),X'FC'             FIX SIGN
          AP    WORK1(2),=P'1'           NEXT ITEM
          LA    R10,PLLINE(,R10)        NEXT LINE
          BCT   R6,L0060
          CLC   ZA#ITL(2),=Y(UACCT1-USTART+1) VERIFY DATA?
          BNL   L0120                    YES
          CLI   HACTION,C'C'             CHANGE?
          BE    L0080                    YES
*
******** ADD SCREEN
*
*****                                                         Y$$TRAIL D
          MVC   UDESPT1(26),HLEGEND
          B     L9000                    SCREEN OUT
*
******** CHANGE SCREEN (GET ITEMS FOR DISPLAY)
*
L0080     LA    R6,20                   LINE COUNTER
          LA    R10,UACCT1               FIRST LINE
                                                              Y$$TRAIL E
          PRINT OFF
          PRINT ON
          MVC   KACCTPAY(15),BLANKS
          MVC   KACCTPAY(2),=C'AI'
          MVC   KACCTPAY+2(6),HTYPE
          MVC   KACCTPAY+8(3),HITMCNT
          MVI   POSITION,C'G'
          LA    R9,EMSG3
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 3 of 29)

```
               BAL    R8,SACCTPAY               SET START OF FILE
      L0090    LA     R9,EMSG4
               BAL    R8,NACCTPAY             READ NEXT AP ITEM
               MVC    ACCTPAYI(165),RACCTPAY  MOVE TO ITEM AREA
               CLC    RACCTPAY+2(6),HTYPE     SAME CHECK?
               BNE    L9000                   FORMAT SCREEN AND OUT
               MVC    0(8,R10),AIACCT
               UNPK   DAMT(10,R10),AIAMT(5)
               CP     AIAMT(5),=P'0'
               BNL    L0100
               MVI    DAMT(R10),C'-'
               OI     DAMT+9(R10),X'F0'
      L0100    MVC    DDESPT(30,R10),AIDESCPT
               MVC    DEMP(4,R10),AIEMP
               LA     R10,ULLINE(,R10)
               BCT    R6,L0090
               B      L9000                   FORMAT SCREEN AND OUT
      ***********************************************************************
      *               VERIFY LINE ITEMS
      ***********************************************************************
      L0120    LA     R6,20                   LINE COUNTER
               LA     R10,PLIN#1
                                                               Y$STRAIL F
               PRINT  OFF
               PRINT  ON
               ST     R10,PROLINE             PROTECTED POINTER
               LA     R10,UACCT1              UNPROTECTED POINTER
               MVC    BRCH(4),UACCT1
               MVC    DESCPTH(30),UDESPT1
      L0130    CLC    0(ULLINE,R10),BLANKS    THIS LINE BLANK?
               BE     L0320                   YES-CHECK FOR ERRORS
                                                               Y$STRAIL G
               PRINT  OFF
               PRINT  ON
               MVC    LAST(1),DXMIT(R10)      SAVE LAST ITEM FLAG
               CLI    DXMIT(R10),C'R'         REVIEW?
               BNE    L0132                   NO
               MVC    REVIEW(1),DXMIT(R10)    SAVE REVIEW REQUEST
      L0132    EQU    *
               CLC    0(4,R10),BLANKS         NO BRANCH?
               BNE    L0140
               MVC    0(4,R10),BRCH           MOVE HOLD BRANCH
      L0140    EQU    *
      *
      *               CHECK ACCOUNT NUMBER
      *
                                                               Y$STRAIL H
               PRINT  OFF
               PRINT  ON
               MVC    KACCTMST(8),0(R10)      HIT ACCOUNT MASTER
      *        ACCOUNT MASTER FILE
               MVC    KACCOUNT(8),0(R10)
               LA     R8,L0150
               BAL    R9,GACCTMST
               MVI    ERR,C'Y'
               L      R9,PROLINE
               OI     DCACCT(R9),X'C1'        ACCT MST ERROR CODE
               MVI    DBACCT(R9),X'1C'
               B      L0200                   CHECK AMOUNT
      *        BRANCH MASTER FILE
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 4 of 29)

```
L0150    CLI    0(R10),C'0'                BRANCH ACCOUNT?
         BNE    L0180                      NO, GENERAL LEDGER ACCOUNT
         MVC    KBRANCHM(3),1(R10)
         MVC    BRCH(4),0(R10)             SAVE ACCOUNT CODE
         LA     R8,L0160                   GET BRANCH
         BAL    R9,GBRANCHM
         MVI    ERR,C'Y'
         L      R9,PROLINE
         OI     DCACCT(R9),X'C2'           BRANCH ERROR CODE
         MVI    DBACCT(R9),X'1C'
*        CHART  OF ACCOUNTS FILE
L0160    MVC    KACCOUNT(4),=C'0000'        BRANCH ACCOUNT
L0180    EQU    *
         LA     R8,L0190
         BAL    R9,GACCOUNT                GET CHART OF ACCOUNTS
         MVI    ERR,C'Y'
         L      R9,PROLINE
         OI     DCACCT(R9),X'C4'            ACCOUNT ERROR CODE
         MVI    DBACCT(R9),X'1C'
         B      L0200                       CHECK AMOUNT
L0190    MVC    INCOME(1),CAINC
         MVC    EXPENS(1),CAEXP
         L      R9,PROLINE
         MVC    DPCOA(1,R9),CACOA           CASH/ACCRUAL CODE
*
*        AMOUNT
*
L0200    MVC    WORK1+5(10),DAMT(R10)       SAVE FIELD
         LA     R1,WORK1+5
                                                           YS$TRAIL I
         PRINT  OFF
         PRINT  ON
         BAL    R7,RJ10
         BZ     L0220
         MVI    ERR,C'Y'
         L      R9,PROLINE
         MVI    DBAMT(R9),X'1C'
L0220    PACK   WORK1(5),WORK1+6(9)
         CLI    CACOA,C'C'                  CASH ACCOUNT
         BE     L0240
         AP     HACCR(5),WORK1(5)
         B      L0260
L0240    AP     HCASH(5),WORK1(5)
L0260    EQU    *
*
*        DESCRIPTION
*
                                                           YS$TRAIL J
         PRINT  OFF
         PRINT  ON
         CLC    DDESPT(30,R10),BLANKS
         BNE    L0265
         MVC    DDESPT(30,R10),DESCPTH      DUP LAST DESCRIPTION
L0265    MVC    DESCPTH(30),DDESPT(R10)     SAVE LAST DESCRIPTION
*
*        EXPENSE/INCOME EMPLOYEE NUMBER
*
L0270    L      R9,PROLINE
         CLI    INCOME,C' '                INCOME ACCOUNT?
         BE     L0280                      NO
         CLC    DEMP(4,R10),BLANKS          ANY EMPLOYEE #?
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 5 of 29)

```
              BNE     L0290                       YES
              OI      DCEMP(R9),X'D4'             INCOME AND NO EMP #
              B       L0298                       FLAG ERROR
     L0280    EQU     *
              CLC     DEMP(4,R10),BLANKS
              BE      L0315                       NO EMP #
     L0290    EQU     *
              MVC     KPAYROLL(4),DEMP(R10)
              MVI     KPAYROLL+4,C'0'
              LA      R8,L0300
              BAL     R9,GPAYROLL                 GT EMPLOYEE
     L0292    L       R9,PROLINE
              OI      DCEMP(R9),X'D1'             EMP NOT FOUND
              B       L0298                       FLAG ERROR
     L0298    MVI     ERR,C'Y'
              L       R9,PROLINE
              MVI     DBEMP(R9),X'1C'
              B       L0315
     L0300    CLI     INCOME,C' '                 INCOME ACCOUNT?
              BNE     L0315                       YES-DO NOT NEED EXPENSE CAL
              MVC     KTABLEMT(8),BLANKS
              MVC     KTABLEMT(3),=C'T10'
              MVC     KTABLEMT+3(3),PMCAL
              LA      R8,L0310
              BAL     R9,GTABLEMT                 GET CLASSIFICATION
              L       R9,PROLINE
              OI      DCEMP(R9),X'D2'             CLAS NOT FOUND
              B       L0298
     L0310    CLI     TMEXP,C' '                  EPENSE CLASS?
              BNE     L0315                       YES-OK
              L       R9,PROLINE
              OI      DCEMP(R9),X'D4'             NOT EXP EMP
              B       L0298                       FLAG ERROR
     L0315    EQU     *
     *
     *        SETUP FOR NEXT LINE
     *
                                                                YS$TRAIL K
              PRINT   OFF
              PRINT   ON
              LA      R10,ULLINE(,R10)            NEXT UNPROTECT LINE
              L       R9,PROLINE
              LA      R9,PLLINE(,R9)              NEXT PROTECT LINE
              ST      R9,PROLINE
              CLI     LAST,C'Y'                   LAST ITEM?
              BE      L0320                       YES
              BCT     R6,L0130                    NEXT LINE
     ****************************************************************************
     *        ADD/UPDATE LINE ITEMS
     ****************************************************************************
     L0320    EQU     *
                                                                YS$TRAIL L
              PRINT   OFF
              PRINT   ON
              LA      R6,20
              LA      R10,PLIN#1                  PROT DATA
              ST      R10,PROLINE                 SAVE ADDRESS
              LA      R10,UACC11
     L0325    CLI     ERR,C'Y'                     ANY ERRORS?
              BE      L9000                        FORMAT AND OUT
              MVC     LAST(1),DXMIT(R10)
     *
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 6 of 29)

```
*************** BUILD RECORD
*
          MVC   KACCTPAY(2),=C'AI'
          MVC   KACCTPAY+2(6),HTYPE
          MVC   KACCTPAY+8(3),HITMCNT
          CLC   0(ULLINE,R10),BLANKS      ITEM?
          BNE   L0328                     NO
          MVI   LAST,C'Y'                SET LAST ITEM
          B     L0400
L0328     MVI   ACCTPAYI,C' '
          MVC   ACCTPAYI+1(164),ACCTPAYI
          MVC   AIRID(2),=C'AI'
          MVC   AITYPE(6),HTYPE
          MVC   AICNT(3),HITMCNT
          MVC   AIVENDOR(5),HVENDOR
          MVC   AIACCT(8),0(R10)
          MVC   WORK1(10),DAMT(R10)
          LA    R1,WORK1
          BAL   R7,RJ10
          BZ    L0330
          MVI   ERR,C'Y'
          L     R9,PROLINE
          OI    DBAMT(R9),X'10'
          B     L0331
L0330     PACK  AIAMT(5),WORK1(10)
L0331     MVC   AIEMP(4),DEMP(R10)
          MVC   AIDESCPT(30),DDESPT(R10)
          L     R9,PROLINE
          MVC   AICOA(1),DPCOA(R9)        CASH/ACCRUAL CODE
          CLI   HACTION,C'C'              CHANGE?
          BE    L0340                     YES
*
*************** ADD RECORD
*
L0335     MVC   AIBATCH(3),HBATCH         BATCH #
          MVC   RACCTPAY(165),ACCTPAYI
          LA    R8,L0390
          BAL   R9,IACCTPAY
          MVI   ZA#PLRI,C'0'              ROLLBACK UPDATES
          MVI   ERR,C'Y'
          L     R9,PROLINE
          MVI   2(R9),X'1C'
          B     L0390
*
*************** UPDATE RECORD
*
L0340     MVC   AIERR(3),HBATCH           CORRECTION BATCH #
          LA    R8,L0380
          BAL   R9,UACCTPAY
                                                   YSSTRAIL M
          PRINT OFF
          PRINT ON
          B     L0335                     ADDING ITEM ON CHANGE
L0360     MVI   ZA#PLRI,C'0'              ROLLBACK UPDATES
          L     R9,PROLINE
          MVI   2(R9),X'1C'
          MVI   ERR,C'Y'
          B     L0390
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 7 of 29)

```
L0360     MVC    RACCTPAY(165),ACCTPAYI
                                                        Y$$TRAIL N
          PRINT  OFF
          PRINT  ON
          LA     R9,L0360
          BAL    R8,PACCTPAY
*
*         UPDATE HEADER DATA
*
L0390     PACK   WORK1(2),HITMCNT(3)
          AP     WORK1(2),=P'1'
          UNPK   HITMCNT(3),WORK1(2)
          OI     HITMCNT+2,X'F0'        FIX SIGN
          AP     HITMTOT(5),AIAMT(5)
          PACK   WORK1(3),HITEMS(4)
          AP     WORK1(3),=P'1'
          UNPK   HITEMS(4),WORK1(3)     NUMBER OF ITEMS
          OI     HITEMS+3,X'F0'         FIX SIGN
          LA     R10,ULLINE(,R10)
          L      R9,PROLINE
          LA     R9,PLLINE(,R9)
          ST     R9,PROLINE
          CLI    LAST,C'Y'              LAST ITEM
          BE     L0400
          BCT    R6,L0325
***********************************************************************
*         SETUP NEXT ACTION
***********************************************************************
L0400     CLI    ERR,C'Y'
          BE     L9000                   FORMAT AND OUT
                                                        Y$$TRAIL O
          PRINT  OFF
          PRINT  ON
          MVI    HERRCDE,C' '
          CP     HITMTOT(5),HAMOUNT(5)
          BE     L0420
          OI     HERRCDE,X'F1'          ITEM TOTAL NOT = CHECK
L0420     CP     HCASH(5),=P'0'
          BE     L0440
          OI     HERRCDE,X'F2'          CSH NOT = 0
L0440     CP     HACCR(5),=P'0'
          BE     L0460
          OI     HERRCDE,X'F4'          ACCRUAL NOT = 0
*
******** DETERMINE SUCCESSOR
*
L0460     CLI    LAST,C'Y'             LAST ITEM?
          BE     L0480                 YES
          MVI    ZA#PSIND,C'D'           EXPECT MORE ITEM NEXT SCREEN
          MVC    ZA#PSID(6),=C'APITMS'
          MVC    OMA+4(6),=C'APITS '   TRANSACTION CODE
          MVC    ZA#OTL(2),=H'14'      LENGTH
          B      L0520
L0480     CLI    HERRCDE,C' '
          BE     L0500
          MVI    ZA#PSIND,C'D'           BALANCE ERRORS-CORRECT CHECK
          MVC    ZA#PSID(6),=C'APCHKS'
          MVC    ZA#OTL(2),=Y(0+LMSG11+8)  LENGTH
          MVC    OMA+4(LMSG11),MSG11     APCKS TRANSACTION
          MVC    OMA+4+DM11A(1),HTYPE
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 8 of 29)

DIALOG TRANSACTION IN BAL: APTIMS PROGRAM

```
             MVC    OMA+4+DM11B(5),HCHECK
             MVC    HITMCNT(3),=C'001'
             MVI    HACTION,C'C'            CHANGE
             MVI    HCOMPL,C' '
             B      L0520
*
*
*
L0500        EQU    *
             AP     HBATOT(5),HAMOUNT(5)   ADD CHECK TO BATCH TOTAL
             CLI    HACTION,C'C'             CHANGE?
             BNE    L0505                    NO
             SP     HBATOT(5),HOLD(5)        CORRECT FOR PREVIOUS AMOUNT
L0505        CLI    REVIEW,C'R'              REVIEW ITEMS?
             BNE    L0510                    NO
             MVI    ZA#PSIND,C'D'           DELAYED INTERNAL SUCCESSION
             MVC    ZA#PSID(6),=C'APAUDT'
             MVC    OMA+4(LMSG12),MSG12       MESSAGE FOR APAUD
             MVC    OMA+4+DM12A(1),HTYPE      CHECK TYPE
             MVC    OMA+4+DM12B(5),HCHECK     CHECK #
             MVC    ZA#OTL(2),=Y(0+LMSG12+8) LENGTH
             B      L0515
L0510        CLI    HPRINT,C'N'
             BNE    L0515
L0512        MVI    ZA#PSIND,C'D'
             MVC    ZA#PSID(6),=C'APCHKS'
             MVC    OMA+4(6),=C'APCKS '   TRANSACTION CODE
             MVC    ZA#OTL(2),=H'14'      LENGTH
L0515        CLI    HACTION,C'A'             ADD/
             BNE    L0518
             PACK   WORK1(2),HCHKS(3)        ADD 1 TO # OF CHECKS
             AP     WORK1(2),=P'1'
             UNPK   HCHKS(3),WORK1(2)
             OI     HCHKS+2,X'F0'
L0518        MVI    HCOMPL,C'C'              COMPLETE
L0520        MVC    KACCTPAY(15),BLANKS
             MVC    KACCTPAY(2),=C'AP'
             LA     R9,EMSG2
             BAL    R8,UACCTPAY
             MVC    RACCTPAY(165),ACCTPAYH
             LA     R9,EMSG2
             BAL    R8,PACCTPAY
L0540        CLI    ZA#PSIND,C'N'             SUCCESSOR?
             BNE    TERM                      YES-TERM
*
**************** CHECK AMOUNT TRANSLATION
*
*
L0600        MVC    KACCTPAY(15),BLANKS       SETUP CHECK PRINT
             MVC    KACCTPAY(2),=C'AC'
             MVC    KACCTPAY+2(6),HTYPE
L0610        LA     R9,EMSG5                  NOT FOUND
             BAL    R8,GACCTPAY               GET CHECK
             MVC    ACCTPAYC(165),RACCTPAY    MOVE TO CHECK AREA
                                                           YS$TRAIL P
    +        PRINT  OFF
    +        PRINT  ON
             CLI    CPRINT,C'N'              NO PRINT?
             BE     L0510
             CP     CAMOUNT(5),=P'0'         NEGATIVE OR ZERO CHECK?
```

Figure C–11. APITMS Action Program Processing a Dialog (Part 9 of 29)

```
          BNH    L0510
          B      L0620
          DC     0Y(0)
          EXTRN  CKGD50
          ENTRY  NMERA
          ENTRY  ALP1
          ENTRY  ALP2
          ENTRY  IDN
NMERA     DC     PL8'0'                       AMOUNT
ALP1      DC     CL50' '                      LINE 1
ALP2      DC     CL50' '                      LINE 2
IDN       DC     C'2'
          DS     0F
L0620     EQU    *
          ZAP    NMERA(8),CAMOUNT(5)
          L      R15,=A(CKGD50)
          BALR   R14,R15
          LTR    R15,R15
          BNZ    EMSG7                        ERRORS
          MVC    PAY10(50),ALP1
          MVC    PAY20(50),ALP2
          MVC    LEGENDO(25),CLEGEND
          MVC    VENDORO(5),CVENDOR
          MVC    CHECKO(5),KACCTPAY+3
          MVC    NAMEO(26),CNAME
          MVC    ADDR10(25),CADDR1
          UNPK   WORK1(7),CDATE(4)
          MVC    DATEO(6),WORK1+1
          MVC    WORK1(14),=X'5C206B2020206B2021204B202060'
          ED     WORK1(14),CAMOUNT
          MVC    AMOUNTO(13),WORK1+1
          CLI    AMOUNTO+12,C'*'            * FROM EDIT
          BNE    L0630                      NO-LEAVE IT
          MVI    AMOUNTO+12,C' '            BLANK IT
L0630     CLC    CADDR2(25),BLANKS          ADDRESS 2?
          BE     L0640                      NO
          MVC    ADDR20(25),CADDR2
          MVC    CITYO(25),CCITY
          MVI    CITYO+18,C' '
          B      L0660
L0640     MVC    ADDR20(25),CCITY
          MVC    CITYO(25),BLANKS
L0660     CP     CZIP(3),=P'0'              ZIP CODE?
          BE     L0680
          UNPK   CITYO+19(5),CZIP(3)
L0680     EQU    *
          MVI    CITYO+25,X'0C'             FORM FEED(TOP OF PAGE)
          MVI    CITYO+26,X'FF'
          MVC    UTRAN(6),=C'APCKS '        TRANSACTION CODE
          MVC    UTRAN+6(4),BLANKS
          MVI    UTRAN+10,X'FF'
          MVI    PSTART,C':'
          MVC    PSTART+1(4),PSTART
          MVI    FILL,C' '
          Y$$OUT 13
          LA     R0,13 .SCREEN NUMBER
          BAL    R8,MOVEOUT .SCREEN AND DATA
          B      TERM
********************************************************************
*                  OUTPUT SCREEN
********************************************************************
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 10 of 29)

```
721 L9000     EQU    *
722                                                          YS$TRAIL Q
723+          PRINT  OFF
733+          PRINT  ON
734           YS$OUT 12
735+          LA     R0,12 .SCREEN NUMBER
736+          BAL    R8,MOVEOUT .SCREEN AND DATA
737                                                          YS$TRAIL R
738+          PRINT  OFF
748+          PRINT  ON
749           B      TERM
750           YS$IOSTS                        .I/O STATUS
752+*********************************************************************
753+*                INTERNAL ROUTINES                                   *
754+*********************************************************************
755+*
756+************** CHECK FILE I/O STATUS
757+*
758+IOSTATUS ORG    *
759+          CLI    ZA#PSC+1,0 .SUCCESSFUL?
760+          BNE    YS$IOS05 .NO
761+          MVI    IOKEY,C' ' .CLEAR KEY
762+          MVC    IOKEY+1(14),IOKEY
763+          BR     R8
764+YS$IOS05 CLI    ZA#PSC+1,1 .INVALID KEY?
765+          BER    R9
766+          CLI    ZA#PDSC+1,5 .FILE NOT DEFINED?
767+          BE     YS$IOS10
768+          CLI    ZA#PDSC+1,6 .FILE CLOSED?
769+          BNE    YS$IOS30
770+YS$IOS10 CLI    IORET,C'Y' .RETURN ON FILE NOT AVAILABLE?
771+          BNE    YS$IOS20
772+          SR     R8,R8 .FLAG FOR FILE NOT AVAILABLE
773+          BR     R9
774+YS$IOS20 MVC    OMA(LIOM2),IOM2 .FILE NOT AVAILABLE
775+          MVC    ZA#OTL(2),=Y(0+LIOM2+4)
776+          MVC    OMA+DIOM2-IOM2(20),IOFILE
777+          B      TERM
778+YS$IOSTR DC     C'0123456789ABCDEFX'
779+IOM1     DC     X'100A18011C'
780+          DC     C'INVALID FILE I/O '
781+DIOM1C   DC     CL5' ' .PIB STATUS
782+DIOM1A   DC     CL21' ' .FILE NAME
783+DIOM1B   DC     CL17' ' .FILE KEY
784+          DC     C'CALL ISD'
785+          DC     X'1D10020000'
786+LIOM1    EQU    *-IOM1
787+IOM2     DC     X'100A18011C'
788+DIOM2    DC     CL21' ' .FILE NAME
789+          DC     C'FILE NOT AVAILABLE'
790+          DC     X'1D10020000'
791+LIOM2    EQU    *-IOM2
792+YS$IOS30 MVC    IOSTS,ZA#PSC
793+          TR     IOSTS,YS$IOSTR .TRANSLATE TO PRINTABLE CHAR
794+          MVC    OMA(LIOM1),IOM1 .FILE NOT AVAILABLE
795+          MVC    OMA+DIOM1A-IOM1(21),IOFILE
796+          MVC    OMA+DIOM1B-IOM1(16),IOKEY
797+          MVC    OMA+DIOM1C-IOM1(4),IOSTS
798+          MVC    ZA#OTL(2),=Y(0+LIOM1+4)
799+          B      SNAP
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 11 of 29)

```
800            YSSMVIN                          .GET IMA DATA
801+*
802+************** MOVE IMA DATA TO SCREEN WORK AREA
803+*
804+MOVEIN     ST     R0,SCREEN# .SCREEN NUMBER
805+          MVC    IOKEY(4),SCREEN# .SCREEN NUMBER
806+          MVI    IOKEY+4,C'G' .GET
807+          MVI    IOFILE,C' '
808+          MVC    IOFILE+1(19),IOFILE .CLEAR TO SPACES
809+          MVC    IOFILE(13),=C'SCREEN FORMAT' .FILE NAME
810+          ZG#CALL MSGIN,(SCRNUM,IN$MSG)
811+          DS     0H
812+          LA     15,SCRNUM
813+          ST     15,PLIST+4*(1-1)
814+          LA     15,IN$MSG
815+          ST     15,PLIST+4*(2-1)
816+          OI     PLIST+4*(2-1),X'80'
817+          LA     1,PLIST
818+          L      15,=V(MSGIN)
819+          BALR   14,15
820+          LA     R9,ABTERM .I/O ERROR ADDRESS
821+          B      IOSTATUS .CHECK I/O STATUS
822            YSSMVOUT                         .PUT OMA DATA
823+*
824+************** MOVE DATA FROM SCREEN WORK AREA TO OMA
825+*
826+MOVEOUT    ST     R0,SCREEN# .SCREEN NUMBER
827+          MVC    IOKEY(4),SCREEN# .SCREEN NUMBER
828+          MVI    IOKEY+4,C'P' .PUT
829+          MVI    IOFILE,C' '
830+          MVC    IOFILE+1(19),IOFILE .CLEAR TO SPACES
831+          MVC    IOFILE(13),=C'SCREEN FORMAT' .FILE NAME
832+          ZG#CALL MSGOUT,(SCRNUM,OUT$MSG,PDATA) .SCREEN AND DATA
833+          DS     0H
834+          LA     15,SCRNUM
835+          ST     15,PLIST+4*(1-1)
836+          LA     15,OUT$MSG
837+          ST     15,PLIST+4*(2-1)
838+          LA     15,PDATA
839+          ST     15,PLIST+4*(3-1)
840+          OI     PLIST+4*(3-1),X'80'
841+          LA     1,PLIST
842+          L      15,=V(MSGOUT)
843+          BALR   14,15
844+          B      YSSMO010
845+MOVEOUTS   ST     R0,SCREEN#
846+          MVC    IOKEY(4),SCREEN# .SCREEN NUMBER
847+          MVI    IOKEY+4,C'P' .PUT
848+          MVI    IOFILE,C' '
849+          MVC    IOFILE+1(19),IOFILE .CLEAR TO SPACES
850+          MVC    IOFILE(13),=C'SCREEN FORMAT' .FILE NAME
851+          ZG#CALL MSGOUT,(SCRNUM)        .SCREEN ONLY (NO DATA)
852+          DS     0H
853+          LA     15,SCRNUM
854+          ST     15,PLIST+4*(1-1)
855+          OI     PLIST+4*(1-1),X'80'
856+          LA     1,PLIST
857+          L      15,=V(MSGOUT)
858+          BALR   14,15
859+YSSMO010   LA     R9,ABTERM .I/O ERROR ADDRESS
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 12 of 29)

```
860+           B      IOSTATUS .CHECK I/O STATUS
861 ACCTPAY    YS$GET 15
862+*
863+*                 GET
864+*
865+GACCTPAY   MVC    IOKEY(15),KACCTPAY .SAVE KEY
866+           MVI    IOKEY+15,C'G' .TYPE OF I/O
867+           ZG#CALL GET,(&FIL.,R&FIL.,K&FIL.)
868+           DS     0H
869+           LA     15,ACCTPAY
870+           ST     15,PLIST+4*(1-1)
871+           LA     15,RACCTPAY
872+           ST     15,PLIST+4*(2-1)
873+           LA     15,KACCTPAY
874+           ST     15,PLIST+4*(3-1)
875+           OI     PLIST+4*(3-1),X'80'
876+           LA     1,PLIST
877+           L      15,=V(GET)
878+           BALR   14,15
879+           AI     #GET,1 .INCREMENT IO COUNT
880+           MVC    IOFILE(20),ACCTPAY+8 .SAVE FILE
881+           B      IOSTATUS .CHECK I/O STATUS
882 ACCTPAY    YS$READ 15
883+*
884+*                 READ (SEQUENTIAL GET)
885+*
886+NACCTPAY   MVC    IOKEY(15),KACCTPAY .SAVE KEY
887+           MVI    IOKEY+15,C'N' .TYPE OF I/O
888+           ZG#CALL GET,(&FIL.,R&FIL.)
889+           DS     0H
890+           LA     15,ACCTPAY
891+           ST     15,PLIST+4*(1-1)
892+           LA     15,RACCTPAY
893+           ST     15,PLIST+4*(2-1)
894+           OI     PLIST+4*(2-1),X'80'
895+           LA     1,PLIST
896+           L      15,=V(GET)
897+           BALR   14,15
898+           AI     #GET,1 .INCREMENT IO COUNT
899+           MVC    IOFILE(20),ACCTPAY+8 .SAVE FILE
900+           B      IOSTATUS .CHECK I/O STATUS
901 ACCTPAY    YS$GETUP 15
902+*
903+*                 GETUP
904+*
905+UACCTPAY   MVC    IOKEY(15),KACCTPAY .SAVE KEY
906+           MVI    IOKEY+15,C'U' .TYPE OF I/O
907+           ZG#CALL GETUP,(&FIL.,R&FIL.,K&FIL.)
908+           DS     0H
909+           LA     15,ACCTPAY
910+           ST     15,PLIST+4*(1-1)
911+           LA     15,RACCTPAY
912+           ST     15,PLIST+4*(2-1)
913+           LA     15,KACCTPAY
914+           ST     15,PLIST+4*(3-1)
915+           OI     PLIST+4*(3-1),X'80'
916+           LA     1,PLIST
917+           L      15,=V(GETUP)
918+           BALR   14,15
919+           AI     #GETUP,1 .INCREMENT IO COUNT
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 13 of 29)

```
920+          MVC    IOFILE(20),ACCTPAY+8 .SAVE FILE
921+          B      IOSTATUS .CHECK I/O STATUS
922 ACCTPAY   YS$PUT 15
923+*
924+*                PUT
925+*
926+PACCTPAY MVC    IOKEY(15),KACCTPAY .SAVE KEY
927+          MVI    IOKEY+15,C'P' .TYPE OF I/O
928+          ZG#CALL PUT,(&FIL.,R&FIL.)
929+          DS     OH
930+          LA     15,ACCTPAY
931+          ST     15,PLIST+4*(1-1)
932+          LA     15,RACCTPAY
933+          ST     15,PLIST+4*(2-1)
934+          OI     PLIST+4*(2-1),X'80'
935+          LA     1,PLIST
936+          L      15,=V(PUT)
937+          BALR   14,15
938+          AI     #PUT,1 .INCREMENT IO COUNT
939+          MVC    IOFILE(20),ACCTPAY+8 .SAVE FILE
940+          B      IOSTATUS .CHECK I/O STATUS
941 ACCTPAY   YS$INSRT 15
942+*
943+*                INSERT
944+*
945+IACCTPAY MVC    IOKEY(15),KACCTPAY .SAVE KEY
946+          MVI    IOKEY+15,C'I' .TYPE OF I/O
947+          ZG#CALL INSERT,(&FIL.,R&FIL.)
948+          DS     OH
949+          LA     15,ACCTPAY
950+          ST     15,PLIST+4*(1-1)
951+          LA     15,RACCTPAY
952+          ST     15,PLIST+4*(2-1)
953+          OI     PLIST+4*(2-1),X'80'
954+          LA     1,PLIST
955+          L      15,=V(INSERT)
956+          BALR   14,15
957+          AI     #INSERT,1 .INCREMENT IO COUNT
958+          MVC    IOFILE(20),ACCTPAY+8 .SAVE FILE
959+          B      IOSTATUS .CHECK I/O STATUS
960 ACCTPAY   YS$SETLK 15
961+*
962+*                SET SEQUENTIAL MODE BY SPECIFIED KEY
963+*
964+SACCTPAY MVC    IOKEY(15),KACCTPAY .SAVE KEY
965+          MVI    IOKEY+15,C'S' .TYPE OF I/O
966+          ZG#CALL SETL,(&FIL.,POSITION,K&FIL.)
967+          DS     OH
968+          LA     15,ACCTPAY
969+          ST     15,PLIST+4*(1-1)
970+          LA     15,POSITION
971+          ST     15,PLIST+4*(2-1)
972+          LA     15,KACCTPAY
973+          ST     15,PLIST+4*(3-1)
974+          OI     PLIST+4*(3-1),X'80'
975+          LA     1,PLIST
976+          L      15,=V(SETL)
977+          BALR   14,15
978+          MVC    IOFILE(20),ACCTPAY+8 .SAVE FILE
979+          B      IOSTATUS .CHECK I/O STATUS
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 14 of 29)

```
 980 ACCTPAY   YSSESETL 15
 981+*
 982+*                  SET RANDOM MODE
 983+*
 984+EACCTPAY MVC     IOKEY(15),KACCTPAY .SAVE KEY
 985+         MVI     IOKEY+15,C'E' .TYPE OF I/O
 986+         ZG#CALL ESETL,(&FIL.)
 987+         DS      0H
 988+         LA      15,ACCTPAY
 989+         ST      15,PLIST+4*(1-1)
 990+         OI      PLIST+4*(1-1),X'80'
 991+         LA      1,PLIST
 992+         L       15,=V(ESETL)
 993+         BALR    14,15
 994+         MVC     IOFILE(20),ACCTPAY+8 .SAVE FILE
 995+         B       IOSTATUS .CHECK I/O STATUS
 996 ACCOUNT  YSSGET 8
 997+*
 998+*                  GET
 999+*
1000+GACCOUNT MVC     IOKEY(8),KACCOUNT .SAVE KEY
1001+         MVI     IOKEY+8,C'G' .TYPE OF I/O
1002+         ZG#CALL GET,(&FIL.,R&FIL.,K&FIL.)
1003+         DS      0H
1004+         LA      15,ACCOUNT
1005+         ST      15,PLIST+4*(1-1)
1006+         LA      15,RACCOUNT
1007+         ST      15,PLIST+4*(2-1)
1008+         LA      15,KACCOUNT
1009+         ST      15,PLIST+4*(3-1)
1010+         OI      PLIST+4*(3-1),X'80'
1011+         LA      1,PLIST
1012+         L       15,=V(GET)
1013+         BALR    14,15
1014+         AI      #GET,1 .INCREMENT IO COUNT
1015+         MVC     IOFILE(20),ACCOUNT+8 .SAVE FILE
1016+         B       IOSTATUS .CHECK I/O STATUS
1017 BRANCHM  YSSGET 3
1018+*
1019+*                  GET
1020+*
1021+GBRANCHM MVC     IOKEY(3),KBRANCHM .SAVE KEY
1022+         MVI     IOKEY+3,C'G' .TYPE OF I/O
1023+         ZG#CALL GET,(&FIL.,R&FIL.,K&FIL.)
1024+         DS      0H
1025+         LA      15,BRANCHM
1026+         ST      15,PLIST+4*(1-1)
1027+         LA      15,RBRANCHM
1028+         ST      15,PLIST+4*(2-1)
1029+         LA      15,KBRANCHM
1030+         ST      15,PLIST+4*(3-1)
1031+         OI      PLIST+4*(3-1),X'80'
1032+         LA      1,PLIST
1033+         L       15,=V(GET)
1034+         BALR    14,15
1035+         AI      #GET,1 .INCREMENT IO COUNT
1036+         MVC     IOFILE(20),BRANCHM+8 .SAVE FILE
1037+         B       IOSTATUS .CHECK I/O STATUS
1038 ACCTMST  YSSGET 8
1039+*
1040+*                  GET
```

Figure C–11. APITMS Action Program Processing a Dialog (Part 15 of 29)

```
+*
+GACCTMST MVC   IOKEY(8),KACCTMST .SAVE KEY
+         MVI   IOKEY+8,C'G' .TYPE OF I/O
+         ZG#CALL GET,(&FIL.,R&FIL.,K&FIL.)
+         DS    0H
+         LA    15,ACCTMST
+         ST    15,PLIST+4*(1-1)
+         LA    15,RACCTMST
+         ST    15,PLIST+4*(2-1)
+         LA    15,KACCTMST
+         ST    15,PLIST+4*(3-1)
+         OI    PLIST+4*(3-1),X'80'
+         LA    1,PLIST
+         L     15,=V(GET)
+         BALR  14,15
+         AI    #GET,1 .INCREMENT IO COUNT
+         MVC   IOFILE(20),ACCTMST+8 .SAVE FILE
+         B     IOSTATUS .CHECK I/O STATUS
 PAYROLL  YS$GET 4
+*
+*              GET
+*
+GPAYROLL MVC   IOKEY(4),KPAYROLL .SAVE KEY
+         MVI   IOKEY+4,C'G' .TYPE OF I/O
+         ZG#CALL GET,(&FIL.,R&FIL.,K&FIL.)
+         DS    0H
+         LA    15,PAYROLL
+         ST    15,PLIST+4*(1-1)
+         LA    15,RPAYROLL
+         ST    15,PLIST+4*(2-1)
+         LA    15,KPAYROLL
+         ST    15,PLIST+4*(3-1)
+         OI    PLIST+4*(3-1),X'80'
+         LA    1,PLIST
+         L     15,=V(GET)
+         BALR  14,15
+         AI    #GET,1 .INCREMENT IO COUNT
+         MVC   IOFILE(20),PAYROLL+8 .SAVE FILE
+         B     IOSTATUS .CHECK I/O STATUS
 TABLEMT  YS$GET 8
+*
+*              GET
+*
+GTABLEMT MVC   IOKEY(8),KTABLEMT .SAVE KEY
+         MVI   IOKEY+8,C'G' .TYPE OF I/O
+         ZG#CALL GET,(&FIL.,R&FIL.,K&FIL.)
+         DS    0H
+         LA    15,TABLEMT
+         ST    15,PLIST+4*(1-1)
+         LA    15,RTABLEMT
+         ST    15,PLIST+4*(2-1)
+         LA    15,KTABLEMT
+         ST    15,PLIST+4*(3-1)
+         OI    PLIST+4*(3-1),X'80'
+         LA    1,PLIST
+         L     15,=V(GET)
+         BALR  14,15
+         AI    #GET,1 .INCREMENT IO COUNT
+         MVC   IOFILE(20),TABLEMT+8 .SAVE FILE
+         B     IOSTATUS .CHECK I/O STATUS
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 16 of 29)

```
1101          YSSNOW                                  .DATE/TIME
1102+*
1103+*************** DATE AND TIME STAMP *************************************
1104+*
1105+DAYTIME  ORG    *
1106+          GETIME S
1107+          DS     0H
1108+          SR     1,1
1109+          SVC    7
1110+          ST     R0,WORK1 .DATE-0YYMMDD+
1111+          UNPK   WORK1+4(7),WORK1(4)
1112+          MVC    YYMMDD(6),WORK1+5
1113+          OI     YYMMDD+5,X'F0' .FIX SIGN
1114+          ST     R1,WORK1 .TIME-0HHMMSS+
1115+          UNPK   WORK1+4(7),WORK1(4)
1116+          MVC    HHMMSS(6),WORK1+5
1117+          OI     HHMMSS+5,X'F0' .FIX SIGN
1118+          BR     R7 .RETURN REGISTER
1119          YSSRJ                                   .RIGHT JUSTIFY
1120+*
1121+*************** RIGHT JUSTIFY *******************************************
1122+*
1123+*
1124+*                R0  = FIELD LENGTH
1125+*                R1  = FIELD ADDRESS
1126+*                R15 = RETURN STATUS
1127+*
1128+RJ1      LA     R0,1 .SET LENGTH
1129+          B      RJ
1130+RJ2      LA     R0,2 .SET LENGTH
1131+          B      RJ
1132+RJ3      LA     R0,3 .SET LENGTH
1133+          B      RJ
1134+RJ4      LA     R0,4 .SET LENGTH
1135+          B      RJ
1136+RJ5      LA     R0,5 .SET LENGTH
1137+          B      RJ
1138+RJ6      LA     R0,6 .SET LENGTH
1139+          B      RJ
1140+RJ7      LA     R0,7 .SET LENGTH
1141+          B      RJ
1142+RJ8      LA     R0,8 .SET LENGTH
1143+          B      RJ
1144+RJ9      LA     R0,9 .SET LENGTH
1145+          B      RJ
1146+RJ10     LA     R0,10 .SET LENGTH
1147+          B      RJ
1148+RJ11     LA     R0,11 .SET LENGTH
1149+          B      RJ
1150+RJ       ST     R7,RJSAVE .SAVE RETURN ADDRESS
1151+          LA     R13,SAVE .PROGRAM SAVE AREA
1152+          DC     0Y(0)
1153+          EXTRN  MODRJ1 .RIGHT JUSTIFY MODULE
1154+          L      R15,=A(MODRJ1)
1155+          BALR   R14,R15 .BRANCH TO RJ
1156+          L      R7,RJSAVE .RESTORE RETURN ADDRESS
1157+          LTR    R15,R15 .SET CONDITION CODE FOR ERRORS
1158+          BR     R7 .RETURN TO CALL
1159 APITMS   YSSSNAP                                 .SNAP DUMP
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 17 of 29)

```
**
*************** SNAP DUMP OF ACTION PROGRAM ***************************
**
SNAPIT   ORG    *
    ZG#CALL SNAP,(ZA#DPIB,EP,ZA#IMH,EI,WORK,EW,ZA#OMH,EO,ENAM.,YS$E)
         DS     0H
         LA     15,ZA#DPIB
         ST     15,PLIST+4*(1-1)
         LA     15,EP
         ST     15,PLIST+4*(2-1)
         LA     15,ZA#IMH
         ST     15,PLIST+4*(3-1)
         LA     15,EI
         ST     15,PLIST+4*(4-1)
         LA     15,WORK
         ST     15,PLIST+4*(5-1)
         LA     15,EW
         ST     15,PLIST+4*(6-1)
         LA     15,ZA#OMH
         ST     15,PLIST+4*(7-1)
         LA     15,EO
         ST     15,PLIST+4*(8-1)
         LA     15,APITMS
         ST     15,PLIST+4*(9-1)
         LA     15,YS$E
         ST     15,PLIST+4*(10-1)
         OI     PLIST+4*(10-1),X'80'
         LA     1,PLIST
         L      15,=V(SNAP)
         BALR   14,15
         BR     R7 .RETURN REGISTER
         YS$TERM                     .PROGRAM TERMINATION
***********************************************************************
**              PROGRAM TERMINATION                                   *
***********************************************************************
TERM     CLI    ISNAP,C'N' .REQUEST NORMAL TERMINATION WITH SNAP?
         BE     SNAP .YES
         CLI    ISNAP,C'S' .REQUEST ABNORMAL TERMINATION WITH SNAP?
         BNE    FINISH .NO-NORMAL TERMINATION
ABTERM   MVI    ZA#PSIND,C'S' .TERMINATE WITH SNAP DUMP
         B      FINISH
SNAP     GETIME M
SNAP     DS     0H
         LA     1,1
         SVC    7
         ST     R1,ETIMS
        ZG#CALL SNAP,(ZA#DPIB,EP,ZA#IMH,EI,WORK,EW,ZA#OMH,EO,YS$B,YS$E)
         DS     0H
         LA     15,ZA#DPIB
         ST     15,PLIST+4*(1-1)
         LA     15,EP
         ST     15,PLIST+4*(2-1)
         LA     15,ZA#IMH
         ST     15,PLIST+4*(3-1)
         LA     15,EI
         ST     15,PLIST+4*(4-1)
         LA     15,WORK
         ST     15,PLIST+4*(5-1)
         LA     15,EW
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 18 of 29)

```
1219+           ST     15,PLIST+4*(6-1)
1220+           LA     15,ZA#OMH
1221+           ST     15,PLIST+4*(7-1)
1222+           LA     15,EO
1223+           ST     15,PLIST+4*(8-1)
1224+           LA     15,Y$$B
1225+           ST     15,PLIST+4*(9-1)
1226+           LA     15,Y$$E
1227+           ST     15,PLIST+4*(10-1)
1228+           OI     PLIST+4*(10-1),X'80'
1229+           LA     1,PLIST
1230+           L      15,=V(SNAP)
1231+           BALR   14,15
1232+FINISH     GETIME M
1233+FINISH     DS     0H
1234+           LA     1,1
1235+           SVC    7
1236+           ST     R1,ETIM$ .ENDING TIME
1237+           ZG#CALL RETURN        .RETURN CONTROL TO IMS
1238+           DS     0H
1239+           L      15,=V(RETURN)
1240+           BALR   14,15
1242            Y$$MSG 1
1243+EMSG1      MVC    OMA(LMSG1),MSG1
1244+           MVC    ZA#OTL(2),=Y(0+LMSG1+4)
1245+           B      TERM
1246            Y$$MSG 2
1247+EMSG2      MVC    OMA(LMSG2),MSG2
1248+           MVC    ZA#OTL(2),=Y(0+LMSG2+4)
1249+           B      TERM
1250            Y$$MSG 3
1251+EMSG3      MVC    OMA(LMSG3),MSG3
1252+           MVC    ZA#OTL(2),=Y(0+LMSG3+4)
1253+           B      TERM
1254            Y$$MSG 4
1255+EMSG4      MVC    OMA(LMSG4),MSG4
1256+           MVC    ZA#OTL(2),=Y(0+LMSG4+4)
1257+           B      TERM
1258            Y$$MSG 5
1259+EMSG5      MVC    OMA(LMSG5),MSG5
1260+           MVC    ZA#OTL(2),=Y(0+LMSG5+4)
1261+           B      TERM
1262            Y$$MSG 7
1263+EMSG7      MVC    OMA(LMSG7),MSG7
1264+           MVC    ZA#OTL(2),=Y(0+LMSG7+4)
1265+           B      TERM
1266            Y$$MSG 10
1267+EMSG10     MVC    OMA(LMSG10),MSG10
1268+           MVC    ZA#OTL(2),=Y(0+LMSG10+4)
1269+           B      TERM
1270 *********************************************************************
1271 *                 CONSTANTS
1272 *********************************************************************
1273 ACCTPAY    DC     C'ACCTPAY ACCOUNTS PAYABLE     '
1274 ACCOUNT    DC     C'ACCOUNT CHART OF ACCOUNTS    '
1275 BRANCHM    DC     C'BRANCHM BRANCH MASTER        '
1276 ACCTMST    DC     C'ACCTMST ACCOUNT SUMMARY MST  '
1277 PAYROLL    DC     C'PAYROLL PAYROLL MASTER       '
1278 TABLEMT    DC     C'TABLEMT SECURITY AND CODE    '
1279 BLANKS     DC     CL80' '
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 19 of 29)

```
MSG1      DC    X'100A18011C'
          DC    C'USE "TRAN UNPROT DISPL"'
          DC    X'1D10020000'
LMSG1     EQU   *-MSG1
*
MSG2      DC    X'100A18011C'
          DC    C'AP HEADER NOT FOUND. CONTACT ISD'
          DC    X'1D10020000'
LMSG2     EQU   *-MSG2
*
MSG3      DC    X'100A18011C'
          DC    C'AP SETLL ERROR'
          DC    X'1D10020000'
LMSG3     EQU   *-MSG3
*
MSG4      DC    X'100A18011C'
          DC    C'ITEM NOT FOUND'
          DC    X'1D10020000'
LMSG4     EQU   *-MSG4
*
MSG5      DC    X'100A18011C'
          DC    C'CHECK NOT FOUND'
          DC    X'1D10020000'
LMSG5     EQU   *-MSG5
*
MSG7      DC    X'100A18011C'
          DC    C'CHECK AMOUNT CANNOT BE TRANSLATED'
          DC    X'1D10020000'
LMSG7     EQU   *-MSG7
*
MSG10     DC    X'100A18011C'
          DC    C'AP ITEMS'
          DC    X'1D10020000'
LMSG10    EQU   *-MSG10
*
MSG11     DC    C'APCKS '
          DC    X'3F3F'
          DC    C'X'                          CHANGE
          DC    X'3F3F'
M11A      DC    X'05'
          DC    X'3F'
M11B      DC    CL5' '
          DC    X'3F05'
LMSG11    EQU   *-MSG11
DM11A     EQU   M11A-MSG11
DM11B     EQU   M11B-MSG11
*
MSG12     DC    C'APAUD '
          DC    CL3' '
          DC    X'3F'
M12A      DC    CL4' '                        CHECK TYPE
          DC    X'3F'
M12B      DC    CL5' '                        CHECK #
          DC    X'3F'
          DC    CL2' '
LMSG12    EQU   *-MSG12
DM12A     EQU   M12A-MSG12
DM12B     EQU   M12B-MSG12
*
          PRINT GEN
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 20 of 29)

```
1340          Y$$PIB                                  .PROGRAM INFORMATION BLOCK
1341+*************************************************************************
1342+*                 LITERAL POOL                                          *
1343+*************************************************************************
1344+         LTORG
1345+                  =C'0000'
1346+                  =A(CKGD50)
1347+                  =V(MSGIN)
1348+                  =V(MSGOUT)
1349+                  =V(GET)
1350+                  =V(GETUP)
1351+                  =V(PUT)
1352+                  =V(INSERT)
1353+                  =V(SETL)
1354+                  =V(ESETL)
1355+                  =A(MODRJ1)
1356+                  =V(SNAP)
1357+                  =V(RETURN)
1358+                  =Y(0+LY$$M1+4)
1359+                  =Y(IMAI-USTART)
1360+                  =Y(UACCT1-USTART+1)
1361+                  =C'AC'
1362+                  =C'AP'
1363+                  =X'40206B2020206B2021204B202060'
1364+                  =C'AI'
1365+                  =C'APITMS'
1366+                  =C'APITS '
1367+                  =H'14'
1368+                  =C'APCHKS'
1369+                  =Y(0+LMSG11+8)
1370+                  =C'APAUDT'
1371+                  =Y(0+LMSG12+8)
1372+                  =C'APCKS '
1373+                  =X'5C206B2020206B2021204B202060'
1374+                  =Y(0+LIOM2+4)
1375+                  =Y(0+LIOM1+4)
1376+                  =Y(0+LMSG1+4)
1377+                  =Y(0+LMSG2+4)
1378+                  =Y(0+LMSG3+4)
1379+                  =Y(0+LMSG4+4)
1380+                  =Y(0+LMSG5+4)
1381+                  =Y(0+LMSG7+4)
1382+                  =Y(0+LMSG10+4)
1383+                  =C'APCHK'
1384+                  =C'T80'
1385+                  =C'   '
1386+                  =C'A   P'
1387+                  =C'APRNT'
1388+                  =P'1'
1389+                  =P'0'
1390+                  =C'T10'
1391+                  =C'001'
1392+                  =C'SCREEN FORMAT'
1393+Y$$E     EQU     * .END OF PROGRAM
1516 WORK     Y$$WORK                                  .WORK AREA
1517+*************************************************************************
1518+*                 WORK AREA                                             *
1519+*************************************************************************
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 21 of 29)

```
+WORK      DSECT
+STIM$     DS     A .START TIME (MILLISECONDS)
+ETIM$     DS     A .END TIME (MILLISECONDS)
+#GET      DS     H .NUMBER OF GET
+#GETUP    DS     H .          GETUP
+#PUT      DS     H .          PUT
+#INSERT   DS     H .          INSERT
+SAVE      DS     18F .PROGRAM SAVE AREA
+PLIST     DS     4A .PARAMETER LIST FOR "CALLS"
+WHO       DS     CL3 .USER INITIALS
+WORK1     DS     2D .WORK FIELD
+PASSKEY   EQU    WORK1,5 .SECURITY RECORD FILE KEY
+IOFILE    DS     CL20 .LAST FILE I/O
+IOKEY     DS     CL20 .LAST FILE I/O KEY
+IOSTS     DS     CL4 .LAST FILE I/O STATUS
+IORET     DS     CL1 .FILE NOT AVAILABLE-RETURN
+ERR       DS     CL1 .ERROR FLAG
+YYMMDD    DS     CL6 .DATE
+HHMMSS    DS     CL6 .TIME
 RJSAVE    DS     A
 EXPENS    DS     CL1
 INCOME    DS     CL1
 PROLINE   DS     A
 POSITION  DS     CL1
 LAST      DS     CL1
 BRCH      DS     CL4
 DESCPTH   DS     CL30
 REVIEW    DS     CL1
 TRAIL$    DS     CL250
 TRAIL$1   DS     A
 TRAIL$2   DS     A
           Y$$SWORK                          .SDMPS WORK SPACE
+*
+************** SDMPS WORK AREA ************************************
+*
+SCRNUM    DS     D .SCREEN NUMBER
+SCREEN#   EQU    SCRNUM+4,4
+SCREENW   DS     CL180 .SCREEN WORK AREA
+MAXITL    EQU    SCREENW,2 .MAXIMUM INPUT TEXT LENGTH
+*
+************** SDMPS I/O AREAS
+*
+UDATA     EQU    *
+OUT$MSG   EQU    * .OUTPUT MESSAGE DATA
+FILL      DS     CL1 .OUTPUT FILL CHARACTER
+IN$MSG    EQU    * .INPUT MESSAGE DATA
 *
 ************** UNPROTECTED DATA
 *
 USTART    EQU    *
 UTRAN     DS     CL5                        .TRANSACTION CODE
 USNAP     DS     CL1                        .SNAP CODE
 USLINE    EQU    *                          .START OF LINE ITEM UNPROT
 IMA1      EQU    *
 UACCT1    DS     CL8                        .ACCOUNT NUMBER
 UAMT1     DS     CL10                       .AMOUNT
 UDESPT1   DS     CL30                       .DESCRIPTION
 UEMP1     DS     CL4                        .EMPLOYEE NUMBER
 UXMIT1    DS     CL2                        .TRANSMIT POSITION
 ULLINE    EQU    *-UACCT1                   .END OF LINE ITEM UNPROT
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 22 of 29)

```
UACCT2    DS    CL8                                 .ACCOUNT NUMBER
UAMT2     DS    CL10                                .AMOUNT
UDESPT2   DS    CL30                                .DESCRIPTION
UEMP2     DS    CL4                                 .EMPLOYEE NUMBER
UXMIT2    DS    CL2                                 .TRANSMIT POSITION
UACCT3    DS    CL8                                 .ACCOUNT NUMBER
UAMT3     DS    CL10                                .AMOUNT
UDESPT3   DS    CL30                                .DESCRIPTION
UEMP3     DS    CL4                                 .EMPLOYEE NUMBER
UXMIT3    DS    CL2                                 .TRANSMIT POSITION
UACCT4    DS    CL8                                 .ACCOUNT NUMBER
UAMT4     DS    CL10                                .AMOUNT
UDESPT4   DS    CL30                                .DESCRIPTION
UEMP4     DS    CL4                                 .EMPLOYEE NUMBER
UXMIT4    DS    CL2                                 .TRANSMIT POSITION
UACCT5    DS    CL8                                 .ACCOUNT NUMBER
UAMT5     DS    CL10                                .AMOUNT
UDESPT5   DS    CL30                                .DESCRIPTION
UEMP5     DS    CL4                                 .EMPLOYEE NUMBER
UXMIT5    DS    CL2                                 .TRANSMIT POSITION
UACCT6    DS    CL8                                 .ACCOUNT NUMBER
UAMT6     DS    CL10                                .AMOUNT
UDESPT6   DS    CL30                                .DESCRIPTION
UEMP6     DS    CL4                                 .EMPLOYEE NUMBER
UXMIT6    DS    CL2                                 .TRANSMIT POSITION
UACCT7    DS    CL8                                 .ACCOUNT NUMBER
UAMT7     DS    CL10                                .AMOUNT
UDESPT7   DS    CL30                                .DESCRIPTION
UEMP7     DS    CL4                                 .EMPLOYEE NUMBER
UXMIT7    DS    CL2                                 .TRANSMIT POSITION
UACCT8    DS    CL8                                 .ACCOUNT NUMBER
UAMT8     DS    CL10                                .AMOUNT
UDESPT8   DS    CL30                                .DESCRIPTION
UEMP8     DS    CL4                                 .EMPLOYEE NUMBER
UXMIT8    DS    CL2                                 .TRANSMIT POSITION
UACCT9    DS    CL8                                 .ACCOUNT NUMBER
UAMT9     DS    CL10                                .AMOUNT
UDESPT9   DS    CL30                                .DESCRIPTION
UEMP9     DS    CL4                                 .EMPLOYEE NUMBER
UXMIT9    DS    CL2                                 .TRANSMIT POSITION
UACCT10   DS    CL8                                 .ACCOUNT NUMBER
UAMT10    DS    CL10                                .AMOUNT
UDESPT10  DS    CL30                                .DESCRIPTION
UEMP10    DS    CL4                                 .EMPLOYEE NUMBER
UXMIT10   DS    CL2                                 .TRANSMIT POSITION
UACCT11   DS    CL8                                 .ACCOUNT NUMBER
UAMT11    DS    CL10                                .AMOUNT
UDESPT11  DS    CL30                                .DESCRIPTION
UEMP11    DS    CL4                                 .EMPLOYEE NUMBER
UXMIT11   DS    CL2                                 .TRANSMIT POSITION
UACCT12   DS    CL8                                 .ACCOUNT NUMBER
UAMT12    DS    CL10                                .AMOUNT
UDESPT12  DS    CL30                                .DESCRIPTION
UEMP12    DS    CL4                                 .EMPLOYEE NUMBER
UXMIT12   DS    CL2                                 .TRANSMIT POSITION
UACCT13   DS    CL8                                 .ACCOUNT NUMBER
UAMT13    DS    CL10                                .AMOUNT
UDESPT13  DS    CL30                                .DESCRIPTION
UEMP13    DS    CL4                                 .EMPLOYEE NUMBER
UXMIT13   DS    CL2                                 .TRANSMIT POSITION
```

Figure C–11. APITMS Action Program Processing a Dialog (Part 23 of 29)

```
UACCT14   DS    CL8                         .ACCOUNT NUMBER
UAMT14    DS    CL10                        .AMOUNT
UDESPT14  DS    CL30                        .DESCRIPTION
UEMP14    DS    CL4                         .EMPLOYEE NUMBER
UXMIT14   DS    CL2                         .TRANSMIT POSITION
UACCT15   DS    CL8                         .ACCOUNT NUMBER
UAMT15    DS    CL10                        .AMOUNT
UDESPT15  DS    CL30                        .DESCRIPTION
UEMP15    DS    CL4                         .EMPLOYEE NUMBER
UXMIT15   DS    CL2                         .TRANSMIT POSITION
UACCT16   DS    CL8                         .ACCOUNT NUMBER
UAMT16    DS    CL10                        .AMOUNT
UDESPT16  DS    CL30                        .DESCRIPTION
UEMP16    DS    CL4                         .EMPLOYEE NUMBER
UXMIT16   DS    CL2                         .TRANSMIT POSITION
UACCT17   DS    CL8                         .ACCOUNT NUMBER
UAMT17    DS    CL10                        .AMOUNT
UDESPT17  DS    CL30                        .DESCRIPTION
UEMP17    DS    CL4                         .EMPLOYEE NUMBER
UXMIT17   DS    CL2                         .TRANSMIT POSITION
UACCT18   DS    CL8                         .ACCOUNT NUMBER
UAMT18    DS    CL10                        .AMOUNT
UDESPT18  DS    CL30                        .DESCRIPTION
UEMP18    DS    CL4                         .EMPLOYEE NUMBER
UXMIT18   DS    CL2                         .TRANSMIT POSITION
UACCT19   DS    CL8                         .ACCOUNT NUMBER
UAMT19    DS    CL10                        .AMOUNT
UDESPT19  DS    CL30                        .DESCRIPTION
UEMP19    DS    CL4                         .EMPLOYEE NUMBER
UXMIT19   DS    CL2                         .TRANSMIT POSITION
UACCT20   DS    CL8                         .ACCOUNT NUMBER
UAMT20    DS    CL10                        .AMOUNT
UDESPT20  DS    CL30                        .DESCRIPTION
UEMP20    DS    CL4                         .EMPLOYEE NUMBER
UXMIT20   DS    CL2                         .TRANSMIT POSITION
DAMT      EQU   UAMT1-USLINE                .DISPLACEMENT OF AMOUNT
DDESPT    EQU   UDESPT1-USLINE             .DISPLACEMENT OF DESCRIPTION
DEMP      EQU   UEMP1-USLINE                .DISPLACEMENT OF EMPLOYEE #
DXMIT     EQU   UXMIT1-USLINE               .DISPLACEMENT OF TRANSMIT
USTOP     DS    CL1
*
************** PROTECTED REPLACEMENT
*
PDATA     EQU   *
PSTART    EQU   *
PSLINE    EQU   *                           .START OF LINE ITEM
PLIN#1    DS    CL3                         .LINE NUMBER
PCACCT1   DS    CL1                         .ACCOUNT ERROR CODE
PBACCT1   DS    CL1                         .ACCOUNT BLINKER
PCOA1     DS    CL1                         .CASH/ACCRUAL
PBAMT1    DS    CL1                         .AMOUNT BLINKER
PBDESP1   DS    CL1                         .DESCRIPTION BLINKER
PCEMP1    DS    CL1                         .EMPLOYEE ERROR CODE
PBEMP1    DS    CL1                         .EMPLOYEE BLINKER
PELINE    EQU   *                           .END OF LINE ITEM
PLLINE    EQU   PELINE-PSLINE
PLIN#2    DS    CL3                         .LINE NUMBER
PCACCT2   DS    CL1                         .ACCOUNT ERROR CODE
PBACCT2   DS    CL1                         .ACCOUNT BLINKER
PCOA2     DS    CL1                         .CASH/ACCRUAL
```

Figure C–11. APITMS Action Program Processing a Dialog (Part 24 of 29)

```
PBAMT2   DS     CL1                                .AMOUNT BLINKER
PBDESP2  DS     CL1                                .DESCRIPTION BLINKER
PCEMP2   DS     CL1                                .EMPLOYEE ERROR CODE
PBEMP2   DS     CL1                                .EMPLOYEE BLINKER
PLIN#3   DS     CL3                                .LINE NUMBER
PCACCT3  DS     CL1                                .ACCOUNT ERROR CODE
PBACCT3  DS     CL1                                .ACCOUNT BLINKER
PCOA3    DS     CL1                                .CASH/ACCRUAL
PBAMT3   DS     CL1                                .AMOUNT BLINKER
PBDESP3  DS     CL1                                .DESCRIPTION BLINKER
PCEMP3   DS     CL1                                .EMPLOYEE ERROR CODE
PBEMP3   DS     CL1                                .EMPLOYEE BLINKER
PLIN#4   DS     CL3                                .LINE NUMBER
PCACCT4  DS     CL1                                .ACCOUNT ERROR CODE
PBACCT4  DS     CL1                                .ACCOUNT BLINKER
PCOA4    DS     CL1                                .CASH/ACCRUAL
PBAMT4   DS     CL1                                .AMOUNT BLINKER
PBDESP4  DS     CL1                                .DESCRIPTION BLINKER
PCEMP4   DS     CL1                                .EMPLOYEE ERROR CODE
PBEMP4   DS     CL1                                .EMPLOYEE BLINKER
PLIN#5   DS     CL3                                .LINE NUMBER
PCACCT5  DS     CL1                                .ACCOUNT ERROR CODE
PBACCT5  DS     CL1                                .ACCOUNT BLINKER
PCOA5    DS     CL1                                .CASH/ACCRUAL
PBAMT5   DS     CL1                                .AMOUNT BLINKER
PBDESP5  DS     CL1                                .DESCRIPTION BLINKER
PCEMP5   DS     CL1                                .EMPLOYEE ERROR CODE
PBEMP5   DS     CL1                                .EMPLOYEE BLINKER
PLIN#6   DS     CL3                                .LINE NUMBER
PCACCT6  DS     CL1                                .ACCOUNT ERROR CODE
PBACCT6  DS     CL1                                .ACCOUNT BLINKER
PCOA6    DS     CL1                                .CASH/ACCRUAL
PBAMT6   DS     CL1                                .AMOUNT BLINKER
PBDESP6  DS     CL1                                .DESCRIPTION BLINKER
PCEMP6   DS     CL1                                .EMPLOYEE ERROR CODE
PBEMP6   DS     CL1                                .EMPLOYEE BLINKER
PLIN#7   DS     CL3                                .LINE NUMBER
PCACCT7  DS     CL1                                .ACCOUNT ERROR CODE
PBACCT7  DS     CL1                                .ACCOUNT BLINKER
PCOA7    DS     CL1                                .CASH/ACCRUAL
PBAMT7   DS     CL1                                .AMOUNT BLINKER
PBDESP7  DS     CL1                                .DESCRIPTION BLINKER
PCEMP7   DS     CL1                                .EMPLOYEE ERROR CODE
PBEMP7   DS     CL1                                .EMPLOYEE BLINKER
PLIN#8   DS     CL3                                .LINE NUMBER
PCACCT8  DS     CL1                                .ACCOUNT ERROR CODE
PBACCT8  DS     CL1                                .ACCOUNT BLINKER
PCOA8    DS     CL1                                .CASH/ACCRUAL
PBAMT8   DS     CL1                                .AMOUNT BLINKER
PBDESP8  DS     CL1                                .DESCRIPTION BLINKER
PCEMP8   DS     CL1                                .EMPLOYEE ERROR CODE
PBEMP8   DS     CL1                                .EMPLOYEE BLINKER
PLIN#9   DS     CL3                                .LINE NUMBER
PCACCT9  DS     CL1                                .ACCOUNT ERROR CODE
PBACCT9  DS     CL1                                .ACCOUNT BLINKER
PCOA9    DS     CL1                                .CASH/ACCRUAL
PBAMT9   DS     CL1                                .AMOUNT BLINKER
PBDESP9  DS     CL1                                .DESCRIPTION BLINKER
PCEMP9   DS     CL1                                .EMPLOYEE ERROR CODE
PBEMP9   DS     CL1                                .EMPLOYEE BLINKER
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 25 of 29)

```
1760 PLIN#10   DS     CL3                         .LINE NUMBER
1761 PCACCT10  DS     CL1                         .ACCOUNT ERROR CODE
1762 PBACCT10  DS     CL1                         .ACCOUNT BLINKER
1763 PCOA10    DS     CL1                         .CASH/ACCRUAL
1764 PBAMT10   DS     CL1                         .AMOUNT BLINKER
1765 PBDESP10  DS     CL1                         .DESCRIPTION BLINKER
1766 PCEMP10   DS     CL1                         .EMPLOYEE ERROR CODE
1767 PBEMP10   DS     CL1                         .EMPLOYEE BLINKER
1768 PLIN#11   DS     CL3                         .LINE NUMBER
1769 PCACCT11  DS     CL1                         .ACCOUNT ERROR CODE
1770 PBACCT11  DS     CL1                         .ACCOUNT BLINKER
1771 PCOA11    DS     CL1                         .CASH/ACCRUAL
1772 PBAMT11   DS     CL1                         .AMOUNT BLINKER
1773 PBDESP11  DS     CL1                         .DESCRIPTION BLINKER
1774 PCEMP11   DS     CL1                         .EMPLOYEE ERROR CODE
1775 PBEMP11   DS     CL1                         .EMPLOYEE BLINKER
1776 PLIN#12   DS     CL3                         .LINE NUMBER
1777 PCACCT12  DS     CL1                         .ACCOUNT ERROR CODE
1778 PBACCT12  DS     CL1                         .ACCOUNT BLINKER
1779 PCOA12    DS     CL1                         .CASH/ACCRUAL
1780 PBAMT12   DS     CL1                         .AMOUNT BLINKER
1781 PBDESP12  DS     CL1                         .DESCRIPTION BLINKER
1782 PCEMP12   DS     CL1                         .EMPLOYEE ERROR CODE
1783 PBEMP12   DS     CL1                         .EMPLOYEE BLINKER
1784 PLIN#13   DS     CL3                         .LINE NUMBER
1785 PCACCT13  DS     CL1                         .ACCOUNT ERROR CODE
1786 PBACCT13  DS     CL1                         .ACCOUNT BLINKER
1787 PCOA13    DS     CL1                         .CASH/ACCRUAL
1788 PBAMT13   DS     CL1                         .AMOUNT BLINKER
1789 PBDESP13  DS     CL1                         .DESCRIPTION BLINKER
1790 PCEMP13   DS     CL1                         .EMPLOYEE ERROR CODE
1791 PBEMP13   DS     CL1                         .EMPLOYEE BLINKER
1792 PLIN#14   DS     CL3                         .LINE NUMBER
1793 PCACCT14  DS     CL1                         .ACCOUNT ERROR CODE
1794 PBACCT14  DS     CL1                         .ACCOUNT BLINKER
1795 PCOA14    DS     CL1                         .CASH/ACCRUAL
1796 PBAMT14   DS     CL1                         .AMOUNT BLINKER
1797 PBDESP14  DS     CL1                         .DESCRIPTION BLINKER
1798 PCEMP14   DS     CL1                         .EMPLOYEE ERROR CODE
1799 PBEMP14   DS     CL1                         .EMPLOYEE BLINKER
1800 PLIN#15   DS     CL3                         .LINE NUMBER
1801 PCACCT15  DS     CL1                         .ACCOUNT ERROR CODE
1802 PBACCT15  DS     CL1                         .ACCOUNT BLINKER
1803 PCOA15    DS     CL1                         .CASH/ACCRUAL
1804 PBAMT15   DS     CL1                         .AMOUNT BLINKER
1805 PBDESP15  DS     CL1                         .DESCRIPTION BLINKER
1806 PCEMP15   DS     CL1                         .EMPLOYEE ERROR CODE
1807 PBEMP15   DS     CL1                         .EMPLOYEE BLINKER
1808 PLIN#16   DS     CL3                         .LINE NUMBER
1809 PCACCT16  DS     CL1                         .ACCOUNT ERROR CODE
1810 PBACCT16  DS     CL1                         .ACCOUNT BLINKER
1811 PCOA16    DS     CL1                         .CASH/ACCRUAL
1812 PBAMT16   DS     CL1                         .AMOUNT BLINKER
1813 PBDESP16  DS     CL1                         .DESCRIPTION BLINKER
1814 PCEMP16   DS     CL1                         .EMPLOYEE ERROR CODE
1815 PBEMP16   DS     CL1                         .EMPLOYEE BLINKER
1816 PLIN#17   DS     CL3                         .LINE NUMBER
1817 PCACCT17  DS     CL1                         .ACCOUNT ERROR CODE
1818 PBACCT17  DS     CL1                         .ACCOUNT BLINKER
1819 PCOA17    DS     CL1                         .CASH/ACCRUAL
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 26 of 29)

```
PBAMT17   DS     CL1                      .AMOUNT BLINKER
PBDESP17  DS     CL1                      .DESCRIPTION BLINKER
PCEMP17   DS     CL1                      .EMPLOYEE ERROR CODE
PBEMP17   DS     CL1                      .EMPLOYEE BLINKER
PLIN#18   DS     CL3                      .LINE NUMBER
PCACCT18  DS     CL1                      .ACCOUNT ERROR CODE
PBACCT18  DS     CL1                      .ACCOUNT BLINKER
PCOA18    DS     CL1                      .CASH/ACCRUAL
PBAMT18   DS     CL1                      .AMOUNT BLINKER
PBDESP18  DS     CL1                      .DESCRIPTION BLINKER
PCEMP18   DS     CL1                      .EMPLOYEE ERROR CODE
PBEMP18   DS     CL1                      .EMPLOYEE BLINKER
PLIN#19   DS     CL3                      .LINE NUMBER
PCACCT19  DS     CL1                      .ACCOUNT ERROR CODE
PBACCT19  DS     CL1                      .ACCOUNT BLINKER
PCOA19    DS     CL1                      .CASH/ACCRUAL
PBAMT19   DS     CL1                      .AMOUNT BLINKER
PBDESP19  DS     CL1                      .DESCRIPTION BLINKER
PCEMP19   DS     CL1                      .EMPLOYEE ERROR CODE
PBEMP19   DS     CL1                      .EMPLOYEE BLINKER
PLIN#20   DS     CL3                      .LINE NUMBER
PCACCT20  DS     CL1                      .ACCOUNT ERROR CODE
PBACCT20  DS     CL1                      .ACCOUNT BLINKER
PCOA20    DS     CL1                      .CASH/ACCRUAL
PBAMT20   DS     CL1                      .AMOUNT BLINKER
PBDESP20  DS     CL1                      .DESCRIPTION BLINKER
PCEMP20   DS     CL1                      .EMPLOYEE ERROR CODE
PBEMP20   DS     CL1                      .EMPLOYEE BLINKER
DCACCT    EQU    PCACCT1-PLIN#1           .DISPLACEMENT OF ACCT ERR CODE
DBACCT    EQU    PBACCT1-PLIN#1           .DISPLACEMENT OF ACCT BLINKER
DPCOA     EQU    PCOA1-PLIN#1             .DISPLACEMENT OF CASH/ACCRUAL
DBAMT     EQU    PBAMT1-PLIN#1            .DISPLACEMENT OF AMOUNT BLINKER
DBDESPT   EQU    PBDESP1-PLIN#1           .DISPLACEMENT OF DESCPT BLINKER
DCEMP     EQU    PCEMP1-PLIN#1            .DISPLACEMENT OF EMP ERR CODE
DBEMP     EQU    PBEMP1-PLIN#1            .DISPLACEMENT OF EMP BLINKER
PTYPE     DS     CL1                      .CHECK TYPE
PCHECK    DS     CL5                      .CHECK NUMBER
PCAMT     DS     CL14                     .CHECK AMOUNT
PCNAME    DS     CL25                     .CHECK PAYEE
PSTOP     DS     CL1
*
*                CHECK PRINT FORMAT
*
PAY10     EQU    PSTART+5,50
PAY20     EQU    PAY10+50,50
LEGENDO   EQU    PAY20+50,25
VENDORO   EQU    LEGENDO+25,5
CHECKO    EQU    VENDORO+5,5
NAMEO     EQU    CHECKO+5,26
ADDR10    EQU    NAMEO+26,25
DATEO     EQU    ADDR10+25,6
AMOUNTO   EQU    DATEO+6,13
ADDR20    EQU    AMOUNTO+13,25
CITYO     EQU    ADDR20+25,25
**********************************************************************
*                RECORD AREAS
**********************************************************************
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 27 of 29)

```
                  Y$SSY104                              .SECURITY RECORD
+*
+************** TABLE MASTER RECORD
+*
+KTABLEMT DS     CL8
+RTABLEMT DS     CL80
+TABSTS   EQU    RTABLEMT+08,1 STATUS
+LIMIT    EQU    RTABLEMT+15,1 PASSWORD LIMIT
+TERMTAB  EQU    RTABLEMT+16 TERMINAL FIELDS
 *
 ************** ACCOUNTS PAYABLE
 *
 KACCTPAY DS     CL15                                 .KEY
 RACCTPAY DS     CL165                                .RECORD
 *
 *        AP100  HEADER
 *
 ACCTPAYH DS     CL165
 HBATOT   EQU    ACCTPAYH+21,5                         BATCH TOTAL
 HCHKCNT  EQU    ACCTPAYH+26,5                         NEXT CHECK COUNTER
 HTYPE    EQU    ACCTPAYH+31,1                         CHECK TYPE
 HCHECK   EQU    ACCTPAYH+32,5                         CHECK NUMBER
 HDATE    EQU    ACCTPAYH+37,6                         CHECK DATE
 HVENDOR  EQU    ACCTPAYH+43,5                         CHECK VENDOR
 HAMOUNT  EQU    ACCTPAYH+48,5  PD2                    CHECK AMOUNT
 HITMTOT  EQU    ACCTPAYH+53,5  PD2                    CHECK ITEM TOTAL
 HITMCNT  EQU    ACCTPAYH+58,3                         CHECK ITEM COUNT
 HNAME    EQU    ACCTPAYH+61,26                        PAYEE
 HLEGEND  EQU    ACCTPAYH+87,26                        LEGEND
 HPRINT   EQU    ACCTPAYH+113,1                        CHECK PRINT
 HBATCH   EQU    ACCTPAYH+114,3                        BATCH NUMBER
 HCHKS    EQU    ACCTPAYH+117,3                        NUMBER OF CHECKS
 HITEMS   EQU    ACCTPAYH+126,4                        ITEM COUNT
 HOLD     EQU    ACCTPAYH+130,5 PD2                    OLD CHECK AMOUNT
 HCASH    EQU    ACCTPAYH+135,6 PD2                    CHECK CASH TOTAL
 HACCR    EQU    ACCTPAYH+140,6 PD2                    CHECK ACCRUAL TOTAL
 HERRCDE  EQU    ACCTPAYH+145,1                        CHECK ERROR CODE
 HACTION  EQU    ACCTPAYH+146,1                        CHECK ACTION CODE
 HCOMPL   EQU    ACCTPAYH+147,1                        CHECK COMPLETION CODE
 *
 *        AP103  CHECK
 *
 ACCTPAYC DS     CL165
 CAMOUNT  EQU    ACCTPAYC+29,5  PD2                    CHECK AMOUNT
 CDATE    EQU    ACCTPAYC+20,4
 CVENDOR  EQU    ACCTPAYC+24,5
 CNAME    EQU    ACCTPAYC+34,26
 CADDR1   EQU    ACCTPAYC+60,25
 CADDR2   EQU    ACCTPAYC+85,25
 CCITY    EQU    ACCTPAYC+110,26
 CZIP     EQU    ACCTPAYC+136,3  PDC
 CLEGEND  EQU    ACCTPAYC+139,25
 CPRINT   EQU    ACCTPAYC+164,1
 *
 *        AP104  ITEM
 *
```

Figure C–11. APITMS Action Program Processing a Dialog (Part 28 of 29)

```
1933 ACCTPAYI DS     CL165
1934 AIRID    EQU    ACCTPAYI+00,2                 .RECORD ID
1935 AITYPE   EQU    ACCTPAYI+02,1                 .CHECK TYPE
1936 AICHECK  EQU    ACCTPAYI+03,5                 .CHECK NUMBER
1937 AICNT    EQU    ACCTPAYI+08,3                 .ITEM COUNT
1938 AIVENDOR EQU    ACCTPAYI+19,5                 .VENDOR
1939 AIACCT   EQU    ACCTPAYI+24,8                 .ACCOUNT NUMBER
1940 AIAMT    EQU    ACCTPAYI+32,5                 .AMOUNT
1941 AIEMP    EQU    ACCTPAYI+53,4                 .EMPLOYEE
1942 AIDESCPT EQU    ACCTPAYI+57,30                .DESCRIPTION
1943 AIBATCH  EQU    ACCTPAYI+87,3                 .BATCH #
1944 AIERR    EQU    ACCTPAYI+90,3                 .ERROR BATCH #
1945 AICOA    EQU    ACCTPAYI+93,1                 .CASH OR ACCRUAL
1946 *
1947 *        GL001  ACCOUNT MASTER
1948 *
1949 KACCTMST DS     CL8
1950 RACCTMST DS     CL80
1951 AMSTS    EQU    RACCTMST+8,1                   STATUS
1952 *
1953 *        SY000  BRANCH MASTER
1954 *
1955 KBRANCHM DS     CL3
1956 RBRANCHM DS     CL250
1957 BMSTS    EQU    RBRANCHM,1                     STATUS
1958 *
1959 *        GL003  CHART OF ACCOUNTS
1960 *
1961 KACCOUNT DS     CL8
1962 RACCOUNT EQU    RBRANCHM+50,80
1963 CASTS    EQU    RACCOUNT+8,1                   STATUS
1964 CACOA    EQU    RACCOUNT+38,1                  CASH OR ACCRUAL
1965 CAEXP    EQU    RACCOUNT+46,1                  EXPENSE ACCOUNT
1966 CAINC    EQU    RACCOUNT+49,1                  INCOME ACCOUNT
1967 *
1968 *        PE010  PERSONNEL MASTER
1969 *
1970 KPAYROLL DS     CL5
1971 RPAYROLL DS     CL421
1972 PMSTS    EQU    RPAYROLL,1                    STATUS
1973 PMCAL    EQU    RPAYROLL+172,1                 CLASSIFICATION
1974 *
1975 *        SY002  PERSONNEL CLASSIFICATION
1976 *
1977 TMSTS    EQU    RTABLEMT+8,1
1978 TMEXP    EQU    RTABLEMT+32,1
1979 OMA      Y$$OMA 2568
1980+EW       EQU    * .END OF WORK AREA
2052 CDA      Y$$CDA
2053+*************************************************************************
2054+*                   CONTINUITY DATA AREA                                *
2055+*************************************************************************
2056+CDA      DSECT
2057+         DS     0H
2058          END
```

Figure C-11. APITMS Action Program Processing a Dialog (Part 29 of 29)

C.5. SAMPLE IMS CONFIGURATION

Programs receive DICE sequences

Figure C-12 is a sample IMS configuration of the SUPPLY, APCHKS, APITMS, and APAUDT action programs. Notice these programs are prepared to receive DICE sequences and therefore the EDIT=NONE parameter is specified in the ACTION sections of this configuration.

```
NETWORK BATCH=NO CONFID=002 NAME=GTN1 PASSWORD=GTN1 TERMS=14
GENERAL AUDITNUM=50 CHRS/LIN=80 LNS/MSG=24 MAXCONT=3880
OPTIONS         CONTOUT=NO DLLOAD=NO FUPDATE=YES
        OPCOM=YES
        RECOVERY=NO RESEND=NO
        SNAPED=NO
        SUBPROG=YES TOMFILE=NO  TOMTRCE=NO
        UNIQUE=TRAN
        UNSOL=NO
TIMEOUTS ACTION=60
        STATUS=30
FILE    BRANCHM FILETYPE=ISAM BLKSIZE=0512 LOCK=TR
               TYPEFLE=RANSEQ UPDATE=YES RECFORM=FIXBLK PCYLOFL=20
               RECSIZE=250 KEYLOC=10 KEYLEN=3
               IOAREA1=BRANCHM KEYARG=BRANCHM WORK1=BRANCHM
               IOROUT=ADDRTR IOREG=8 WORKS=YES
               INDAREA=BRANCHM INDSIZE=256
FILE    TABLEMT FILETYPE=ISAM BLKSIZE=0512 LOCK=TR
               TYPEFLE=RANSEQ UPDATE=YES RECFORM=FIXBLK PCYLOFL=30
               RECSIZE=080 KEYLOC=0 KEYLEN=8
               IOAREA1=TABLEMT KEYARG=TABLEMT WORK1=TABLEMT
FILE    SCRFIL  FILETYPE=DAMR BLKSIZE=2560 IOAREA1=SCRFIL READID=YES
               RELATIVE=R SEEKADR=SCRFIL WRITEID=YES LOCK=UP
FILE    PAYROLL FILETYPE=ISAM BLKSIZE=1280 LOCK=TR
               TYPEFLE=RANSEQ UPDATE=YES RECFORM=FIXBLK PCYLOFL=30
               RECSIZE=421 KEYLOC=6 KEYLEN=5
               IOAREA1=PAYROLL KEYARG=PAYROLL WORK1=PAYROLL
               IOROUT=ADDRTR IOREG=8 WORKS=YES
FILE    ACCOUNT FILETYPE=ISAM BLKSIZE=0512 LOCK=TR
               TYPEFLE=RANSEQ UPDATE=YES RECFORM=FIXBLK PCYLOFL=10
               RECSIZE=080 KEYLOC=0 KEYLEN=8
               IOAREA1=ACCOUNT KEYARG=ACCOUNT WORK1=ACCOUNT
               IOROUT=ADDRTR IOREG=8 WORKS=YES
FILE    ACCTMST FILETYPE=ISAM BLKSIZE=0512 LOCK=TR
               TYPEFLE=RANSEQ UPDATE=YES RECFORM=FIXBLK PCYLOFL=20
               RECSIZE=080 KEYLOC=0 KEYLEN=8
               IOAREA1=ACCTMST KEYARG=ACCTMST WORK1=ACCTMST
               IOROUT=ADDRTR IOREG=8 WORKS=YES
FILE    ACCTPAY FILETYPE=ISAM BLKSIZE=1022 LOCK=TR
               TYPEFLE=RANSEQ UPDATE=YES RECFORM=FIXBLK PCYLOFL=40
               RECSIZE=165 KEYLOC=0 KEYLEN=15
               IOAREA1=ACCTPAY KEYARG=ACCTPAY WORK1=ACCTPAY
               IOROUT=ADDRTR IOREG=8 WORKS=YES
```

Figure C-12. Sample IMS Configuration (Part 1 of 2)

```
FILE      VENDORM  FILETYPE=ISAM BLKSIZE=1022 LOCK=TR
                   TYPEFLE=RANSEQ UPDATE=YES RECFORM=FIXBLK PCYLOFL=10
                   RECSIZE=199 KEYLOC=0 KEYLEN=5
                   IOAREA1=VENDORM KEYARG=VENDORM WORK1=VENDORM
                   IOROUT=ADDRTR IOREG=8 WORKS=YES
FILE      TRANACT  FILETYPE=ISAM BLKSIZE=1022 LOCK=TR
                   TYPEFLE=RANSEQ UPDATE=YES RECFORM=FIXBLK PCYLOFL=30
                   RECSIZE=165 KEYLOC=0 KEYLEN=15
                   IOAREA1=TRANACT KEYARG=TRANACT WORK1=TRANACT
                   IOROUT=ADDRTR IOREG=8 WORKS=YES
TERMINAL TM08     IMSREADY=NO
TERMINAL TM07     IMSREADY=NO
TERMINAL TM06     IMSREADY=NO
TERMINAL TM09     IMSREADY=NO
TERMINAL TM04
TERMINAL TM10     IMSREADY=NO
TERMINAL TM05     IMSREADY=NO
TERMINAL TM11
TERMINAL TM03
TERMINAL TM12      MASTER=YES
TERMINAL TM01
TRANSACT APCKS    ACTION=APCHKS
TRANSACT APITS    ACTION=APITMS
TRANSACT APAUD    ACTION=APAUDT
TRANSACT SUPLY    ACTION=SUPPLY
ACTION    APITMS   EDIT=NONE
                   FILES=ACCOUNT,ACCTMST,ACCTPAY,BRANCHM,PAYROLL
                   FILES=SCRFIL,TABLEMT,VENDORM
                   INSIZE=STAN MAXSIZE=9472 OUTSIZE=2568 WORKSIZE=3584
                   ALLRNT=NO  BYPASS=2 MAXUSERS=1
ACTION    APAUDT   EDIT=NONE
                   FILES=ACCOUNT,ACCTPAY,BRANCHM,SCRFIL,PAYROLL
                   FILES=TABLEMT,VENDORM
                   INSIZE=STAN MAXSIZE=8960 OUTSIZE=2568 WORKSIZE=3072
                   ALLRNT=NO  BYPASS=2 MAXUSERS=1
ACTION    APCHKS   EDIT=NONE
                   FILES=ACCTPAY,PAYROLL,SCRFIL,TABLEMT,VENDORM
                   INSIZE=STAN MAXSIZE=7936 OUTSIZE=2568 WORKSIZE=2048
                   ALLRNT=NO  BYPASS=2 MAXUSERS=1
ACTION    SUPPLY   EDIT=NONE
                   FILES=BRANCHM,TABLEMT,TRANACT
                   INSIZE=STAN MAXSIZE=2304 OUTSIZE=STAN WORKSIZE=1024
                   ALLRNT=NO  BYPASS=5 MAXUSERS=1
PROGRAM   APITMS ERET=YES TYPE=SER
PROGRAM   APCHKS ERET=YES TYPE=RNT
PROGRAM   APAUDT ERET=YES TYPE=RNT
PROGRAM   SUPPLY ERET=YES TYPE=SER
```

Figure C-12. Sample IMS Configuration (Part 2 of 2)

Appendix D. Status and Detailed Status Codes

Results from function call execution

IMS returns a status code and sometimes both status and detailed status codes after each function call issued by your action program. IMS places these codes in the STATUS-CODE and DETAILED-STATUS-CODE fields of the program information block. Your action program then tests the contents of these program information block fields and performs routines to handle the conditions indicated by them.

Status codes

Table D-1 shows the status codes and their meaning for sequential and random functions issued to sequential, relative, indexed, and defined files.

Detailed status codes

Table D-2 shows detailed status codes IMS returns with invalid key status code 1.

Table D-3 describes detailed status codes IMS returns with status code 3 for invalid request errors.

Table D-4 lists detailed status codes returned by IMS with status code 6 for internal message control errors.

Table D-5 explains detailed status codes returned with status code 7 for screen formatting errors.

Table D-1. Status Codes for I/O Function Calls

Status Codes	Sequential Functions												Random Functions															Status Code Meaning
	Seq. Files		Relative Files			Indexed File				Defined Files			Relative Files					Indexed Files					Defined Files					
	GET	PUT	SETL	GET	ESETL	SETL	SETK	GET	ESETL	SETL	GET	ESETL	GET	GETUP	PUT	INSERT	DELETE	GET	GETUP	PUT	INSERT	DELETE	GET	GETUP	PUT	INSERT	DELETE	
0	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Successful
0											X																	Detail cycle
1			X										X	X	X	X	X											Invalid record number
1						X												X	X				X	X		X		Invalid key
1				X							X																	Invalid record type
2	X			X				X																				End of file (DAM files only)
2	X	X		X				X					X	X	X	X	X	X	X	X	X	X						Unallocated optional file (MIRAM files only)
2											X																	Total cycle
3	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Invalid request
4	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	I/O error
5																							X	X	X	X	X	Violation of data definition

Table D-2. Detailed Status Codes for Invalid Key Errors (Status Code 1)

Code (Hexadecimal)	Description	Meaning
01	Invalid duplicate key count	Duplicate key count value on random GET function is zero or exceeds number of duplicate keys.
E1	No identifier supplied	Insert an IDENTIFIER statement in the data definition.
E2	Identifier too long	Identifier must be 1 to 30 alphanumeric characters.
E4	Identifier out of range	Value entered at terminal is not in range of VALUE clause specified in Data Definition.

Table D–3. Detailed Status Codes for Invalid Requests (Status Code 3) (Part 1 of 2)

Code (Hexadecimal)	Description	Meaning
01	Incorrect number of parameters	The number of parameter addresses contained in a request parameter list in inconsistent with the function requested. This error can result from the failure of BAL action programs to set the sign bit in the final address word in a request parameter list as required by standard linkage conventions.
02	Function code out of legal range	This error may occur when an action program inadvertently writes into the IMS link module that is linked to a serially reusable or sharable action program, or control passes improperly from an action program to IMS.
03	Incorrect parameter value	The parameter list address passed to IMS on a request is 0, or an address contained in the parameter list is 0, or the actual value of a parameter is incorrect. This error can also occur when an I/O area for a DAM file was not half-word aligned.
04	Shared record not in use by this transaction	This code does not apply to user action program requests.
05	File not defined	A logical or defined file named in a request to IMS is not configured or defined via the data definition processor.
06	File not open	The ZZCLS master terminal command closed a logical file named in a request to IMS or data management closed a logical file as the result of an unrecoverable error.
07	Function invalid for type of file	The function specified in a request to IMS is not valid for the type of file named. For example, the action program issued a SETL function call for a nonindexed file.
08	Record(s) not locked	The action program issued an UNLOCK function when no locks exited.
09	PUT or DELETE request not preceded by a GETUP request	The function sequence for an update operation is not valid.

Table D-3. Detailed Status Codes for Invalid Requests (Status Code 3) (Part 2 of 2)

Code (Hexadecimal)	Description	Meaning
0A	Illegal function requested	The requested function is not consistent with the DTF or RIB parameters in the configuration.
0B	File not assigned to this action	The action program requested a logical file that was not named in the configured definition of the action making the request, or the preceding action did not name a defined file.
0C	Required module not included in configuration	The action program requested a feature not included in the IMS load module at configuration time.
0D	Capacity exceeded on INSERT request	An action program requested insertion of a record into a MIRAM or ISAM file, but insufficient space exists to contain the new record.
0E	Insufficient space in main	User must allocate more main storage.
0F	Update not permitted in configuration	An action program requested an update function, but update was disallowed at configuration time.
10	Update suppressed for files	The requested update is not permitted because of an I/O error in the audit file.
11	Trace file down	File recovery is not operational; only file displays are allowed.
12	Record was locked by another transaction (single-thread only)	Under single-thread, an action program issued either a GETUP or INSERT request on a record, but this record was already locked by some other transaction.
14	Work-area address invalid or SETLOAD was not issued before GETLOAD	Check the order in which you issued SETLOAD and GETLOAD calls; make sure that work area is word aligned.
15	Data buffer too small (less than 10 bytes)	Make sure the value specified on the size parameter of the GETLOAD call is greater than 10.
16	Another SETLOAD call was issued between the initial SETLOAD and the GETLOAD call	Check that an additional SETLOAD call was not issued before the GETLOAD call.

Table D-4. Detailed Status Codes for I/O Errors (Status Code 4)

File Type	Error Code Description	
MIRAM	DMnn	nn is the hexadecimal value of data management area error code contained in the first byte of the detailed status code. The second byte of detailed status code is error subcode interpretation. (See 3.6 and system messages programmer/operator reference, UP-8076 (current version).)
DAM SAM ISAM	filenameC+2	Is the value in the detailed status code. For interpretation, refer to data management user guide, UP-8068 (current version).

Table D-5. Detailed Status Codes for Internal Message Control Errors (Status Code 6) (Part 1 of 2)

Detailed Status Code (Hexadecimal)	Description	Meaning
02	Destination terminal busy, on hold, or down	Output-for-input queueing was requested and: 1. Destination terminal is in interactive mode. 2. Destination terminal has an input message on queue. 3. ZZHLD or ZZDWN command was entered for destination terminal. 4. Destination terminal is marked physically down to ICAM. 5. IMS cannot allocate main storage buffer (multithread) only; INBUFSIZ specification inadequate.
03	Destination terminal physically or logically down; message queued	SEND function was issued for message switching. Message is queued at destination terminal and is retransmitted when terminal becomes operational.
04	Invalid specification in output message header	Invalid destination terminal-id or auxiliary-device-id; or aux-function field contains X'C3', X'F3', or X'F7' (not valid with SEND function).
05	No ICAM network buffer available	Insufficient buffer space allocated in ICAM network definition.

Table D-5. Detailed Status Codes for Internal Message Control Errors (Status Code 6) (Part 2 of 2)

Detailed Status Code (Hexadecimal)	Description	Meaning
06	Disk error	Output error occurred on attempt to write message to disk; error passed to IMS by ICAM.
07	Invalid length specification	In delayed internal succession or output-for-input queueing, output message length was larger than the input buffer pool.

Table D-6. Detailed Status Codes for Screen Formatting Errors (Status Code 7) (Part 1 of 2)

Detailed Status Code (Hexadecimal)	Description	Meaning
00	Validation error; all error fields in variable data area replaced by hexadecimal F's and affected field-error statuses set in the output-status area	Check validation error codes returned in status byte for invalid field
01	Buffer address indicates a format area not large enough to receive the screen format	Check the length field in output message header portion of format area to find actual length required for the format described
02	Variable data area not large enough	Check data-size parameter on BUILD function
03	Insufficient number of terminals configured for SFS	Check SFS parameter in the OPTIONS section of configurator
04	Variable data specified when no variable data area exists	Variable-data parameter specified in BUILD function but no output fields or option indicator bytes described in action program.
05	Format dimensions are greater than screen dimensions	Check screen format generation for length of screen format.
06	Fatal error; I/O error reading format file	Get DM error message from console; refer to OS/3 system messages programmer/ operator reference, UP-8076 (current verson).

Table D–6. Detailed Status Codes for Screen Formatting Errors (Status Code 7) (Part 2 of 2)

Detailed Status Code (Hexadecimal)	Description	Meaning
07	REBUILD not allowed	User issued output-only screen and can issue a REBUILD only with input fields.
08	Invalid field in variable data area	On REBUILD, data description in action program doesn't match screen format generation.
09	Variable-data parameter specified but no error field detected	Screen coordinator checked all data in variable-data area and no fields of hexadecimal F's found.
0A	Screen format incorrectly generated	On BUILD, data description in action program doesn't match screen format generation.
0B	SFS failed	System error. Take dump and write software user report (SUR).
10	SFS failed during input conversion	Inadequate main storage in system; or format contains protected fields and terminal does not have protect feature or is not in protect mode.
11	Screen format services error	Take IMS job dump and write SUR.
12	Screen format can't be transmitted because this is a program-initiated DDP transaction.	Action program processing DDP transaction attempted to send screen format to initiating action program.

Appendix E. Generating Edit Tables

E.1. PURPOSE

The edit table generator offers a convenient means for converting unformatted input received from terminal operators into fixed formats required by action programs and checking this input for types of data, value ranges, and presence of required fields.

Edit table generator output

The output of the edit table generator is written to the named record file (NAMEREC). From there it is loaded at the appropriate time by IMS. Each edit table is associated with a particular action at configuration time via the EDIT parameter in an ACTION section. The edit table utility can be run either before or after configuration, but the NAMEREC file must be previously initialized.

E.2. GENERATOR INPUT CODING RULES FOR EDIT TABLE

Edit table generator input parameters

Input to the edit table generator is in the form of keyword parameters that define the edit table, the fields you want edited, and the edit criteria for each field. Note that the statement conventions in Appendix A also apply.

EDIT TABLE GENERATOR CODING RULES

To code input to the edit table generator, apply the following rules:

Sequence numbers

1. Input entries must contain sequence numbers in columns 77 through 80, in ascending order. The lowest permissible sequence number is 0001.

Where to code parameters

2. Parameters can be coded in any column between 1 and 76. Blanks are ignored and are permitted anywhere in the edit table definition.

Example:

```
1                                                         77   8Ø
SEP=;ETAB=ETABTST;KEY=1;POS=Ø;MAN=Y;LEN=5;                Ø 1 Ø Ø
KEY=2;FIL= ;JUS=L;LEN=15;MAN=Y;TYP=A;POS=5;               Ø 2 Ø Ø
KEY=3;FIL= ;JUS=L;LEN=2Ø;POS=2Ø;TYP=M;;                   Ø 3 Ø Ø
```

Spanning lines

3. Specifications for an edit table and for each field can span more than one line. However, a keyword and its value must be contained on one line.

Example:

INCORRECT

```
SEP=;ETAB=ETABTST;KEY=1;POS=     Ø1ØØ
0;MAN=Y;LEN=5;MAN=Y;LEN=5;;      Ø2ØØ
```

KEYWORD AND VALUE NOT ON SAME LINE

CORRECT

```
SEP=;ETAB=ETABTST;KEY=1;POS=Ø;    Ø1ØØ
```

New line

4. A new edit table specification must start on a new line. Each field need not begin on a new line.

Example:

INCORRECT

```
SEP=;ETAB=ETABTST;KEY=1;POS=Ø;   Ø1ØØ
MAN=Y;LEN=5;                     Ø2ØØ
KEY=2;FIL= ;JUS=L;LEN=15;MAN=Y;  Ø3ØØ
TYP=A;POS=5;;SEP=,ETAB=TABL1,    Ø4ØØ
KEY=1,LEN=20,POS=2Ø,,            Ø5ØØ
```

NEW EDIT TABLE NOT SPECIFIED ON NEW LINE

CORRECT

```
SEP=;ETAB=ETABTST;KEY=1;POS=Ø;        Ø1ØØ
MAN=Y;LEN=5;KEY=2;FIL= ;JUS=L;        Ø2ØØ
LEN=15;MAN=Y;TYP=A;POS=5;;            Ø3ØØ
SEP=,ETAB=TABL1,KEY=1,LEN=2Ø,         Ø4ØØ
POS=2Ø,,                              Ø5ØØ
```

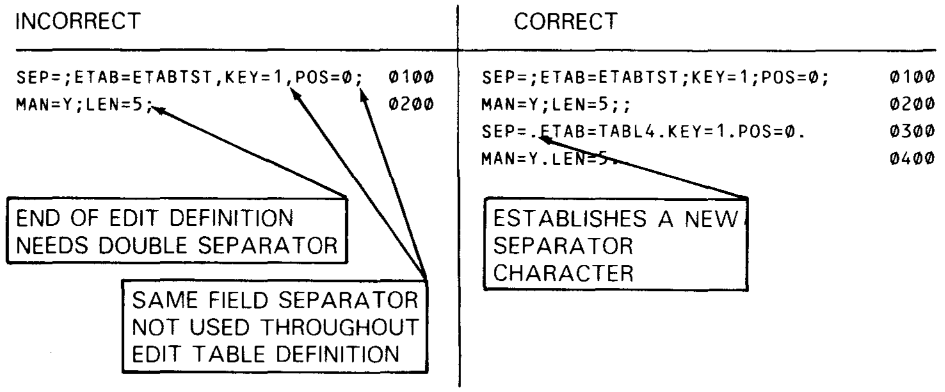
NEW FIELD NEED NOT START ON NEW LINE

Field separator character

Changing separator character

5. The field separator character specified by the SEP keyword parameter must be used as the field separator throughout the edit table specification, as well as in the input message to be edited. Double separator characters indicate the end of the edit definition. A new edit table can establish a different separator character.

Example:


INCORRECT
CORRECT
END OF EDIT DEFINITION NEEDS DOUBLE SEPARATOR
SAME FIELD SEPARATOR NOT USED THROUGHOUT EDIT TABLE DEFINITION
ESTABLISHES A NEW SEPARATOR CHARACTER

INCORRECT

```
SEP=;ETAB=ETABTST,KEY=1,POS=Ø;  Ø1ØØ
MAN=Y;LEN=5;                    Ø2ØØ
```

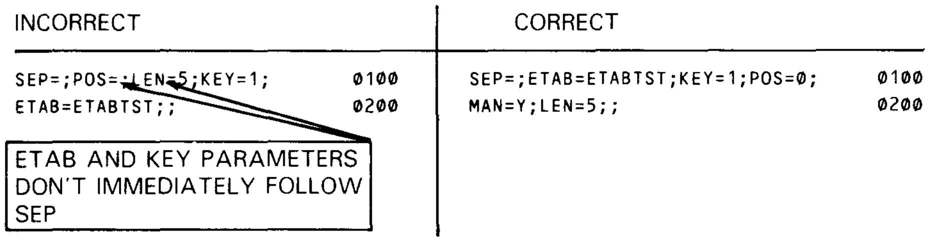
CORRECT

```
SEP=;ETAB=ETABTST;KEY=1;POS=Ø;     Ø1ØØ
MAN=Y;LEN=5;;                      Ø2ØØ
SEP=.ETAB=TABL4.KEY=1.POS=Ø.       Ø3ØØ
MAN=Y.LEN=5..                      Ø4ØØ
```

Order of parameters

6. The SEP, ETAB, and KEY parameters must be coded in the prescribed order; the remaining keyword parameters can be specified in any order. SEP and ETAB are coded once for each edit table. The remaining parameters are repeated for each field in the input message to be edited.

Example:



INCORRECT

```
SEP=;POS=;LEN=5;KEY=1;       Ø1ØØ
ETAB=ETABTST;;               Ø2ØØ
```

CORRECT

```
SEP=;ETAB=ETABTST;KEY=1;POS=Ø;     Ø1ØØ
MAN=Y;LEN=5;;                      Ø2ØØ
```

Numeric values

7. Numeric values are positive unless preceded by a minus sign (-). The plus sign (+) is not permitted in numeric values.

Example:

INCORRECT	CORRECT
`SEP=;ETAB=TABL1;KEY=1;LEN=5;    Ø1ØØ`	`SEP=;ETAB=TABL1;KEY=1;LEN=5;    Ø1ØØ`
`POS=0;MAX=+20000Ø;MIN=-1;;      Ø2ØØ`	`POS=0;MAX=2ØØØØ;MIN=-1;;        Ø2ØØ`
PLUS SIGN NOT ALLOWED	
NUMBER OF CHARACTERS EXCEEDS LENGTH GIVEN IN LEN PARAMETER	

E.3. EDIT TABLE GENERATOR PARAMETERS

Input parameter format

The input parameters you give to the edit table generator should follow this format:

```
SEP=separator-character
ETAB=tablename
KEY={keyword }
    {position}
LEN=field-length
POS=starting-position
[FIL=fill-character]
[JUS={L}]
[    {R}]
[MAN={N}]
[    {Y}]
[MAX=maximum-value]
[MIN=minimum-value]
[TYP={A}]
[    {B}]
[    {M}]
[    {N}]
[    {P}]
```

Separator character (SEP)

The separator parameter specifies the field separator character for both the edit table definition and the input message to be edited. It cannot be a blank, equal sign, or minus sign. This parameter is required, must be the first entry on the first line of the edit table definition, and can be specified only once per edit table.

Edit table name (ETAB)

The edit table name parameter names the edit table and must immediately follow the SEP parameter. This specification associates the edit table with an action at configuration, via the EDITtablename option in the ACTION section.

Key field identification (KEY)

The key field parameter identifies the input message field for which edit criteria are specified in subsequent parameters and must be the first parameter specified for each field. The edit table generator associates all subsequent specifications with this field until it encounters another KEY parameter. Input fields can be positional or keyword. Positional fields precede keyword fields.

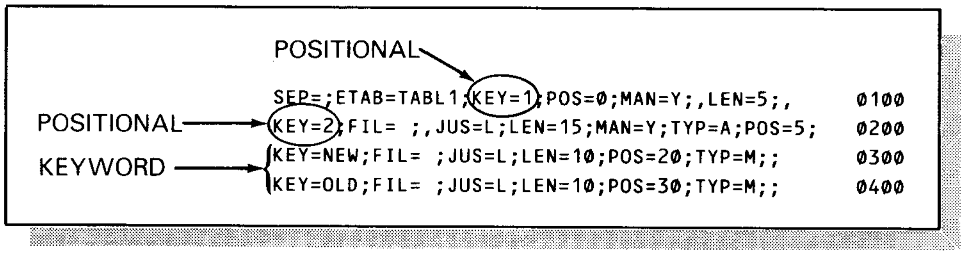
Positional fields

KEY=*position* specifies the relative position of the field as it appears in the input message. Positional fields must be defined in numeric order, starting with 1.

Keyword fields

KEY=*keyword* specifies a 1- to 3-character alphanumeric identification. The first character must be alphabetic, for a keyword field in the input message. The terminal operator enters keyword fields in the form *keyword=data.* For example, when you specify KEY=OLD, the terminal operator might enter OLD=57500 for this field. Once a keyword field is identified in the edit table definition, all subsequent fields must be defined as keyword fields.

Figure E-1 shows the correct coding for positional and keyword parameters to the edit table generator.



```
SEP=;ETAB=TABL1;KEY=1;POS=Ø;MAN=Y;,LEN=5;,         Ø1ØØ
KEY=2;FIL= ;,JUS=L;LEN=15;MAN=Y;TYP=A;POS=5;       Ø2ØØ
KEY=NEW;FIL= ;JUS=L;LEN=1Ø;POS=2Ø;TYP=M;;          Ø3ØØ
KEY=OLD;FIL= ;JUS=L;LEN=1Ø;POS=3Ø;TYP=M;;          Ø4ØØ
```

Figure E-1. Edit Table Parameter Description with Positional and Keyword Parameters

Edited field length (LEN)

The length parameter specifies the length of the edited field and is a required parameter. You may specify a maximum of 255 characters for alphanumeric fields and four characters for binary fields. Ten characters is the maximum length for numeric fields unless you specify both MIN and MAX parameters for this field. If you identify a numeric field in the action program as packed decimal, you can specify up to 16 characters in the LEN parameter.

NOTES:

Field-length longer than screen width

1. *If the field-length is larger than the width of the screen on which data is to be entered, IMS removes the DICE code at the end of each line of terminal input and replaces it with a blank character. You must provide for these additional blank characters in the action program and include them in the field-length specified by the LEN parameter.*

Binary and packed field lengths

2. *The length specified for binary (TYP=B) and packed (TYPP) fields is the maximum length for the field in the input message, not the length of the field in your program. For example, if a field is defined as packed with a LEN=3, the largest number the terminal operator can key in is 999, even though 1000 may be represented in a packed field in 3 bytes.*

Transaction codes under five characters

3. *If the transaction code (the first field in the input message) is less than five characters, the terminal operator must key in a space before entering the separator character for the next field. You must include the space in the field-length specified by the LEN parameter.*

TRANSACTION CODE IS PAY

SO

OPERATOR ENTERS

AND LEN=4;

Transaction code field larger than five characters

The length of the first field can be greater than five characters, but only the first five characters are used in the transaction code. The LEN parameter should specify the actual length of the field.

Field starting position (POS)	The starting position parameter specifies the starting position of this field as it appears in the edited message and is a required parameter. The first field starts at 0.
Fill character identification (FIL)	The fill character parameter optionally specifies the fill character inserted in the edited field when the field the terminal operator enters as input is shorter than the field-length specified by the LEN parameter. The default fill character is 0. If you want to fill with spaces (X'40'), code either FIL= or FIL=△; i.e., you can include or omit a space before the separator character for the next field. Binary fields are always filled with binary zeros; therefore, this parameter is ignored if specified for a binary field.
Field justification (JUS)	JUS=L left-justifies this field in the edited message. Binary and packed fields are always right-justified; therefore, this parameter is ignored if specified for binary or packed fields. JUS=R right-justifies this field in the edited message and is the default assumed.
Mandatory field (MAN)	MAN=N indicates that this field is not mandatory in the edited message for input to be acceptable. MAN=Y indicates that this field is mandatory in the edited message.
Maximum value limitation (MAX)	The maximum value parameter specifies the maximum value allowed for the field in the input message. This parameter applies only to numeric fields. The highest value allowed is 2 to the thirty-first power minus 1, (2^{31}-1). The number of characters in this value must not exceed the length specified by the LEN parameter.
Minimum value limitation (MIN)	The minimum value parameter specifies the minimum value allowed for the field in the input message. This parameter applies only to numeric fields. The lowest value allowed is minus 2 to the thirty-first power minus 1 (-(2^{31}-1)). The number of characters in this value must not exceed the length specified by the LEN parameter.
Data type (TYP)	The type parameter describes the type of data to be contained in the edited field.

TYP=A specifies alphabetic data. A field defined to the editor as alphabetic is treated as an alphanumeric field.

TYP=B specifies binary data.

TYP=M specifies alphanumeric data and is the default value.

TYP=N specifies numeric data.

TYP=P specifies packed decimal data.

E.4. EXECUTING THE EDIT TABLE GENERATOR

Job control stream

Once you code input parameters describing the edit table format and the NAMEREC file is initialized, you can execute the ZH#EDT edit table generator using the control stream illustrated in Figure E-2.

```
// JOB ADDEDT,,A000
// DVC 20 // LFD PRNTR
// OPTION DUMP
// DVC 50 // VOL DS9999 // LBL NAMEREC,DS9999 // LFD NAMEREC
// EXEC ZH_EDT
/$
      input  parameters
        .         .
        .         .
        .         .
      input  parameters
/*
/&
// FIN
```

Figure E-2. Sample Execution of Edit Table Generator

When execution is successful

If the input definition is acceptable, the generated edit table is written to the NAMEREC file and the following message is issued:

```
tablename ADDED
```

Duplicate edit table name

If the edit table has the same name as a table already existing in the NAMEREC file, the new edit table replaces the existing table, and the following message is issued:

```
TABLE ADDED, DUPLICATE DELETED
```

Errors in edit table generation

If errors cause rejection of the edit table, the following message is issued:

```
tablename REJECTED
```

UPSI byte values

Another way to determine edit table errors is to look at the UPSI byte. The following UPSI byte values pertain to the edit table error status:

UPSI Byte Contents	Meaning
00	No errors
40	Warning. ZH#EDT continues processing edit table input parameters but no edit table is built.
80	Fatal error. Edit table processing terminates.

E.5. ERROR PROCESSING

Warning errors

When the edit table generator encounters a file I/O error or certain types of input errors, it terminates and prints a message in the output listing. The resulting value in the UPSI byte is 80. Most types of input errors do not cause termination. Processing and validation continues, but an error message is printed and the edit table is rejected. Input specifications for the edit table generator are not printed in the output listing. This type of error results in an UPSI byte value of 40.

Fatal errors

If an I/O error occurs while reading input to the edit table generator, the following message is issued, and the program terminates with an UPSI byte value of 80:

```
INPUT READ ERROR, SCAN TERMINATED
```

If an error occurs while opening, reading, or closing the named record file, the following error message is issued and the program terminates with an UPSI byte value of 80:

```
FILE ERROR, SCAN TERMINATED
```

Error message format

Errors in the input statements are reported in the following format:

```
nnnn cc error-message-text
```

where:

`nnnn`
: Is the sequence number in columns 77 through 80 of the card containing the error.

`cc`
: Is the column number of the beginning of the input text that is in error. This column number is suppressed if the error is detected during final validation of all parameters for a given field.

`error-message-text`
: Is the description of the error as listed in Table E-1.

Error message example

An example of an input statement error and the resultant error message follows:

Input:

```
SEP=,ETAB=EDIT1,KEY=1,LEN=5,POS=Ø,JUS=X,MAN=Y,       ØØØ2
```

Error message:

```
ØØØ2 39 JUSTIFICATION ILLEGAL
```

Table E-1 lists alphabetically the message texts inserted into the input statement error message. In each case, processing continues, unless otherwise indicated in the explanation column.

Table E-1. Edit Table Diagnostic Messages (Part 1 of 2)

Error Message Text	Explanation
B TYPE LENGTH GR THAN 4	Four characters (one full word) is maximum
CARDS NOT IN SEQUENCE	Scan terminated, run aborted*
DOUBLE SEPARATOR MISSING	Warning only; end-of-file encountered while searching for separator
DUPLICATE NAME	Duplicate name for nonpositional field
FIELD NOT ACCEPTED, KEYS STARTED	Positional parameters not allowed after nonpositionals started
FIELD NOT IN SEQUENCE	Positional parameters must be in sequence
FILLER MUST BE SINGLE CHARACTER	Self-explanatory
ILLEGAL FIELD TYPE	Only A, B, M, N, or P accepted
INVALID MAN SPECIFICATION	Only Y or N accepted
INVALID NAME	Name too long or contains invalid characters
INVALID SEPARATOR	Scan terminated, run aborted; = and - are not allowed as separators*
JUSTIFICATION ILLEGAL	Only R or L accepted
KEYWORD ETAB MISSING	Self-explanatory
KEYWORD INVALID	Self-explanatory

* These errors set the UPSI byte to 80; all other errors in this table result in an UPSI byte value of 40.

Table E-1. Edit Table Diagnostic Messages (Part 2 of 2)

Error Message Text	Explanation
KEYWORD KEY= MISSING	Self-explanatory
KEYWORD SEP= MISSING	Scan terminated, run aborted*
LEN OR POS EXCEEDS MAX	Maximum length is 255; maximum position is 32,767
LEN OR POS MISSING	Required parameters
LEN ZERO	Length must be at least 1
MAX OR MIN ABSOLUTE VALUE TOO LARGE	$2^{31}-1$ is largest absolute value allowed
N TYPE LENGTH GR THAN 10	Ten characters is maximum unless MAX and MIN both specified
NO DEFAULT FOR THIS FIELD	Parameter value must be specified
NO FIELDS DEFINED	Empty table not allowed
P TYPE LENGTH GR THAN 16	Sixteen characters maximum for packed decimal field
REPEATED FIELD	Parameter already specified
SEPARATOR CHARACTER MISSING	Self-explanatory
SEQUENCE NUMBER NOT NUMERIC	Scan terminated, run aborted*
= SIGN MUST FOLLOW KEYWORD	Self-explanatory
TOO MANY FIELDS	Scan terminated, run aborted; output buffer overflow*
xxx OVERLAPS yyy	Warning only; overlapping fields permitted

* These errors set the UPSI byte to 80; all other errors in this table result in an UPSI byte value of 40.

E.6. ENTERING INPUT MESSAGES FROM TERMINAL

When the terminal operator enters an input message for which you've generated an edit table, an IMS component called the expanded input editor processes it. The following considerations apply when entering input messages from the terminal:

Transaction code first

- When an input message contains a transaction code, the transaction code must always be the first field. If the transaction code is less than five characters, enter a space before keying in the separator character.

Beginning positional fields

Omitting positional fields

- Positional fields begin with the first nonblank character and extend to the next separator. Positional fields must appear in the same order as specified in the edit table definition. If you omit a positional field, enter an additional separator character in its position. A positional field entered as input may not contain an equal sign.

Keyword fields

- Keywords must be followed by an equal sign with no intervening blanks. Data starts immediately after the equal sign and extends to the next field separator.

Invalid plus sign

- Numeric values are positive unless preceded by a minus sign. The plus sign (+) is an invalid character.

Error messages screen placement

- Error messages are displayed on the first line of the display terminal; therefore, we recommend that you start input messages on the second line so that the input is not erased by an error message.

Continuing fields

- If you continue fields from one line to another, IMS removes the DICE code at the end of each line and replaces it with a blank character, which it sends to the action program as part of the data. Always enter on one line fields that do not exceed the width of the screen. If a field exceeds the screen width and must be continued from one line to another, avoid splitting a word between lines.

Ending input with positional parameters

- If the terminal input ends with a positional parameter (no keyword parameters are specified), enter a separator character at the end of the input message; otherwise, the input message could be partially deleted. A correct terminal entry is:

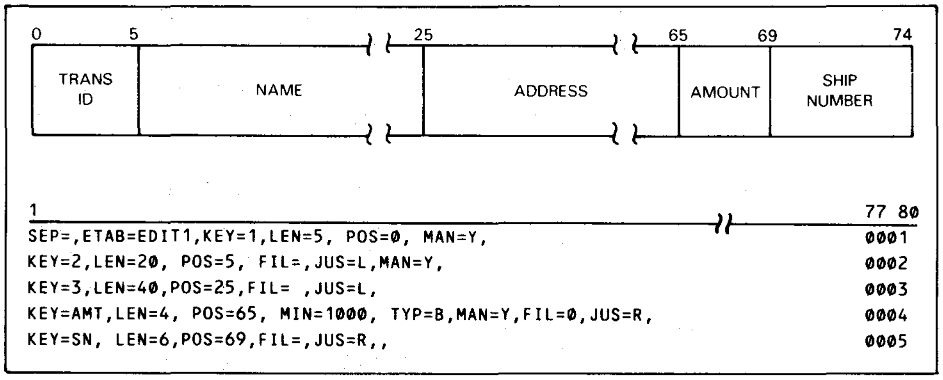
```
INFOR,BIOLOGY,CLASS2,MARY J. BLISS,
```

  When terminal input ends with a keyword parameter, this is not necessary.

E.7. SAMPLE EDIT TABLE APPLICATION USING POSITIONAL AND KEYWORD PARAMETERS

Example edit table input

Figure E-3 and Table E-2 describe sample input to the edit table generator for an accounts receivable application and the format in which the edited input is delivered to the action program.



```
SEP=,ETAB=EDIT1,KEY=1,LEN=5, POS=0, MAN=Y,                           0001
KEY=2,LEN=20, POS=5, FIL=,JUS=L,MAN=Y,                               0002
KEY=3,LEN=40,POS=25,FIL= ,JUS=L,                                     0003
KEY=AMT,LEN=4, POS=65, MIN=1000, TYP=B,MAN=Y,FIL=0,JUS=R,            0004
KEY=SN, LEN=6,POS=69,FIL=,JUS=R,,                                    0005
```

Figure E-3. Sample Input to Edit Table Generator and Format of Input Delivered to Action Program

Table E-2. Description of Sample Input to Edit Table Generator (Part 1 of 2)

Line	Parameter	Explanation
1	SEP=,	The field separator is a comma for both the edit specification and input from the terminal.
	ETAB=EDIT1	The edit table name is EDIT1.
	KEY=1	The first field described is positional. It must be the first field in the input message.
	LEN=5	The edited field is five characters long.
	POS=0	In the edited message the field begins in position 0.
	MAN=Y	The field must be present for the message to be acceptable.
2	KEY=2	The field is positional. It must be the second field in the input message.
	LEN=20	The edited field is 20 characters long.
	POS=5	In the edited message the field begins in position 5.
	FIL=	The field is to be blank filled in the edited message.
	JUS=L	The field is to be left-justified in the edited message.
	MAN=Y	The field must be present for the message to be acceptable.

Table E-2. Description of Sample Input to Edit Table Generator (Part 2 of 2)

Line	Parameter	Explanation
3	KEY=3	The field is positional. It must be the third field in the input message.
	LEN=40	The edited field is 40 characters long.
	POS=25	In the edited message, the field begins in position 25.
	FIL=	The field is to be blank filled in the edited message.
	JUS=L	The field is to be left-justified in the edited message.
4	KEY=AMT	The field is a keyword field. AMT=n must be specified in the input message.
	LEN=4	The edited field is three characters long.
	POS=65	In the edited message, the field begins in position 65.
	MIN=1000	The minimum level allowed for the message to be acceptable is $10.00 (entered as 1000).
	TYP=B	In the edited message, the field is to be converted to binary.
	MAN=Y	The field must be present for the message to be acceptable.
	FIL=0	The field is to be zero filled in the edit message. (This parameter could have been omitted.)
	JUS=R	The field is to be right-justified in the edited message. (This parameter could have been omitted.)
5	KEY=SN	The field is a keyword field.
	LEN=6	The edited field is six characters long.
	POS=69	In the edited message, the field begins in position 68.
	FIL=	The field is to be blank filled in the edited message.
	JUS=R	The field is to be right-justified in the edited message. (This parameter could have been omitted.)
	-	End of edit definition.

Example freeform input

The following examples show freeform input from the terminal and the resulting messages sent to the action program in accordance with the edit table specifications or, in case of error, the output message displayed at the terminal. Note that in the edited messages, the 4-character binary field specified for the AMT entry is represented by an underlined, 4-hexadecimal-digit field. Spaces between each delimiter and the first character of the next field are ignored.

Terminal input

```
PAYMT, JOHN D. SMITH,1112 BREEZE DR. PHILA.PA. 19160,
AMT=2500,SN=123456
```

Edited message received by action program

```
PAYMTJOHN△D.△SMITH△△△△△△△1112△BREEZE△DR.△PHILA.△PA.△19160
△△△△△△△△09C4123456
```

Terminal input

```
PAYMT,JOHN D. SMITH,,SN=123456,AMT=2500
```

Edited message received by action program

```
PAYMTJOHN△D.△SMITH△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△
△△△△△△△△△△△△△09C4123456
```

Explanation

The address field was not specified as mandatory in the edit table input and is omitted here; an additional comma is coded in its position. The AMT and SN fields are keyword fields and need not be entered in the order defined in the edit table input.

Terminal input

```
PAYMT ,JOHN D. SMITH,1112 BREEZE DR. PHILA. PA. 19160,
AMT=2500,SN=123456
```

Output message

```
ILLEGAL INPUT
```

Explanation The transaction code field is longer than the LEN specification.

Terminal input

```
PAYMT,JOHN D. SMITH,1112 BREEZE DR. PHILA. PA.1916Ø,
AMT=7ØØ,SN=123456
```

Output message

```
AMT IS BELOW MIN
```

Explanation Edit table specifies AMT must be at least 1000.

Terminal input

```
PAYMT, JOHN D. SMITH,1112 BREEZE DR. PHILA. PA. 1916Ø,SN=123456
```

Output message

```
AMT MISSING
```

Explanation AMT was specified as mandatory.

E.8. SAMPLE EDIT TABLE APPLICATION INCLUDING ACTION PROGRAM

Sample input parameters This sample application describes an edit table for a customer purchase/payment application and includes the action program that uses edit table input.

Edit Table for the Purchase/Payment Application

Figure E-4 describes the input to the edit table generator.

```
SEP=;ETAB=ETABTST;KEY=1;POS=0;MAN=Y;LEN=5;                              0100
KEY=2;FIL= ;JUS=L;LEN=15;MAN=Y;TYP=A;POS=5;                             0200
KEY=3;FIL= ;JUS=L;LEN=20;POS=20;TYP=M;                                  0300
KEY=4;MIN=0001;MAX=9999;TYP=B;LEN=4;POS=40;MAN=Y;                       0400
KEY=5;MIN=-99999999;MAX=99999999;TYP=P;POS=44;LEN=8;MAN=Y;              0500
KEY=6;FIL=0;MIN=-20000;MAX=999999999;TYP=N;POS=52;LEN=10;MAN=Y;;        0600
```

Figure E-4. Sample Input to Edit Table Generator

Input message description

Line 100 designates a semicolon as the field separator for both the edit specification and the input from the terminal. The edit table is named ETABTST. The first input field is positional and is the transaction code. The field begins in position 0, is mandatory, and is 5 characters long.

Line 200 describes the second input field as positional with blank-fill where the input entry is shorter than 15 characters. This second field is left-justified, 15 characters long, mandatory, alphanumeric, and begins in position 5.

Line 300 describes the third input field as positional with blank-fill, left-justified, 20 characters long and alphanumeric. The TYPM parameter is not required because it is the default.

Line 400 describes the fourth input field as positional and allows a value of not less than 1 and not more than 9999 with a length of 4 characters. In the edited message, the field is converted to binary and begins in position 40. The field is mandatory.

Line 500 describes the fifth input field as positional with a minimum value of -99999999 and a maximum value of 99999999 in packed decimal format. The field begins in position 44, is 8 characters long, and is mandatory.

Line 600 describes the sixth input field as positional with a zero fill character, minimum value of -20000 and maximum value of 999999999 in numeric format beginning in position 52 for a length of 10 characters. The field is mandatory.

Action Program (EDITST) for Purchase/Payment Application

Figure E-5 provides the EDITST action program coding that processes the input message received from the edit table and issues an output message to the terminal.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EDITST.
INSTALLATION. SPERRY-UNIVAC,BLUE BELL,PA.
DATE-WRITTEN. FEBRUARY 1978.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. UNIVAC-OS3.
OBJECT-COMPUTER. UNIVAC-OS3.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  CRT                       PIC X(4) VALUE IS  '    '.
01  NXT-LNE                   PIC X(4) VALUE IS '    '.
01  DEPOSIT                   PIC X(8)  VALUE IS 'PURCHASE'.
01  WITHDRAW                  PIC X(7)  VALUE IS 'PAYMENT'.
01  LINE3-HEAD.
    05  NAME                  PIC X(4)  VALUE 'NAME'.
    05  FILLER                PIC X(26) VALUE SPACE.
    05  ADDRESS               PIC X(7)  VALUE 'ADDRESS'.
    05  FILLER                PIC X(23) VALUE SPACE.
    05  ACCOUNT               PIC X(7)  VALUE 'ACCOUNT'.
    05  FILLER                PIC X(13) VALUE SPACE.
01  LINE6-HEAD.
    05  TRANSACT              PIC X(8)  VALUE 'TRANSACT'.
    05  FILLER                PIC X(12) VALUE SPACE.
    05  AMOUNT                PIC X(6)  VALUE 'AMOUNT'.
    05  FILLER                PIC X(14) VALUE SPACE.
    05  BALANCEO              PIC X(12) VALUE 'BALANCE(OLD)'.
    05  FILLER                PIC X(8)  VALUE SPACE.
    05  BALANCEN              PIC X(12) VALUE 'BALANCE(NEW)'.
    05  FILLER                PIC X(8)  VALUE SPACE.
LINKAGE SECTION.
01  PIB.     COPY     PIB74.
    02  STATUS-CODE                      PIC 9(4) COMP-4.
    02  DETAILED-STATUS-CODE             PIC 9(4) COMP-4.
    02  RECORD-TYPE REDEFINES DETAILED-STATUS-CODE.
        03 PREDICTED-RECORD-TYPE     PIC X.
        03 DELIVERED-RECORD-TYPE     PIC X.
    02  SUCCESSOR-ID                     PIC X(6).
    02  TERMINATION-INDICATOR            PIC X.
    02  LOCK-ROLLBACK-INDICATOR          PIC X.
    02  TRANSACTION-ID.
        03 YEAR                          PIC 9(4) COMP-4.
        03 TODAY                         PIC 9(4) COMP-4.
        03 HR-MIN-SEC                    PIC 9(9) COMP-4.
    02  DATA-DEF-REC-NAME                PIC X(7).
    02  DEFINED-FILE-NAME                PIC X(7).
    02  STANDARD-MSG-LINE-LENGTH     PIC 9(4) COMP-4.
    02  STANDARD-MSG-NUMBER-LINES PIC 9(4) COMP-4.
    02  WORK-AREA-LENGTH                 PIC 9(4) COMP-4.
    02  CONTINUITY-DATA-INPUT-LENGTH  PIC 9(4) COMP-4.
```

Figure E-5. Sample Action Program (EDITST) Using Edit Table Generator Input (Part 1 of 3)

SAMPLE EDIT TABLE APPLICATIONS

```
00051              02  CONTINUITY-DATA-OUTPUT-LENGTH  PIC 9(4) COMP-4.
00052              02  WORK-AREA-INC               PIC 9(4) COMP-4.
00053              02  CONTINUITY-DATA-AREA-INC    PIC 9(4) COMP-4.
00054              02  SUCCESS-UNIT-ID.
00055                  03 TRANSACTION-DATE.
00056                     04  YEAR                 PIC 99.
00057                     04  MONTH                PIC 99.
00058                     04  TODAY                PIC 99.
00059                  03 TIME-OF-DAY.
00060                     04  HOUR                 PIC 99.
00061                     04  MINUTE               PIC 99.
00062                     04  SECOND               PIC 99.
00063                  03 FILLER                   PIC XXX.
00064              02  SOURCE-TERMINAL-CHARS.
00065                  03 SOURCE-TERMINAL-TYPE  PIC X.
00066                  03 SOURCE-TERM-MSG-LINE-LENGTH   PIC 9(4) COMP-4.
00067                  03 SOURCE-TERM-MSG-NUMBER-LINES  PIC 9(4) COMP-4.
00068              02  DDP-MODE                    PIC X.
00069          01  IMA.     COPY     IMA74.
00070              02 SOURCE-TERMINAL-ID               PIC X(4).
00071              02 DATE-TIME-STAMP.
00072                 03  YEAR                         PIC 9(4) COMP-4.
00073                 03  TODAY                        PIC 9(4) COMP-4.
00074                 03  HR-MIN-SEC                   PIC 9(9) COMP-4.
00075              02 TEXT-LENGTH                      PIC 9(4) COMP-4.
00076              02 AUXILIARY-DEV-ID.
00077                 03  FILLER                       PIC X.
00078                 03  AUX-DEV-NO                   PIC X.
00079              02  LINE-1-IN.
00080               07  TRANSACT          PIC X(5).
00081               07  IN-NAME           PIC A(15).
00082               07  IN-ADDR           PIC X(20).
00083               07  IN-ACC-NO         PIC 9(8)  COMP.
00084               07  IN-AMOUNT         PIC S9(13)V99 COMP-3.
00085               07  IN-BALANCE        PIC S9(8)V99.
00086          01  OMA.     COPY     OMA74.
00087              02  DESTINATION-TERMINAL-ID       PIC X(4).
00088              02  SFS-OPTIONS                   PIC X(2).
00089              02  FILLER                        PIC X(2).
00090              02  CONTINUOUS-OUTPUT-CODE        PIC X(4).
00091              02  TEXT-LENGTH                   PIC 9(4)    COMP-4.
00092              02  AUXILIARY-DEVICE-ID.
00093                  03  AUX-FUNCTION              PIC X.
00094                  03  AUX-DEVICE-NO             PIC X.
00095              02   OUTPUT-MSG-TEXT.
00096              03   LINE1-DICE        PIC X(4).
00097              03   LINE1-OUT         PIC X(60).
00098              03   LINE2-DICE        PIC X(4).
00099              03   LINE3-DICE        PIC X(4).
00100              03   LINE3-HEADER      PIC X(60).
00101              03   LINE4-DICE        PIC X(4).
00102              03   LINE4-OUT.
00103               05  NAMEALP           PIC A(15).
00104               05  FILLER            PIC X(15).
00105               05  ADDR-ALPNUM       PIC X(20).
00106               05  FILLER            PIC X(10).
00107               05  ACC-NO-BIN        PIC 9(8).
00108               05  FILLER            PIC X(12).
00109              03   LINE5-DICE        PIC X(4).
00110              03   LINE6-DICE        PIC X(4).
```

Figure E-5. Sample Action Program (EDITST) Using Edit Table Generator Input (Part 2 of 3)

```
    03  LINE6-HEADER       PIC X(80).
    03  LINE7-DICE         PIC X(4).
    03  LINE7-OUT.
     05 TYPE-TRANS         PIC X(8).
     05 FILLER             PIC X(12).
     05 AMOUNT-PAC         PIC 9(14).99CR.
     05 FILLER             PIC X(2).
     05 BAL-OLD-NUM        PIC 9(8).99CR.
     05 FILLER             PIC X(8).
     05 BAL-NEW-NUM        PIC 9(8).99CR.
    03  LINE8-DICE         PIC X(4).
01  WORK.
    05  UNPAC-AMT          PIC 9(14)V99.
PROCEDURE DIVISION USING PIB IMA WORK OMA.

HOUSEKEEPING.
    MOVE CRT TO LINE1-DICE.
    MOVE NXT-LNE TO LINE2-DICE, LINE3-DICE, LINE4-DICE,


        LINE5-DICE, LINE6-DICE, LINE7-DICE, LINE8-DICE.
    MOVE TRANSACT OF LINE-1-IN TO LINE1-OUT.
    MOVE LINE3-HEAD TO LINE3-HEADER.
    MOVE LINE6-HEAD TO LINE6-HEADER.
INPUT-CHECK.
    MOVE IN-NAME TO NAMEALP.
    MOVE IN-ADDR TO ADDR-ALPNUM.
    MOVE IN-ACC-NO TO ACC-NO-BIN.
    IF IN-AMOUNT IS LESS THAN 0 THEN MOVE WITHDRAW TO TYPE-TRANS
        ELSE MOVE DEPOSIT TO TYPE-TRANS.
    MOVE IN-AMOUNT TO AMOUNT-PAC.
    MOVE IN-BALANCE TO BAL-OLD-NUM.
    ADD IN-AMOUNT , IN-BALANCE
            GIVING BAL-NEW-NUM.
    MOVE 430 TO TEXT-LENGTH OF OMA.
EXIT-PROG.
    CALL 'RETURN'.
```

Figure E-5. Sample Action Program (EDITST) Using Edit Table Generator Input (Part 3 of 3)

Processing the Purchase/Payment Application

Unformatted terminal input

When the terminal operator enters the unformatted input – transaction code, name, address, account number, amount, and balance as follows:

```
WIDEP;JAN HALS;1422 AMBER LN PHILA;472;11ØØØ;35ØØØ
```

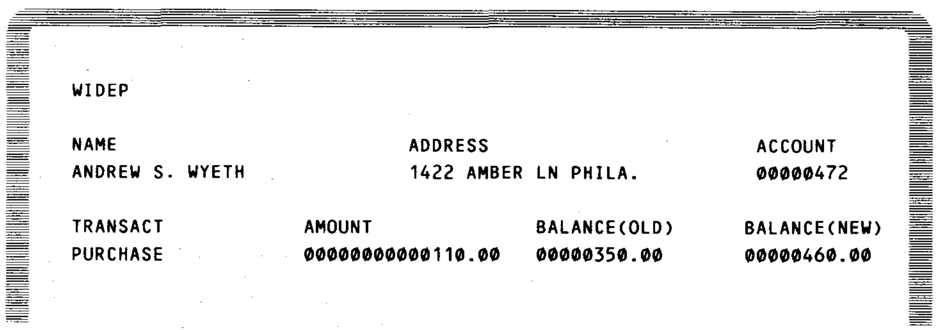
the edit table generator formats the input according to your edit table input parameters (Figure E-4), and the action program EDITST (Figure E-5) receives this edited input in its input message area as follows:

```
WIDEP;JAN△HALS△△△△△△△;1422△AMBER△LN△PHILA△;Ø1D8;
 ØØØ11ØØØ;ØØØØØ35ØØØ
```

Note that for easier identification in this example, the binary account field expected as input to the action program is shown here as a hexadecimal value and underlined.

Formatted input received by EDITST

The EDITST action program receives this input message giving the old balance and payment amount, computes a new balance, and generates a 5-line output message as follows:



```
Line 1   WIDEP
     2
     3   NAME                         ADDRESS                      ACCOUNT
     4   ANDREW S. WYETH              1422 AMBER LN PHILA.         00000472
     5
     6   TRANSACT       AMOUNT                BALANCE(OLD)      BALANCE(NEW)
     7   PURCHASE       0000000000110.00      00000350.00       00000460.00
```

Generating output message

In the Procedure Division, EDITST moves the transaction code into the first line of the output message, double spaces, moves the NAME-ADDRESS-ACCOUNT header to line 3, double spaces, moves the TRANSACT-AMOUNT-BALANCES header to line 6, and begins computations based on your terminal input.

EDITST places the name, address, and account number entered at the terminal in line 4 of the output message. Note that the account number entered at the terminal is decimal; however, the edit table generator converts this number to binary and EDITST receives it as a binary field.

Accommodating packed and binary fields

Note that in your action program, any fields describing decimal values keyed in at the terminal must be defined large enough to accommodate the field as received from the edit table generator. For example, an 8-digit decimal number entered as an amount from the terminal and defined by LEN=8 and TYP=P in the edit table parameters (Figure E-4, line 500) is defined in the program's input and output message texts as a 16-byte packed field (Figure E-5, line 84 and 116). This field sizing also applies to binary values.

Next, EDITST tests the amount field (IN-AMOUNT) entered as input to see if it is less than zero. If the amount entered was negative, it was for payment; otherwise, it was for purchase. EDITST moves these respective constants to the output message area.

After this, the program moves the input amount and old balance to the output message area and adds either the negative payment amount or the positive purchase amount to the old balance giving the new balance.

Finally, the total output message text length is moved to the output message area TEXT-LENGTH field before the RETURN function ends the transaction. When the RETURN function executes, EDITST sends the type transaction, amount of payment or purchase, old balance, and new balance to line 7 of the output message and, the entire output message text to the designated lines.

Appendix F. Using Device Independent Control Expressions and Field Control Characters

F.1. GENERAL

Using DICE for formatting

You use device independent control expressions (DICE sequences) to format input and output messages handled by action programs. These codes are needed to control various operations, such as cursor positioning and carriage return, on a terminal screen.

Scope of section

This appendix supplies all DICE sequences and their interpretations, describes how to use them in formatting messages in your action programs, and discusses the DICE macroinstructions used in BAL action programs to create the DICE sequences. In addition, it presents limited information concerning the use of field control characters.

F.2. FORMATTING MESSSAGES

Output Messages

There are numerous methods for formatting output messages. The action program can use:

Ways to format messages

1. Screen format services. For a complete discussion of how to use screen format services, see Section 7.
2. Device independent control expressions
3. Format control expressions with UNISCOPE 100 and 200 display terminals
4. Field control characters (FCCs) with workstations and Universal Terminal System terminals

DICE and FCCs

Format control expressions

This appendix supplies information on DICE sequences and how to use them. Also included is information concerning field control characters. For detailed information concerning format control expressions, consult the UNISCOPE display terminal programmer reference, UP-7807 (current version).

Use of format control characters

When a program uses format control expressions, it must include a different formatting routine for each type of terminal receiving the output. Figure F-1 illustrates this.

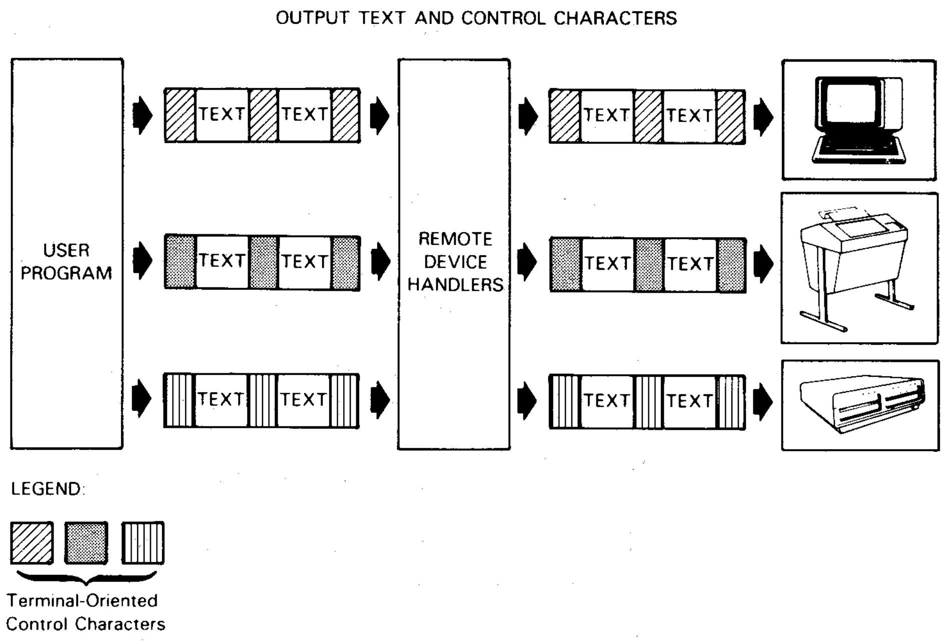

OUTPUT TEXT AND CONTROL CHARACTERS
USER PROGRAM
TEXT
TEXT
REMOTE DEVICE HANDLERS
LEGEND:
Terminal-Oriented Control Characters

Figure F-1. Using Terminal-Oriented Control Characters to Format Messages

Handling DICE sequences

Using DICE sequences to format messages eliminates this problem. The remote device handler converts DICE sequences to control characters for each destination terminal, regardless of type. Some of the control character functions are:

Functions performed

- Line feed – cursor movement to the first space of a new line
- Form feed – cursor to the home position of a new page
- Carriage return – cursor to the beginning of the same line
- Cursor movement to a specific row and column on a display

DICE placement

You can place DICE sequences anywhere in a message. As you can see in Figure F-2, DICE sequences simplify message formatting.

Coding with DICE

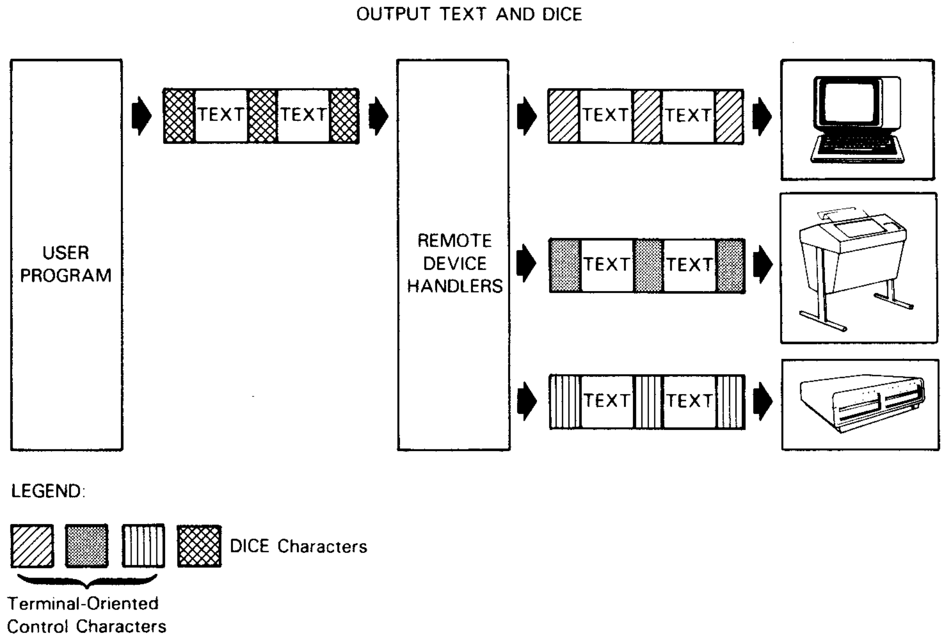


Figure F–2. Using Dice Sequences to Format Messages

Input Messages

Using input DICE

For input, the remote device handler converts control characters received in a message into DICE sequences. For certain terminals, your program can analyze these sequences to determine cursor position. In addition, input DICE is handy for message switch applications because control characters in each input message are converted to DICE sequences. The remote device handler converts these sequences into the appropriate control characters for the destination terminal.

Stripping DICE

When you specify EDIT=c or EDIT=tablename in the ACTION section of the IMS configuration, input DICE is stripped from your input message. You should specify EDIT=c or EDIT=tablename in your IMS configuration. (Specify EDIT=tablename only when you generate an edit table for the action. See Appendix E.)

F.3. DICE AND ICAM

Defining DICE at network definition

You can turn DICE on or off when you define your communications network with the DICE operand of the TERM macroinstruction.

$$\texttt{DICE=}\left(\begin{Bmatrix}\underline{\texttt{ON}}\\ \texttt{OFF}\end{Bmatrix}\right)$$

where:

DICE=(ON)

```
DICE=ON
```
Remote device handler creates input DICE according to your input terminal cursor movements.

DICE=(OFF)

```
DICE=OFF
```
Remote device handler does not create input DICE.

DICE=(ON) is recommended

The default is DICE=(ON). We recommend that you specify DICE=(ON) or omit this operand because many IMS features require the use of input DICE. Certain terminal commands and IMS transaction codes are not available when you specify DICE=(OFF).

See ICAM concepts and facilities, UP-8194 (current version), for a detailed explanation of input DICE creation, and the IMS system support functions user guide, UP-8364 (current version), for specific IMS considerations.

F.4. THE FORMAT OF DICE SEQUENCES

The format of a DICE sequence is:

DICE format

select character	function code	m field	n field

select character

Hexadecimal character (10) designates the start of a DICE sequence.

function code

Defines the device control sequence that is recognized by the remote device handlers on input. On output, this code is a 1-character field defining the operation to be performed on the text message. DICE function codes are listed in Table F-1.

m field and n field

Horizontal/vertical positioning

These fields are treated as parameters to the DICE function code. Their actual definition varies and is determined by the individual DICE macroinstruction. Generally, m relates to vertical positioning and n refers to horizontal positioning.

Text message alignment

Cursor movement

These fields may be expressed in absolute values (m_a and n_a) or relative displacement values (m_r and n_r). The absolute values align the text message to the actual location (row and column) on a page or screen. The relative displacement values give a relative location from the present position of the cursor, that is, move cursor two rows down and one column to the right. All values are expressed in hexadecimal notation. If you choose to use DICE macroinstructions, these parameters must be specified.

F.5. USING DICE MACROINSTRUCTIONS IN BAL PROGRAMS

Purpose

DICE macroinstructions let you create DICE sequences (DICE constants) in the same way you would create constants in your program; when the assembler expands a DICE macroinstruction, your program creates a constant at that location.

Output DICE code conversion

On output, when your program is ready to send a message, it moves the DICE constants created from the DICE macroinstructions into the appropriate places in your message before it issues the output request. The remote device handler converts the DICE constants into the corresponding control characters to produce the necessary positioning.

Input DICE code conversion

On input, DICE sequences are automatically created by the remote device handlers unless you specify the DICE=(OFF) parameter in your network definition. Table F-1 lists the DICE macroinstructions, function code generated, and m and n coordinates as they apply to particular devices on input and output.

Specifying m and n coordinates

You must specify m and n coordinates in your program according to the absolute and relative values expressed in Table F-1. m_a and n_a are absolute values of m and n; m_r and n_r are relative displacements of m and n. For CRT terminals, the home position is $(m_a, n_a)=(1,1)$. For character- or page-oriented devices that allow position to top of form, the top-of-form position is $(m_a,n_a)(1,1)$.

Absolute positions

- Absolute Positions

  Absolute positions of m_a and n_a may range as follows:

  m_a ranges 1 to r

  where:

  r = maximum number of rows (CRT), or maximum number of lines per page.

  n_a ranges 1 to c

  where:

  c = maximum number of columns (CRT), or maximum number of character positions per line.

Relative positions

- Relative Displacement

  Relative displacements of m_r and n_r may begin at zero and range to the bottom and right margin of the screen or page.

  If a value of m or n falls outside of the legal range, that value of m or n will cause the following action:

  m_a or n_a = 0 is interpreted as m_a or n_a = 1

Specifying an absolute or relative value for m or n that is greater than the screen or page size causes unpredictable results.

F.6. GENERATING DICE CODES

Macroinstructions are issued to generate the DICE codes.

DICE macro format

LABEL	△OPERATION△	OPERAND
`[symbol]`	`dice-macroinstruction`	`m,n`

where:

Label

`[symbol]`
: An optional alphanumeric character string, from one to eight characters long, that identifies the specific instruction line.

Operation

`dice-macroinstruction`
: You specify the appropriate name from the macroinstruction column of Table F-1 for the desired DICE sequence.

Positional parameter 1

`m`
: A decimal number (0 to 255) indicating the number of lines or rows the terminal should advance before starting output of the message (Table F-1).

Positional parameter 2

`n`
: A decimal number (0 to 255) indicating the number of spaces or columns to the right the terminal should space before starting output of the message (Table F-1).

Examples

```
    1        10     16
   ------------------------
1. NEWLINE  ZO#POS 0,0
2. COORDI   ZO#COORD 5,10
```

1. This DICE sequence causes movement to a new line.

2. New text starts at line 5, column 10.

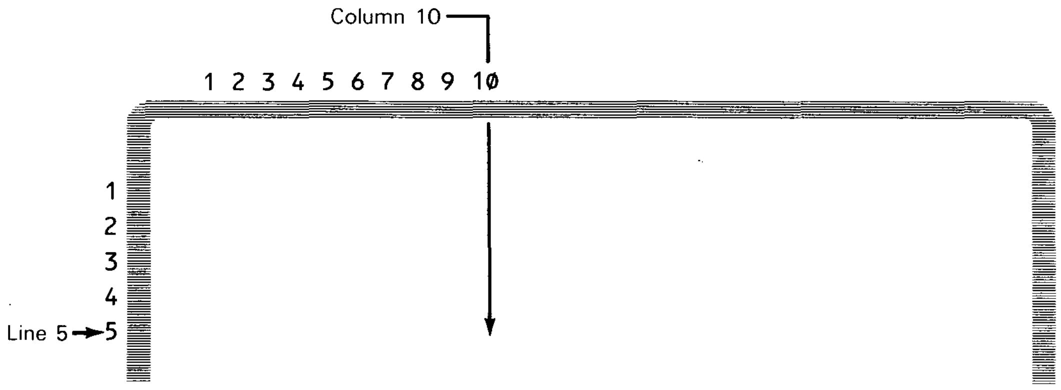


Table F-1. Dice Input/Output Commands, Codes, and Device Interpretation (Part 1 of 4)

DICE Macro-instruction	Function	Function Code Value	I/O	m	n	Character-oriented Devices①	CRT Devices	Page Printing Devices (n is Not Interpreted)	Communications Output Printer (COP) or Terminal Printer (TP)
ZO#COORD	Set coordinates	01_{16}	INPUT	m	n	Not used	m and n represent the start-of-entry (SOE) cursor coordinates.	Not used	Not used
			OUTPUT	m_a	n_a	Action is optional.②	Move cursor to row m and column n.	Action is optional.②	Action is optional.②
ZO#FORM	Forms control	02_{16}	INPUT	01	01	Form feed	Form feed	Not used	Not used
			OUTPUT	m_a	n_a	Form feed, carriage return, and advance to line m and column n (m-1 line feeds and n—1 spaces to the right)	Move cursor to row m and column n.	Top of form and advance to line m (m-1 line feeds)	Form feed, line feed, and advance to line m and column n (m—1 line feeds and n—1 spaces to the right)
ZO#FORMC	Forms control with clear unprotected data	03_{16}	INPUT	—	—	Not used	Not used	Not used	Not used
			OUTPUT	m_a	n_a	Action is optional.②	Move cursor to row m and column n, and clear unprotected data to end of screen.	Action is optional②	Action is optional.②
ZO#POS	New line control	04_{16}	INPUT	00	00	Carriage return, line feed	Cursor return	Not used	Not used
			OUTPUT	m_r	n_r	Carriage return, line feed, followed by m line feeds and n spaces to the right.	Move cursor to beginning of next line. Then move cursor m lines down and n columns to the right	Advance (m+1) lines.	Line feed, followed by m line feeds and n spaces to the right.
ZO#POSC	New line control with clear	05_{16}	INPUT	—	—	Not used	Not used	Not used	Not used
			OUTPUT	m_r	n_r	Carriage return, line feed, followed by m line feeds and n spaces to the right	Same as 04_{16} except area between start and end positions is cleared.	Advance (m+1) lines.	Line feed, followed by m line feeds and n spaces to the right

Table F-1. Dice Input/Output Commands, Codes, and Device Interpretation (Part 2 of 4)

DICE Macro-instruction	Function	Function Code Value	I/O	m	n	Character-oriented Devices①	CRT Devices	Page Printing Devices (n is Not Interpreted)	Communications Output Printer (COP) or Terminal Printer (TP)
ZO#CUR	Current position control	06_{16}	INPUT	01	00	Line feed	Line feed	End of input card	Not used
			OUTPUT	m_r	n_r	m line feeds and n spaces to the right	Move cursor m lines down and n columns to the right.	Advance m lines.	Insert n spaces if nonsignificant space suppression is allowed. If not, insert n DC3 characters; m is not interpreted.③
ZO#CURC	Current position control with clear	07_{16}	INPUT	—	—	Not used	Not used	Not used	Not used
			OUTPUT	m_r	n_r	m line feeds and n spaces to the right	Insert n spaces if nonsignificant space suppression is allowed. If not, insert n DC3 characters; m is not interpreted.③	Advance m lines.	Insert n spaces if nonsignificant space suppression is allowed. If not, insert n DC3 characters; m is not interpreted.③
ZO#BEG	**Beginning of current line control**	**08_{16}**	**INPUT**	00	00	Carriage return	Not used	Not used	Not used
			OUTPUT	m_r	n_r	Carriage return followed by m line feeds and n spaces to the right	Move cursor to beginning of current line. Then move cursor m lines down and n columns to the right.	Advance m lines.	m line feeds and n spaces to the right.
ZO#TABS	Set tab stop at an absolute position④	09_{16}	INPUT	—	—	Not used	Not used	Not used	Not used
			OUTPUT	m_a	n_a	No line feed, space to right.	Set tab stop at row m and column n.	Advance m lines.	Not used
ZO#FORMA	Forms control with clear; protected/unprotected data	• $0A_{16}$	INPUT	—	—	Not used	Not used	Not used	Not used
			OUTPUT	m_a	n_a	Action is optional.②	Move cursor to row m and column n and clear protected/unprotected data to end of screen.	Action is optional.②	Action is optional.②

Table F-1. Dice Input/Output Commands, Codes, and Device Interpretation (Part 3 of 4)

DICE Macro-instruction	Function	Function Code Value	I/O	m	n	Character-oriented Devices(1)	CRT Devices	Pages Printing Devices (n is Not Interpreted)	Communications Output Printer (COP) or Terminal Printer (TP)(1)
ZO#ERSLN	Erase to end of line	$0B_{16}$	INPUT	—	—	Not used	Not used	Not used	Not used
			OUTPUT	m_a	n_a	No action	Cursor does not move. Unprotected data to the end of a line or to the end of the first unprotected field is cleared, whichever comes first.	Advance 0 lines.	Not used

NOTES:

(1) Most character-oriented terminals can be strapped to handle the carriage return (CR) character and the line feed (LF) character as follows:

- CR
  1. print mechanism moves to beginning of the same line; or
  2. print mechanism moves to the beginning of the same line followed by a line feed.
- LF
  1. line feed (no column change); or
  2. line feed followed by return of the print mechanism to the beginning of the new line.

To achieve device independence between terminal types, the character-oriented terminals must use the first option for CR and the first option for LF if the device macroinstruction is ZO#CUR or ZO#BEG.

Use the first option when the character-oriented terminals are a part of a message switch environment.

Certain terminals do not have a form feed capability (i.e., some teletypewriters). For these terminals, the DICE expressions that specify form feed will line feed.

Table F-1. Dice Input/Output Commands, Codes, and Device Interpretation (Part 4 of 4)

② The set coordinates macroinstruction (ZO#COORD) or the forms control with clear macroinstruction (ZO#FORMC), when acted upon by character-oriented or page-printing terminals, will vary in its action, depending on the usage of the DICE keyword parameter of the TERM macroinstruction at network definition time:

$$\textbf{TERM} \ \ldots\textbf{,DICE=}\left(\begin{Bmatrix}\textbf{FORMS}\\ \textbf{NEWLINE}\end{Bmatrix}\right)\textbf{,}\ldots$$

When FORMS is specified, the set coordinates macroinstruction is interpreted as the forms control macroinstruction.

When NEWLINE is specified, the set coordinates macroinstruction and the forms control with clear macroinstruction result in a carriage return, line feed for character-oriented terminals, or advance one line for page-oriented terminals; m and n are not interpreted.

When the DICE parameter is not specified, the default option is NEWLINE.

③ The UNISCOPE display terminal suppresses nonsignificant spaces on each line (except for the line containing the cursor) when text is transmitted to the processor or printed locally on the COP or TP.

Your program may send data to the UNISCOPE screen containing significant blank segments that include the last column of the screen. If this data is transmitted from the terminal to the processor or is printed locally on the COP or TP, the blank segments must consist of nonspace characters that are nondisplayable. The DC3 character meets these qualifications. The ICAM interface provides your program with the capability to prevent nonsignificant space suppression on the UNISCOPE display terminal. The "current position control with clear" is the only DICE macroinstruction that can perform a clear function if your program is preventing nonsignificant space suppression.

NOTE:

The ASCII-to-EBCDIC translation table is modified so that the DC3 character is translated to space 40_{16} for input from the UNISCOPE display terminal.

④ Using DICE function code 09_{16} for setting a tab stop, m=0 and n=0 results in a tab stop being placed at the current cursor location (no cursor positioning is performed). This applies to UNISCOPE and UTS 400 devices only. For teletypewriters and DCT 500 terminals, a space character is inserted.

When m or n is greater than the maximum allowable m or n, action varies depending on the remote terminal:

- UNISCOPE display terminals – wraparound occurs on the screen.
- Character-oriented terminals – gives different results depending on device characteristics.

F.7. INTERPRETING DICE SEQUENCES

Device independent

When using DICE, your program does not need to be aware of the terminal type. A particular DICE denotes the same positioning on any terminal. There are some exceptions that result from terminal limitations.

Factors controlling interpretation of DICE sequences

The interpretation of a DICE by the remote device handler is controlled by:

1. DICE function code
2. DICE m and n fields
3. The terminal involved
4. The particular device on the terminal being used.

Remote terminals supported

The remote device handlers currently provide device-independent support for three classes of remote terminal devices:

Hard copy character-oriented devices

1. Hard copy character-oriented devices, such as the SPERRY UNIVAC Data Communications Terminal 475 (DCT 475), Data Communications Terminal 500 (DCT 500), Data Communications Terminal 524 (DCT 524), and Data Communications Terminal 100 (DCT 1000), and Teletype teletypewriter models 28, 32, 33, 35, and 37.

Hard copy page printer devices

2. Hard copy page printer type device, such as the SPERRY UNIVAC 1004 Card Processor System, Data Communications Terminal 2000 (DCT 2000), and the IBM 2780.

CRT terminals

3. CRT-type terminals, such as the UNISCOPE 100 and 200 and the UTS 400 Display Terminals.

Primary devices

Table F-2 defines the primary output device and the primary input device for each terminal type.

Table F-2. DICE Primary Devices

Terminal Type	Primary Output Device	Primary Input Device
Character-oriented terminals	Printer	Keyboard
Page printing terminals	Printer	Card reader
CRT terminals	Screen	Keyboard

Auxiliary devices supported In addition to the specified primary devices, each terminal has the ability to support one or more auxiliary devices. The auxiliary devices suggested by each terminal are listed in Table F-3.

Table F-3. DICE Usage for Auxiliary Devices (Part 1 of 2)

Remote Terminals	Auxiliary Device	DICE Usage
UNISCOPE	Tape cassette (TCS) Communications output printer (COP) 800 terminal printer (TP)	DICE is applied to the COP. ①
DCT 1000 DCT 500/TTY	Card reader/card punch Paper tape reader/punch Paper tape reader/punch	DICE is applied as if the output/input is to/from the primary device, even though it is for the auxiliary device. ②
DCT 524	Tape cassette (TCS) in paper tape read and write only	
Batch terminals	Punch	DICE is used for end of network buffer sentinel. No forms control action is taken.

NOTES:

① When the print transparent option is not used, DICE is applied to the UNISCOPE screen even though the output is sent to an auxiliary device of the UNISCOPE terminal. In this case, the format of the data printed on the COP or TP is identical to the screen format. Nonsignificant space suppression by the UNISCOPE terminal may have to be prevented to keep the formats identical.

The full capability of DICE cannot be applied to to the COP because of hardware characteristics. All data to a UNISCOPE auxiliary device passes through the UNISCOPE terminal. When DICE is applied to the COP, the use of print transparent mode means that no carriage returns are transferred to the COP. Line feeds and form feeds take a storage position in the UNISCOPE storage and are nondisplayable. These characters are passed to the COP where:

- an LF causes a line feed followed by return of the print mechanism to the beginning of the new line; and
- an FF causes a page eject and positioning of the print mechanism at the beginning of the first line of the form.

The COP has no tabbing capability.

These characteristics are reflected in the interpretation of DICE output function codes for the COP as shown in Table F-2.

For messages sent to a UNISCOPE auxiliary device with transparent transfer, the cursor to home (ESC e) sequence is inserted at the beginning of the text by the RDH.

Table F-3. DICE Usage for Auxiliary Devices (Part 2 of 2)

② The control characters that are generated from the DICE macroinstructions are always created for the primary device of a character-oriented device, even though your program is sending to an auxiliary device. The message and these control characters (carriage returns, line feeds, form feeds, and spaces) will be punched/written by the output auxiliary device that was specified by your program or was switch-selected by the terminal operator. If the punched/written data is later read by the terminal's input auxiliary device, the carriage returns, line feeds, and form feeds are converted to input DICE as specified in Table F-1.

F.8. USING DICE SEQUENCES IN A COBOL ACTION PROGRAM

Though COBOL action programs do not issue DICE macroinstructions, they do use the function code values in PICTURE clauses to position messages and control the cursor. Table F–1 lists and explains the possible DICE input/output commands. The following example of output message coding (Figure F–3) illustrates a COBOL action program's use of DICE sequences to issue the terminal message shown following the code (Figure F–4).

```
01 O-M-A              COPY OMA.
   02  DESTINATION-TERMINAL-ID        PIC X(4).
   02  SFS-OPTIONS.
       03  SFS-TYPE                   PIC X(2).
       03  SFS-LOCATION               PIC X(2).
   02  FILLER                         PIC X(2).
   02  CONTINUOUS-OUTPUT-CODE         PIC X(4).
   02  TEXT-LENGTH                    PIC 9(4)    COMP-4.
   02  AUXILIARY-DEVICE-ID.
       03  AUX-FUNCTION               PIC X.
       03  AUX-DEVICE-NO              PIC X.
   02  OUTPUT-TEXT.
       03  DICE-SEQ-1                 PIC X(4)   VALUE ='100A0A1E'.
       03  LINE-1                     PIC X(22)  VALUE 'YOU USE DICE
                                                  SEQUENCES'.
       03  DICE-SEQ-2                 PIC X(4)   VALUE ='10010C20'.
       03  LINE-2                     PIC X(18)  VALUE 'ON THE OUTPUT FORM'.
       03  DICE-SEQ-3                 PIC X(4)   VALUE ='10040E22'.
       03  LINE-3                     PIC X(14)  VALUE 'TO FORMAT YOUR'.
       03  DICE-SEQ-4                 PIC X(4)   VALUE ='10081026'.
       03  LINE-4                     PIC X(7)   VALUE 'MESSAGE'.
```

Figure F–3. COBOL Action Program Using DICE Sequences to Format Output Message

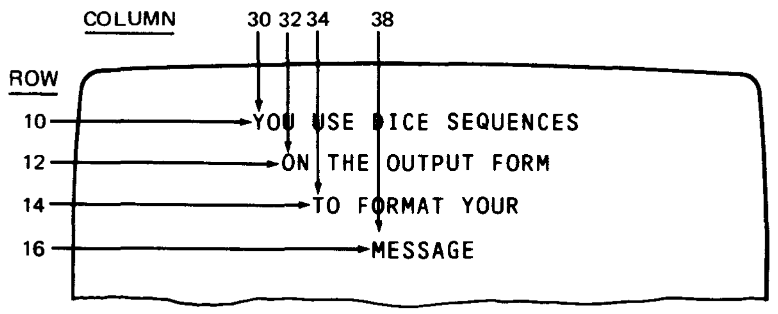


Figure F–4. A DICE Formatted Output Message on the Terminal Screen

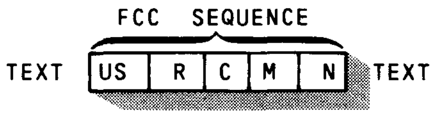
Here is a brief description of the DICE sequences used in Figure F-3.

DICE Sequence	Description
100A0A1E	The select character 10 signals the start of the DICE sequence. The function code (0A) clears all protected and unprotected data from the terminal screen. The m field (0A) and the n field (1E) position the cursor to row 10, column 30.
10010C20	The select character 10 is always the same and signals the start of the DICE sequence. The function code (01) sets coordinates as directed by the m and n fields of the DICE sequence. The m field (0C) and the n field (20) position the cursor at row 12, column 32.
10040E22	The select character is the same as before. The function code (04) moves the cursor to the beginning of the next line and then sets the coordinates as directed by the m and n fields. The m field (0E) and the n field (22) position the cursor two rows below where it presently is and in column 34.
10081026	The select character is again the same. The function code (08) returns the cursor to the beginning of the current line. The m field (10) and the n field (26) position the cursor two rows below the current line and in column 38.

F.9. USING FIELD CONTROL CHARACTERS

Field control character format

Each field control character (FCC) sequence contains a preface control character, a screen row number, screen column number, and two character places that define the screen operations being performed by the sequence. The field control character sequence format is:



US - preface control character

US is the control character that signals the start of a field control character sequence. It corresponds to a hexadecimal 1F.

R - row number

R is the number of the row in which the field control character is placed. This is the hexadecimal value equivalent to the row code for the screen row indicated in Figure F-5.

C - column number

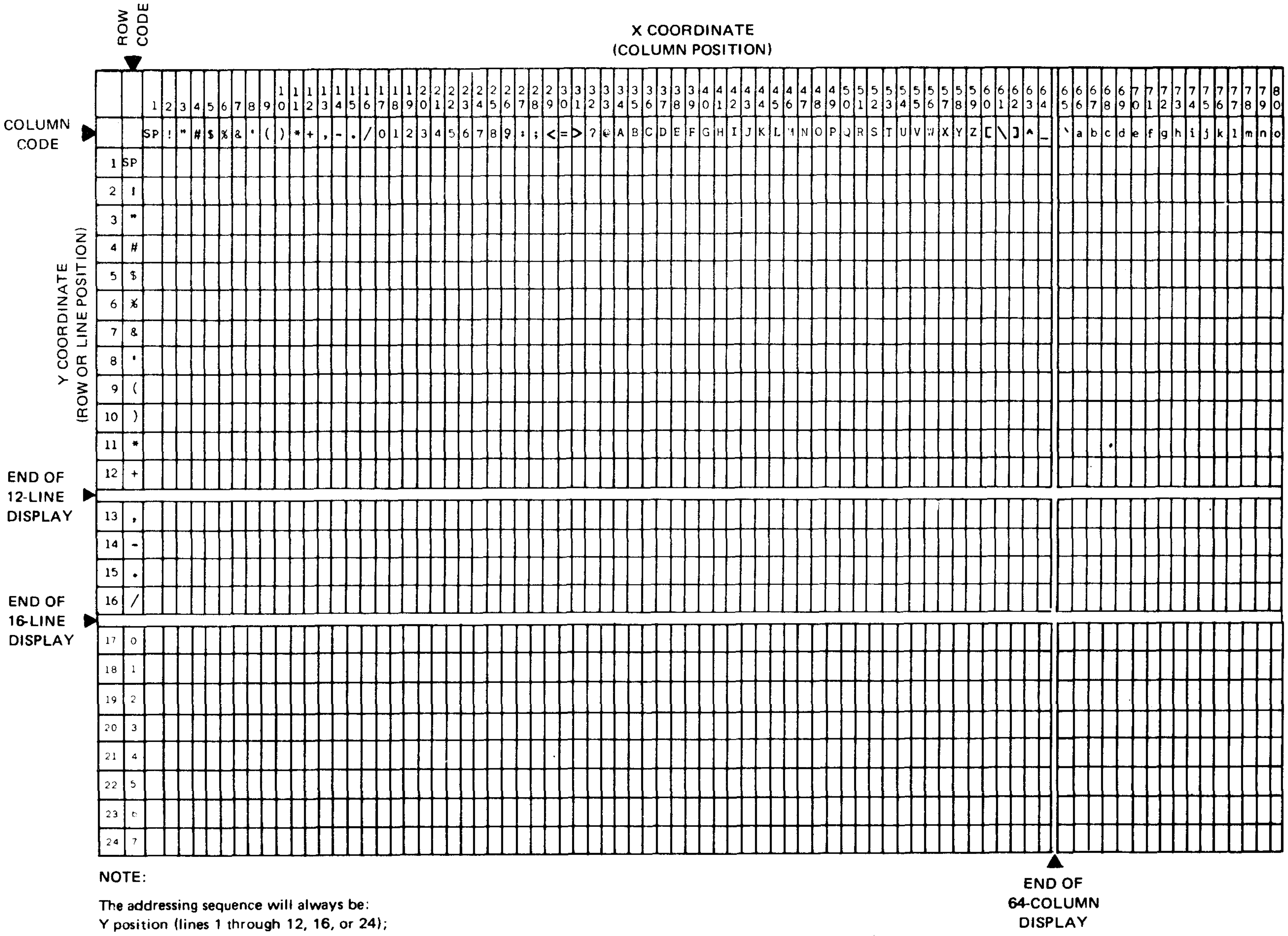
C is the number of the column in which the field control character is placed. This is the hexadecimal value equivalent to the column code for the screen column indicated in Figure F-5.

M - operation

M is a hexadecimal value placed in the sequence to define bits 4, 5, 6, and 7 of the field control character operation. Table F-4 lists the hexadecimal codes you can use.

N - operation

N is a hexadecimal value placed in the sequence to define bits 0, 1, 2, and 3 of the field control character operation. Table F-5 lists the hexadecimal codes you can use.


ROW CODE
X COORDINATE (COLUMN POSITION)
COLUMN CODE
Y COORDINATE (ROW OR LINE POSITION)
END OF 12-LINE DISPLAY
END OF 16-LINE DISPLAY
END OF 64-COLUMN DISPLAY

X	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Column code	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3

X	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
Column code	4	5	6	7	8	9	:	;	<	=	>	?	@	A	B	C	D	E	F	G

X	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
Column code	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[

X	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
Column code	\	]	^	_	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o

Y	Row code
1	SP
2	!
3	"
4	#
5	$
6	%
7	&
8	'
9	(
10	)
11	*
12	+
13	,
14	-
15	.
16	/
17	0
18	1
19	2
20	3
21	4
22	5
23	6
24	7

NOTE:

The addressing sequence will always be:
Y position (lines 1 through 12, 16, or 24);
then X position (columns 1 through 64 or 80).

Figure F-5. Row and Column Coordinate Values Used in Field Control Sequences

Table F-4. Hexadecimal Codes Used as M in the FCC Sequence

ASCII Character	Hexadecimal Code	Field Characteristics
0	30	Tab stop, normal intensity, changed field*
1	31	Tab stop, display off (no intensity), changed field*
2	32	Tab stop, low intensity, changed field*
3	33	Tab stop, blinking display, changed field*
4	34	Tab stop, normal intensity
5	35	Tab stop, display off (no intensity)
6	36	Tab stop, low intensity
7	37	Tab stop, blinking display
8	38	Not tab stop, normal intensity, changed field*
9	39	Not tab stop, display off (no intensity), changed field*
:	3A	Not tab stop, low intensity, changed field*
;	3B	Not tab stop, blinking display, changed field*
<	3C	Not tab stop, normal intensity
=	3D	Not tab stop, display off (no intensity)
>	3E	Not tab stop, low intensity
?	3F	Not tab stop, blinking display

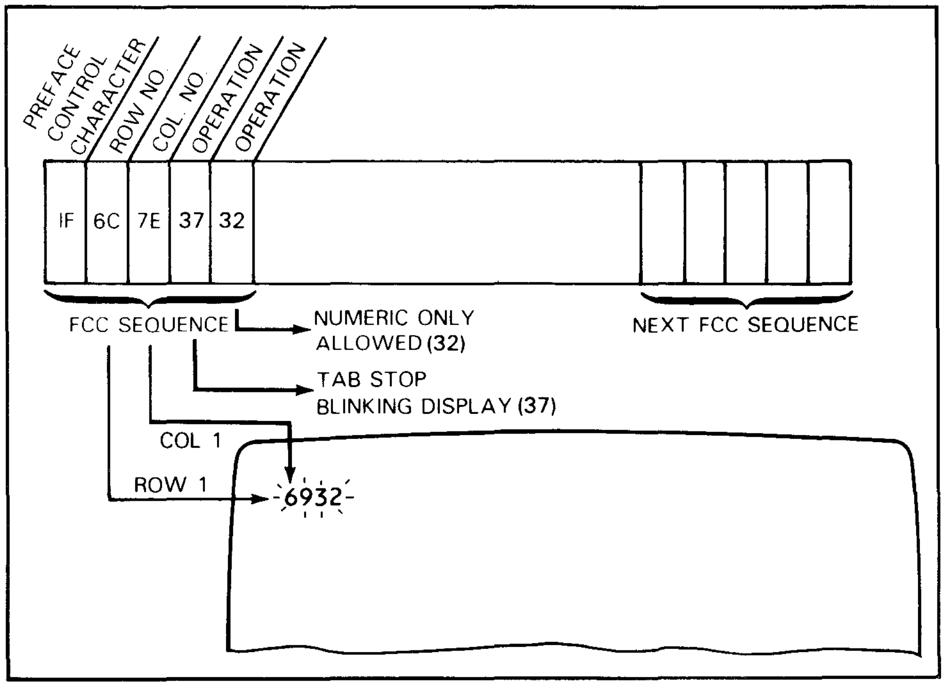
* Normally, when an FCC is generated by the host processor, the changed-field designator is cleared. However, the host processor can generate individual FCCs with the changed-field designator set; this capability may be used for selective transfer or transmission of fields which were not in fact changed by the terminal operator. By sending an ESC u code to the terminal in a text message, the host processor can clear the changed-field designators in all FCCs without regenerating each FCC and without altering the data within the fields.

Table F-5. Hexadecimal Codes Used as N in the FCC Sequence

ASCII Character	Hexadecimal Code	Field Characteristics
0	30	Any input allowed
1	31	Alpha only allowed
2	32	Numeric only allowed
3	33	Protected (no entries and no changes allowed)
4	34	Any input allowed, right-justified
5	35	Alpha only allowed, right-justified
6	36	Numeric only allowed, right-justified

Example

The following diagram illustrates a field control character sequence and the resulting output display of a numeric field to which this sequence is applied. Notice the 1F preface control character is followed by a row and column positioning of the field at 6 rows down ($6C_{16}$) and 30 columns across ($7E_{16}$) the screen. At this screen location, the next character, the operation value, (37_{16}, Table F-4) specifies a tab stop with blinking display. The last character (32_{16}, Table F-5) specifies numeric fields only allowed. For detailed information on using field control characters, consult the UTS 400 programmer reference, UP-8359 (current version).



Appendix G. Differences Between Extended COBOL and 1974 American National Standard COBOL

G.1. DIFFERENCES

If you use the extended COBOL compiler, there are three main differences in coding, compiling, and linking your action programs. Table G-1 explains.

Table G-1. Differences for Extended and 1974 COBOL Action Programs

Extended COBOL	1974 COBOL
Shared code parameter format is: // PARAM OUT=(M)	Shared code parameter format is: // PARAM IMSCOD=YES
Linkage editor INCLUDE statement: INCLUDE prog-id00	Linkage editor INCLUDE statement: INCLUDE prog-id
I/O function code format is: ENTER LINKAGE. CALL statement. ENTER COBOL.	I/O function code format is: CALL statement.
DICE code sequences expressed as DICE value multipunch equivalent. (See Figure G-3.)	DICE code sequences expressed as DICE value hexadecimal equivalent.
Restricted reserved words different from 1974 COBOL. (See 2.3.)	Restricted reserved words different from extended COBOL. (See G.6.)

G.2. SHARED CODE PARAMETER

Purpose

Using the shared code parameter allows the extended or 1974 COBOL compilers to check the program for conformance to IMS syntax and to issue appropriate compilation diagnostics. If you use this option along with the configurator parameters, TYPE=SHR and SHRDSIZE, programs are allowed to run as shared under multithread IMS.

For shared code parameter formats for extended and 1974 COBOL, see Table G-1. Section 11 provides more details about compiling sharable and nonsharable action COBOL programs.

G.3. OBJECT MODULE NAME IN LINKAGE EDITOR CONTROL STREAM

INCLUDE coding format for extended COBOL

When the extended COBOL compiler compiles your action program, it appends the first six characters of your program-id with zeros. Thus, when naming the object modules on your linkage editor INCLUDE statement, you must append the two zeros.

INCLUDE coding format for 1974 COBOL

The 1974 COBOL object module name is composed of the first six characters of the program-id. Thus, the object name on the INCLUDE statement should be the same.

G.4. ENTER STATEMENTS

CALL coding format

When you use the extended COBOL compiler, each I/O function call you issue from your action program must be preceded by an ENTER LINKAGE statement and followed by an ENTER COBOL statement. For example, if you issued a CALL 'GET' function, you must use the following coding format:

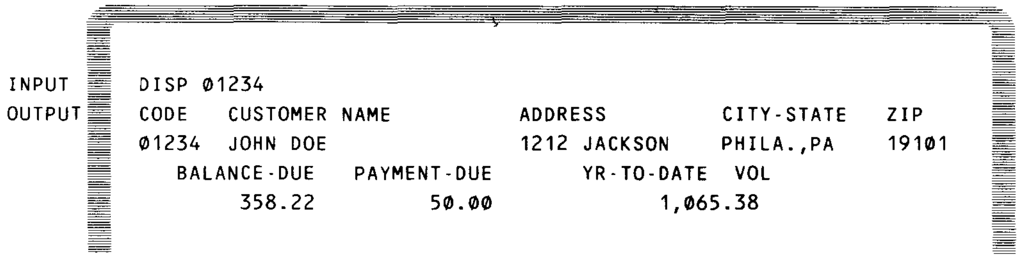
```
ENTER LINKAGE.
CALL 'GET' USING filename  record-area  key.
ENTER COBOL.
```

For compiling action programs with the 1974 COBOL compiler, only the I/O function call is needed. The ENTER statements are accepted by the compiler but cause warning diagnostics.

Figure G-2 illustrates the extended COBOL coding required for the DISP action program. In addition, Figure G-3 illustrates the multipunch DICE code equivalents that DISP copies from the IMS COPY library (Figure G-2, line 12).

Initiating DISP program

You initiate the DISP action program by entering the transaction code, DISP (in this case the same name as the program), and the 5-digit numeric key of the record desired. Figure G-1 shows the input message and corresponding output display.



```
INPUT    DISP Ø1234
OUTPUT   CODE    CUSTOMER NAME          ADDRESS         CITY-STATE    ZIP
         Ø1234   JOHN DOE               1212 JACKSON    PHILA.,PA     191Ø1
            BALANCE-DUE    PAYMENT-DUE        YR-TO-DATE  VOL
                 358.22          5Ø.ØØ             1,Ø65.38
```

Figure G-1. Sample Transaction Displaying Customer Record

DISP coding description

DISP retrieves a record from the customer file (CUSTFIL) and displays it at the terminal (Figure G-2, line 75). In case of an invalid record key in the input message, or any error condition detected by IMS, the program moves an error message to the output message area and terminates the transaction (line 77 and 86-95).

Note that DISP uses DICE, previously coded and filed in a copy library (Figure G-3) for homing the cursor, clearing the screen, and repositioning the cursor to a new line (line 70–72).

```
00001        IDENTIFICATION DIVISION.
00002        PROGRAM-ID.  DISP.
00003        ENVIRONMENT DIVISION.
00004        CONFIGURATION SECTION.
00005        SOURCE-COMPUTER. UNIVAC-9Ø3Ø.
00006        OBJECT-COMPUTER. UNIVAC-9Ø3Ø.
00007        DATA DIVISION.
00008        WORKING-STORAGE SECTION.
00009        77  CUSTFIL   PIC X(7) VALUE 'CUSTFIL'.
00010        77  TEXT-1    PIC X(32) VALUE 'PROCESSING ERROR.STATUS CODE =
00011        77  TEXT-2    PIC X(23) VALUE 'DETAILED STATUS CODE = '.
00012        Ø1  DICE COPY DICE.
00013        Ø1  CUSHDR1.
00014            Ø2  CUSHD1   PIC A(6)  VALUE ' CODE '.
00015            Ø2  CUSHD2   PIC A(2Ø) VALUE 'CUSTOMER NAME       '.
00016            Ø2  CUSHD3   PIC A(15) VALUE 'ADDRESS        '.
00017            Ø2  CUSHD4   PIC A(15) VALUE 'CITY-STATE     '.
00018            Ø2  CUSHD5   PIC A(5)  VALUE 'ZIP  '.
00019        Ø1  CUSHDR2.
00020            Ø2  CUSHD6   PIC A(15) VALUE '   BALANCE-DUE '.
00021            Ø2  CUSHD7   PIC A(15) VALUE '  PAYMENT-DUE  '.
00022            Ø2  CUSHD8   PIC A(15) VALUE ' YR-TO-DATE VOL'.
00023        LINKAGE SECTION.
00024        Ø1  PROGRAM-INFORMATION-BLOCK. COPY PIB.
00025        Ø1  INPUT-MESSAGE-AREA. COPY IMA.
```

Figure G-2. Sample Extended COBOL Action Program DISP (Part 1 of 3)

```
    Ø2  TRANSAC-CDE   PIC X(4).
    Ø2  FILLER        PIC X.
    Ø2  REC-KEY       PIC X(5).
    Ø2  REC-NO REDEFINES REC-KEY  PIC 9(5).
Ø1  WORK-AREA.
    Ø2  CUS-REC.
        Ø3  CDE            PIC X(5).
        Ø3  NAME           PIC X(2Ø).
        Ø3  ADDR           PIC X(15).
        Ø3  CTY-STE        PIC X(15).
        Ø3  ZIP            PIC 9(5).
        Ø3  BLNCE-DUE      PIC S9(9)V99  COMP-3.
        Ø3  DUE-IN         PIC S9(9)V99  COMP-3.
        Ø3  YTD-VOL        PIC 9(6)V99.
    Ø2  ERROR-MSGE.
        Ø3  TXT-1          PIC X(32).
        Ø3  STAT           PIC 9(4).
        Ø3  TXT-2          PIC X(23).
        Ø3  DSTAT          PIC 9(4).
Ø1  OUTPUT-MESSAGE-AREA COPY OMA.
    Ø2  LINE-0        PIC X(4).
    Ø2  LINE-1        PIC X(64).
    Ø2  CR-1          PIC X(4).
    Ø2  LINE-2.
        Ø3  CDE       PIC X(5).
        Ø3  FILLER    PIC X.
        Ø3  NAME      PIC X(2Ø).
        Ø3  ADDR      PIC X(15).
        Ø3  CTY-STE   PIC X(15).
        Ø3  ZIP       PIC X(5).
    Ø2  CR-2              PIC X(4).
    Ø2  LINE-3            PIC X(45).
    Ø2  CR-3              PIC X(4).
    Ø2  LINE-4.
        Ø3  FILLER        PIC X.
        Ø3  OUT-BAL       PIC ZZZ.ZZZ.ZZ9.99.
        Ø3  FILLER        PIC X(5).
        Ø3  OUT-DUE       PIC ZZZ.ZZZ.ZZZ.99.
        Ø3  FILLER        PIC X(5).
        Ø3  OUT-VOL       PIC ZZZ.ZZZ.99.
    Ø2  CR-4              PIC X(4).
    Ø2  LINE-13           PIC X(4).
PROCEDURE DIVISION USING PROGRAM-INFORMATION-BLOCK
     INPUT-MESSAGE-AREA WORK-AREA OUTPUT-MESSAGE-AREA.
STRT-CDE-SECT.
    MOVE CURS-COORD TO LINE-0.
```

Figure G-2. Sample Extended COBOL Action Program DISP (Part 2 of 3)

```
        MOVE CURS-HME TO LINE-13.
        MOVE CR TO CR-1, CR-2, CR-3, CR-4.
    CUSTOMER-FILE-SECT.
        ENTER LINKAGE.
        CALL 'GET' USING CUSTFIL CUS-REC REC-KEY.
        ENTER COBOL.
        IF STATUS-CODE IS NOT = Ø GO TO PROCESS-ERROR.
        MOVE CUSHDR1 TO LINE-1.
        MOVE CORR CUS-REC TO LINE-2.
        MOVE CUSHDR2 TO LINE-3.
        MOVE BLNCE-DUE TO OUT-BAL.
        MOVE DUE-IN TO OUT-DUE.
        MOVE YTD-VOL TO OUT-VOL.
        GO TO NORMAL-TERM.
    PROCESS-ERROR.
        MOVE TEXT-1 TO TXT-1.
        MOVE STATUS-CODE TO STAT.
        MOVE TEXT-2 TO TXT-2.
        MOVE DETAILED-STATUS-CODE TO DSTAT.
        MOVE ERROR-MSGE TO LINE-1.
        MOVE REC-KEY TO ADDR OF LINE-2.
    NORMAL-TERM.
        ENTER LINKAGE.
        CALL 'RETURN'.
        ENTER COBOL.
```

Figure G–2. Sample Extended COBOL Action Program DISP (Part 3 of 3)

```
 Ø1  DICE COPY DICE.
*    DICE SPECIAL CHARACTERS FOR PROGRAM DISP.
*
*    FORMS CONTROL & CLEAR. CURSOR TO ROW Y. COLUMN X. AND CLEAR
*    SCREEN. X'1ØØØ3Ø2Ø1'
*    MULTIPUNCHES 12-11-9-8-1. 12-9-3. 12-9-2. 12-9-1.
*
     Ø2  CURS-COORD.
         Ø3  DICE-1    PIC X(2) VALUE '  '.
         Ø3  ROW-Y1    PIC X(1) VALUE '  '.
         Ø3  COL-X1    PIC X(1) VALUE '  '.
*
     POSITION CONTROL NEW LINE.X'1ØØ4ØØØØ'.
*    MULTIPUNCHES 12-11-9-8-1. 12-9-4.  12-Ø-9-8-1. 12-Ø-9-8-1.
*
```

Figure G–3. Example of DICE Sequences Filed in a COPY Library (Part 1 of 2)

```
00016       77  CR                  PIC X(4) VALUE '    '.
00017       *
00018       *   SET COORD-CURSOR TO HOME. X'1ØØ1ØØØØ'.
00019       *   MULTIPUNCHES 12-11-9-8-1. 12-9-8-1. 12-Ø-9-8-1. 12-Ø-9-8-1.
00020       *
00021       77  CURS-HME            PIC X(4) VALUE '    '.
00022       *
00023       *   POSITION CONTROL & CLEAR. CLEAR TO END OF LINE & NEW LINE.
00024       *   X '1ØØ5ØØØØ'.
00025       *   MULTIPUNCHES 12-11-9-8-1. 12-9-5. 12-Ø-9-8-1. 12-Ø-9-8-1.
00026       *
00027       77  CLR-LINE            PIC X(4) VALUE '    '.
00028       *
00029       *   APPENDING CODE FOR UNISCOPE-1ØØ COP. X'12'.
00030       *   MULTIPUNCH 11-9-2.
00031       *
00032       77  DC                  PIC X(1) VALUE ' '.
00033       *
00034       *   START OF ENTRY CHARACTER SOE. X'1E'.
00035       *   MULTIPUNCH 11-9-8-6.
00036       *
00037       77  SOE                 PIC X(1) VALUE ' '.
```

Figure G-3. Example of DICE Sequences Filed in a COPY Library (Part 2 of 2)

G.5. DICE CODES

Multipunch DICE equivalents

When you compile an action program with the extended COBOL compiler, you must express DICE sequences using the multipunch equivalents of the DICE values. Figure G-3 shows an example of the statement describing multipunch DICE values used in the DISP action program (Figure G-2, line 12). The comments in this copy library module explain the hexadecimal values equivalent to the blank multipunch values.

Hexadecimal DICE equivalents

The 1974 COBOL compiler permits you to use the hexadecimal DICE values directly in the action program. The following examples illustrate three possible applications of hexadecimal DICE values that conform to 1974 standards.

Example 1

```
01 DICE
   03 FIELD-1 PIC X.
   03 FIELD-2 PIC X.
   03 FIELD-3 PIC X.
   03 FIELD-4 PIC X.
MOVE ='10' TO FIELD-1.
MOVE ='03' TO FIELD-2.
MOVE ='01' TO FIELD-3.
MOVE ='01' TO FIELD-4.
```

Example 2

```
   03 DICE PIC X(4).
MOVE ='10030101' TO DICE.
```

Example 3

```
77 DICE PIC X(4) VALUE ='10030101'.
```

For more detail about DICE code sequences, their interpretation, and use, see Appendix F.

G.6. EXTENDED COBOL LANGUAGE RESTRICTIONS

Illegal syntax

Some COBOL verbs, clauses, and sections are illegal in extended COBOL action programs. If you compile them with the shared code parameter, PARAM OUT=(M), the compiler locates and deletes them from your program. (See Section 11.)

The following reserved words are illegal in extended COBOL action programs:

Reserved words

ALTER	REWRITE
CLOSE	SEEK
DECLARATIVE SECTION	SEGMENT-LIMIT
ENTRY	SORT
EXHIBIT	STOP
EXIT-PROGRAM	SYSCHAN-t
FILE SECTION	SYSCONSOLE
INPUT-OUTPUT SECTION	SYSERR [-m]
INSERT	SYSIN
OPEN	SYSIN-96
READ	SYSIN-128
READY TRACE	SYSLOG
RELEASE	SYSLST
RESET TRACE	WRITE
RETURN	

Illegal verbs with working-storage items

Other COBOL verbs must not have working-storage items as receiving operands. These verbs are:

ADD	PERFORM (varying option)
COMPUTE	SEARCH (varying option)
DIVIDE	SET
EXAMINE (replacing option)	SUBTRACT
MOVE	TRANSFORM
MULTIPLY	

Precautionary diagnostics

If you compile your action program with the shared code parameter, the compiler flags the erroneous statement and issues a precautionary diagnostic.

Index

SPERRY UNIVAC

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

(Document Title)

(Document No.) *(Revision No.)* *(Update No.)*

Comments:

Cut along line.

From:

(Name of User)

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

FOLD

CUT