SPERRY UNIVAC
COMPUTER SYSTEMS

PUBLICATIONS UPDATE

Operating System/3 (OS/3)

Information Management System 90 (IMS 90) Applications

User Guide/ Programmer Reference

UP-8614 Rev. 1-B

This Library Memo announces the release and availability of Updating Package B to "SPERRY UNIVAC Operating System/3 (OS/3) Information Management System 90 (IMS 90) Applications User Guide/Programmer Reference", UP-8614 Rev. 1.

Update B for release 7.1 makes several technical changes concerning the random GETUP function call, downline load processing, batch processing, the ZZRSD and ZZHLD terminal commands, and the SWTCH transaction code. These changes are applicable to software prior to release 7.1.

Copies of Updating Package B are now available for requisitioning. Either the updating package only or the complete manual with the updating package may be requisitioned by your local Sperry Univac representative. To receive only the updating package, order UP-8614 Rev. 1-B. To receive the complete manual, order UP-8614 Rev. 1.

REPLACED BY:

✓ UP9205 IMS CONCEPTS AND FACILITIES
✓ UP9206 IMS ACTION PROGRAMMING IN RPG
✓ UP-9207 " " " COBOL AND ASSEMBLER
✓UP-9208 " TERMINAL USER'S GUIDE
✓UP-9209 . DATA DEFINITION AND UNIQUE USER GUIDE

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ and MZ	Mailing Lists 18, 18U, 19, 19U, 20, 20U, 21, 21U, 28U, 29U, 75, 75U, 76 and 76U (Package B to UP-8614 Rev. 1, 24 pages plus Memo)	Library Memo for UP-8614 Rev. 1-B RELEASE DATE: December, 1981

SPERRY UNIVAC
COMPUTER SYSTEMS

PUBLICATIONS UPDATE

Operating System/3 (OS/3)

Information Management System 90 (IMS 90) Applications

User Guide/Programmer Reference

UP-8614 Rev. 1–A

This Library Memo announces the release and availability of Updating Package A to "SPERRY UNIVAC Operating System/3 (OS/3) Information Management System 90 (IMS 90) Applications User Guide/Programmer Reference", UP-8614 Rev. 1.

Update A contains the following new or changed items for release 7.1:

- IRAM files are supported only when defined as MIRAM files.
- Hexadecimal and character values for contents of AUX-FUNCTION field in the output message header for continuous output are changed.
- Special considerations must be made when using the input options or report address option in the CONTINUOUS-OUTPUT-CODE field of the OMA.
- Screen formats can be displayed on the UTS 20 and on auxiliary devices.
- The UPSI byte can be used to determine edit table errors.
- The ZZOPN, ZZCLS, and ZZPCH terminal commands can be issued from the system console in single-thread IMS 90.
- In addition to cancelling the currently active transaction, the ZZCNC terminal command clears all output queued to the source terminal.
- Three different status messages can be issued in response to the SWTCH transaction code: one for message sent, one for message not sent, and one for message queued.
- Output from the ZSTAT transaction can be sent to a tape cassette or diskette.
- The terminal status display has been changed.
- One new unrecoverable and two new recoverable ZSTAT error messages are added.
- ZSTAT places a message in the OMA if it receives an unrecognized delivery notice.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ and MZ	Mailing Lists 18, 18U, 19, 19U, 20, 20U, 21, 21U, 28U, 29U, 75, 75U, 76 and 76U (Package A to UP-8614 Rev. 1, 198 pages plus Memo)	Library Memo for UP-8614 Rev. 1–A RELEASE DATE: September, 1981

- The IBM 3270 terminal is supported.

Other changes include expanded descriptions, clarifications, or corrections that apply to the software before the current release.

Copies of Updating Package A are now available for requisitioning. Either the updating package only or the complete manual with the updating package may be requistioned by your local Sperry Univac representative. To receive only the updating pacakage, order UP-8614 Rev. 1–A. To receive the complete manual, order UP-8614 Rev. 1.

SPERRY UNIVAC
COMPUTER SYSTEMS

Operating System/3 (OS/3)

Information Management System 90 (IMS 90) Applications

User Guide/Programmer Reference

UP-8614 Rev. 1

This Library Memo announces the release and availability of "SPERRY UNIVAC® Operating System/3 (OS/3) Information Management System 90 (IMS 90) Applications User Guide/Programmer Reference", UP-8614 Rev. 1.

This revision includes changes to program examples, IMS 90 internal tables, and new sections describing:

- File I/O functions;
- Cassette/diskette devices and additional auxiliary function byte settings in the OMA;
- Screen formatting services;
- The BUILD and REBUILD function calls used to construct screen buffer or error screen formats;
- Three formats of the ZSTAT transaction code used to display statistics about files, programs, transactions, and terminals; and
- ZSTAT recoverable and unrecoverable error messages.

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

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS:
Mailing Lists BZ, CZ and MZ	Mailing Lists 18, 18U, 19, 19U, 20, 20U, 21, 21U, 28U, 29U, 75, 75U, 76 and 76U (Covers and 418 pages)	Library Memo
		RELEASE DATE: October, 1980

Information Management System 90 (IMS 90)

Applications

OS/3

User Guide/
Programmer Reference

H

SPERRY UNIVAC

UP-8614 Rev. 1

This document contains the latest information available at the time of preparation. Therefore, it may contain descriptions of functions not implemented at manual distribution time. To ensure that you have the latest information regarding levels of implementation and functional availability, please consult the appropriate release documentation or contact your local Sperry Univac representative.

Sperry Univac reserves the right to modify or revise the content of this document. No contractual obligation by Sperry Univac regarding level, scope, or timing of functional implementation is either expressed or implied in this document. It is further understood that in consideration of the receipt or purchase of this document, the recipient or purchaser agrees not to reproduce or copy it by any means whatsoever, nor to permit such action by others, for any purpose without prior written permission from Sperry Univac.

Sperry Univac is a division of the Sperry Corporation.

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.

PRINTED IN U.S.A.

PAGE STATUS SUMMARY

ISSUE: Update B – UP-8614 Rev. 1
RELEASE LEVEL: 7.1 Forward

Part/Section	Page Number	Update Level
Cover/Disclaimer		Orig.
PSS	1	B
Preface	1	Orig.
Contents	1 thru 4	Orig.
	5	A
	6	Orig.
	7	A
	8	Orig.
	8 thru 12	A
1	1	A
	2 thru 5	Orig.
2	1	A
	2 thru 13	Orig.
	14	A
	15 thru 20	Orig.
	21	A
	22 thru 28	Orig.
	29	A
	30 thru 41	Orig.
	42	A
	43 thru 60	Orig.
	61	A
	62 thru 73	Orig.
3	1 thru 9	Orig.
	10	A
	10a	A
	11 thru 18	Orig.
	19 thru 21	A
	22, 23	Orig.
	24, 25	A
	26 thru 35	Orig.
	36	A
	37	Orig.
	38	A
	38a	Orig.
	39 thru 43	Orig.
	44	A
	45 thru 48	Orig.
	49, 50	A
	51	Orig.
	52	A
	53, 54	Orig.
	55	A
	56, 57	Orig.
	58	B
	59	A
	60 thru 74	Orig.
	75	A
	76 thru 79	Orig.
	80	A
	81 thru 83	Orig.
	84, 85	A
	86, 87	Orig.
	88	A
	89	Orig.
	90	A
	91	Orig.
	92	A
	93 thru 95	Orig.
	96	A
	97 thru 100	Orig.
	101	A
	102 thru 109	Orig.
	110, 111	B
	112	A
	112a	A
	113	B
	114	A
	114a	B
	115	B
	116	A
	116a	A
	117, 118	A
	118a, 118b	A
	118c	B
	118d	B**
	119	A
	120 thru 124	Orig.
	125, 126	A
	126a	A
	127, 128	A
	128a	A
	129 thru 140	A
	140a thru 140i	A
	140j	B
	141 thru 152	A
	152a thru 152e	A
	153, 154	A
4	1 thru 5	Orig.
	6	A
	6a	A
	7, 8	A
	9 thru 12	Orig.
5	1, 2	Orig.
	3 thru 6	A
	7	B
	8 thru 11	A
	12, 13	Orig.
	14, 15	A
	16	Orig.
	17, 18	A
	18a	A
	19	A
	20	B
	20a	B
	21, 22	A
	22a	A
	23 thru 27	A
	28	Orig.
	29 thru 32	A
6	1 thru 3	Orig.
	4	A
	5 thru 13	Orig.
	14, 15	A
	16 thru 18	Orig.
	18a	Orig.
	19 thru 21	Orig.
	22, 23	A
	24 thru 34	Orig.
	35	A
7	1 thru 9	Orig.
	10	B
	11 thru 17	Orig.
Appendix A	1, 2	Orig.
Appendix B	1 thru 3	Orig.
Appendix C	1 thru 7	Orig.
	8	A
	9 thru 12	Orig.
Appendix D	1 thru 6	Orig.
Appendix E	1, 2	Orig.
	3, 4	A
	4a	A
	5	A
	6 thru 12	Orig.
Glossary	1 thru 13	Orig.
Index	1	Orig.
	2	A
	3	Orig.
	4	A
	5	Orig.
	6, 7	A
	8	Orig.
	9 thru 12	A
User Comment Sheet		

*New pages **Deleted pages

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.

Preface

This document is one of a series describing the SPERRY UNIVAC Information Management System 90 (IMS 90) for users of Operating System/3 (OS/3). An introduction to IMS 90, UP-8816 (current version), provides an overview of IMS 90 software and its use. The system support functions user guide/programmer reference, UP-8364 (current version), is directed to systems analysts and IMS 90 administrators. It describes communications network structuring, pre-online processing, and IMS 90 initiation, execution, and recovery procedures. This applications user guide/programmer reference (UP-8614) is intended for use with the systems support functions manual and is directed to IMS 90 application programmers and terminal operators who must prepare and process user applications. Subjects described are:

- Preparation of data definitions for use by the uniform inquiry update element (UNIQUE) or user-written action programs
- Preparation of action programs in COBOL, RPG II, or basic assembly language (BAL)
- Preparation of edit tables for use with user-written action programs
- Terminal operation, including operation of the master terminal and user terminals
- Transaction processing via UNIQUE
- Batch transaction processing

IMS 90 users wishing to access a DMS 90 data base from action programs should read the IMS 90/DMS 90 interface user guide/programmer reference, UP-8748 (current version). Also available is the IMS 90 terminal user commands, UP-8741 (current version), which is a pocket reference card.

Contents

APPENDIXES

E. DEVICE INDEPENDENT CONTROL EXPRESSIONS

GLOSSARY

INDEX

USER COMMENT SHEET

FIGURES

TABLES

1. Introduction

1.1. OVERVIEW

The SPERRY UNIVAC Information Management System 90 (IMS 90) is an interactive, transaction-oriented file management system. It is interactive because it operates on the principle that a question-and-answer dialog is maintained between the terminal operator and IMS 90.

The basic unit of work in IMS 90 is an action. An input message, the processing of it by one or more action programs, and at least one output message define an action. Accordingly, a sequence of one or more related actions defines a transaction. IMS 90 is transaction-oriented because all processing is triggered by an input message or question that requires at least one output message or answer.

To process transactions, Sperry Univac supplies a set of action programs called the uniform inquiry update element (UNIQUE), activated by commands from remote terminals, as a component of IMS 90. For convenience in processing messages, you can either employ the IMS 90 UNIQUE action programs or write your own action programs in COBOL, report program generator II (RPG II), or basic assembly language (BAL).

IMS 90 manages *user logical* and *defined files. User logical* files are collections of logical records created on physical devices and accessed via the standard access methods (DAM, MIRAM, ISAM, or SAM. To access IRAM files, you must define them as MIRAM files at configuration time.) In contrast, *defined files* are collections of defined records that the defined record management component of IMS 90 composes from one or more logical disk records according to a user-supplied data definition. You can also protect defined files by assigning passwords. Built into the IMS 90 modular components are data verification and protection procedures, and scheduling and queueing procedures. IMS 90 file access techniques are compatible with existing programming and file structures. IMS 90 also allows you to access data base management system 90 (DMS 90) data bases.

The interactive transaction processing capabilities of IMS 90 rely on its communications/data management support. IMS 90 uses the SPERRY UNIVAC Integrated Communications Access Method (ICAM) Transaction Control Interface (TCI) to support terminal communications. Refer to the IMS 90 system support functions user guide/programmer reference, UP-8364 (current version) for information about setting up a communications interface.

IMS 90 provides a terminal command repertoire to assist terminal operators in using remote terminals. A master terminal command repertoire, also provided by IMS 90, enables the master terminal operator to control terminals assigned to IMS 90 and monitor the system.

When inquiries or updates to files are initiated from remote terminals, IMS 90 components such as action scheduling, file management, and internal message control facilitate rapid processing. Application, administration, and operation of the IMS 90 transaction processing system are performed during pre-online, online, and offline processing by IMS 90 operations and user activities.

1.1.1. IMS 90 Operations

1.1.1.1. Pre-online Processing

IMS 90 pre-online processing employs the IMS 90 utilities and processors that prepare and tailor the system for processing transactions online. IMS 90 pre-online processing includes:

- initialization of the named record file via the NAMEREC utility;
- definition of passwords via the same NAMEREC file utility;
- processing of user data definitions by the data definition processor;
- configuration of the online IMS 90 system from user-specified parameters; and
- generation of edit tables.

1.1.1.2. Online Processing

IMS 90 online processing employs components that control the interactive processing of transactions.

Online processing includes:

- system startup and shutdown procedures;
- internal message control including terminal control functions;
- scheduling and loading of UNIQUE or user-written action programs;
- IMS 90 and user file management; and
- activation record control.

1.1.1.3. Offline Processing

IMS 90 offline processing handles the recovery of user files left damaged or in an inconsistent state. It includes:

- the offline recovery utility (ZC#TRC); and
- the tape copy routine (ZC#TCP).

A more detailed description of the IMS 90 components and their roles in pre-online, online, and offline processing is provided in the IMS 90 system support functions user guide/programmer reference, UP-8364 (current version). System preparation functions, startup/shutdown procedures, and recovery processing are also described in UP-8364. Applications preparation and processing, including action programming, data definition, edit table generation, and terminal operation, are described in this document.

1.1.2. User Activities

The most important activities you perform are defining data (required for UNIQUE, optional for user action programs); configuring the online IMS 90 system (mandatory); writing action programs (optional); and processing your transactions from remote terminals, using UNIQUE or user action programs.

In addition, user activities include the following optional pre-online and offline operations:

- running the NAMEREC utility to initialize the named record file and define passwords;
- defining edit tables for input message formatting and data validation; and
- recovering user files, if necessary.

1.1.2.1. Defining Data

One of your first tasks is to define the data to be used in your IMS 90 application. To do this, you must write a data definition if you are going to use UNIQUE or you may optionally write one for use with your own action programs. (User action programs can also access conventional DAM, MIRAM, IRAM, ISAM, and SAM files.) A data definition describes a defined file, containing defined records. These defined records are redefinitions of user logical records from existing physical files. Because the actual data for these defined records exists as logical records in one or more user files, the defined record management component of IMS 90 needs to know only where all parts of each defined record can be located in your files so that it can construct the defined record when an action program calls for it. This location information is contained in a data definition record which the data definition processor places in the IMS 90 internal file, NAMEREC.

1.1.2.2. Configuring IMS 90

The user configures the IMS 90 system by preparing the job control stream and configurator input, and by running the configurator job. A job control proc (jproc), IMSCONF, performs five job steps: communications control area (CCA) linkage, internal file initialization, configuration, assembly, and online linkage.

User-specified configurator input consists of the following sections that define IMS 90 characteristics.

NETWORK TERMINAL OPTIONS ACTION FILE

GENERAL TRANSACT TIMEOUTS PROGRAM DRCRDMGT

For detailed discussion of IMS 90 configuration procedures, see the IMS 90 system support functions user guide/programmer reference, UP-8364 (current version).

1.1.2.3. Writing Action Programs

Action programs are written to process transactions (sequences of one or more actions). Action programs can be user-written or supplied by IMS 90. All action programs operate under the application management component of IMS 90 to process input messages and generate output messages.

User action programs may be written in COBOL, Report Program Generator II (RPG II), or Basic Assembly Language (BAL). The series of action programs supplied by IMS 90 is identified as the uniform inquiry update element (UNIQUE). No user action programming is required if you plan to process all your transactions through UNIQUE.

1.1.2.4. Using Uniform Inquiry Update Element (UNIQUE)

The Sperry Univac supplied series of action programs called UNIQUE accesses and updates defined files via a group of UNIQUE commands issued by the user at the terminal. The defined record management (DRM) component of IMS 90 handles the defined file accessing operations. Section 6 of this document provides a detailed discussion of UNIQUE.

1.1.2.5. Operating Terminals

There are two sets of terminal commands – master and standard. Master terminal commands control and monitor the overall system and communications network. The standard terminal commands control message processing at the terminal. All terminal commands begin with the letters ZZ. Section 5 of this document discusses terminal operation in detail.

1.2. APPLICABILITY

IMS 90 is ideally suited for inquiry/update-oriented file processing applications. Some of the most typical applications are:

- Inventory control
- Order shipping
- Insurance
- Medical/hospital
- Interactive data collection
- Information management
- Library recall
- Warehouse management
- Computer-aided instruction
- One-time report generation
- Job shop operation

IMS 90 is especially suited to any application where ease of use, reliable performance, and integrity in data manipulation and information management are the primary concerns.

2. Data Definition

2.1. INTRODUCTION

You write a data definition to describe a defined file. IMS 90 constructs defined files from elements of existing ISAM, MIRAM, and DAM disk files and from data base subschemas according to the data definition you write and submit to the data definition processor.

If you decide to use the IMS 90-supplied uniform inquiry update element (UNIQUE), you must write a data definition because UNIQUE accesses user files only via defined files. When writing your own action programs, you can use defined files or you can call on IMS 90 file management to access your conventional DAM, ISAM, MIRAM, or SAM files directly. (To access IRAM files, you must define them as MIRAM files at configuration time.) Your COBOL action programs can also directly access a DMS 90 data base. Refer to the IMS 90/DMS 90 interface user guide/programmer reference, UP-8748 (current version).

In addition to defining records and file structures in your data definition, you define allowable functions (retrieve, modify, add, delete). The defined record management (DRM) portion of IMS 90 file management provides strong data integrity during defined file manipulation. Before executing any operation that changes disk files, DRM verifies that the change is allowed and that new values are within the limits you defined in the data definition.

IMS 90 accesses defined records by keys. For this reason, IMS 90 must construct defined records from indexed files (ISAM or MIRAM) or a data base subschema. It can also use nonindexed files (DAM or MIRAM), but only in combination with indexed files or a subschema.

Because the formats for data definition are similar to COBOL data division formats, you should have some knowledge of COBOL before you write a data definition.

2.1.1. Data Flow in IMS 90

Action programs access user data files through IMS 90 file management. When a user-written action program requests a logical input record from IMS 90, the file management component of IMS 90 issues a request for the record to OS/3 data management. Data management, interfacing between IMS 90 and the user data file, retrieves the physical record (or block) containing the logical record wanted, and passes the logical record to IMS 90 file management. In turn, IMS 90 file management passes the desired logical record to the action program. Similarly, data management receives logical record output from IMS 90 and transfers physical records to the user data files. These relationships are illustrated in Figure 2-1.

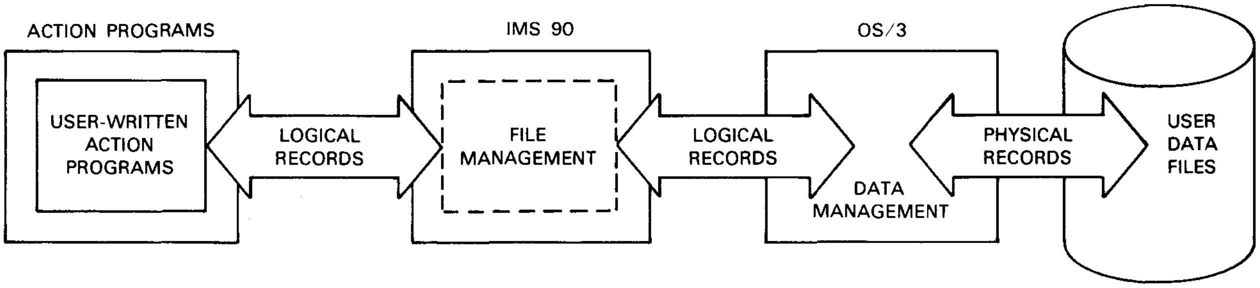


Figure 2—1. Data Flow Between Action Programs and User Data Files

There also is a different type of record that can be requested by a user-written action program – the *defined record.* A defined record comprises:

- all or part of a logical record from one of the user data files;
- all or parts of several logical records from the same user data file; or
- all or parts of several logical records from different user data files.

It is the DRM portion of IMS 90 file management that handles requests from action programs for defined records. DRM determines which logical records and what parts of them are required and from which user data files they are to be retrieved by data management. DRM then issues the necessary calls to data management, receives the logical records as input, and constructs the defined record that is expected by the action program. Finally, DRM passes the defined record to the program requesting it. Figure 2-2 illustrates these activities.

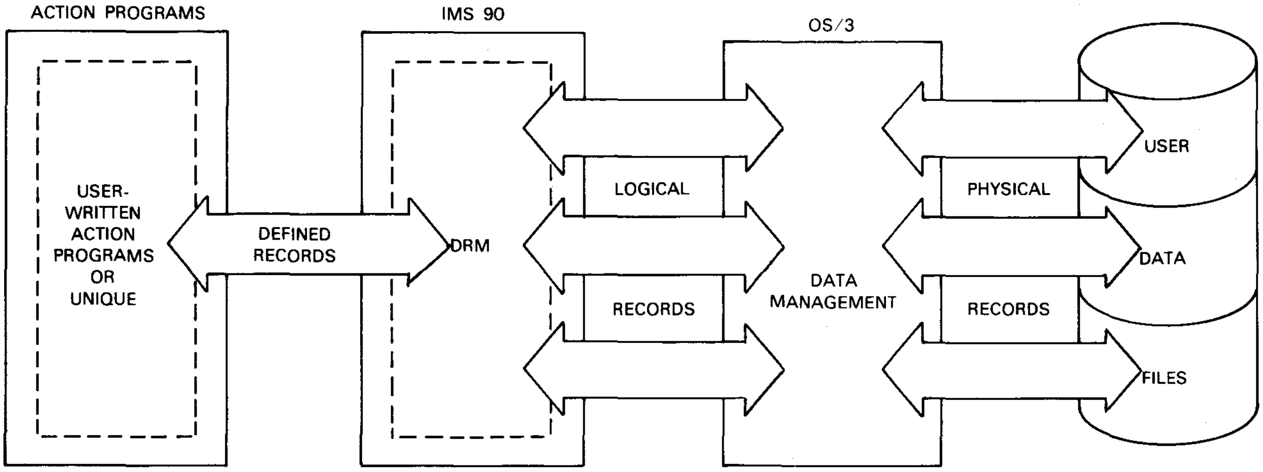


Figure 2—2. Flow of Defined Records to Action Programs via IMS 90

The information DRM needs for calling logical records to construct defined records, requested by action programs, is contained in the *data definition record.* When action programs request defined records in response to some terminal input, DRM accesses the data definition record (2.1.2) in an internal IMS 90 disk file called the named record (NAMEREC) file.

A terminal operator who is using the commands of the UNIQUE language for file query operations receives only defined records, displayed or listed at the terminal as specified by the IMS 90 action programs that implement UNIQUE. Using UNIQUE, the terminal operator accesses the user data files only through DRM. User-written action programs, on the other hand, provide the terminal operator with defined records, obtained through DRM, or with logical records, obtained from the user data files through the other components of IMS 90 file management. Similarly, updating information input to IMS 90 through a UNIQUE action program is received by DRM as defined records. A user-written action program provides both defined records to DRM and logical records to IMS 90 file management. Action programs using both defined records and logical records, however, cannot directly access a logical record that is accessed by the defined record. Figure 2-3 illustrates the data flow between both types of action programs and disk storage, indicating in schematic form the complete processing of logical and defined records.

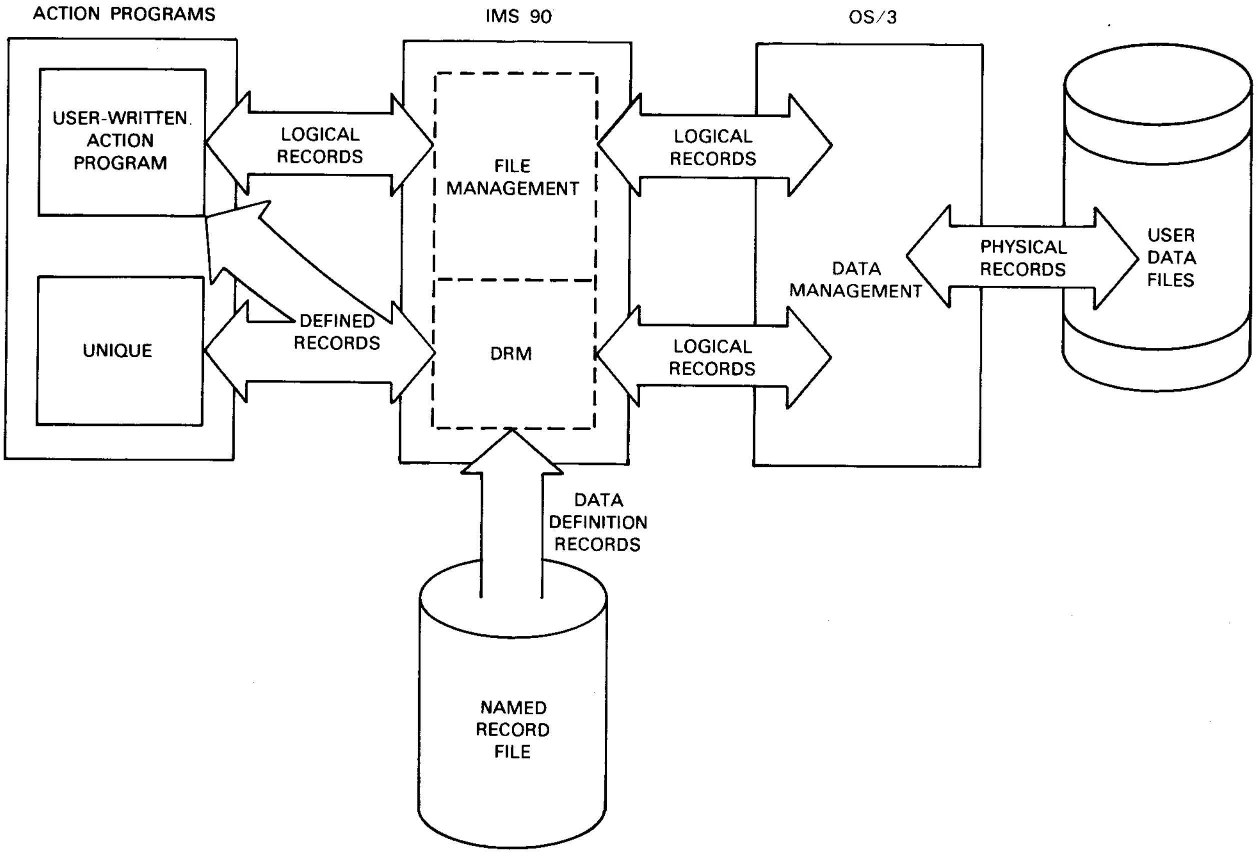


Figure 2—3. Data Flow Between Action Programs and Disk Storage

→ The collection of defined records extracted from user data files and passed by DRM to action programs forms the *defined file.*

2.1.2. Creating Data Definition Records

The data definition records needed by DRM to access defined files are created by writing data definitions, using the data definition language described in 2.3. These data definitions are processed by the data definition processor which is similar to a COBOL compiler. The data definition processor writes the patterns for all defined records and defined files into the NAMEREC file and produces a diagnostic listing. Figure 2-4 illustrates the use of the data definition processor to create data definition records.

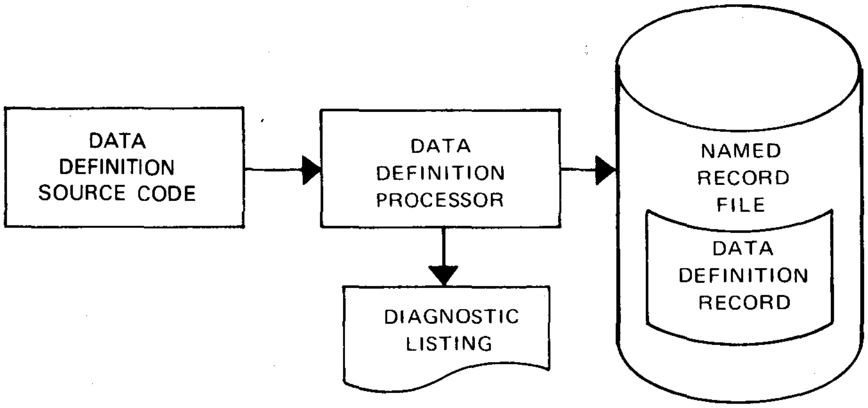


Figure 2—4. Data Definition Processing

2.2. DEFINED FILE

The defined file can be regarded as an indexed sequential file containing defined records tailored to the needs of an application. The tailoring begins with preliminary offline processing when you define to the IMS 90 data definition processor the data you want extracted for use in your specific application. This data definition simplifies both user action programs and UNIQUE because an action program need address only one defined file rather than several logical files whose records have to be matched and processed together in main storage. Defined files require no additional storage because they exist only by description and consist of all or parts of existing user data files extracted by DRM. The only storage space needed is disk storage space (NAMEREC file) to hold the data definition records.

Action programmers think of a defined file as an indexed sequential file containing records to be accessed via IMS 90 file I/O functions in the action program. It is possible for a defined file to be identical to an ISAM file. Other defined files can consist of different descriptions of the same user data file.

Terminal operators display or list defined records on the terminal. The image they see informs them of the defined file structure that they are querying and updating.

2.2.1. Hierarchical Structure

A defined file that contains more than one type of defined record has a hierarchical structure in which defined records have parent, child, and fraternal relationships. In Figure 2-5, defined record A1 is a parent to the child records B1, B2, and B3. But B1 is also a parent to C1, C2, and C3, and B3 is a parent to C4 and C5. In addition, C4 is a parent to D1.

Fraternal records are at the same level in the hierarchy; they can have the same parent or no parent. Thus, defined records A1 and A2 are fraternal; B1, B2, and B3 are fraternal; C1, C2, and C3 are fraternal; and C4 and C5 are fraternal. However, C1, C2, and C3 are not fraternal to C4 and C5 because they have a different parent.

In practice, most defined files contain few types of defined records. This example contains many for illustration.

Parent, child, and fraternal records must be defined in a prescribed order in the data definition and appear in that order in the defined file. A parent record is defined first, followed by each of the child records belonging to that parent. Each of these child records is followed by any child record to which it is a parent. Figures 2-6 and 2-7 show the order in which the defined records of the hierarchy shown in Figure 2-5 are defined. Figure 2-6 also illustrates the parent-child relationships in the defined file, and Figure 2-7 shows the fraternal relationships.

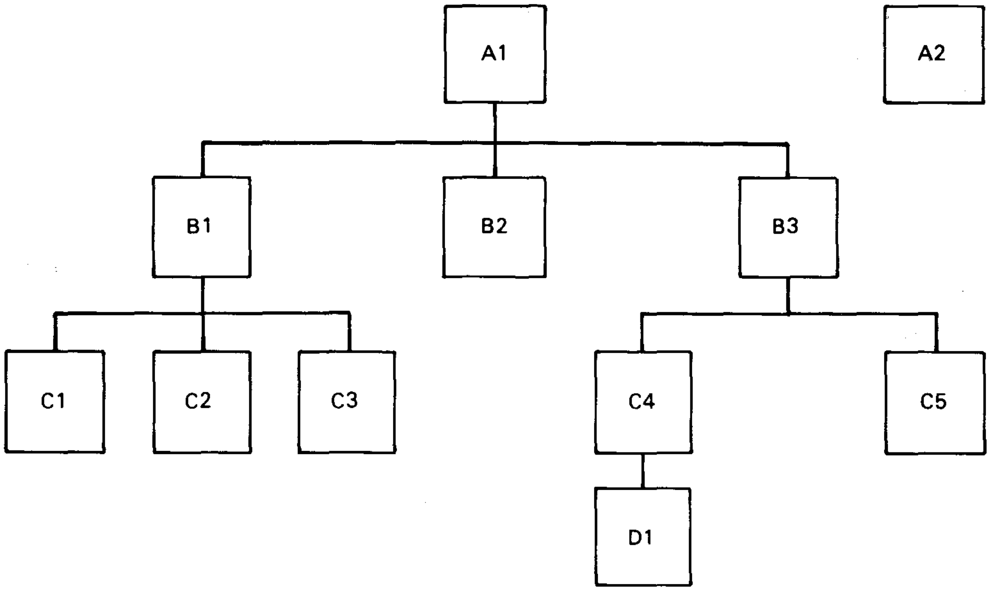


Figure 2—5. Hierarchical Structure of a Defined File

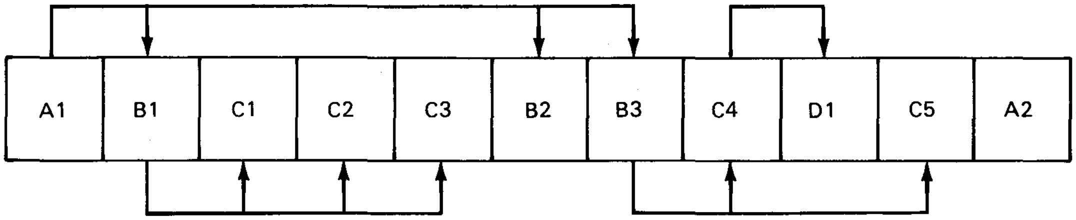


Figure 2—6. Parent/Child Relationships in a Defined File

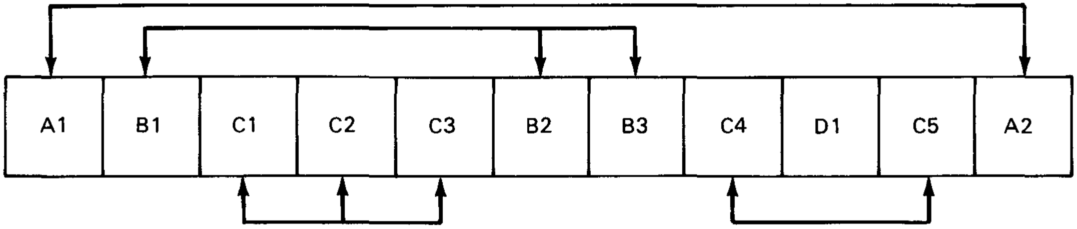


Figure 2—7. Fraternal Relationships in a Defined File

2.2.2. Defined Records

Defined records constitute the defined file. They contain the specific data needed by a particular application. You describe your defined records to the data definition processor, which processes the description for use by online IMS 90. In turn, DRM constructs the defined records and passes them to the action program or UNIQUE. DRM constructs the defined record containing record items in the same order specified in your defined record description.

Each defined record contains a record identifier. Similar to data management's use of ISAM keys to locate records, DRM uses the record identifiers to build keys that it passes to data management. Data management then uses these keys to extract data from the user logical data file.

The programmer describing his defined record to the data definition processor may specify a new item name on the IDENTIFIER statement of the item definition. He also specifies the name of the logical record from which the identifier item is being extracted. Figure 2-8 shows the derivation of the identifier from the logical record.

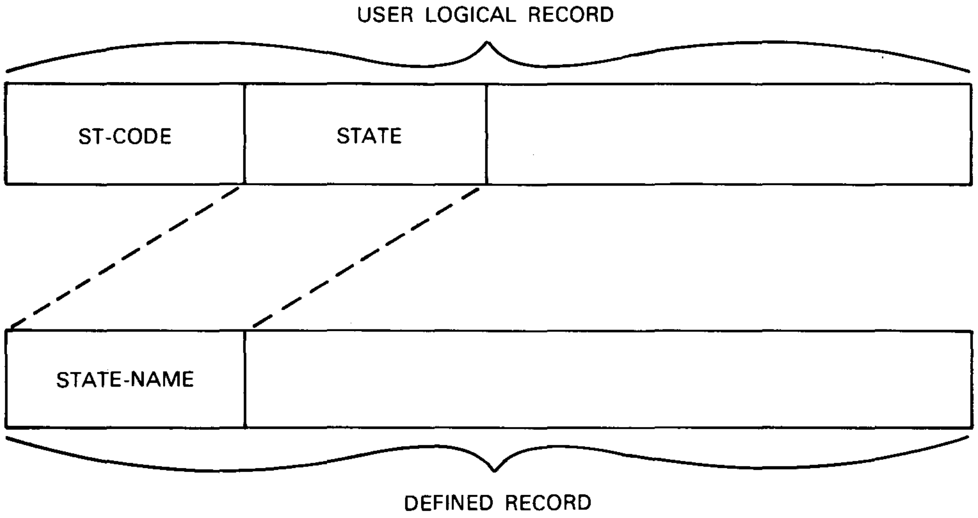
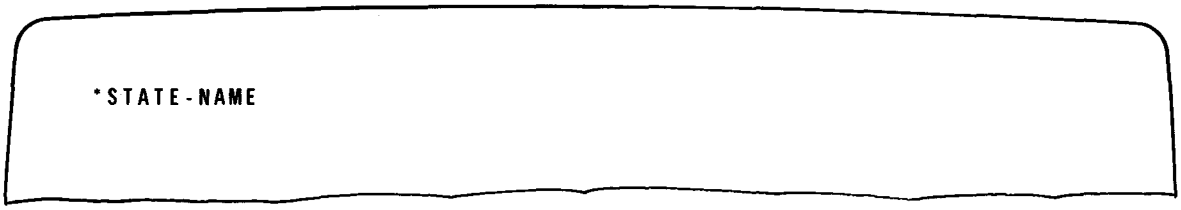


Figure 2—8. Defined Record Identifier Redefined from User Logical Record

The identifier item names appear on the terminal as column headers to the terminal operator using UNIQUE. Figure 2-9 illustrates a column header on the terminal display screen. (Column headers are normally followed by data as shown in Figure 2-11.)



```
*STATE-NAME
```

Figure 2—9. Terminal Display of Defined Record Identifier as Column Header

Each defined record can have only one identifier, which may have up to 39 items. To the action programmer, a defined record identifier consists of the first few items of the defined record; however, these identifier items can come from any data field of the logical record that is part of the key. Identifier items are arranged from left to right in major-to-minor order. Because identifier items are derived from key fields, only records in an indexed file or a data base subschema can supply these identifier items. Additional data items in a defined record are derived from the same record that supplies the identifier or from other records in conventional files or a subschema by defining *supplements* to the data definition. (See 2.3.7.) Nonindexed files may be used as a source for defined records only in combination with indexed files or subschemas. Defined record supplements may name a key or need not name a key. Thus, they may be derived from indexed or nonindexed files.

The identifier associated with the defined record is used by DRM to access that record. Figure 2-10 illustrates the derivation of a defined record identifier from one logical record in a simple defined file.

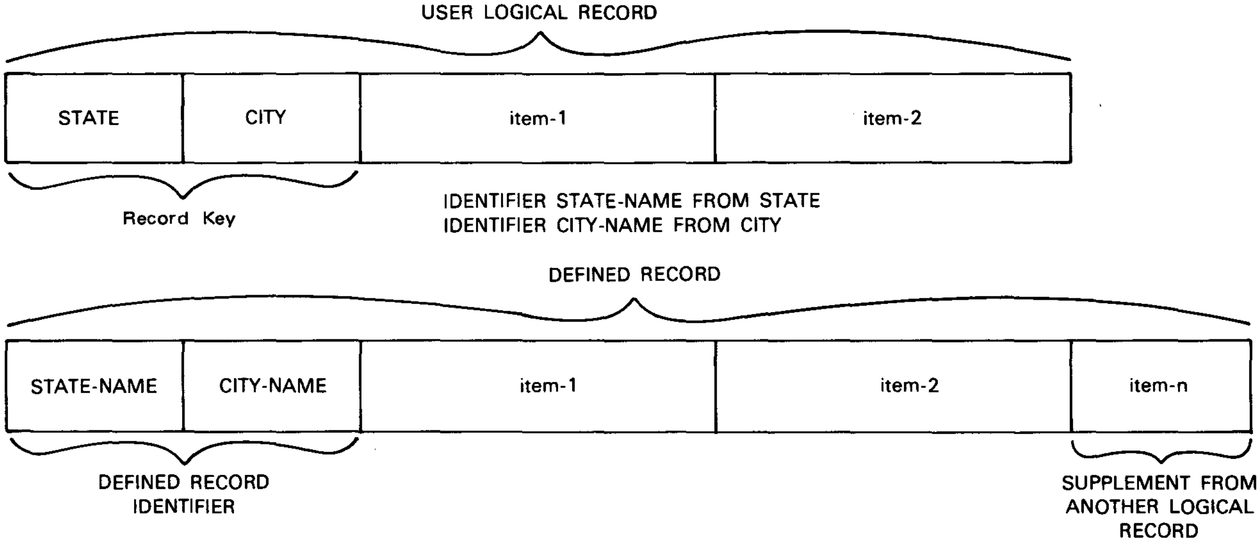
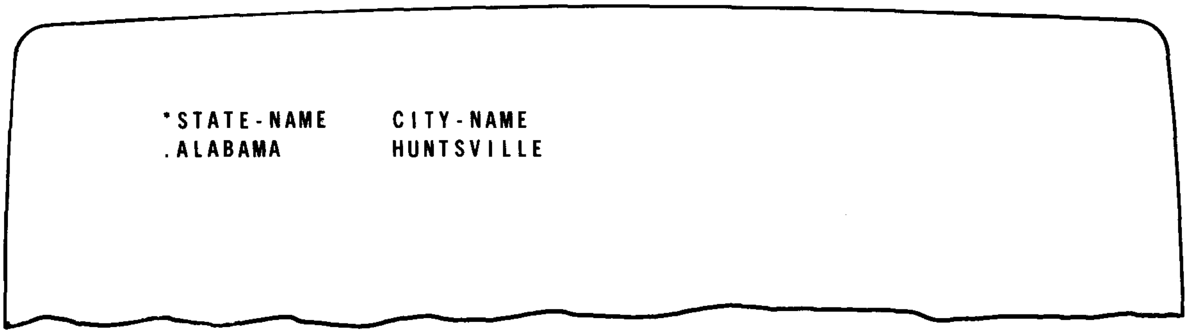

USER LOGICAL RECORD
STATE
CITY
item-1
item-2
Record Key
IDENTIFIER STATE-NAME FROM STATE
IDENTIFIER CITY-NAME FROM CITY
DEFINED RECORD
STATE-NAME
CITY-NAME
item-1
item-2
item-n
DEFINED RECORD IDENTIFIER
SUPPLEMENT FROM ANOTHER LOGICAL RECORD

Figure 2—10. Defined Record Identifier in a Simple Defined File

Figure 2-11 illustrates a defined record with item names used as column headers plus data comprising the defined record.



```
*STATE-NAME     CITY-NAME
.ALABAMA        HUNTSVILLE
```

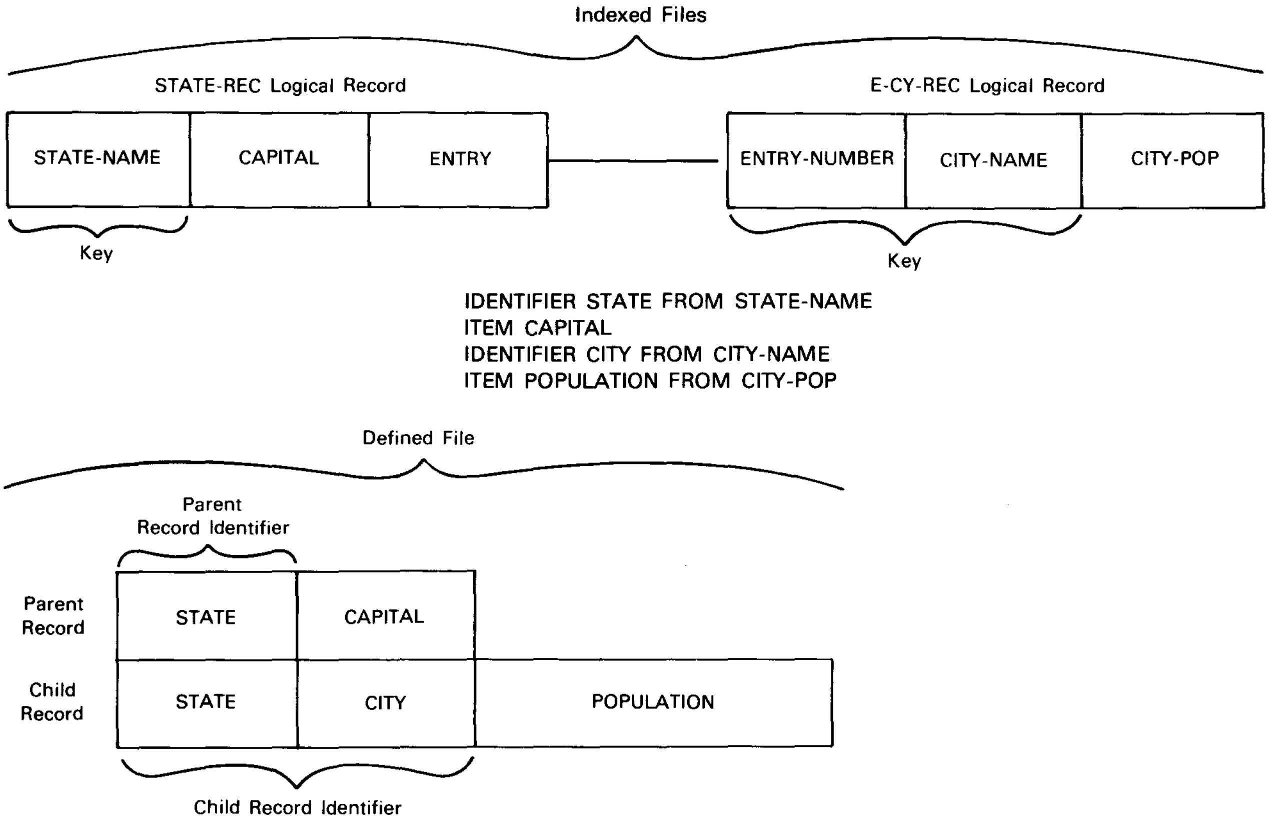
Figure 2—11. Terminal Display of Column Headers and Data Items

A hierarchical defined file contains two or more types of defined records that are related to each other in a parent/child relationship. Several occurrences of the child-type record are associated with each occurrence of the parent-type record.

The child record identifier is always longer than the parent record identifier because the entire value of the parent identifier is repeated at the beginning of the child identifier associated with that parent record. Hence, when listed at the terminal, those child records appear in order immediately following the parent record.

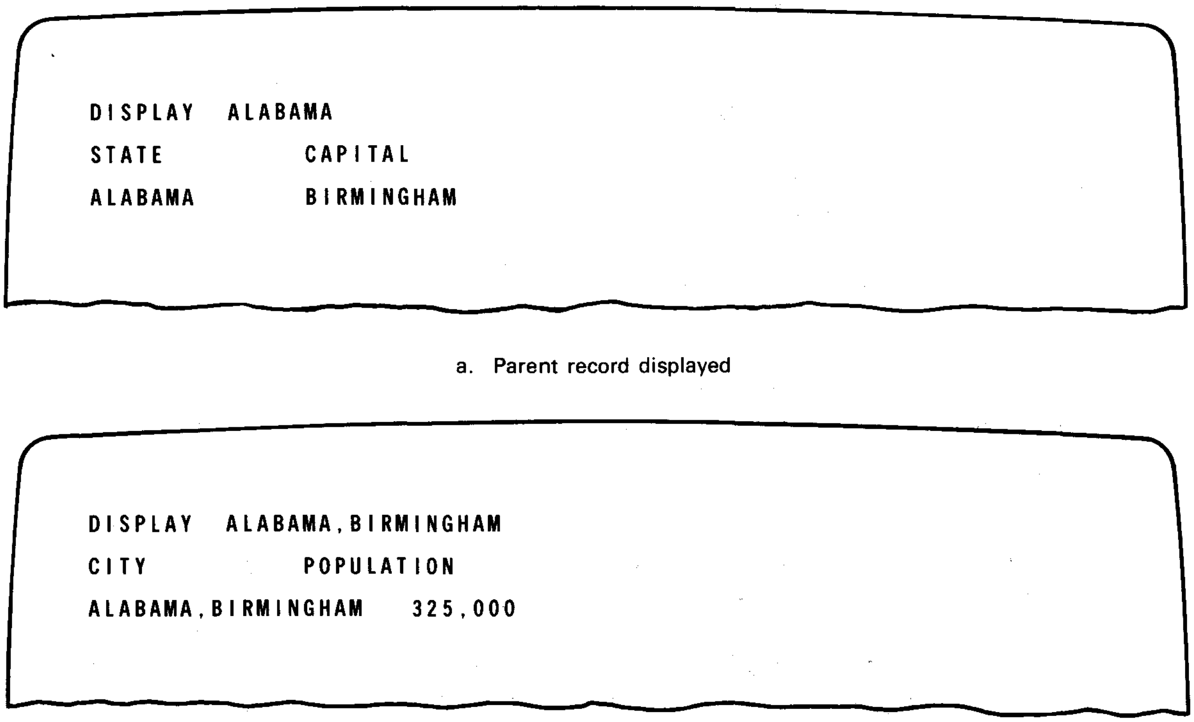
Additional items used in a child record identifier distinguish one child record occurrence from another. Note that the portion of the child identifier that is unique to each child record occurrence may come from a different file than the file that provided the parent record identifier. Figure 2-12 compares the parent and child record identifiers in a hierarchical defined file and shows their derivation from logical records in two different indexed files.

Figure 2-13 shows the terminal display of parent/child defined record identifiers in response to a request for the parent record or the child record. (See also Figure 2-32.)



```
IDENTIFIER STATE FROM STATE-NAME
ITEM CAPITAL
IDENTIFIER CITY FROM CITY-NAME
ITEM POPULATION FROM CITY-POP
```

Figure 2—12. Parent/Child Defined Record Identifiers



```
DISPLAY  ALABAMA
STATE         CAPITAL
ALABAMA       BIRMINGHAM
```

a. Parent record displayed

```
DISPLAY  ALABAMA,BIRMINGHAM
CITY          POPULATION
ALABAMA,BIRMINGHAM   325,000
```

b. Child record displayed

Figure 2—13. Terminal Displays of Column Headers and Data Items for Parent and Child Records

2.3. DATA DEFINITION LANGUAGE

The data definition language is a COBOL-like language used to describe the defined files accessed by user action programs or UNIQUE through defined record management. Each data definition describes one defined file in terms of one or more logical files. These may be indexed files or a combination of indexed and nonindexed files. You can also use a DMS 90 data base subschema as a source in a data definition. The data definition language for a data definition that uses subschema records as source differs from that described here. The syntax for such a data definition is documented in the IMS 90/DMS 90 interface user guide/programmer reference, UP-8748 (current version). Any number of defined files can be created through multiple runs of the data definition processor.

2.3.1. Format Presentation and Coding Rules

In addition to the statement conventions presented in Appendix A, the following rules apply to the presentation of formats in this section and the coding of input for the data definition processor.

1. Underlined lowercase terms, such as record-description and defined-record-definition, are names of formats detailed in subsequent illustrations.

2. Uppercase words are all reserved words.

   - All underlined uppercase words are required when the statements or clauses of which they are a part are used.

   - Uppercase words that are not underlined are optional, i.e., they may or may not be coded in the source program.

   - Uppercase words, whether underlined or not, must be spelled exactly as in the format.

3. A user-supplied word can be any sequence of not more than 30 characters, except for reserved words. Each character is taken from the set A through Z, 0 through 9, and the hyphen (-). The hyphen may not appear as the first or last character in a word.

4. User words are represented in the formats by generic terms where they are initially defined and also where they refer to previously defined user words. All references are in the backward direction; the definition always precedes the references.

5. Alphabetic and alphanumeric literals must be enclosed by single quotes to distinguish them from user words. Numeric literals are not enclosed by quotes.

6. Record definitions, and item definitions within them, are described in a defined file definition in exactly the same order as the logical data they represent appear in the defined file.

7. The standard COBOL coding form should be used for coding the data definition processor input. Certain statements must begin in margin A, as noted in the descriptions of those statements later in this section. Margin A is column 8 of the coding form. All other statements must begin at column 12 (margin B) or beyond.

8. Statements in the identification division and the data division are followed by periods. Periods and semicolons are optional throughout the defined file definition and are ignored by the data definition processor.

A list of reserved words that must not be used as user words in the definition division of the input to the data definition processor is presented in Table 2-1. Except for the words DEFINITION and DIVISION, these reserved words may be used in the data division. The COBOL reserved word list documented in the OS/3 extended COBOL supplementary reference, UP-8059 (current version) and OS/3 American National Standard COBOL programmer reference, UP-8613 (current version) applies to the data division of the data definition.

Table 2—1. Data Definition Reserved Words

ADD
ALL
ALLOW
ALSO
AND
ARE
AS
ASSUME
ASSUMES
BREAK
BY
CALC
CHANGE
CONTAINS
CONTROL
CONTROLLED
CONTROLLING
COUNT

DBS
DEFINED
DEFINITION
DELETE
DIVISION
DMS
DUPLICATE
FILE
FILL
FOLLOWS
FROM
GROUP
HIDDEN
IDENTIFIER
IN
IS
ITEM
KEY

KEY-NAME
MANUAL
MUST
NAME
NEUTRAL
NEXT-MEMBER-POINTER
OF
ONLY
OWNED
OWNING
PARENT
PASSWORD
PERIOD
POINTER
PREFIX
RECORD
REPEATING

ROLE
SELECTIVE
SEMICOLON
SET
SUBFILE
SUBRECORD
SUPPLEMENT
THROUGH
THRU
TO
TOTAL
TYPE
UPDATE
USING
VALUE
VALUES
VIA
WITHIN

2.3.2. Data Definition Structure

A data definition contains three divisions: the identification division, the data division, and the definition division. The identification division and the data division are intentionally similar to the COBOL identification and data divisions. The definition division is unique to IMS 90. The data division is used for describing the logical files from which the defined file is to be extracted; the definition division describes the defined file.

Figure 2-14 illustrates the overall structure of the data definition. The *record-description* and *defined-file-definition* entries are names of group formats that are expanded in Figures 2-15 and 2-16.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.data-definition-name.
[AUTHOR.comment-entry.]
[INSTALLATION.comment-entry.]
[DATE-WRITTEN.comment-entry.]
[DATE-COMPILED.comment-entry.]
[SECURITY.comment-entry.]
[REMARKS.comment-entry.]

DATA DIVISION.
FILE SECTION.
FD file-name-1.record-description [record-description]...
[FD file-name-2.record-description [record-description]...]...

DEFINITION DIVISION.
defined-file-definition
```

Figure 2—14. Overall Format of a Data Definition

2.3.2.1. Identification Division

The identification division must begin with the reserved words IDENTIFICATION DIVISION followed by a period and spaces to column 72. The PROGRAM-ID statement specifies the data definition name that appears on the diagnostic listing and serves to identify the contents of the listing. A data-definition-name is required on the PROGRAM-ID statement following the division header. The data-definition-name must be alphanumeric and begin with an alphabetic character. Although you may use more characters in the data-definition-name, the system uses only the first six characters to identify the object program on the compiler listing. Therefore, it is difficult to identify your programs when the first six characters are not unique. The first six characters of each data-definition-name within a given program library must be unique only to produce a unique object module. Each statement must begin in margin A (column 8) of the coding form. Each optional comment entry can contain any printable characters. If it exceeds a single line, additional lines must begin beyond column 11 in the coding form.

2.3.2.2. Data Division

The data division contains only a file section, in which the user logical files are described. It is similar to the standard COBOL file section, except that it cannot contain the VALUE clause. Other clauses are the same as in the OS/3 implementation of *American National Standard COBOL, X3.4—1968*.

You may specify multiple logical files, each containing multiple logical record descriptions.

The data division must begin with the reserved words DATA DIVISION followed by a period and spaces to column 72. DATA DIVISION, FILE SECTION, FD statements, and 01 level record descriptions must begin in margin A. All other entries must begin beyond column 11. The expressions *filename-1, filename-2,* etc, may be one to seven alphanumeric characters in length and must begin beyond column 11. The same file names must be specified by *filename* positional parameters in the configurator FILE section.

The *record-description* specification is illustrated and discussed in 2.3.3.

2.3.2.3. Definition Division

The definition division, unique to the data definition language, describes the defined file by designating a defined file name and describing each defined record, item, supplement, subrecord, subitem, and subfile.

The definition division must begin in margin A with the reserved words DEFINITION DIVISION followed by a period and spaces to column 72.

The *defined-file-definition* is described in 2.3.4.

2.3.3. Logical Data Record Description

The logical records that constitute conventional files from which the defined file is extracted are described in FD statements in the file section of the data division. (See Figure 2-14.) Nonindexed files can be used as source in a data division only in combination with indexed files.

Figure 2-15 illustrates the two formats that can be used to describe user logical records. The formats are intentionally similar to record descriptions in a COBOL data division. The first format copies data descriptions from an existing library; the second format describes the logical record items from which the defined records will be formed.

Format 1:

```
01 data-name-1; COPY library-name

   [REPLACING word-1 BY {word-2        }
                        {identifier-1  }
                        {literal-1     }

   [, word-3 BY {word-4       }] ...]
                {identifier-2 }
                {literal-2    }
```

Format 2:

```
level-number {data-name-2}
             {FILLER     }

     [;REDEFINES data-name-3]
     [;BLANK WHEN ZERO]
     [;{JUST     } RIGHT]
       {JUSTIFIED}
     [;{PIC    } IS character-string]
       {PICTURE}
     [;OCCURS {integer-1 TO integer-2 TIMES [DEPENDING ON data-name-4]}
              {integer-2 TIMES                                        }
      [{ASCENDING }KEY IS data-name-5 [,data-name-6] ...]
       {DESCENDING}
      [INDEXED BY index-name-1 [,index-name-2]...]                      ]
     [;{SYNC        } {LEFT }]
       {SYNCHRONIZED} {RIGHT}
     [; MAP IS integer-3 CHARACTERS]
     [; [SIGN is]{LEADING }[SEPARATE CHARACTER]]
                 {TRAILING}
     [; [USAGE IS] {COMP            }]
                   {COMPUTATIONAL   }
                   {COMP-3          }
                   {COMP-4          }
                   {COMPUTATIONAL-3 }
                   {COMPUTATIONAL-4 }
                   {DISPLAY         }
                   {INDEX           }
```

Figure 2—15. Logical Record Description Formats

2.3.4. Defined File Definition

The defined file definition contains the defined record; item; and optional supplement, subrecord,and subfile definitions that IMS 90 uses to construct the defined file from user logical data files (Figure 2-16). The defined file definition is coded in the definition division. IMS 90 requires that only one defined file be included in a data definition.

In the consolidated format for the defined file definition illustrated in Figure 2-16, statements enclosed in solid-line rectangles are required in a data definition, and statements enclosed in broken-line rectangles optionally can be included. Solid-line rectangles within broken-line rectangles indicate that the statements in the inner rectangle must be included if the statement in the outer rectangle is included.

The rectangles within rectangles illustrate a nested structure, with the defined record and subfile definitions subordinate to the DEFINED FILE definition and the item, supplement, and subrecord definitions subordinate to the defined record definition. (See also Figure 2-17.) All subordinate definitions can be used repeatedly within the larger definitions.

Periods and semicolons are optional throughout the defined file definition. The statements in each box of Figure 2-16 are described in 2.3.4.1 through 2.3.10.2.

2.3.4.1. DEFINED FILE Statement

The DEFINED FILE statement indicates the beginning of a defined file definition and supplies a name for future reference in password definition records and action programs. The DEFINED FILE statement must begin in margin A and be the first statement in the definition division.

Format:

```
DEFINED FILE defined-file-name[PASSWORD]
```

where:

`defined-file-name`
: Names the defined file and must be one to seven characters in length. The name must be different from that of any logical file assigned to IMS 90. The data definition processor truncates a defined-file-name longer than 7 characters and uses the first seven characters for the defined-file-name. There is no error message when truncation occurs.

`PASSWORD`
: Specifies that *defined-file-name* is to be used by terminal operators as the password in the UNIQUE OPEN command to gain access to this defined file. If PASSWORD is omitted, terminal operators using UNIQUE can access the defined file only if a password is defined by means of the NAMEREC file utility.

Note that multiple passwords may be used to access a defined file and that a password defined by means of the NAMEREC utility does not negate a password defined in the data definition unless the passwords are the same. If access to the defined file is to be limited to specific terminals, the PASSWORD option should be omitted and the NAMEREC utility should be used.

```
DEFINED FILE defined-file-name [PASSWORD]

   DEFINED RECORD defined-record-name-1

      {FROM stored-record-name-1                    }
      {FROM CONTROL BREAK IN stored-record-name-2   }
      {FROM REPEATING GROUP data-name-1             }
      [TYPE IS  literal-1 ]
      [PARENT IS defined-record-name-2]
      [PREFIX IS  literal-2 ]
      [{POINTER IS item-name-1[,item-name-2]]         }]
      [{FOLLOWS {defined-record-name-3}               }]
      [{        {supplement-name-1    }               }]
      [FILL KEY TO  literal-3 ]
      [ALLOW {ADD           }OF RECORD]
      [      {DELETE        }         ]
      [      {ADD AND DELETE}         ]

         {IDENTIFIER [item-name-1 FROM] data-name-1}
         {ITEM       [item-name-2 FROM] data-name-2}

         [HIDDEN]
         [MUST ADD]
         [ALLOW CHANGE]
         [{VALUE IS  }literal-1[{THROUGH} literal-2][ literal-3 [{THROUGH} literal-4]]...]
         [{VALUES ARE}         [{THRU   }          ][           [{THRU   }          ]]   ]

         SUPPLEMENT supplement-name-1

            {FROM stored-record-name-1        }
            {FROM REPEATING GROUP data-name-1 }
            [POINTER IS item-name-1 [,item-name-2]...]
            [FILL KEY TO  literal-1 ]
            [ASSUMES{CONTROLLING}ROLE IN UPDATE]
            [       {CONTROLLED }              ]
            [       {NEUTRAL    }              ]

               ITEM  [item-name-2 FROM] data-name-2

                  [HIDDEN]
                  [MUST ADD]
                  [ALLOW CHANGE]
                  [{VALUE IS  } literal-1 [{THROUGH} literal-2][,literal-3 [{THROUGH}literal-4]]...]
                  [{VALUES ARE}           [{THRU   }          ][           [{THRU   }          ]]   ]

      [ALSO [item-alias-1 FROM] item-name-3
            [,[item-alias-2 FROM] item-name-4]...]

         SUBRECORD subrecord-name-1 [OF subrecord-name-2]

            [ALLOW{ADD           }          ]
            [     {DELETE        }          ]
            [     {ADD AND DELETE}OF RECORD ]

               ITEM  [item-alias-1 FROM]{item-name-1 }
                                        {item-alias-2}

               [MUST ADD]
               [ALLOW CHANGE]
               [{VALUE IS  }literal-1 [{THROUGH}literal-2] [,literal-3[{THROUGH}literal-4]]...]
               [{VALUES ARE}          [{THRU   }         ] [          [{THRU   }         ]]   ]

   SUBFILE subfile-name-1 [PASSWORD]

         CONTAINS{defined-record-name-1}[,{defined-record-name-2}]...
                 {subrecord-name-1     }[ {subrecord-name-2     }]
```

Figure 2—16. Consolidated Format of Defined File Definition

Programming Note:

The defined file name also is used to create a record key for the data definition record. Other references to defined file names and subfile names outside the data definition itself include:

- keyword parameters DFILE and DDRECORD* in the ACTION section of the configuration;
- keyword parameters FN and DDN* in the password definition input to the NAMEREC file utility;
- the defined-file-name parameter in action program function calls to defined record management;
- the DEFINED-FILE-NAME and DATA-DEF-REC-NAME* fields in the program information block for COBOL action programs; and
- the ZA#PDFN and ZA#PDDRN* labels in the program information block DSECT for BAL action programs.

Examples:

```
   8   12
1. DEFINED FILE PAYROLL PASSWORD
2. FILE PAYROLL PASSWORD
3. FILE PAYROLL
```

Examples 1 and 2 perform exactly the same function. The word DEFINED can be omitted because it is not underlined in the format. The defined file name is PAYROLL, and the password used to access the file via UNIQUE also is PAYROLL.

In example 3, PASSWORD is omitted, precluding access to the file by a terminal operator using UNIQUE. Either a password is generated by the NAMEREC utility or the defined file is accessed only by user action programs, not UNIQUE.

*These can be defined file names but not subfile names.

2.3.5. Defined Record Definition

The defined record definition describes the source and contents of each defined record and the allowable operations. Figure 2–17 illustrates the format of the defined record definition. The underlined lowercase terms (item-definition, supplement-definition, and subrecord-definition) are names of group formats that are described in separate subsections. Items appear in the defined record in the same order that their item definitions appear in the defined record definition.

```
DEFINED RECORD defined-record-name-1
      ⎧FROM stored-record-name-1                           ⎫
      ⎨FROM CONTROL BREAK IN stored-record-name-2          ⎬
      ⎩FROM REPEATING GROUP data-name-1                    ⎭
      [TYPE IS literal-1]
      [PARENT IS defined-record-name-2]
      [PREFIX IS literal-2]
      ⎡⎧POINTER IS item-name-1[,item-name-2]...⎫⎤
      ⎢⎨FOLLOWS ⎧defined-record-name-3⎫        ⎬⎥
      ⎣⎩        ⎩supplement-name-1    ⎭        ⎭⎦
      [FILL KEY TO literal-3]
      ⎡ALLOW ⎧ADD             ⎫ OF RECORD⎤
      ⎢      ⎨DELETE          ⎬          ⎥
      ⎣      ⎩ADD AND DELETE  ⎭          ⎦
      item-definition [item-definition]...
      [supplement-definition]...
      [ALSO [item-alias-1 FROM] item-name-3
          [,[item-alias-2 FROM] item-name-4]...]
      [subrecord-definition]...
```

Figure 2—17. Defined Record Definition Format

2.3.5.1. DEFINED RECORD Statement

The DEFINED RECORD statement indicates the beginning of a defined record definition and assigns a name to the record being defined. It must be the first statement following the DEFINED FILE statement or another defined record definition and must begin in margin A of the coding form (column 8).

Format:

```
DEFINED RECORD defined-record-name-1
```

where:

`defined-record-name-1`
Is a 1- to 30-character name, unique within the data definition, that identifies the defined record.

Examples:

```
      8   12
   ┌──────────────────────────
1. │ DEFINED RECORD EMPLOYEE
2. │ RECORD EMPLOYEE
```

Again, both examples are the same; DEFINED is omitted in the second example because it is not required in the format. The first defined record in the defined file, PAYROLL, is named EMPLOYEE. If record types other than EMPLOYEE records exist in the defined file, one or more additional DEFINED RECORD statements must be coded in the definition division.

Programming Note:

IMS 90 accepts a maximum of 78 ITEM and IDENTIFIER statements for each defined record.

2.3.5.2. FROM Statement

The FROM statement identifies the logical record that supplies the primary part of this defined record. The logical record supplying the primary part must be from an indexed file. The primary part of a defined record contains the record identifier (ISAM or MIRAM key) and any items coming from the same logical record.

Format:

```
FROM stored-record-name-1
```

where:

`stored-record-name-1`
Refers to an 01 level entry in the file section of the data division.

Programming Notes:

1. The FROM statement must be the first statement following the DEFINED RECORD statement.

2. *Stored-record-name-1* must be a unique name or be fully qualified. (Rules for a fully qualified name are the same as for a COBOL fully qualified name.)

3. Any item within *stored-record-name-1* may be included in the primary part of this defined record if:

   - it meets constraints on length and usage (see 2.3.6.2); and

   - its position within *stored-record-name-1* comes before any item defined with an OCCURS clause.

Example:

```
8   12
DEFINED RECORD EMPLOYEE FROM EMPLOYEE-REC
```

In this example, the primary part of the defined record EMPLOYEE will be supplied by the logical record EMPLOYEE-REC, which is an 01 level entry in the file section of the data division of this data definition.

2.3.5.3. FROM CONTROL BREAK Statement

The FROM CONTROL BREAK statement specifies that the primary part of this defined record comes from the first of a sequence of logical records, all of which contain the same value in the leftmost characters of their record keys. In other words, the primary part of the defined record consists of the identifier only. The typical use of the FROM CONTROL BREAK statement is to access a specified portion of a defined file using, for example, the FOR parameter of a UNIQUE LIST or DETAIL command. The FROM CONTROL BREAK statement also can enable UNIQUE statistical functions to provide subtotals for subsets of child defined records associated with occurrences of the control break.

Format:

```
FROM CONTROL BREAK IN stored-record-name-2
```

where:

`stored-record-name-2`

Refers to an 01 level entry in the file section of the data division. The source record must be from an indexed file. When necessary to avoid ambiguity, *stored-record-name-2* must be fully qualified.

As indexed records are read sequentially, a new occurrence of the current defined record is generated each time a new value appears in those left-hand character positions of the logical record key that contributes this defined record's identifier. Typically, the indexed record that contains the identifier value (and is therefore the source of the current defined record) also will be the source of the next occurrence of the first subordinate defined record.

Programming Notes:

1. The FROM CONTROL BREAK statement must be the first statement following the DEFINED RECORD statement.

2. The current defined record must be named subsequently as the parent of at least one subordinate record. The first subordinate defined record must name *stored-record-name-2* as its source in a FROM statement or another FROM CONTROL BREAK statement.

Examples:

```
     8   12
1.  DEFINED RECORD EMPLOYEE
        FROM CONTROL BREAK IN EMPLOYEE-REC
2.  RECORD EMPLOYEE FROM BREAK EMPLOYEE-REC
```

Example 1 uses the long form of both the DEFINED RECORD and FROM CONTROL BREAK statements; example 2 illustrates the short form, using only the underlined reserved words in the format. Both perform the same function.

The defined record named EMPLOYEE will receive only an identifier from the logical record, EMPLOYEE-REC, which is a 01 level entry in the data division file section of this data definition.

2.3.5.4. FROM REPEATING GROUP Statement

The FROM REPEATING GROUP statement specifies the group item, described in the data division with an OCCURS clause, that supplies the primary part of this defined record. This statement must immediately follow the DEFINED RECORD statement.

Format:

```
FROM REPEATING GROUP data-name-1
```

where:

`data-name-1`
Refers to a data name defined in the data division with both an OCCURS clause and a KEY clause. When necessary to avoid ambiguity, *data-name-1* must be fully qualified.

Programming Notes:

1. The logical record that contains data-name-1 must not be created by a UNIQUE ADD command or a defined file INSERT function; otherwise, the value of data-name-1 is binary zeros and, therefore, cannot contain a unique key.

2. Any item within *data-name-1* may be included in the primary part of this defined record if:

   - it meets constraints of length and usage (see 2.3.6.2); and

   - its position within *data-name-1* precedes any item (other than *data-name-1* itself) defined as an OCCURS clause.

Example:

```
8   12
RECORD DEPENDENTS FROM GROUP DEPENDENT-REC
```

In this example, the primary part of the defined record DEPENDENTS is supplied by the repeating group item DEPENDENT-REC, which is an 02 level entry in the logical record EMPLOYEE-REC. (See example in 2.3.5.2.) IMS 90 requires that the item DEPENDENT-REC be described in the data division file section with both an OCCURS clause and a KEY clause. For example:

```
8   12
    Ø2 DEPENDENT-REC OCCURS 5 TIMES
       ASCENDING KEY IS DEP-NAME
       Ø3  DEP-NAME      PIC X(15)
```

2.3.5.5. TYPE Statement

The TYPE statement provides user-written action programs with an indicator identifying the record type delivered as a result of a SETL and sequential GET function. The type indicator is presented in the detailed status code of the program information block (PIB). This statement is applicable only when there is more than one record type in a given defined file and the file is being accessed by user action programs.

Format:

```
TYPE IS literal-1
```

where:

literal-1
: Is a single alphanumeric character. It is the actual value that is delivered in the DETAILED-STATUS-CODE field in the PIB. Each defined record type must be assigned a unique character identification.

Example:

```
8   12
    TYPE IS 'A'
```

2.3.5.6. PARENT Statement

The PARENT statement establishes the relationship of a defined record to other defined records in the hierarchical organization of the defined file. A defined record definition must always contain a PARENT statement unless the record being defined is at the highest level in the hierarchy.

Format:

```
PARENT IS defined-record-name-2
```

where:

defined-record-name-2

Refers to a defined record previously described in this data definition. It must have been defined in the immediately preceding defined record definition or be one of the direct ancestors of the immediately preceding defined record.

The position of defined records in the defined file reflects the order of their defined record definitions within the defined file definition.

The first defined record definition never contains a PARENT statement. It is at the highest level in the hierarchy and, therefore, cannot be subordinate to a parent-type record. If a subsequent defined record definition does not contain a PARENT statement, that defined record also is considered to be at the highest level in a hierarchy. All record types that have no parents are considered to be fraternal.

More than one defined record definition can name the same parent. Following each definition of a parent record, defined records subordinate to the parent record and fraternal to each other are defined.

Programming Notes:

A defined record can be named as the parent of another defined record only if one of four relationships exists between their sources in the logical records read from disk:

1. The source of the child record is a repeating group item occurring within the group that is the source of the parent or one of the parent's supplements. (See Figure 2-31.)

2. The sources of the parent record (or one of the parent's supplements) and the child are two distinct record types (01 level entries) that exist, collated, in the same indexed file. (See Figure 2-29.)

3. The source of the parent record is a control break that is detected while reading the source of the child record.

4. The source of the child record is a sequence of indexed records that exist somewhere entirely remote from the source of the parent or any of its supplements. In this case, IMS 90 requires a POINTER statement in the defined record definition for the child record. (See Figure 2-32.)

Example:

```
8   12
________________________
    PARENT IS EMPLOYEE
```

In this example, the parent of the child record being defined is the previously defined record EMPLOYEE.

2.3.5.7. PREFIX Statement

The PREFIX statement inserts an additional character or characters that are not present in any physical record into the identifier of a defined record, thus enabling identifier values to reflect the order of accessing or listing fraternal record types. Either the PREFIX statement or the VALUE statement (or both) must be included for each defined record that is fraternal to another defined record. The PREFIX statement must be used if the range of values for the identifier items of fraternal record types overlap. This can be the case if the records have sources in different files or in different repeating group items.

Format:

```
PREFIX IS literal-2
```

where:

literal-2

Is a constant inserted into the identifier of a fraternal-type record and must be enclosed by single quotes.

Programming Notes:

1. The values of successively defined prefixes for fraternal record types must be in ascending order and must also be the same length.

2. The prefix defined by this statement appears in the identifier, as seen by the action program and the terminal operator, of every occurrence of this defined record. It immediately precedes the identifier item defined in the first item definition for this defined record.

Example:

```
8   12
___________________
    PREFIX IS 'A'
```

In this example, assume that the defined file PAYROLL contains three types of records: EMPLOYEE, DEPENDENTS, and PAYDATA, with EMPLOYEE being the parent of both DEPENDENTS and PAYDATA, and DEPENDENTS and PAYDATA being fraternal records. When listing or accessing the file sequentially, all DEPENDENTS child-type records for an EMPLOYEE parent-type record are delivered before any PAYDATA records. Prefixes are identified as 'A' for DEPENDENTS records and 'B' for PAYDATA records. Because A comes before B alphabetically, and DEPENDENTS records occur before PAYDATA records in the defined file, the prefixes support the requirement that successive occurrences of defined records always contain identifiers with ascending values.

2.3.5.8. POINTER Statement

The POINTER statement defines the leftmost characters of the search key for the source of the defined record's primary part. The POINTER statement is used only if there is a PARENT statement for this defined record and the source of the primary part of this defined record and its parent exist in different files or at randomly different locations within the same file.

Format:

```
POINTER IS item-name-1 [,item-name-2]
```

where:

`item-name`
Has been defined in a direct ancestor of this defined record.

To retrieve the primary part of this defined record, IMS 90 concatenates the values of *item-name-1, item-name-2...* to form a character string. Then:

1. if the source of this defined record's primary part is an indexed record, that character string comprises the characters to the left of the identifier items in the record key; and

2. if the source is a repeating group item, then the leftmost characters of the character string are used to locate the occurrence of the record that contains the source. If this repeating group item is nested within a larger one, additional characters will be in the pointer and will be used to locate the larger group item by its key.

Example:

```
8     12
_____________________________
         POINTER IS EMP-NR
```

In this example, assume that the parent record is EMPLOYEE and you are defining a child record called DEPENDENTS. The source of the primary part of DEPENDENTS is an indexed file ordered by employee number and dependent name. The item EMP-NR of EMPLOYEE contains the employee number that points to the appropriate dependent records and is used to position the dependent's file.

2.3.5.9. FOLLOWS Statement

The FOLLOWS statement specifies that the source of this defined record is to be read sequentially following the source of a previously defined primary part or supplement in the same indexed file. It is required only if the source of the defined record does not follow the source most recently mentioned in a defined record definition or supplement definition involving the same indexed file.

Format:

```
FOLLOWS {defined-record-name-3}
        {supplement-name-1     }
```

where:

defined-record-name-3
Identifies a previous defined record so that the source of the primary part of the current defined record sequentially follows the source of the primary part of that defined record.

supplement-name-1
Identifies a previous supplement so that the source of the primary part of the current defined record sequentially follows the source of that supplement.

Programming Notes:

1. This statement never appears in the first defined record definition for a defined file or when the source of the defined record is a repeating group item.

2. This statement is used only if the source of the defined record is an indexed file already named as a source in a previous defined record definition or supplement definition.

Example:

```
8   12
---------------------
    FOLLOWS BRAND-RCD
```

In this example, assume that a liquor wholesale application has a logical ISAM file containing two types of user logical records: brand records and stock records. The file layout is a brand record followed by corresponding stock records. The stock records are the inventory record by unit of issue (i.e., pint, fifth, etc).

A defined record named BRAND-RCD has an item named SUBSTITUTE, which is a pointer to a supplement (i.e., the brand record "JIM BEAM" could have as a substitute "OLD CROW"). Another defined record named STOCK-RCD is defined after BRAND-RCD. Its definition uses the statement in this example to indicate that IMS 90 should read logical file stock records that follow the "JIM BEAM" record, rather than the logical file stock records that follow the "OLD CROW" record.

2.3.5.10. FILL KEY Statement

The FILL KEY statement specifies the rightmost characters of an indexed record key to differentiate between record types in an indexed logical file when they are identical for all records of the same type.

Format:

```
FILL KEY TO literal-3
```

where:

literal-3

Identifies the rightmost character or characters of the indexed file's key and must be enclosed by single quotes.

Programming Notes:

1. The FILL KEY statement is required only if the indexed file record key is longer than the combined length of the pointer items (if any) and the identifier items and if the remaining characters in the key are not all spaces (hexadecimal 40). When creating a search key, IMS 90 fills all remaining character positions with spaces and then moves *literal-3,* if specified, into the rightmost character positions of the record key.

   The length of literal-3 following the FILL KEY clause must not be longer than the part of the key not specified by IDENTIFIER or POINTER statements.

2. Indexed records not previously mentioned in the data definition are permitted to intervene in the source sequence if the FILL KEY statement is present.

Example:

```
8   12
___________________
    FILL KEY TO 'E'
```

In this example, assume there are two types of records in an indexed file being used as a source of defined records: EMPLOYEE-REC and DEPENDENT-REC. The key to the employee records is xxxxE, and the key to the dependent records is xxxxn. By defining a record EMPLOYEE with the FILL KEY statement in this example, you can access the EMPLOYEE-REC records simply by specifying a key of xxxx.

2.3.5.11. ALLOW ADD AND DELETE Statement

The ALLOW ADD AND DELETE statement permits the addition and/or deletion of occurrences of this defined record. This statement cannot be used if the FROM CONTROL BREAK or FROM REPEATING GROUP statement is included for this defined record.

Format:

```
ALLOW {ADD             } OF RECORD
      {DELETE          }
      {ADD AND DELETE  }
```

NOTE:

The absence of these statements in a record definition disables the corresponding functional capability. For example, if ALLOW ADD or ALLOW ADD AND DELETE is not specified in the EMPLOYEE record definition, any input of the UNIQUE ADD command or any action program issuance of the CALL INSERT function involving an EMPLOYEE record is rejected as invalid.

Example:

```
8   12
_________________________
    ALLOW ADD AND DELETE
```

This example allows the defined record EMPLOYEE (see example in 2.3.5.1) to be added to or deleted from the defined file.

2.3.5.12. ALSO Statement

The ALSO statement specifies that the defined record is to include copies of items described in definitions of its direct ancestors. Without the ALSO statement, these items can be included in the defined record only by defining a supplement (by means of the supplement definition) whose source is the same as the source of the direct ancestor record. Note that the ALSO statement must follow the item definition and any supplement definitions for this defined record.

Format:

```
ALSO [ITEM-alias-1 FROM] item-name-3 [,[item-alias-2 FROM] item-name-4]
```

where:

item-alias
: Specifies a unique name for an item within this defined file from 1 to 30 characters in length.

item-name
: Refers to an item defined in an item definition within a defined record definition for a direct ancestor of this defined record.

Example:

```
8   12
_____________________________
        ALSO EMP-NAME FROM NAME
```

In this example, the item EMP-NAME is included in the record being defined after the ancestor record item, NAME.

2.3.6. Item Definition

The defined record consists of elements or items known by their item names. A separate item definition is required to describe each item name. Figure 2–18 shows the item definition format.

IMS 90 accepts a maximum of 78 ITEM and IDENTIFIER statements. If more than 78 are processed, the data definition processor issues the following message:

MAX TABLE AREA FOR ITEM STATUS IS 78.

The data definition processor then terminates by indicating that it could not create the data definition record described.

If the items being defined are to be accessed via UNIQUE, it is important to consider carefully the size and meaning of the item names, because these item names are used as headings in all UNIQUE command response output. Indiscriminate assignment of item names can result in the inefficient use of CRT display screen space and erroneous interpretation of item contents.

Note that when using UNIQUE, allow one extra byte for UNIQUE to insert a tab stop control character.

```
{IDENTIFIER [item-name-1 FROM] data-name-1}
{ITEM       [item-name-2 FROM] data-name-2}
[HIDDEN]
[MUST ADD]
[ALLOW CHANGE]
[{VALUE IS  } literal-1 [{THROUGH} literal-2]
[{VALUES ARE}           [{THRU   }          ]

    [,literal-3 [{THROUGH} literal-4]] ...]
    [           [{THRU   }          ]]    ]
```

Figure 2—18. Item Definition Format

2.3.6.1. IDENTIFIER Statement

The IDENTIFIER statement indicates the beginning of an item definition and specifies the identifier for the defined record being described.

Format:

```
IDENTIFIER [item-name-1 FROM] data-name-1
```

where:

`item-name-1`
: Identifies the item, is 1 to 30 alphanumeric characters in length, and must be unique within the defined file definition. This name appears as a column header on the terminal when UNIQUE is used.

`data-name-1`
: Refers to a data name described in the data division as a part of the logical record or repeating group item that is the source of the primary part of this defined record. If the source of the primary part of this defined record is a logical record (either the FROM statement or the FROM CONTROL BREAK statement is employed), then *data-name-1* must be part of the record key of that logical record. If the source is a repeating group item, *data-name-1* must be part of the key of that item.

Programming Notes:

1. If *item-name-1* is identical to *data-name-1,* then *item-name-1* and the word FROM can be omitted.
2. Definitions of identifier items must precede those of all other items.
3. If more than one IDENTIFIER statement is present, items must be defined in major-to-minor order.

Example:

```
8   12
_____________________________________
    IDENTIFIER EMP-NR FROM EMPL-NR
    IDENTIFIER EMP-NM FROM EMPL-NAME
```

This example illustrates the specification of multiple IDENTIFIER statements, the first indicating the major identifier, the second indicting a minor identifier. The value of the record key in the record that is the source of this defined record is:

EMPL-NR△△△EMPL-NAME△△△△△△△△△△△

The sequence of items defined by the IDENTIFIER statement appears at the terminal as a string of variable-length items separated by commas:

```
EMP-NR,EMP-NM
```

When more than one IDENTIFIER statement is used, UNIQUE uses a single item name to refer to the entire identifier string. This item name is derived from the final IDENTIFIER statement. If the two defined items are individually named EMP-NR and EMP-NM, as in the example, then the item name for the combination, as seen by the terminal operator, is simply EMP-NM.

2.3.6.2. ITEM Statement

The ITEM statement indicates the beginning of an item definition and includes in the defined record an item described in the data division file section.

Format:

```
ITEM [item-name-2 FROM] data-name-2
```

where:

`item-name-2`
Identifies the item, may be 1 to 30 characters in length, and must be unique within the defined file definition. This name appears as a column header on the terminal when UNIQUE is used.

`data-name-2`
Refers to a data name described in the data division as a part of the logical record or repeating group item that is the source of the primary part or supplement of this defined record. *Data-name-2* must never be qualified; qualification by the name of the source is always implied.

Programming Notes:

1. If *item-name-2* is identical to *data-name-2, item-name-2* and the word FROM may be omitted.

2. The item named in this statement should not exceed 72 bytes. Moreover, it should not exceed line length minus 2 if UNIQUE is to display this item at any terminal whose line length is shorter than 74 characters.

3. Defined record management moves the values of items to a new or updated source record in the order in which they are defined. If data-name-2 overlaps the source of another item (either item is a group item that contains the other), then the second item moved covers up the first. Therefore, if either item is to be changed, that item must be defined after the other item.

4. The data definition language permits a single item value on disk to be the source of two items within the same defined record. If you attempt to update those items to different new values, the resulting value on disk is unpredictable.

5. *Data-name-2* must be one of the following:

   - an elementary item with a USAGE clause value other than COMP-1 or COMP-2; or

   - a group item that can be treated as if it were an elementary alphanumeric item; i.e., it contains only alphabetic, alphanumeric, or unsigned numeric items whose usage is DISPLAY.

Example:

```
8   12
________________________
    ITEM EMP-NAME FROM NAME
    ITEM AGE
```

The items NAME and AGE from the current logical record are included in the defined record being described. The item NAME is assigned the name EMP-NAME, and the item age retains the name AGE.

2.3.6.3. HIDDEN Option

The HIDDEN option prevents the data item defined by the ITEM statement from being displayed at the terminal in response to a UNIQUE command. The purpose of this option is to allow a subsequent POINTER statement to refer to the item without allowing that item to be displayed. The HIDDEN option has no effect on a defined record accessed by user-written action programs.

Another use of the HIDDEN clause assures the validity of a numeric field in a logical record on which another program may later want to perform arithmetic. When you add a defined record that does not include all fields in a logical record, binary zeros are inserted in the missing fields. To avoid this problem, include the field in the defined record with an ITEM statement and restrict its use with the HIDDEN clause.

Format:

```
HIDDEN
```

Programming Notes:

1. If the current item definition begins with an IDENTIFIER statement, the specification of the HIDDEN option is ignored.

2. When a terminal operator using UNIQUE adds a defined record having an item definition for which the HIDDEN option is specified, IMS 90 automatically inserts spaces or zeros in the corresponding item of the record on disk. If the item is defined as alphanumeric, IMS 90 inserts spaces; if numeric, IMS 90 inserts zeros in the data format appropriate to the declared usage of the item.

Example:

```
8   12
------------------------
    ITEM DEP-KEY HIDDEN
```

Assume that DEP-KEY is a pointer to the supplemental dependent record in a nonindexed file. There would be no value in displaying this data at a terminal.

2.3.6.4. MUST ADD Option

The MUST ADD option specifies that this item must be present and contain a valid value for a record to be added to the defined file by the terminal operator. If defined as numeric, the item must be nonzero; if alphanumeric, it must contain other than all spaces.

Format:

```
MUST ADD
```

Programming Notes:

1. This option has meaning only in item definitions that begin with the ITEM statement; identifier items always must be present to add a defined record.

2. If this item is in a supplement, the ROLE IN UPDATE for that supplement should be CONTROLLED.

3. This option is inoperative unless the ALLOW ADD statement is specified in the defined record definition.

Example:

```
8   12
---------------------------
    ITEM EMP-NAME MUST ADD
```

Before an EMPLOYEE record (see example in 2.3.5.1) can be added to the defined file, the item EMP-NAME must contain a valid value. Obviously, an employee record would be of little value without an employee name. Probably an item called AGE would not have the MUST ADD option because the item is not critical to the record's validity and can be added later via update.

2.3.6.5. ALLOW CHANGE Option

The ALLOW CHANGE option permits changes to be made to the current item from the terminal. If this option is not specified, IMS 90 refuses to carry out any requested changes to records on disk.

Format:

```
ALLOW CHANGE
```

Programming Notes:

1. If ALLOW CHANGE is not specified and the action program calls upon the PUT function and delivers to IMS 90 a record in which the value of this item has been changed, IMS 90 returns control to the action program with an invalid request indicator (003) in the program status code.

2. This option has meaning only in item definitions that begin with the ITEM statement; identifier items cannot be changed.

3. If this item is in a supplement, the ROLE IN UPDATE for that supplement should be CONTROLLED.

Example:

```
8   12
____________________________________
    ITEM MARITAL-STATUS ALLOW CHANGE
```

This example specifies that the item MARITAL-STATUS can be changed in the defined record to which it belongs.

2.3.6.6. VALUE Statement

The VALUE statement specifies the valid ranges of values an item may have when it is being added or changed. IMS 90 checks the validity of the item when any type of update (ADD, CHANGE, PUT, or INSERT) is requested and carries out the requested function only if the value of the item lies within the specified ranges. If the VALUE statement is omitted, any value consistent with the PICTURE and USAGE specified in the data division for the source of this item is acceptable.

Format:

```
{VALUE IS  } literal-1 [{THROUGH} literal-2]
{VALUES ARE}           [{       }          ]
     [, literal-3 {THROUGH} literal-4] ...
     [            {THRU   }          ]
```

where:

literal-1, literal-2,...

Specify the allowable values or ranges of values for an item being added or changed. The values of *literal-1, literal-2,* ... must be in ascending order. Their lengths must be exactly equal to each other and to the item named by *data-name-1* or *data-name-2* in the ITEM statement. Alphanumeric literals must be enclosed in single quotes; numeric literals are not.

If the item being defined is an identifier item, IMS 90 employs the ranges specified in the VALUE statement to recognize the defined record type. When an identifier is supplied to IMS 90 by an action program or by a terminal operator using UNIQUE, these range tests are applied to the string of items comprising the identifier. When IMS 90 is asked to produce the next defined record in a sequence, it applies the range tests to the items comprising the record key in the records read from disk. The effect of applying the VALUE clause to an identifier item in this manner is to disable access to records with keys outside the range specified. This could be an effective tool for file segmentation; i.e., processing A through E, F through L, M through R, and S through Z segments of a payroll file in stages.

Programming Notes:

1. The VALUE statement must be used for fraternal record types having the same source (i.e., their primary parts come from successive occurrences of the same indexed record) and their value ranges must not overlap.

2. When the VALUE statement is used for fraternal record types from different sources and the value ranges of their identifiers overlap, the PREFIX statement must also be included.

3. The VALUE statement must be used if occurrences of an indexed record contributing to this defined record must be distinguished from successive occurrences of the same indexed records that do not contribute to the defined file.

Example:

```
8   12
    ITEM HOURLY-RATE ALLOW CHANGE VALUE IS Ø225 THRU 15ØØ
```

In this example, the item HOURLY-RATE can be changed, but new values must fall between 225 and 1500 or the update is rejected.

2.3.7. Supplement Definition

IMS 90 determines the existence of a defined record by obtaining the source of its primary part. A defined record can contain additional items from other logical records or repeating groups. Items coming from a logical record or repeating group other than the source of the primary part must be defined in a supplement definition. Figure 2-19 shows the format of the supplement definition. Statements in the supplement definition must follow the same sequence shown in Figure 2-19. The item definition shown in the format is required and is the same as the item definition for a defined record (2.3.6 through 2.3.6.6). The item definition for a supplement, however, cannot start with an IDENTIFIER statement; it must begin with an ITEM statement.

```
SUPPLEMENT supplement-name-1
    {FROM stored-record-name-1                }
    {FROM REPEATING GROUP data-name-1         }
    [POINTER IS item-name-1[,item-name-2] ...]
    [FILL KEY TO literal-1]
    [ASSUMES {CONTROLLING}ROLE IN UPDATE ]
    [        {CONTROLLED  }                ]
    [        {NEUTRAL     }                ]
    item-definition [item-definition]...
```

Figure 2—19. Supplement Definition Format

2.3.7.1. SUPPLEMENT Statement

The SUPPLEMENT statement indicates the beginning of a supplement definition and supplies a name for future reference within the data definition. This statement must begin in margin A (column 8) and be the first statement in the supplement definition.

Format:

```
SUPPLEMENT supplement-name-1
```

where:

`supplement-name-1`
Identifies the supplement, is 1 to 30 characters in length, and must be unique within the data definition.

Example:

```
8   12
_____________________
SUPPLEMENT DEPENDENT
```

This example identifies a supplement named DEPENDENT.

2.3.7.2. FROM Statement

The FROM statement designates the record description in the data division that describes the source of this supplement. The FROM statement must immediately follow the SUPPLEMENT statement.

Format:

```
FROM stored-record-name-1
```

where:

```
stored-record-name-1
```

Refers to a 01 level group item in the data division file section.

Programming Note:

Any item within *stored-record-name-1* may be included in this supplement if:

- it meets constraints of length and usage (see 2.3.6.2); and
- its position within *stored-record-name-1* comes before any item defined with the OCCURS clause.

Example:

```
8   12
___________________________________________
SUPPLEMENT DEPENDENT FROM DEPENDENT-RECORD
```

The contents of the supplement DEPENDENT are supplied by the logical record DEPENDENT-RECORD, which is a 01 entry in the data division file section.

2.3.7.3. FROM REPEATING GROUP Statement

The FROM REPEATING GROUP statement designates the item, described in the data division with an OCCURS clause, that is to be the source of this supplement. If used, it must immediately follow the SUPPLEMENT statement.

Format:

```
FROM REPEATING GROUP data-name-1
```

where:

```
data-name-1
```

Refers to a data name defined in the data division with both an OCCURS clause and a KEY clause.

Programming Notes:

1. If *data-name-1* is contained within one or two larger group items in the data division that are also described with the OCCURS clause, IMS 90 requires those descriptions to include the KEY clause.

2. The logical record that contains *data-name-1* must not be created by a UNIQUE ADD command or a defined file INSERT function; otherwise, the value of *data-name-1* is binary zeros and therefore, cannot contain a unique key.

3. Any item within *data-name-1* may be included in this supplement if:

   - it meets constraints on length and usage (see 2.3.6.2); and

   - its position within *data-name-1* comes before any item (other than *data-name-1* itself) defined with an OCCURS clause.

Example:

```
8   12
____________________________________________________
SUPPLEMENT DEPENDENT FROM REPEATING GROUP DEPENDENTS
```

In this example, the contents of the supplement DEPENDENT are supplied by the repeating-group item, DEPENDENTS, which is a 02 level entry in the file section of the data division. DEPENDENTS is described in the data division with both an OCCURS clause and a KEY clause as follows:

```
8   12
____________________________________________________________
    Ø2 DEPENDENTS OCCURS 5 TIMES ASCENDING KEY IS DEP-NAME
       03  DEP-NAME          PIC X(15)
```

2.3.7.4. POINTER Statement

The POINTER statement designates the items whose values are employed to locate a particular occurrence of this supplement's source record. This statement is required when the source of this supplement is a repeating group item or a record in a nonindexed file.

Format:

```
POINTER IS item-name-1[,item-name-2] ...
```

where:

`item-name-1,item-name-2`
Refer to items previously defined in the current defined record definition or in a direct ancestor of the current defined record.

The values of *item-name-1, item-name-2* ... are concatenated from left to right to form a character string called the POINTER. Then, if the source of this supplement is a repeating group item, as many right-hand characters as necessary are used to match against key items. The remaining left-hand characters are used to create a record key. These characters are justified to the left when constructing a reference key for accessing an indexed record. They are right-justified to create a relative record number in a nonindexed file. Relative record numbers are filled to the left with binary 0's. An indexed record key is filled to the right with spaces (hexadecimal 40). Finally, the rightmost characters of an indexed record key are made equal to literal-1 as specified in the FILL KEY statement, if any.

If the source of this supplement is a record in an indexed file and that record is one of a sequence of records that contribute items to the same defined record, the POINTER statement is employed only if the source of this supplement is the first record of that sequence. The values of the record keys of those records must differ only in their least significant (rightmost) character positions, as specified by literal-1 of the FILL KEY statement. The values of the most significant characters of the record are the same for all records in the sequence and are determined according to the definition of the first source in the sequence. If that is the source of a supplement rather than the primary part of this defined record, a POINTER statement must be included in that supplement definition.

Examples:

```
     8   12
1.  |    POINTER IS REC-KEY
2.  |    POINTER IS EMP-NO,DEP-NAME
```

In the first example, assume the primary part is EMPLOYEE and the supplement is DEPENDENT. Thus, REC-KEY contains a relative record number pointing to the logical record in a DAM file that contains the DEPENDENT data for the particular EMPLOYEE identifier in the primary part.

In the second example, assume the same situation except that the DEPENDENT data comes from a repeating group item whose key is equal to DEP-NAME and which is contained in a record with a key equal to EMP-NO.

2.3.7.5. FILL KEY Statement

The FILL KEY statement specifies the rightmost characters of a record key. It is required if there is no POINTER statement or if the POINTER statement does not specify all the characters of a record key, and the remaining right-hand characters must have a value other than spaces (hexadecimal 40).

Format:

```
FILL KEY TO literal-1
```

where:

literal-1
: Specifies the rightmost character or characters of a record key and must be enclosed in single quotes.

If there is no POINTER statement, the value of literal-1 must be greater than spaces (greater than hexadecimal 40) and also greater than the value of literal-1 in the FILL KEY statement, if any, in the immediately preceding supplement definition. This is because each indexed record must have a record key whose value is greater than that of the record key of the immediately preceding record in the file.

The length of literal-1 following the FILL KEY clause must not be longer than the part of the key not specified by IDENTIFIER or POINTER statements.

Example:

```
8   12
------------------
    FILL KEY TO 'P'
```

This example applies to two applications. In the first, assume that payroll and dependent records are included in an indexed file. The EMPLOYEE record keys are emp-no,△, the DEPENDENT record keys are emp-no,D, and the PAYROLL records are emp-no,P. By specifying FILL KEY TO 'P', no POINTER statement is necessary to generate a pointer to the source of the supplement, PAYROLL, assuming the EMPLOYEE record was already named as a source for this defined record.

Second, assume that the source of the supplement is a separate indexed file containing two types of records: PAYROLL and DEPENDENT. The same principle applies as in the first situation except that the POINTER statement is required, but only for the first supplement.

2.3.7.6. ROLE IN UPDATE Statement

The ROLE IN UPDATE statement specifies how the source of this supplement affects or is affected by the addition or deletion of an occurrence of this defined record.

Format:

```
ASSUMES ⎧CONTROLLING⎫ ROLE IN UPDATE
        ⎨CONTROLLED ⎬
        ⎩NEUTRAL    ⎭
```

where:

CONTROLLING
: Specifies that addition of an occurrence of the defined record is not to take place unless the corresponding occurrence of the source of this supplement already exists. In this way, the values of the items that point to this supplement can be validated. Deletion of the defined record is not affected.

CONTROLLED

Specifies that the occurrence of the source of this supplement is to be added or deleted whenever an occurrence of the defined record is added or deleted. The source of this supplement must not be a repeating group item. If the source of this supplement already exists when addition is requested, a new occurrence of that source will replace the old.

NEUTRAL

Indicates that the source of this supplement neither affects nor is affected by the addition or deletion of an occurrence of the defined record. This option is selected by default when the ROLE IN UPDATE statement is absent.

Examples:

```
      8   12
     ________________________________________
1.  |      ASSUMES CONTROLLING ROLE IN UPDATE
2.  |      ASSUMES CONTROLLED
3.  |      ASSUMES NEUTRAL
```

In the first example, assume that a primary part is an inventory record and the supplement being defined is the vendor information pertaining to this inventory record. The CONTROLLING option is used because normally you do not want to allow adding of an inventory record if no vendor information is available.

In the second example, assume that a primary part is an employee record and the supplement being defined is the payroll information pertaining to that employee. The CONTROLLED option is specified because the adding or deleting of the employee information always requires the corresponding adding or deleting of the payroll information.

In the third example, assume that a primary part is a requisition record and the supplement being defined is from an inventory master record. The NEUTRAL option is specified because normally you do no want the deletion or addition of a requisition record that is of a temporary nature to affect the status of the inventory master record that is of a permanent nature. The same effect also is obtained by omitting the ROLE IN UPDATE statement.

2.3.8. Subrecord Definition

Two or more variants of the same defined record can exist in the same data definition. They can differ in number of items included, positioning of the items, spelling of item names (used as column headers by UNIQUE), and authorization to update. A subrecord definition is required for each variant. The format of the subrecord definition is shown in Figure 2-20.

```
SUBRECORD subrecord-name-1[OF subrecord-name-2]
    ┌ALLOW┌ADD            ┐OF RECORD┐
    │     │DELETE         │         │
    └     └ADD AND DELETE ┘         ┘
    [subitem-definition]. . . .
```

Figure 2—20. Subrecord Definition Format

All identifier items of the defined record are automatically included in each of its subrecords. The IDENTIFIER statement is used in a subrecord definition only when a new item name is desired. Any other item is included in the subrecord only if a subitem definition is present.

2.3.8.1. SUBRECORD Statement

The SUBRECORD statement indicates the beginning of a subrecord definition and supplies a name for future reference within the data definition. It must begin in margin A (column 8).

Format:

```
SUBRECORD subrecord-name-1
```

where:

`subrecord-name-1`
: Identifies the subrecord, is 1 to 30 characters in length, and must be unique within the data definition.

Example:

```
8    12
____________________
SUBRECORD EMPLOYEE-SUB1
```

In this example, a variant of the defined record EMPLOYEE is defined as a subrecord called EMPLOYEE-SUB1. A possible reason is that information on dependents is contained in a nonindexed file, which cannot be the source of a defined record's primary part, and the dependent information must be presented interspersed with information from the defined record's primary part. This cannot be done in a defined record. Thus, a variant of the defined record must be named in the SUBRECORD statement.

2.3.8.2. OF Statement

The OF statement is used only when other subrecord definitions have already appeared for a defined record. It simplifies the writing of subitem definitions when item names of *subrecord-name-1* and *subrecord-name-2* are mostly the same.

Format:

```
OF subrecord-name-2
```

where:

subrecord-name-2
Refers to a previously defined subrecord within this defined record definition.

Example:

```
8   12
____________________________________
SUBRECORD EMPLOYEE-SUB2 OF EMPLOYEE-SUB1
```

In this example, the subrecord EMPLOYEE-SUB2 is defined. The OF statement indicates that the items that will make up EMPLOYEE-SUB2 are identified in the definition of subrecord EMPLOYEE-SUB1.

2.3.8.3. ALLOW ADD AND DELETE Statement

The ALLOW ADD AND DELETE statement permits the addition and deletion of occurrences of this subrecord. If this statement is not included, addition or deletion of the defined record is not possible when it is accessed through a subfile containing this record.

Format:

```
ALLOW(ADD              )OF RECORD
     {DELETE           }
     (ADD AND DELETE   )
```

NOTE:

The absence of these statements in a subrecord definition disables the corresponding functional capability. For example, if ALLOW ADD or ALLOW ADD AND DELETE is not specified in the DEPENDENT-SUB1 subrecord definition, any input of the UNIQUE ADD or DELETE commands or any issuance of the CALL INSERT or DELETE functions involving the EMPLOYEE-SUB1 subrecord is rejected as invalid.

Example:

```
8   12
_____________________
    ALLOW ADD AND DELETE
```

This example allows occurrences of the subrecord identified by the SUBRECORD statement to be added to or deleted from the defined file.

2.3.9. Subitem Definition

The components of the subrecord correspond to items previously defined in item definitions in the defined record definition. Except for identifier items, which are included automatically, a subitem definition is required for each item in the subrecord. Figure 2-21 shows the format of the subitem definition.

```
ITEM [item-alias-1 FROM] {item-name-1 }
                         {item-alias-2}
[MUST ADD]
[ALLOW CHANGE]
[{VALUE IS  } literal-1 [{THROUGH} literal-2]
[{VALUES ARE}           [{THRU   }          ]
              [,literal-3 [{THROUGH} literal-4]]...]
              [           [{THRU   }          ]]   ]
```

Figure 2—21. Subitem Definition Format

2.3.9.1. ITEM Statement

The ITEM statement includes into the subrecord a previously identified specific item from a preceding defined record or subrecord and optionally supplies a new name (alias) by which the item is known within this subrecord. The ITEM statement must be the first statement in the subitem definition.

Format:

```
ITEM item-alias-1 FROM {item-name-1 }
                       {item-alias-2}
```

where:

`item-alias-1`
: Provides a name for the subitem, is 1 to 30 characters in length, and must be unique within the subrecord definition. If specified, this name appears as a column header to the terminal operator accessing this subrecord through UNIQUE.

`item-name-1`
: Refers to an item previously defined within this defined record definition. If this subrecord definition does not include an OF statement, the *item-name-1* option is required.

`item-alias-2`
: Refers to a subitem named in a previous subrecord definition; that subrecord must be defined as *subrecord-name-2* in the OF statement for the current subrecord. This option is required if the OF statement is included.

Programming Note:

If *item-alias-1* is identical to *item-name-1* (or item-alias-2), *item-alias-1* and the word FROM can be omitted.

Examples:

```
      8   12
   ┌──────────────────────────────
1. │       ITEM LAST-NAME
2. │       ITEM LNAME FROM LAST-NAME
```

Both examples include the item LAST-NAME that was previously defined in a record definition or subrecord definition. Example 1 retains the same name for the item; example 2 assigns a unique name.

2.3.9.2. MUST ADD Option

The MUST ADD option specifies that this subitem must be present and contain a valid value in order to add an occurrence of the defined record if it is accessed via a subfile containing this subrecord. If defined as numeric, the item must be nonzero; if alphanumeric, it must contain other than all spaces. This option is valid only when the ALLOW ADD or ALLOW ADD AND DELETE statement has been applied to the subrecord definition.

Format:

```
MUST ADD
```

See 2.3.6.4 for an example.

2.3.9.3. ALLOW CHANGE Option

The ALLOW CHANGE option permits changes to be made to the current item when the defined record is accessed through a subfile containing this subrecord.

Format:

```
ALLOW CHANGE
```

Refer to 2.3.6.5 for additional information.

2.3.9.4. VALUE Statement

The VALUE statement specifies the valid ranges of values an item may have when it is added or changed, if the defined record is accessed through a subfile containing the current subrecord. If the VALUE statement is omitted, any value consistent with the PICTURE and USAGE specified in the data division for the source of this item is acceptable.

Format:

```
{VALUE IS  }literal-1 [{THROUGH}literal-2]
{VALUES ARE}          [{THRU   }         ]
      [,literal-3 [{THROUGH}literal-4]]...
      [           [{THRU   }         ]]
```

where:

literal-1, literal-2,...

Specify the allowable values or ranges of values for a subitem being added or changed. The values of *literal-1, literal-2,* ... must be in ascending order. Alphanumeric literals must be enclosed in single quotes; numeric literals are not.

Programming Notes:

1. The number of literals specified may not exceed 64.

2. IMS 90 will not update the defined record if the resulting value of this item is new and is not within one of the specified ranges. Instead, the status code 003 (invalid operation) is returned in the program information block.

2.3.10. Subfile Definition

Two or more variants of the same defined file can exist in the same data definition. They can differ in number of defined record types and in the makeup of each type of defined record. A subfile definition is required for each additional variant. It describes a subset of a defined file independently and provides the means of accessing subrecords. Subrecords are accessible only via a subfile. Figure 2-22 shows the format of a subfile definition.

```
SUBFILE subfile-name-1[PASSWORD]
    CONTAINS{defined-record-name-1} [,{defined-record-name-2}] ...
            {subrecord-name-1     } [ {subrecord-name-2     }]
```

Figure 2—22. Subfile Definition Format

2.3.10.1. SUBFILE Statement

The SUBFILE statement indicates the beginning of a subfile definition and supplies an identifying name for future reference in password definition records and action programs. This statement must begin in margin A (column 8).

Format:

```
SUBFILE subfile-name-1[PASSWORD]
```

where:

`subfile-name-1`
: Identifies the subfile, is one to seven characters in length, and must be unique among defined file and subfile names within the data definition. It also must be different from the name of any conventional user file assigned to IMS 90.

`PASSWORD`
: Specifies that *subfile-name-1* is to be used as the password in the UNIQUE OPEN command to gain access to this subfile. If PASSWORD is omitted, terminal operators using UNIQUE can access the subfile only if a password is defined by means of the NAMEREC file utility.

Programming Note:

Subfile names are used in the same way as defined file names within the data definition and where reference is made to them outside the data definition. Refer to the programming note in 2.3.4.1.

Example:

```
8   12

SUBFILE EMP-FILE PASSWORD
SUBFILE PAY-FILE
```

Assume a defined file has two record types, EMPLOYEE and PAYROLL. To restrict access to the PAYROLL file, two subfiles, EMP-FILE and PAY-FILE, are defined. The EMP-FILE can be accessed by all terminal operators, using the name of the subfile as the password. The PAY-FILE can be accessed only by those terminals named in a password definition submitted to the NAMEREC file utility.

2.3.10.2. CONTAINS Statement

The CONTAINS statement identifies the defined records and subrecords included in this subfile.

Format:

```
CONTAINS {defined-record-name-1} [,{defined-record-name-2}] ...
         {subrecord-name-1     } [ {subrecord-name-2     }]
```

where:

defined-record-name-1,defined-record-name-2,...
Identify defined records included in this subfile.

subrecord-name-1,subrecord-name-2,...
Identify subrecords included in this subfile.

Programming Notes:

1. No more than one entry can be included for each defined record; it can be either the defined record name or a subrecord name.

2. Entries must be in the same order as their corresponding defined record definitions appeared previously in the data definition.

3. Before an entry is submitted for a defined record, an entry must be submitted for every direct ancestor of that defined record.

Example:

```
8   12

    CONTAINS EMPLOYEE-SUB
```

In this example, the subfile EMP-FILE consists of just one type of defined record, under its subrecord name EMPLOYEE-SUB.

2.4. DATA DEFINITION EXAMPLES

2.4.1. Example of Simple Defined File

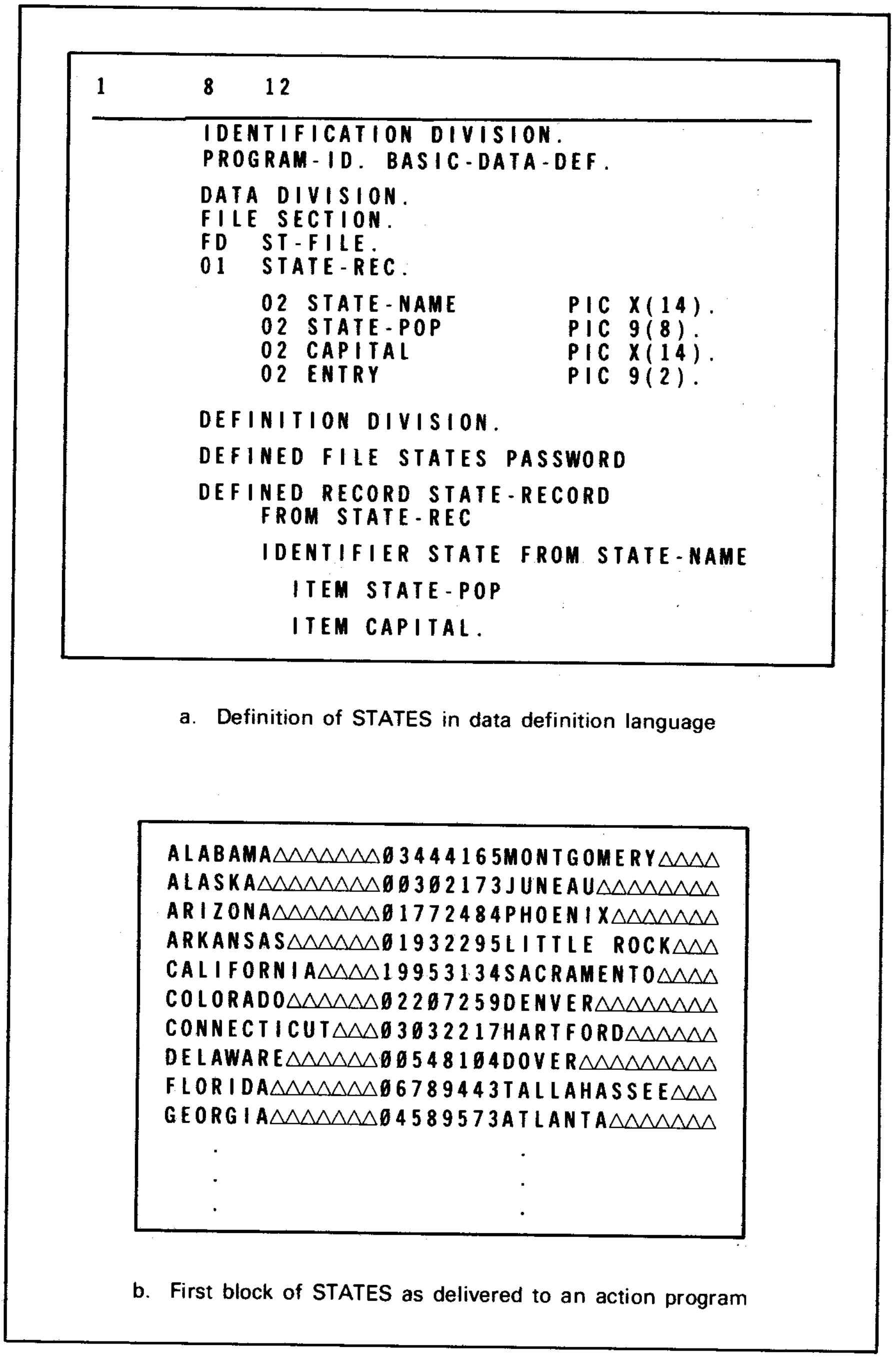
Data definition language used with a simple defined file named STATES is shown in Figures 2-23 and 2-24. The STATES file is derived from ST-FILE, whose first few records are shown in Figure 2-23. A 14-byte field at the beginning of each record contains its key. The record is named STATE-REC and the key is named STATE-NAME.

The data definition coding is shown in Figure 2-24a. The first few records of STATES are shown in Figure 2-24b as IMS 90 delivers them to action programs, and in Figure 2-24c as UNIQUE lists them at terminals. Each record contains an identifier item named STATE, and two other items named STATE-POP and CAPITAL. These names appear at terminals as column headers.

The defined record is named STATE-RECORD. The programmer accessing the defined file, STATES, must provide a place for the defined record to be received in his action program. Figure 2-24d shows how a record area for STATE-RECORD is described on a coding form in a COBOL action program. Figure 2-24e shows the same for BAL.

```
ALABAMA△△△△△△△Ø3444165MONTGOMERY△△△△22
ALASKA△△△△△△△△ØØ3Ø2173JUNEAU△△△△△△△△49
ARIZONA△△△△△△△Ø1772484PHOENIX△△△△△△△48
ARKANSAS△△△△△△Ø1932295LITTLE ROCK△△△25
CALIFORNIA△△△△19953134SACRAMENTO△△△△31
COLORADO△△△△△△Ø22Ø7259DENVER△△△△△△△△38
CONNECTICUT△△△Ø3Ø32217HARTFORD△△△△△△Ø5
DELAWARE△△△△△△ØØ5481Ø4DOVER△△△△△△△△△Ø1
FLORIDA△△△△△△△Ø6789443TALLAHASSEE△△△27
GEORGIA△△△△△△△Ø4589573ATLANTA△△△△△△△Ø4
     .                          .               .
     .                          .               .
     .                          .               .
```

Figure 2—23. Excerpt from a Sample Indexed State File (ST-FILE)



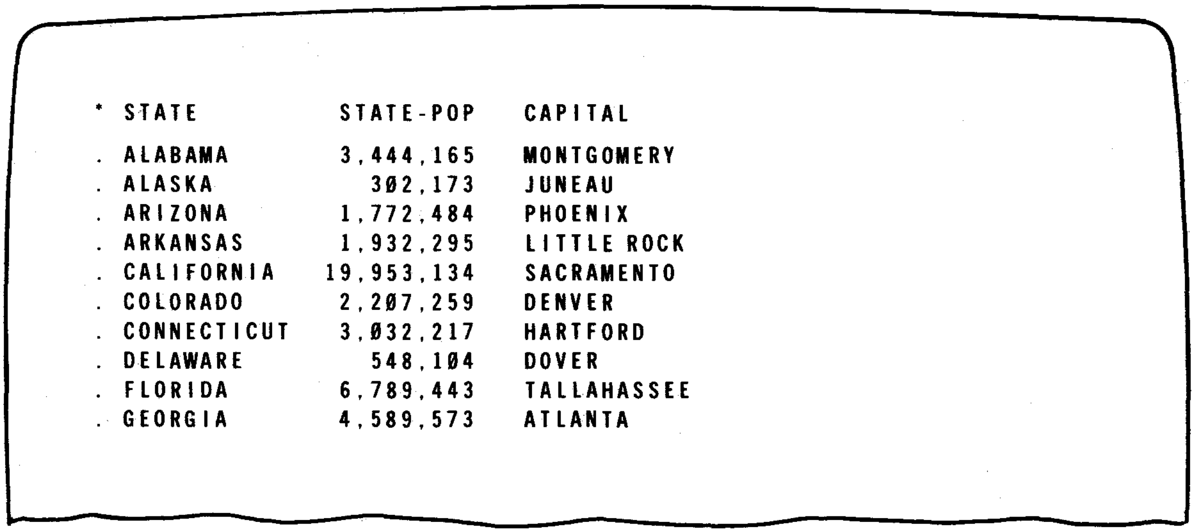
```
1      8   12
----------------------------------------------------
       IDENTIFICATION DIVISION.
       PROGRAM-ID. BASIC-DATA-DEF.
       DATA DIVISION.
       FILE SECTION.
       FD  ST-FILE.
       01  STATE-REC.
           02 STATE-NAME        PIC X(14).
           02 STATE-POP         PIC 9(8).
           02 CAPITAL           PIC X(14).
           02 ENTRY             PIC 9(2).
       DEFINITION DIVISION.
       DEFINED FILE STATES PASSWORD
       DEFINED RECORD STATE-RECORD
           FROM STATE-REC
           IDENTIFIER STATE FROM STATE-NAME
             ITEM STATE-POP
             ITEM CAPITAL.
```

a. Definition of STATES in data definition language

```
ALABAMA△△△△△△△Ø3444165MONTGOMERY△△△△
ALASKA△△△△△△△△ØØ3Ø2173JUNEAU△△△△△△△△
ARIZONA△△△△△△△Ø1772484PHOENIX△△△△△△△
ARKANSAS△△△△△△Ø1932295LITTLE ROCK△△△
CALIFORNIA△△△△19953134SACRAMENTO△△△△
COLORADO△△△△△△Ø22Ø7259DENVER△△△△△△△△
CONNECTICUT△△△Ø3Ø32217HARTFORD△△△△△△
DELAWARE△△△△△△ØØ5481Ø4DOVER△△△△△△△△△
FLORIDA△△△△△△△Ø6789443TALLAHASSEE△△△
GEORGIA△△△△△△△Ø4589573ATLANTA△△△△△△△
    .                   .
    .                   .
    .                   .
```

b. First block of STATES as delivered to an action program

Figure 2—24. Defined File STATES (Part 1 of 2)



```
* STATE          STATE-POP   CAPITAL

. ALABAMA        3,444,165   MONTGOMERY
. ALASKA           3Ø2,173   JUNEAU
. ARIZONA        1,772,484   PHOENIX
. ARKANSAS       1,932,295   LITTLE ROCK
. CALIFORNIA    19,953,134   SACRAMENTO
. COLORADO       2,2Ø7,259   DENVER
. CONNECTICUT    3,Ø32,217   HARTFORD
. DELAWARE         548,1Ø4   DOVER
. FLORIDA        6,789,443   TALLAHASSEE
. GEORGIA        4,589,573   ATLANTA
```

c. First block of STATES as listed at a terminal display by UNIQUE

```
1      8   12
------------------------------------------
       Ø1  WORK-AREA
           Ø2 STATE-RECORD.
              Ø3 STATE PIC X(14).
              Ø3 STATE-POP PIC 9(8).
              Ø3 CAPITAL PIC X(14).
           Ø2 S-STATE-RECORD.
              Ø3 S-STATE     PIC X.
              Ø3 S-STATE-POP PIC X.
              Ø3 S-CAPITAL   PIC X.
```

d. Description of STATE-RECORD in COBOL action program

```
1      8   12
----------------------------------------------------
WORK     DSECT      WORK AREA
RECORD   EQU    *
SNAME    DS     XL14 STATE NAME
SPOP     DS     XL8  STATE POPULATION
SCAPITAL DS     XL14 STATE CAPITAL

SNAME#S  DS     X    STATE NAME STATUS BYTE
SPOP#S   DS     X    STATE POPULATION STATUS BYTE
SCAP#S   DS     X    STATE CAPITAL STATUS BYTE
```

e. Description of STATE-RECORD (labeled RECORD) in BAL action program

Figure 2—24. Defined File STATES (Part 2 of 2)

2.4.2. Example of Subfile

An example of the use of a subfile definition to restrict access to a defined file is shown in Figure 2–25. Compare this example to the example shown in Figure 2–24. Both data definitions deal with the same source data (the ISAM logical file ST-FILE in Figure 2–23). Both data definitions make the defined file STATES available to action programs and, via UNIQUE, to the terminal operator.

Where a subrecord is defined, it can be accessed only via the subfile, which must be described in the SUBFILE and CONTAINS statements (Figure 2–25a). Figures 2–24b through 2–24e still apply to the new data definition as well as the old, where data was accessed via the defined file name STATES. The new data definition, however, also makes the subfile, SUBFIL, available to action programs (including UNIQUE). Thus, b and c in Figure 2–25 illustrate the data that can be accessed via the subfile name, SUBFIL. In this case, only two items are delivered to the action program. Their item names, as employed by UNIQUE, are changed from STATE and STATE-POP to NAME-OF-STATE and POPULATION, respectively. Figures 2–25d and 2–25e show how the programmer provides a place for subrecords to be received in a COBOL or BAL action program.

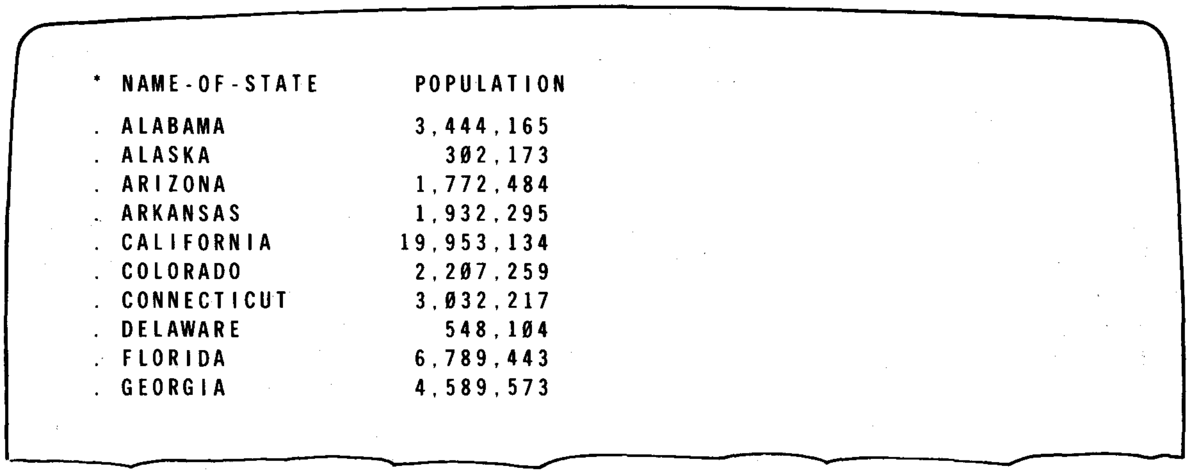
```
1      8   12
-----------------------------------------------------
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SUB-DEF.
       DATA DIVISION.
       FILE SECTION:
       FD  ST-FILE.
       01  STATE-REC.
           02 STATE-NAME        PIC X(14).
           02 STATE-POP         PIC 9(8).
           02 CAPITAL           PIC X(14).
           02 ENTRY             PIC 9(2).
       DEFINITION DIVISION.
       DEFINED FILE STATES PASSWORD
       DEFINED RECORD STATE-RECORD
           FROM STATE-REC
           IDENTIFIER STATE FROM STATE-NAME
           ITEM STATE-POP
           ITEM CAPITAL
       SUBRECORD SUB-STATES
           IDENTIFIER NAME-OF-STATE FROM STATE
           ITEM POPULATION FROM STATE-POP
       SUBFILE SUBFIL PASSWORD
           CONTAINS SUB-STATES.
```

a. Definition of STATES and SUBFIL

Figure 2—25. Subfile Definition Restricting Access to a Defined File (Part 1 of 2)

```
ALABAMA△△△△△△△Ø3444165
ALASKA△△△△△△△△ØØ3Ø2173
ARIZONA△△△△△△△Ø1772484
ARKANSAS△△△△△△Ø1932295
CALIFORNIA△△△△19953134
COLORADO△△△△△△Ø22Ø7259
CONNECTICUT△△△Ø3Ø32217
DELAWARE△△△△△△ØØ5481Ø4
FLORIDA△△△△△△△Ø6789443
GEORGIA△△△△△△△Ø4589573
      .             .
```

b. First block of SUBFIL as delivered to an action program



```
* NAME-OF-STATE        POPULATION
. ALABAMA               3,444,165
. ALASKA                  3Ø2,173
. ARIZONA               1,772,484
. ARKANSAS              1,932,295
. CALIFORNIA           19,953,134
. COLORADO              2,2Ø7,259
. CONNECTICUT           3,Ø32,217
. DELAWARE                548,1Ø4
. FLORIDA               6,789,443
. GEORGIA               4,589,573
```

c. First block of SUBFIL as listed at a terminal by UNIQUE

```
1      8   12
---------------------------------------------
       Ø1  WORK-AREA.
           Ø2  SUB-STATES.
               Ø3 NAME-OF-STATE PIC X(14).
               Ø3 POPULATION PIC 9(8).
           Ø2  S-SUB-STATES.
               Ø3 S-NAME-OF-STATE PIC X.
               Ø3 S-POPULATION    PIC X.
```

d. Description of SUB-STATES in COBOL action program

```
1        10     16
------------------------------------------------------
WORK     DSECT        WORK AREA
SUBREC   EQU    *
SNAME    DS     XL14   STATE NAME
SPOP     DS     XL8    STATE POPULATION
SNAME#S  DS     X      STATE NAME STATUS BYTE
SPOP#S   DS     X      STATE POPULATION STATUS BYTE
```

e. Description of SUB-STATES (labeled SUB-REC) in BAL action program

Figure 2—25. Subfile Definition Restricting Access to a Defined File (Part 2 of 2)

2.4.3. Example of Supplements in Defined File

The data definition for CITIES in Figure 2-27a illustrates the use of supplements in a defined file. Each defined record in CITIES comes from three different logical records on disk. Two of these come from the indexed file CI-FILE, an excerpt of which is shown in Figure 2-26. Another comes from ST-FILE, shown in Figure 2-23.

The records of the indexed file CI-FILE occur in pairs. There are two records for ABERDEEN, two for ABILENE, etc. The first supplies the primary part of CITY-RECORD; the second supplies a supplement. Both have record keys in the same character positions: 1 through 22. The values of these keys differ only in character position 22. The first record contains the space character and the second record contains the number 1. These values cause the logical records to be in ascending order and identify the type of logical record (CITY-REC or CITY-REC-TRAILER), a necessary function in case one of the pair is missing. IMS 90 operates on the single entity represented by the defined record. Therefore, it adds, deletes, and displays both types of records together. If the first record is missing, the second is ignored by IMS 90. If the second record is missing, IMS 90 supplies spaces for the item named STATE.

The second file ST-FILE contains a single type of logical record, STATE-REC. that contributes a supplementary part of the defined record. It is accessed by means of a pointer. As a record in an indexed file, STATE-REC contains a record key. The pointer that IMS 90 constructs from NAME-STATE in the CITY-REC-TRAILER logical record is used as a search key to match against the record key STATE-NAME in the STATE-REC record. In this way, the secondary part is located. If ST-FILE is a nonindexed file, there will be no record key in the STATE-REC record. The pointer will be a file relative record number instead. It still will be constructed in the same way and used for the same purpose as an indexed file key. If IMS 90 fails to find a STATE-REC record, it supplies zeros for the item named STATE-POP in the defined record supplement.

```
ABERDEEN△△△△△△△△△△△△△△ØØ26467△△△△△△△
ABERDEEN△△△△△△△△△△△△△1SOUTH△DAKOTA△△
ABILENE△△△△△△△△△△△△△△△ØØ89653△△△△△△△
ABILENE△△△△△△△△△△△△△△1TEXAS△△△△△△△△△
ALAMEDA△△△△△△△△△△△△△△△ØØ7Ø968△△△△△△△
ALAMEDA△△△△△△△△△△△△△△1CALIFORNIA△△△△
ALBANY△△△△△△△△△△△△△△△△Ø175781△△△△△△△
ALBANY△△△△△△△△△△△△△△△1NEW△YORK△△△△△△
    .                    .
    .                    .
```

Figure 2—26. Indexed File with Two Logical Records for Each City (CI-FILE)

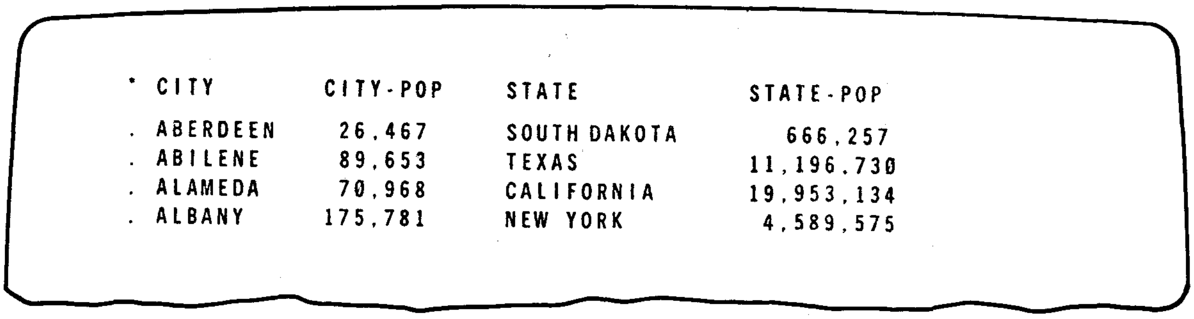
```
1      8   12
-------------------------------------------------------------------
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SEC-PART-DEF.
       DATA DIVISION.
       FILE SECTION.
       FD  CI-FILE.
       01  CITY-REC.
           02 CITY-NAME          PIC X(21).
           02 RCD-TYPE           PIC X.
           02 CITY-POP           PIC 9(7).
           02 FILLER             PIC X(7).
       01  CITY-REC-TRAILER.
           02 CITY-ID            PIC X(21).
           02 TYPE-RCD           PIC X.
           02 NAME-STATE         PIC X(14).
       FD  ST-FILE.
       01  STATE-REC.
           02 STATE-NAME         PIC X(14).
           02 STATE-POP          PIC 9(8).
           02 CAPITAL            PIC X(14).
           02 ENTRY              PIC 9(2).
       DEFINITION DIVISION.
       DEFINED FILE CITIES
       DEFINED RECORD CITY-RECORD
           FROM CITY-REC
           ALLOW ADD AND DELETE
           IDENTIFIER CITY FROM CITY-NAME
           ITEM CITY-POP
       SUPPLEMENT CITY-PART-1
           FROM CITY-REC-TRAILER
           FILL KEY TO '1' ASSUMES CONTROLLED ROLE IN UPDATE
           ITEM STATE FROM NAME-STATE
       SUPPLEMENT CITY-PART-2
           FROM STATE-REC
           POINTER IS STATE
           ITEM STATE-POP.
```

a. Data definition of CITIES showing supplements

Figure 2—27. Defined File CITIES (Part 1 of 2)

```
.ABERDEEN△△△△△△△△△△△△△ØØ26467SOUTH△DAKOTA△△ØØ666257
.ABILENE△△△△△△△△△△△△△△ØØ89653TEXAS△△△△△△△△△1119673Ø
.ALAMEDA△△△△△△△△△△△△△△ØØ7Ø968CALIFORNIA△△△△19953134
.ALBANY△△△△△△△△△△△△△△△Ø175781NEW△YORK△△△△△△Ø4589575
```

b. CITIES as delivered to an action program



```
* CITY         CITY-POP    STATE           STATE-POP
. ABERDEEN      26,467     SOUTH DAKOTA       666,257
. ABILENE       89,653     TEXAS           11,196,73Ø
. ALAMEDA       7Ø,968     CALIFORNIA      19,953,134
. ALBANY       175,781     NEW YORK         4,589,575
```

c. CITIES as listed at a terminal by UNIQUE

```
1      8   12
-------------------------------------
       Ø1  WORK-AREA.
           02 CITY-RECORD.
              Ø3 CITY       PIC X (21).
              Ø3 CITY-POP   PIC 9 (7).
              Ø3 STATE      PIC X (14).
              Ø3 STATE-POP  PIC 9 (8).
           Ø2 S-CITY-RECORD.
              Ø3 S-CITY       PIC X.
              Ø3 S-CITY-POP   PIC X.
              Ø3 S-STATE      PIC X.
              Ø3 S-STATE-POP  PIC X.
```

d. Description of CITY-RECORD in COBOL action program

```
1        10     16
-----------------------------
WORK     DSECT       WORK AREA
CTYREC   EQU    *
SCTYNM   DS     XL21
SCTYPOP  DS     XL7
SSTATE   DS     XL14
SSTPOP   DS     XL8
SCTYNM#S DS     X
SCPOP#S  DS     X
SSTATE#S DS     X
SSTPOP#S DS     X
```

e. Description of CITY-RECORD (labeled CTY-REC) in BAL action program

Figure 2—27. Defined File CITIES (Part 2 of 2)

2.4.4. Examples of Hierarchical Records in Defined Files

Figures 2-28 through 2-32 show portions of defined files and the data definitions needed to define:

- a logical indexed file containing two record types;
- a logical indexed file containing a repeating group item; and
- two logical indexed files.

Note that three alternative data definitions describe the same defined file taken from these various sources. The sources themselves differ in content and organization. Nevertheless, in these examples there is no difference in the resulting defined file that is delivered to the action program regardless of the defined file's source, nor is there any difference in the way the defined file appears at the terminal when accessed via UNIQUE. Actually, the logical files can be reorganized and the defined file redefined without any change to action programs or terminal operating procedures.

2.4.4.1. Hierarchical Defined Records Using Several Record Types as Source

Figure 2-28 illustrates the first few logical records in the indexed file, ST-CITY, from which the defined records of the BIGCITY defined file are taken. Figure 2-29 provides the data definition for the defined file BIGCITY, which contains records arranged in a hierarchical structure within set occurrences.

The order of records in the BIGCITY defined file is identical to the order of their primary parts in the logical file ST-CITY. In fact, the order of records in a defined file is derived from the corresponding record sequence in the logical file. In this case, the source of each parent record is followed directly by the logical source of its child records. This is just one of several ways the sources of parent- and child-type defined records can be related in logical files.

The resulting records defined in this data definition are delivered to the user action program and appear at the terminal via UNIQUE, as shown in Figures 2-33, 2-34, and 2-35.

2.4.4.2. Hierarchical Defined Records Using Repeating Group Item as Source

The logical indexed file, ST-RG, shown in Figure 2-30, contains the same information as the ST-CITY logical file (Figure 2-28), but is organized differently. Here the city information is contained in a table within each state record. The data definition in Figure 2-31 shows how the BIGCITY defined file would be described if its source were the logical file ST-RG. The resulting records defined in this data definition are delivered to the user action program and appear at the terminal via UNIQUE as shown in Figures 2-33, 2-34, and 2-35.

```
ALABAMA△△△△△△△MONTGOMERY△△△△
ALABAMA△△△△△△△BIRMINGHAM△△△△△△△△△△△△△△△Ø325ØØØ△△△△△△△
ALABAMA△△△△△△△HUNTSVILE△△△△△△△△△△△△△△△△Ø143ØØØ△△△△△△△
ALABAMA△△△△△△△MOBILE△△△△△△△△△△△△△△△△△△△Ø215ØØØ△△△△△△△
ALABAMA△△△△△△△MONTGOMERY△△△△△△△△△△△△△△△Ø152ØØØ△△△△△△△
ALASKA△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△JUNEAU△△△△△△△△
ALASKA△△△△△△△△ANCHORAGE△△△△△△△△△△△△△△△△ØØ52ØØØ△△△△△△△
ALASKA△△△△△△△△FAIRBANKS△△△△△△△△△△△△△△△△ØØØ19ØØ△△△△△△△
ALASKA△△△△△△△△JUNEAU△△△△△△△△△△△△△△△△△△△ØØØ68ØØ△△△△△△△
ARIZONA△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△PHOENIX△△△△△△△
ARIZONA△△△△△△△PHOENIX△△△△△△△△△△△△△△△△△△Ø515ØØØ△△△△△△△
ARIZONA△△△△△△△TUCSON△△△△△△△△△△△△△△△△△△△Ø24ØØØØ△△△△△△△
ARKANSAS△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△LITTLE△ROCK△△△
```

Figure 2—28. ST-CITY Indexed File with Two Record Types

```
1      8   12
-----------------------------------------------------
       IDENTIFICATION DIVISION.
       PROGRAM-ID. BIG-CITY-DEF.
       DATA DIVISION.
       FILE SECTION.
       FD  ST-CITY.
       01  STATE-REC.
           02 STATE             PIC X(14).
           02 FILLER            PIC X(25).
           02 CAPITAL           PIC X(14).
       01  CITY-REC.
           02 STATE             PIC X(14).
           02 CITY              PIC X(25).
           02 POPULATION        PIC 9(7).
           02 FILLER            PIC X(7).
       DEFINITION DIVISION.
       DEFINED FILE BIGCITY
       DEFINED RECORD STATE-RECORD
           FROM STATE-REC
           IDENTIFIER STATE
           ITEM CAPITAL
       DEFINED RECORD CITY-RECORD
           FROM CITY-REC
           PARENT IS STATE-RECORD
           IDENTIFIER CITY
           ITEM POPULATION.
```

Figure 2—29. Data Definition for the Defined File BIGCITY

```
ALABAMA△△△△△△△MONTGOMERY△△△△ØØØ4
BIRMINGHAM△△△△△△△△△△△△△△△Ø325ØØØ
MOBILE△△△△△△△△△△△△△△△△△△△Ø215ØØØ
HUNTSVILLE△△△△△△△△△△△△△△△Ø143ØØØ
MONTGOMERY△△△△△△△△△△△△△△△Ø152ØØØ

ALASKA△△△△△△△△JUNEAU△△△△△△△△ØØØ3
ANCHORAGE△△△△△△△△△△△△△△△△ØØ52ØØØ
FAIRBANKS△△△△△△△△△△△△△△△△ØØ19ØØØ
JUNEAU△△△△△△△△△△△△△△△△△△△ØØØ68ØØ

ARIZONA△△△△△△△PHOENIX△△△△△△△ØØØ2
PHOENIX△△△△△△△△△△△△△△△△△△Ø515ØØØ
TUCSON△△△△△△△△△△△△△△△△△△△Ø24ØØØØ

ARKANSAS△△△△△△LITTLE△ROCK△△△ØØØ1
LITTLE△ROCK△△△△△△△△△△△△△△Ø135ØØØ
```

Figure 2—30. STATE-RG Indexed File Containing a Repeating Group Item

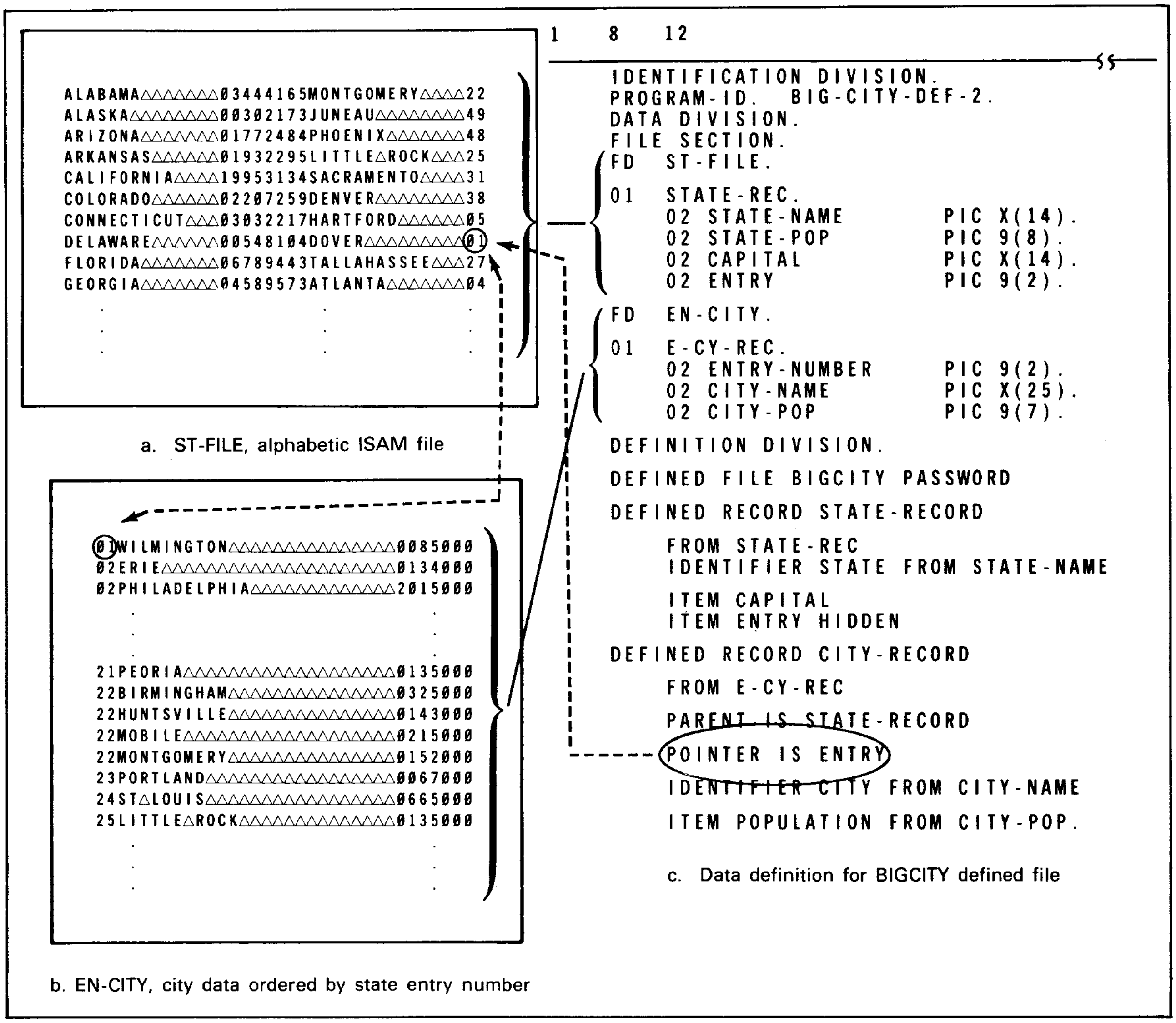
```
1      8   12
-------------------------------------------------------
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  BIG-CITY-DEF-1.
       DATA DIVISION.
       FILE SECTION.
       FD  ST-RG.
       01  STATE-REC.
           02 STATE               PIC X(14).
           02 CAPITAL             PIC X(14).
           02 COUNT               PIC 9(4).
           02 CITY-ENTRY;
              OCCURS 0 TO 5 TIMES
              DEPENDING ON COUNT
              ASCENDING KEY IS CITY.
              03 CITY             PIC X(25).
              03 POPULATION       PIC 9(7)..
       DEFINITION DIVISION.
       DEFINED FILE BIGCITY
       DEFINED RECORD STATE-RECORD
           FROM STATE-REC
           IDENTIFIER STATE
           ITEM CAPITAL
       DEFINED RECORD CITY-RECORD
           FROM REPEATING GROUP CITY-ENTRY
           PARENT IS STATE-RECORD
           IDENTIFIER CITY
           ITEM POPULATION.
```

Figure 2—31. BIGCITY Data Definition Derived from a Repeating Group Item

2.4.4.3. Hierarchical Defined Records Using Two ISAM Files as Source

A third data definition of the BIGCITY defined file is given in Figure 2-32c. In this example, the city and state records come from sources that are in two different ISAM files; the state records come from the logical file ST-FILE, and city records come from the logical file EN-CITY. Each state record contributes a pointer that IMS 90 uses to locate the set of city records that are its child-type records in the hierarchy. Figure 2-32 shows the relationship of the two ISAM files to each other and to the defined file.

Figures 2-33, 2-34, and 2-35 show the resulting records defined in this data definition as they are delivered to the user action program and as they appear at the terminal via UNIQUE.



```
ALABAMA△△△△△△△Ø3444165MONTGOMERY△△△△22
ALASKA△△△△△△△△ØØ3Ø2173JUNEAU△△△△△△△△49
ARIZONA△△△△△△△Ø1772484PHOENIX△△△△△△△48
ARKANSAS△△△△△△Ø1932295LITTLE△ROCK△△△25
CALIFORNIA△△△△19953134SACRAMENTO△△△△31
COLORADO△△△△△△Ø22Ø7259DENVER△△△△△△△△38
CONNECTICUT△△△Ø3Ø32217HARTFORD△△△△△△Ø5
DELAWARE△△△△△△ØØ5481Ø4DOVER△△△△△△△△△Ø1
FLORIDA△△△△△△△Ø6789443TALLAHASSEE△△△27
GEORGIA△△△△△△△Ø4589573ATLANTA△△△△△△△Ø4
   .                     .            .
   .                     .            .
   .                     .            .
```

a. ST-FILE, alphabetic ISAM file

```
Ø1WILMINGTON△△△△△△△△△△△△△△△ØØ85ØØØ
Ø2ERIE△△△△△△△△△△△△△△△△△△△△△Ø134ØØØ
Ø2PHILADELPHIA△△△△△△△△△△△△△2Ø15ØØØ
   .                              .
   .                              .
   .                              .
21PEORIA△△△△△△△△△△△△△△△△△△△Ø135ØØØ
22BIRMINGHAM△△△△△△△△△△△△△△△Ø325ØØØ
22HUNTSVILLE△△△△△△△△△△△△△△△Ø143ØØØ
22MOBILE△△△△△△△△△△△△△△△△△△△Ø215ØØØ
22MONTGOMERY△△△△△△△△△△△△△△△Ø152ØØØ
23PORTLAND△△△△△△△△△△△△△△△△△ØØ67ØØØ
24ST△LOUIS△△△△△△△△△△△△△△△△△Ø665ØØØ
25LITTLE△ROCK△△△△△△△△△△△△△△Ø135ØØØ
   .                              .
   .                              .
   .                              .
```

b. EN-CITY, city data ordered by state entry number

```
1      8   12
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  BIG-CITY-DEF-2.
       DATA DIVISION.
       FILE SECTION.
       FD  ST-FILE.
       01  STATE-REC.
           02 STATE-NAME        PIC X(14).
           02 STATE-POP         PIC 9(8).
           02 CAPITAL           PIC X(14).
           02 ENTRY             PIC 9(2).
       FD  EN-CITY.
       01  E-CY-REC.
           02 ENTRY-NUMBER      PIC 9(2).
           02 CITY-NAME         PIC X(25).
           02 CITY-POP          PIC 9(7).
       DEFINITION DIVISION.
       DEFINED FILE BIGCITY PASSWORD
       DEFINED RECORD STATE-RECORD
           FROM STATE-REC
           IDENTIFIER STATE FROM STATE-NAME
           ITEM CAPITAL
           ITEM ENTRY HIDDEN
       DEFINED RECORD CITY-RECORD
           FROM E-CY-REC
           PARENT IS STATE-RECORD
           POINTER IS ENTRY
           IDENTIFIER CITY FROM CITY-NAME
           ITEM POPULATION FROM CITY-POP.
```

c. Data definition for BIGCITY defined file

Figure 2—32. Derivation of BIGCITY Defined File from Two Distinct Files Using Pointers

2.4.4.4. Defined File Resulting from Different Logical File Sources

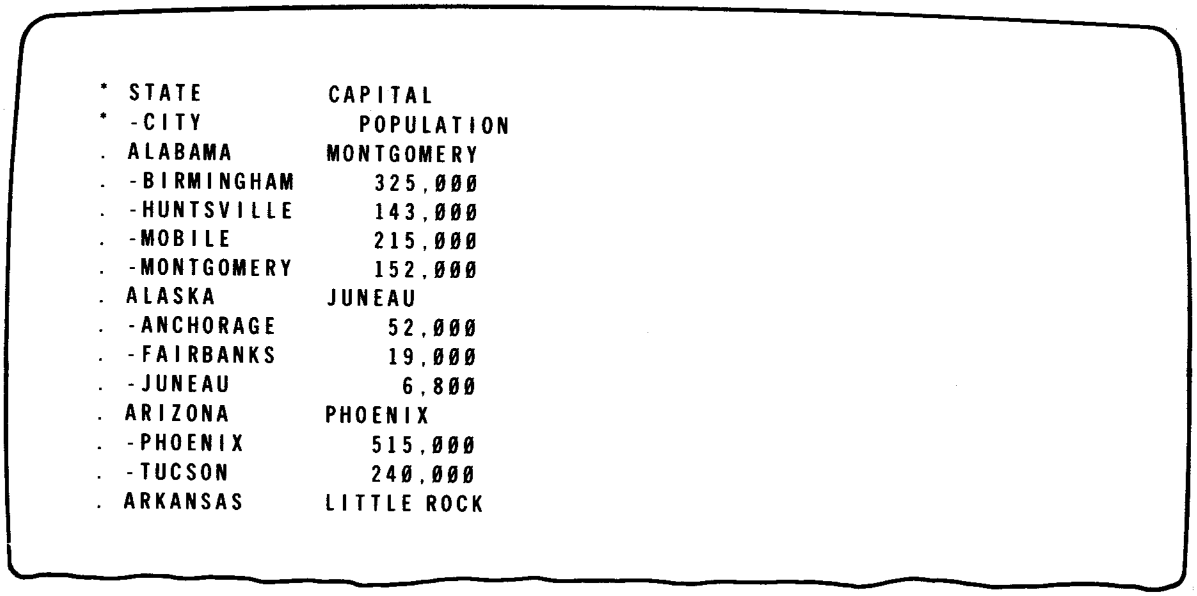
Figure 2-33 illustrates the first defined records of the BIGCITY defined file that IMS 90 delivers in main storage to the user action program in response to a series of GET function calls. When each GET function call is issued, all fields of a defined record plus one status byte per field are moved to the action program.

```
ALABAMA△△△△△△△MONTGOMERY△△△△
ALABAMA△△△△△△△BIRMINGHAM△△△△△△△△△△△△△△△Ø325ØØØ
ALABAMA△△△△△△△HUNTSVILLE△△△△△△△△△△△△△△△Ø143ØØØ
ALABAMA△△△△△△△MOBILE△△△△△△△△△△△△△△△△△△△Ø215ØØØ
ALABAMA△△△△△△△MONTGOMERY△△△△△△△△△△△△△△△Ø152ØØØ
ALASKA△△△△△△△△JUNEAU△△△△△△△△
ALASKA△△△△△△△△ANCHORAGE△△△△△△△△△△△△△△△△ØØ52ØØØ
ALASKA△△△△△△△△FAIRBANKS△△△△△△△△△△△△△△△△ØØ19ØØØ
ALASKA△△△△△△△△JUNEAU△△△△△△△△△△△△△△△△△△△ØØØ68ØØ
ARIZONA△△△△△△△PHOENIX△△△△△△△
ARIZONA△△△△△△△PHOENIX△△△△△△△△△△△△△△△△△△Ø515ØØØ
ARIZONA△△△△△△△TUCSON△△△△△△△△△△△△△△△△△△△024ØØØØ
ARKANSAS△△△△△△LITTLE△ROCK△△△
```

Figure 2—33. Defined Records from the BIGCITY File as Delivered to Action Programs

Figure 2-34 is the terminal display of the BIGCITY file in response to a UNIQUE LIST command. Note that a leading asterisk indicates a line of item names that serve as column headers. A period indicates a line of item values that comprise an occurrence of a defined record. Because the identifier of each city record consists of both state and city names, UNIQUE replaces the state name with a hyphen to conserve screen space.

Figure 2-35 and 2-36 illustrate the corresponding description of the receiving space that accommodates the parent and child defined records in the user COBOL and BAL action programs.



```
* STATE          CAPITAL
* -CITY            POPULATION
. ALABAMA        MONTGOMERY
. -BIRMINGHAM      325,ØØØ
. -HUNTSVILLE      143,ØØØ
. -MOBILE          215,ØØØ
. -MONTGOMERY      152,ØØØ
. ALASKA         JUNEAU
. -ANCHORAGE        52,ØØØ
. -FAIRBANKS        19,ØØØ
. -JUNEAU            6,8ØØ
. ARIZONA        PHOENIX
. -PHOENIX         515,ØØØ
. -TUCSON          24Ø,ØØØ
. ARKANSAS       LITTLE ROCK
```

Figure 2—34. Defined Records from the BIGCITY File as Listed at the Terminal by UNIQUE

```
1      8   12
------------------------------------------
       01  WORK-AREA.
           Ø2 STATE-RECORD.
              Ø3 STATE       PIC X (14).
              Ø3 CAPITAL     PIC X (14).
           Ø2 S-STATE-RECORD.
              Ø3 S-STATE     PIC X.
              Ø3 S-CAPITAL   PIC X.
           Ø2 CITY-RECORD.
              Ø3 STATE       PIC X (14).
              Ø3 CITY        PIC X (25).
              Ø3 POPULATION  PIC X (7).
           Ø2 S-CITY-RECORD.
              Ø3 S-STATE       PIC X.
              Ø3 S-CITY        PIC X.
              Ø3 S-POPULATION  PIC X.
```

Figure 2—35. Description of STATE-RECORD and CITY-RECORD in COBOL Action Program

```
1        10    16
WORK     DSECT       WORK AREA
STREC    EQU   *
SSTATE   DS    XL14
SCAPIT   DS    XL14
SSTATE#S DS    X
SCAPIT#S DS    X
CITY#REC EQU   *
SCSTATE  DS    XL14
SCCITY   DS    XL25
SCPOP    DS    XL7
SCSTAT#S DS    X
SCCITY#S DS    X
SCPOP#S  DS    X
```

Figure 2—36. Description of STATE-RECORD and CITY-RECORD in BAL Action Program

2.5. EXECUTING DATA DEFINITION PROCESSOR

After the data definition is prepared, it must be submitted to the data definition processor (data definition processor), whose module name in OS/3 is DT3DF. The data definition processor writes a data definition record into the named record file (NAMEREC) and produces a diagnostic listing. Multiple defined files must be created in separate runs of the data definition processor but can be written to the same NAMEREC file. The data definition processor cannot write to the NAMEREC file while IMS 90 is accessing NAMEREC. The NAMEREC file must be initialized before the data definition processor is executed for the first time. Initialization procedures are described in the IMS 90 system support functions user guide/programmer reference, UP-8364 (current version). Note that if the NAMEREC file is reinitialized at any time, all data definitions must be recompiled.

2.5.1. Data Definition Processor Options

You can present parameters to the data definition processor via the optional PARAM statement. The format is:

```
// PARAM parameter-1 [,...,parameter-n]
```

PARAM statements should be placed immediately following the EXEC job control statement (// EXEC DT3DF) in the execution job control stream. The data definition processor prints these on the first page of the diagnostic listing. If a PARAM statement format error or an illegal parameter is encountered, a system console message is produced and the data definition run is terminated. Only one blank precedes the P of the word PARAM.

To produce a single-spaced diagnostic and source listing, you must specify the following PARAM statement:

```
// PARAM LST=(L,S)
```

The format for the source library PARAM job control statement is:

```
// PARAM IN=program-name/file-name
```

where:

program-name
: Is a 1- to 8-character name of your source data definition program.

file-name
: Is a 1- to 8-character name used to identify the file on which your source data definition program resides. This name must appear on the LFD job control statement you used to define this device. If the *file-name* is omitted, the name $Y$SRC is automatically supplied.

The format of the PARAM statement for copy library input is:

```
// PARAM LIN=file-name
```

where:

file-name
: Is a 1- to 8-character name used to identify the file on which your COPY library resides. This name must appear on the LFD job control statement you used to define the device. If the *file-name* is omitted, the name COPY$ is automatically supplied. You supply the COPY *element-name* in your source data definition program via the COPY clause.

There are no output options available for the data definition processor. (// PARAM OUT=(M) is specified only for a COBOL action program, never for the data definition processor.)

2.5.2. Execution Run Streams

To execute the data definition processor, having previously allocated the NAMEREC file using the ZP#NRU utility or the configurator, you code and execute a job such as DATADF. (See sample control stream in Figure 2-37.) The main storage requirement for the data definition processor is 50K bytes.

```
// JOB DATADF,,CØØØ
// DVC 2Ø // LFD PRNTR
// OPTION DUMP
// DVC 5Ø // VOL DS9999 // LBL NAMEREC,DS9999 // LFD ISAMNRF
// WORK1
// WORK2
// WORK3
// EXEC DT3DF
/$
     source cards
       .     .
       .     .
       .     .
     source cards
/*
/&
// FIN
```

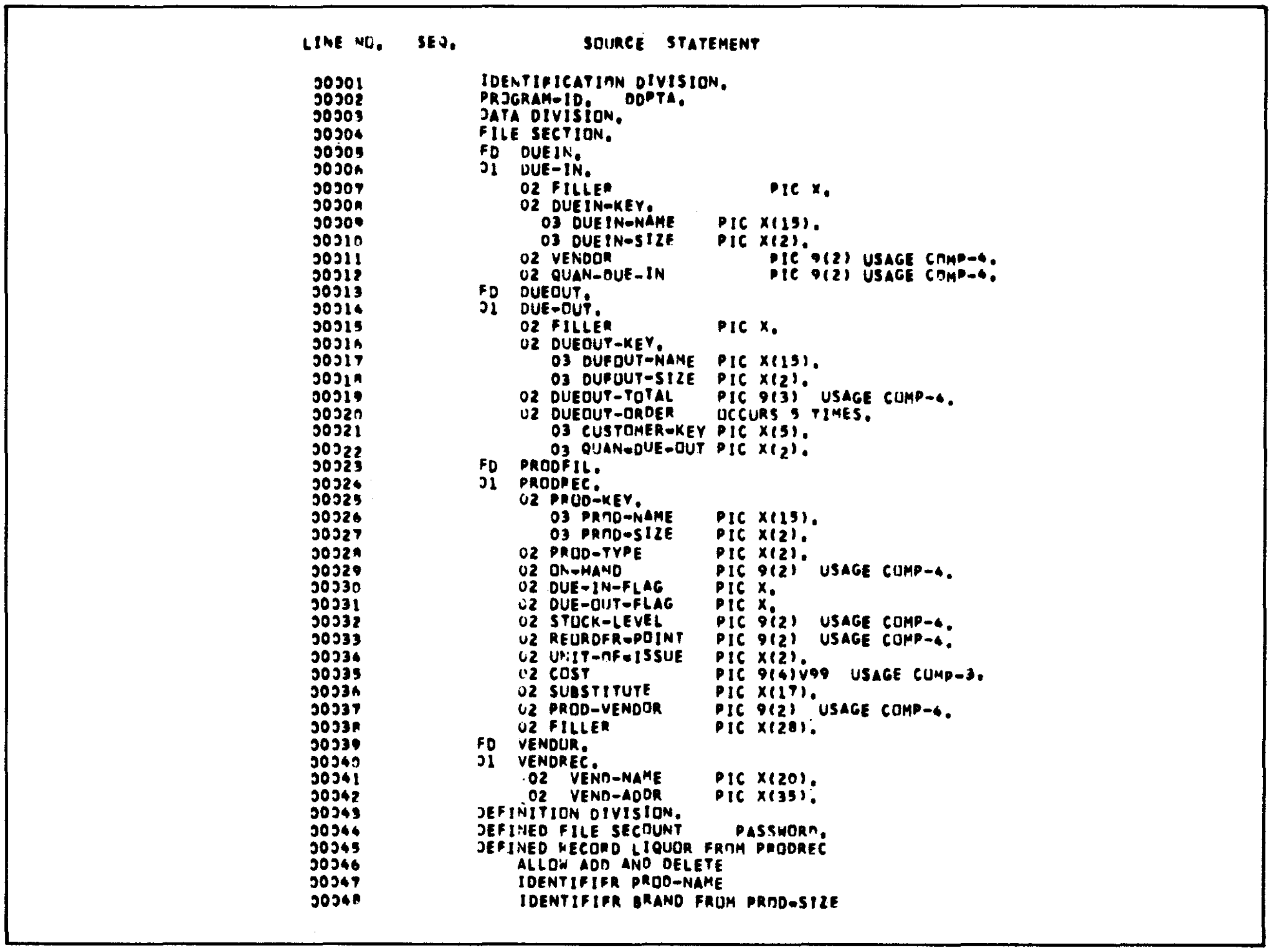
Figure 2—37. Executing the Data Definition Processor (DT3DF)

In addition to the job control and PARAM statements already discussed, the most important part of your input to the data definition processor is your source statements (2.3). Figure 2-16 provides a consolidated format of the defined file definition which can be used when studying the sample diagnostic listing produced by the OS/3 data definition processor (see Figures 2-38 and 2-39).

2.5.3. Data Definition Processor Output Listing

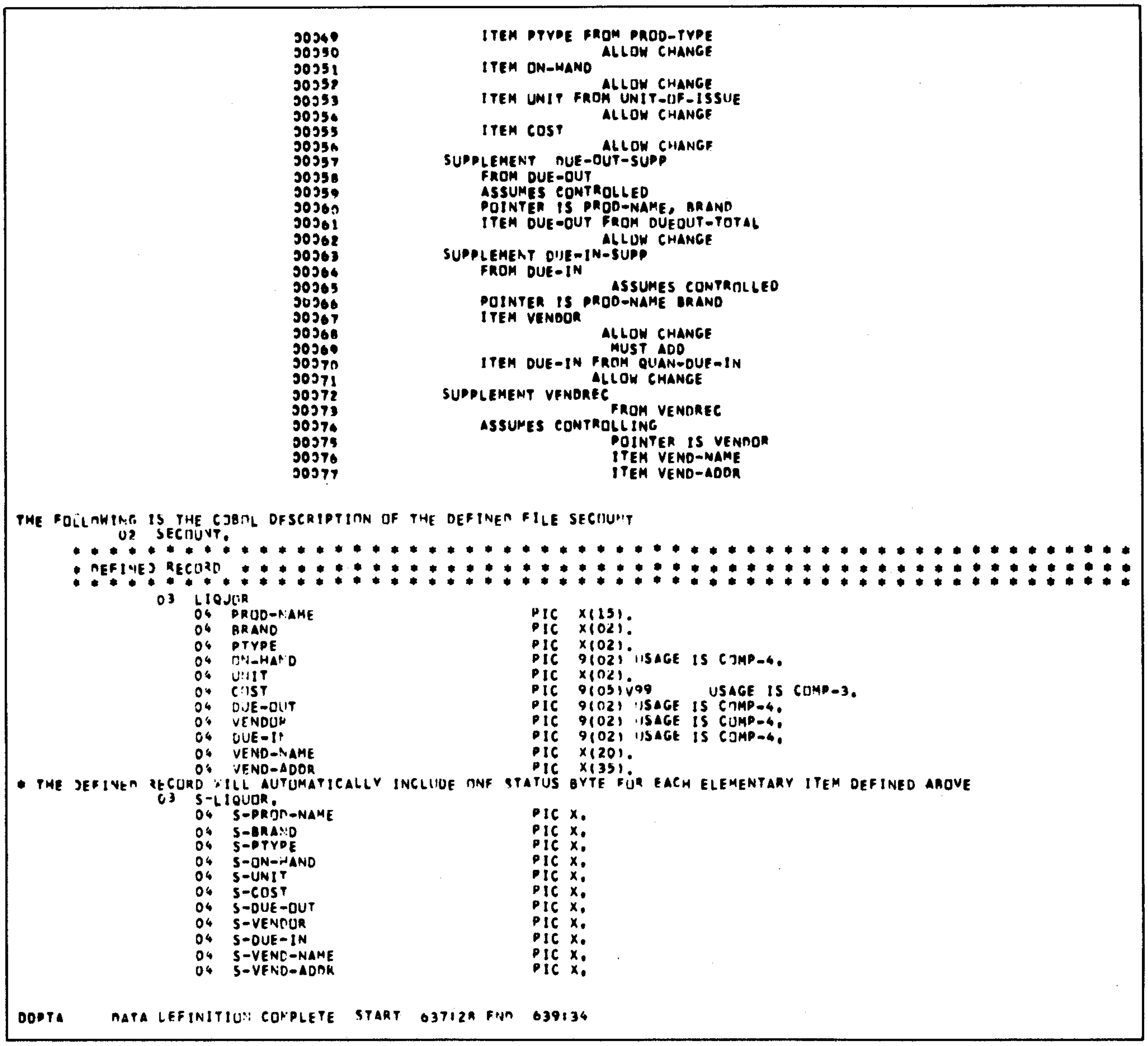
The printed output provided by the data definition processor comprises a source listing of the input to the data definition processor and, when the processor has successfully created a data definition record, a COBOL description of the defined file. Each defined record and subrecord is described as a COBOL group item such as required in a COBOL-written action program accessing the file. Included with each defined record description, the processor listing describes the item status bytes, one for each elementary item defined in the data definition record above it. The processor generates each item status byte data name by prefixing the data name of the corresponding elementary item with 'S'-. This provides a data name for accessing each item's status byte if a test is made for the completeness and validity of data transfer after retrieving a record from the defined file in the action program.

Figure 2-38 is a listing compiled by the data definition processor for the defined file SECOUNT. Part 1 lists the source input. Part 2 shows the COBOL description of the defined file, the description of the defined record LIQUOR, and the description of the item status bytes. The last line of output contains the statement DATA DEFINITION COMPLETE, followed by compilation time figures. When a subfile definition is input to the data definition processor, the last pages of the output listing have the format shown in Figure 2-39, in which a defined file and subfile (ZR and CH-ZR) are described.



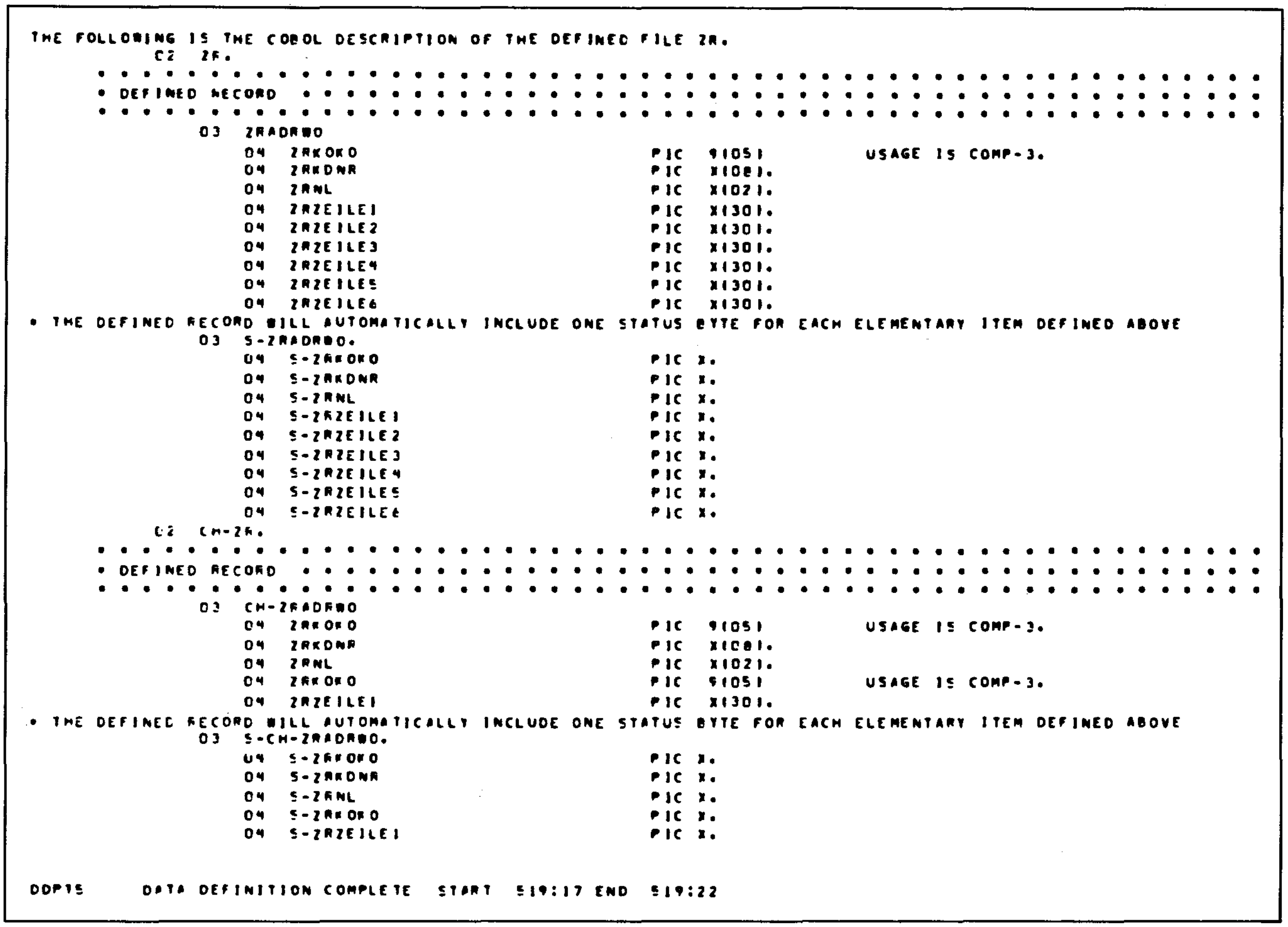
```
LINE NO.   SEQ.            SOURCE  STATEMENT

00301              IDENTIFICATION DIVISION.
00302              PROGRAM-ID.   DDPTA.
00303              DATA DIVISION.
00304              FILE SECTION.
00305              FD  DUEIN.
00306              01  DUE-IN.
00307                  02 FILLER                 PIC X.
00308                  02 DUEIN-KEY.
00309                     03 DUEIN-NAME     PIC X(15).
00310                     03 DUEIN-SIZE     PIC X(2).
00311                  02 VENDOR                 PIC 9(2) USAGE COMP-4.
00312                  02 QUAN-DUE-IN            PIC 9(2) USAGE COMP-4.
00313              FD  DUEOUT.
00314              01  DUE-OUT.
00315                  02 FILLER            PIC X.
00316                  02 DUEOUT-KEY.
00317                     03 DUFOUT-NAME    PIC X(15).
00318                     03 DUFOUT-SIZE    PIC X(2).
00319                  02 DUEOUT-TOTAL      PIC 9(3)  USAGE COMP-4.
00320                  02 DUEOUT-ORDER      OCCURS 5 TIMES.
00321                     03 CUSTOMER-KEY   PIC X(5).
00322                     03 QUAN-DUE-OUT   PIC X(2).
00323              FD  PRODFIL.
00324              01  PRODREC.
00325                  02 PROD-KEY.
00326                     03 PROD-NAME      PIC X(15).
00327                     03 PROD-SIZE      PIC X(2).
00328                  02 PROD-TYPE         PIC X(2).
00329                  02 ON-HAND           PIC 9(2)  USAGE COMP-4.
00330                  02 DUE-IN-FLAG       PIC X.
00331                  02 DUE-OUT-FLAG      PIC X.
00332                  02 STOCK-LEVEL       PIC 9(2)  USAGE COMP-4.
00333                  02 REORDER-POINT     PIC 9(2)  USAGE COMP-4.
00334                  02 UNIT-OF-ISSUE     PIC X(2).
00335                  02 COST              PIC 9(4)V99  USAGE COMP-3.
00336                  02 SUBSTITUTE        PIC X(17).
00337                  02 PROD-VENDOR       PIC 9(2)  USAGE COMP-4.
00338                  02 FILLER            PIC X(28).
00339              FD  VENDOR.
00340              01  VENDREC.
00341                  02  VEND-NAME        PIC X(20).
00342                  02  VEND-ADDR        PIC X(35).
00343              DEFINITION DIVISION.
00344              DEFINED FILE SECOUNT     PASSWORD.
00345              DEFINED RECORD LIQUOR FROM PRODREC
00346                  ALLOW ADD AND DELETE
00347                  IDENTIFIER PROD-NAME
00348                  IDENTIFIER BRAND FROM PROD-SIZE
```

Figure 2—38. Complete Data Definition Processor Output Listing (Part 1 of 2)
]



```
                         00349                 ITEM PTYPE FROM PROD-TYPE
                         00350                             ALLOW CHANGE
                         00351                 ITEM ON-HAND
                         00352                             ALLOW CHANGE
                         00353                 ITEM UNIT FROM UNIT-OF-ISSUE
                         00354                             ALLOW CHANGE
                         00355                 ITEM COST
                         00356                             ALLOW CHANGE
                         00357             SUPPLEMENT  DUE-OUT-SUPP
                         00358                 FROM DUE-OUT
                         00359                 ASSUMES CONTROLLED
                         00360                 POINTER IS PROD-NAME, BRAND
                         00361                 ITEM DUE-OUT FROM DUEOUT-TOTAL
                         00362                             ALLOW CHANGE
                         00363             SUPPLEMENT DUE-IN-SUPP
                         00364                 FROM DUE-IN
                         00365                              ASSUMES CONTROLLED
                         00366                 POINTER IS PROD-NAME BRAND
                         00367                 ITEM VENDOR
                         00368                             ALLOW CHANGE
                         00369                              MUST ADD
                         00370                 ITEM DUE-IN FROM QUAN-DUE-IN
                         00371                            ALLOW CHANGE
                         00372             SUPPLEMENT VENDREC
                         00373                              FROM VENDREC
                         00374                 ASSUMES CONTROLLING
                         00375                              POINTER IS VENDOR
                         00376                              ITEM VEND-NAME
                         00377                              ITEM VEND-ADDR

THE FOLLOWING IS THE COBOL DESCRIPTION OF THE DEFINED FILE SECOUNT
       02  SECOUNT,
     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     * DEFINED RECORD  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
               03  LIQUOR
                   04  PROD-NAME                    PIC  X(15).
                   04  BRAND                        PIC  X(02).
                   04  PTYPE                        PIC  X(02).
                   04  ON-HAND                      PIC  9(02) USAGE IS COMP-4.
                   04  UNIT                         PIC  X(02).
                   04  COST                         PIC  9(05)V99      USAGE IS COMP-3.
                   04  DUE-OUT                      PIC  9(02) USAGE IS COMP-4.
                   04  VENDOR                       PIC  9(02) USAGE IS COMP-4.
                   04  DUE-IN                       PIC  9(02) USAGE IS COMP-4.
                   04  VEND-NAME                    PIC  X(20).
                   04  VEND-ADDR                    PIC  X(35).
* THE DEFINED RECORD WILL AUTOMATICALLY INCLUDE ONE STATUS BYTE FOR EACH ELEMENTARY ITEM DEFINED ABOVE
               03  S-LIQUOR,
                   04  S-PROD-NAME                  PIC X.
                   04  S-BRAND                      PIC X.
                   04  S-PTYPE                      PIC X.
                   04  S-ON-HAND                    PIC X.
                   04  S-UNIT                       PIC X.
                   04  S-COST                       PIC X.
                   04  S-DUE-OUT                    PIC X.
                   04  S-VENDOR                     PIC X.
                   04  S-DUE-IN                     PIC X.
                   04  S-VEND-NAME                  PIC X.
                   04  S-VEND-ADDR                  PIC X.


DDPTA      DATA DEFINITION COMPLETE  START  637128 END  639134
```

Figure 2—38. Complete Data Definition Processor Output Listing (Part 2 of 2)



```
THE FOLLOWING IS THE COBOL DESCRIPTION OF THE DEFINED FILE ZR.
        02  ZR.
      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      * DEFINED RECORD  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
            03  ZRADRWO
                04  ZRKOKO                          PIC  9(05)       USAGE IS COMP-3.
                04  ZRKDNR                          PIC  X(08).
                04  ZRNL                            PIC  X(02).
                04  ZRZEILE1                        PIC  X(30).
                04  ZRZEILE2                        PIC  X(30).
                04  ZRZEILE3                        PIC  X(30).
                04  ZRZEILE4                        PIC  X(30).
                04  ZRZEILE5                        PIC  X(30).
                04  ZRZEILE6                        PIC  X(30).
* THE DEFINED RECORD WILL AUTOMATICALLY INCLUDE ONE STATUS BYTE FOR EACH ELEMENTARY ITEM DEFINED ABOVE
            03  S-ZRADRWO.
                04  S-ZRKOKO                        PIC X.
                04  S-ZRKDNR                        PIC X.
                04  S-ZRNL                          PIC X.
                04  S-ZRZEILE1                      PIC X.
                04  S-ZRZEILE2                      PIC X.
                04  S-ZRZEILE3                      PIC X.
                04  S-ZRZEILE4                      PIC X.
                04  S-ZRZEILE5                      PIC X.
                04  S-ZRZEILE6                      PIC X.
        02  CH-ZR.
      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      * DEFINED RECORD  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
            03  CH-ZRADRWO
                04  ZRKOKO                          PIC  9(05)       USAGE IS COMP-3.
                04  ZRKDNR                          PIC  X(08).
                04  ZRNL                            PIC  X(02).
                04  ZRKOKO                          PIC  9(05)       USAGE IS COMP-3.
                04  ZRZEILE1                        PIC  X(30).
* THE DEFINED RECORD WILL AUTOMATICALLY INCLUDE ONE STATUS BYTE FOR EACH ELEMENTARY ITEM DEFINED ABOVE
            03  S-CH-ZRADRWO.
                04  S-ZRKOKO                        PIC X.
                04  S-ZRKDNR                        PIC X.
                04  S-ZRNL                          PIC X.
                04  S-ZRKOKO                        PIC X.
                04  S-ZRZEILE1                      PIC X.

DDP15     DATA DEFINITION COMPLETE  START  519:17 END  519:22
```

Figure 2—39. Last Page of Data Definition Processor Listing Showing COBOL Description of a Defined File and a Subfile

2.5.4. Error Processing by Data Definition Processor

In processing your input, the data definition processor acts like a COBOL compiler; it subjects the entire input to scrutiny for syntactical errors and issues diagnostics.

The data definition processor applies the rules of COBOL to the data division of the input and issues COBOL diagnostics for this division. The COBOL reserved word list applies to the data division of the input to the data definition processor as well. When processing the definition division, however, the data definition processor applies not only the appropriate standard COBOL rules, but also rules of its own. (See 2.3.1.) Detected violations of these rules result in the issuance of diagnostics from a set unique to the data definition processor. These diagnostics are listed in Table 2-2.

Each diagnostic message contains the processor-generated line number on which the error occurred, the diagnostic severity code, the diagnostic number, and the diagnostic message text, in that order.

Table 2—2. Compilation Time Diagnostics Unique to the IMS 90 Data Definition Processor (Part 1 of 2)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
139	U	–SUSPEND CHECKING INVALID SOURCE STATEMENTS ON THIS LINE.	Beginning at this source line, the data definition processor does not recognize source input as data definition language.	No validity checking for syntax of data definition source statements occurs until some succeeding statement is recognized.	If preceded by another diagnostic for the same line number, recovery for that diagnostic will usually suffice, but the remainder of this line *might* contain another error. Otherwise, this line contains an error that that is not embedded in a recognized statement type.
140	U	–RESUME CHECKING SOURCE STATEMENTS ON THIS LINE.	Having previously issued diagnostic 139, the data definition processor again recognizes source input as data definition language.	Error processing continues, beginning with this source line.	None required. All lines for which validity checking was skipped should be scanned for *possible* error, before recompiling.
159	U	REFERENCE TO insert INVALID	Self-explanatory	Refer to 2.3 for the rules for each statement.	Correct the data definition and recompile.
160	U	DEFINITION IS TOO LARGE	The length of the data definition record exceeds the blocksize specified for the NAMEREC file.	Block size for the NAMEREC file specified with the NBLK keyword parameter of the IMSCONF jproc or the BLKSZE parameter of the ZP#NRU utility. It ranges between 1024 and 12,800 bytes but most not exceed disk track size.	Reduce size of the data definition record and recompile. The line number indicated is the one which caused the overflow. Alternatively, specify a larger block size for the NAMEREC file and reconfigure.
161	C	CHANGE TO NEUTRAL SUPPLEMENT IS ILLEGAL.	The processor has encountered the ALLOW CHANGE option specified in a supplement for which no ROLE IN UPDATE is specified, or whose update ROLE is specified as NEUTRAL.	If the ALLOW CHANGE option is specified for an item, its ROLE IN UPDATE must be CONTROLLED.	Correct the data definition and recompile. Your action program logic may also require revision.
162	C	CHANGE TO CONTROLLING SUPPLEMENT IS ILLEGAL.	The processor has encountered the ALLOW CHANGE option specified in a supplement whose ROLE IN UPDATE is specified as CONTROLLING.	If the ALLOW CHANGE option is specified for an item, its ROLE IN UPDATE must be CONTROLLED.	Correct the data definition and recompile. Your action program logic may also require revision.

Table 2—2. Compilation Time Diagnostics Unique to the IMS 90 Data Definition Processor (Part 2 of 2)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
163	C	ADD TO NEUTRAL SUPPLEMENT IS ILLEGAL.	The processor has encountered a MUST ADD statement specified for a supplement for which no update role is specified or for which ASSUMES NEUTRAL ROLE IN UPDATE is specified.	If the MUST ADD option is specified for an item, its ROLE IN UPDATE must be CONTROLLED.	Same as 161.
164	C	ADD TO CONTROLLING SUPPLEMENT IS ILLEGAL.	The processor has encounterd a MUST ADD statement specified in a supplement for which ASSUMES CONTROLLING ROLE IN UPDATE is specified.	Same as 163.	Same as 161.
165	U	CANNOT ADD OR DELETE CONTROL BREAK.	The processor has encountered one of three options of the ALLOW ADD AND DELETE statement specified for a defined record for which a FROM CONTROL BREAK statement is also specified.	The ALLOW ADD AND DELETE statement cannot be specified for a defined record for which FROM CONTROL BREAK is also specified.	Same as 161.
166	U	CANNOT ADD OR DELETE REPEATING GROUP.	The processor has encountered one of the three options of the ALLOW ADD AND DELETE statement specified for a defined record for which a FROM REPEATING GROUP statement has been specified.	The ALLOW ADD AND DELETE statement cannot be specified for a defined record for which FROM REPEATING GROUP is also specified.	Same as 161.
167	U	SEE CONSOLE FOR DMXX	OS/3 ISAM has issued a numbered data management error message to the system console, reflecting an error detected during processing of the NAMEREC file by the data definition processor.	None. The actual OS/3 data management message number, the prefix of which is "DM", will appear at the system console and on the console output printer (COP) sheet for the run.	According to the nature of the error detected or reported to ISAM. Refer to the OS/3 system messages operator/programmer reference, UP-8076 (current version) and to the data management user guide, UP-8068 or programmer reference, UP-8159 (current versions).

Definitions of diagnostic severity are as follows:

- P (precautionary)

  No source language error detected, but an unusual or potentially undesirable condition noted by the data definition processor.

- C (changed)

  A character, word, clause, entry, or statement in the source program is omitted or used incorrectly. To compensate for the error, the item has been changed by the data definition processor to avoid its deletion and reduce the probability of error propagation. A data definition record is not created.

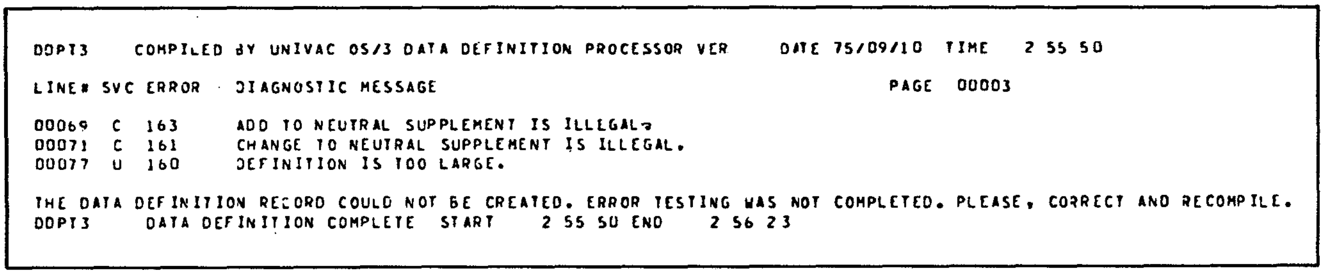
- U (uncorrectable)

  A source language error detected causing the data definition processor to delete a character, word, clause, entry, or statement from the source program. The compilation continues, but other errors may result because of the deleted item. A data definition record is not created.

- S (compiler restriction exceeded)

  The compilation continues but, to generate code for the excessive items, a recompilation is necessary after source program modification or with more storage assigned to the compiler.

Figure 2-40 shows the last page of output from an unsuccessful run of the data definition processor, during which it was not possible to construct the desired data definition record and error testing was not completed. The diagnostic messages listed happen to be limited to the set that is unique to the data definition processor; however, COBOL diagnostics would be listed in the same manner and location.



```
DDPT3    COMPILED BY UNIVAC OS/3 DATA DEFINITION PROCESSOR VER    DATE 75/09/10  TIME   2 55 50

LINE# SVC ERROR   DIAGNOSTIC MESSAGE                                        PAGE  00003

00069  C  163     ADD TO NEUTRAL SUPPLEMENT IS ILLEGAL.
00071  C  161     CHANGE TO NEUTRAL SUPPLEMENT IS ILLEGAL.
00077  U  160     DEFINITION IS TOO LARGE.

THE DATA DEFINITION RECORD COULD NOT BE CREATED. ERROR TESTING WAS NOT COMPLETED. PLEASE, CORRECT AND RECOMPILE.
DDPT3     DATA DEFINITION COMPLETE  START    2 55 50 END    2 56 23
```

Figure 2—40. Last Page of Data Definition Processor Listing from Unsuccessful Run

3. User-Written Action Programs

3.1. DESCRIPTION

Action programs operate under the application management component of IMS 90; they process input messages and generate output messages. Action programs operate as subprograms to the configured IMS 90 online program. Some action programs are provided as a standard part of IMS 90. For example, the uniform inquiry update element (UNIQUE) is a series of action programs.

You can also write your own programs for applications that cannot be implemented with UNIQUE. Some applications require more extensive data validation for update operations than UNIQUE can provide. Other applications require special output formats. These requirements and others are accommodated within the general framework provided for user-written action programs. This section describes the framework within which you write your own action programs.

Action programs can be written in COBOL, RPG II, or BAL. The current versions of the following OS/3 documents can be referenced when preparing these programs:

- extended COBOL supplementary reference, UP-8059
- 1974 American National Standard COBOL programmer reference, UP-8613
- report program generator (RPG II) programmer reference, UP-8044
- assembler programmer reference, UP-8227

If your action programs access a DMS 90 data base, see the current versions of the following manuals:

- IMS 90/DMS 90 interface user guide/programmer reference, UP-8748
- DMS 90 data description language user guide/programmer reference, UP-8022
- DMS 90 data manipulation language user guide/programmer reference, UP-8036

COBOL and BAL action programs also can call user-written resident subprograms.

3.1.1. Action Program Environment

An action program is scheduled for execution either because an input message is received that must be processed by the action program or because one action program has designated another action program as its immediate successor. In either situation, when execution of the action begins, a standard environment exists in which it carries out its processing. The environment is shown in Figure 3-1.

The environment consists of certain main storage areas, each of which has a specialized function, and a set of standard interfaces to IMS 90 for all file input/output, message output, and program termination requests.

Because of the requirement that action programs must be at least serially reusable, no I/O areas or other main storage areas with varying contents can be compiled or assembled into an action program. When application management activates an action to process an input message, the variable main storage areas that are required are dynamically allocated from the IMS 90 main storage pool. Accordingly, when the action scheduling component of application management calls the action program, it passes a list of addresses of the dynamically allocated areas.

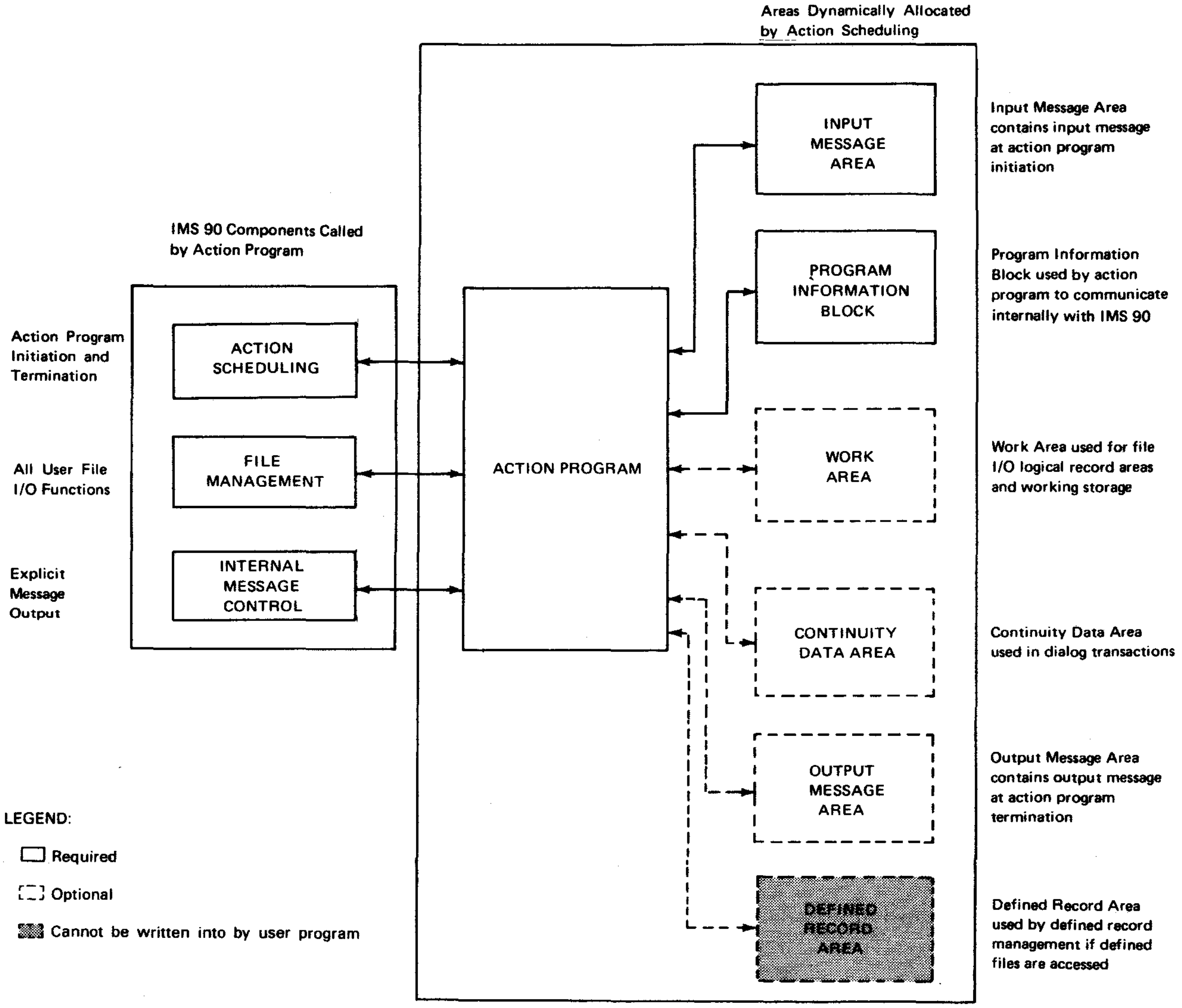

Areas Dynamically Allocated by Action Scheduling
IMS 90 Components Called by Action Program
Action Program Initiation and Termination
All User File I/O Functions
Explicit Message Output
ACTION SCHEDULING
FILE MANAGEMENT
INTERNAL MESSAGE CONTROL
ACTION PROGRAM
INPUT MESSAGE AREA
PROGRAM INFORMATION BLOCK
WORK AREA
CONTINUITY DATA AREA
OUTPUT MESSAGE AREA
DEFINED RECORD AREA
Input Message Area contains input message at action program initiation
Program Information Block used by action program to communicate internally with IMS 90
Work Area used for file I/O logical record areas and working storage
Continuity Data Area used in dialog transactions
Output Message Area contains output message at action program termination
Defined Record Area used by defined record management if defined files are accessed
LEGEND:
□ Required
⬚ Optional
▨ Cannot be written into by user program

Figure 3—1. Action Program Environment

Two main storage areas are always present: the program information block and the input message area. The program information block is used to communicate information between IMS 90 functions and an action program at initiation, after I/O functions, and at termination. The input message area contains the input message that caused the action program to be scheduled.

The work area, output message area, continuity data area, and defined record area are optional. The work area is a general-purpose area used for working storage, logical record areas during file I/O functions, or output messages sent explicitly. The output message area normally is used to build an output message that is sent to the originating terminal when the action terminates. The continuity data area is used to pass data from one action to another. It is saved by action scheduling at the termination of one action and restored upon initiation of the next action in a dialog. The defined record area is used by defined record management if the action program accesses defined records. It cannot be written into by the user program.

The choice of the specialized main storage areas that are to be associated with a given action program, and what their sizes are to be, is made at configuration time and, in some cases, by the immediately preceding action at execution time.

All file I/O, explicit message output, and program termination requests made by an action program must be made through IMS 90 function calls. Transfer of control to a user-written resident subprogram also is made in this manner.

3.1.2. Transaction Structures

A transaction is one action or a sequence of related actions. A simple transaction consists of one action; a dialog transaction consists of two or more related actions.

The structure of a transaction depends on whether that transaction is performed by a single action program, by more than one action program processing a single action, or by several actions dynamically sequenced. Transaction structures differ according to the type of termination specified in the action programs that process the transaction. There are four types of termination:

1. Normal

2. External succession

3. Immediate internal succession

4. Delayed internal succession

The type of action program termination is specified in the termination indicator of the program information block (PIB). (See 3.6.1.)

3.1.2.1. Simple Transaction

The structure of a simple transaction is shown in Figure 3-2. A message is input from a terminal and contains a transaction code as the first one to five characters. The transaction code is used to look up the name of the action program that is to process the input message. For each transaction code, there can be only one corresponding action program. More than one transaction code, however, can designate the same action program. Transaction codes must be defined to IMS 90 at configuration time.

The input messge is placed on a queue for the action program until scheduling occurs. The scheduling process consists of allocating the main storage areas required by the action program from the IMS 90 main storage pool, loading the action program from a disk library file if it is not already resident, moving the input message from the queue into the input message area, and passing control to the action program. Because Figure 3-2 is a simple transaction with a single action program, the use of the continuity data area does not apply.

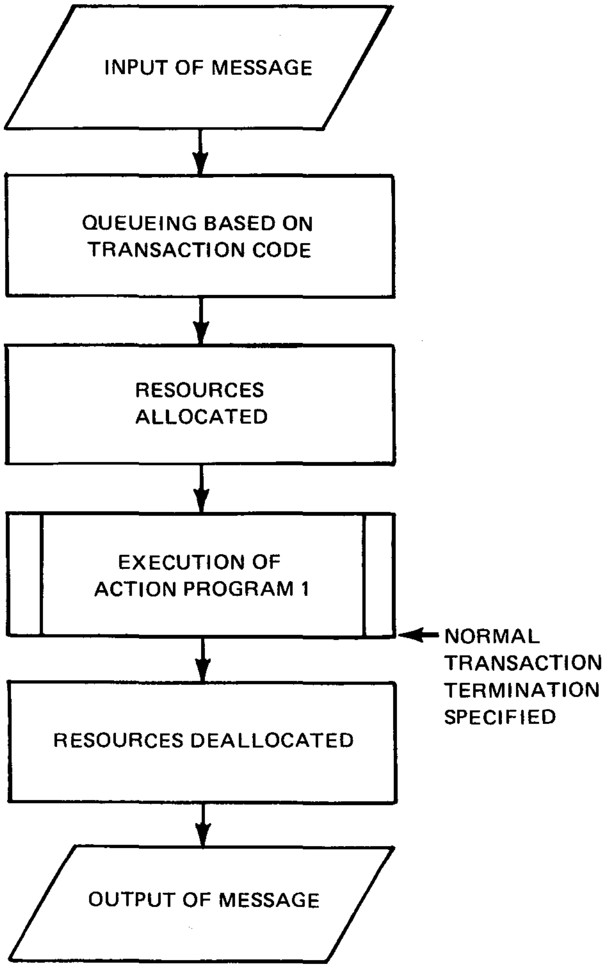

INPUT OF MESSAGE
QUEUEING BASED ON TRANSACTION CODE
RESOURCES ALLOCATED
EXECUTION OF ACTION PROGRAM 1
NORMAL TRANSACTION TERMINATION SPECIFIED
RESOURCES DEALLOCATED
OUTPUT OF MESSAGE

Figure 3—2. Simple Transaction

The action program accesses files by calling on IMS 90 file management to request that the logical or defined records be placed in the work area. The text of the output message is built in the output message area. The action program then requests a normal transaction termination. The type of termination is specified in the PIB.

This sequence of events is called an action within IMS 90. It always begins with the input of a message and ends with the output of a response. The simple transaction in Figure 3-2 consists of a single action, performed by a single action program. (An action also can be performed by more than one action program as shown in Figure 3-4.)

3.1.2.2. External Succession

IMS 90 can establish a relationship between two or more successive actions. A transaction that makes use of this relationship is called a dialog transaction, which can consist of an arbitrary number of actions. The dialog transaction in Figure 3-3 consists of two actions.

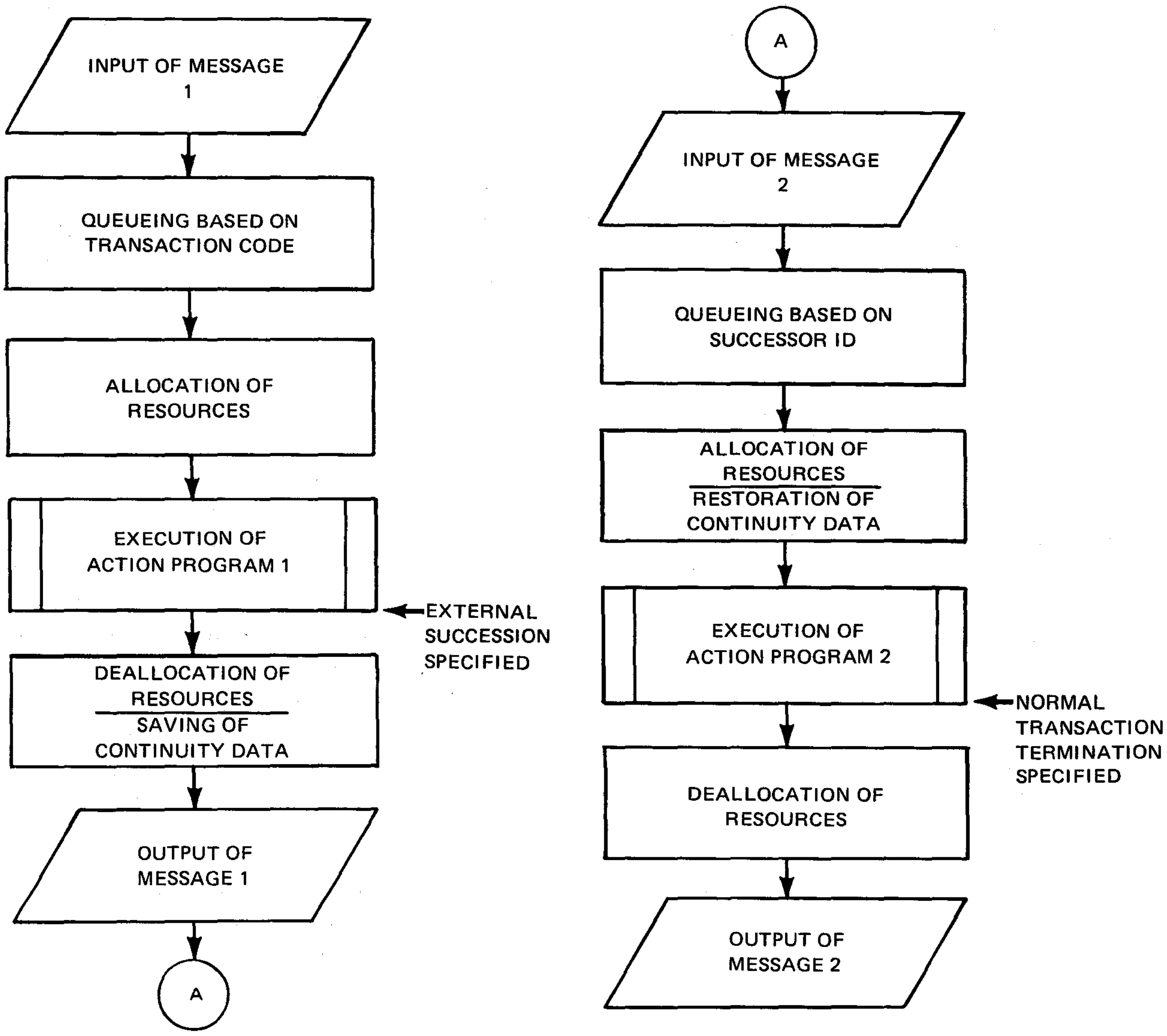

INPUT OF MESSAGE 1
QUEUEING BASED ON TRANSACTION CODE
ALLOCATION OF RESOURCES
EXECUTION OF ACTION PROGRAM 1
EXTERNAL SUCCESSION SPECIFIED
DEALLOCATION OF RESOURCES
SAVING OF CONTINUITY DATA
OUTPUT OF MESSAGE 1
A
A
INPUT OF MESSAGE 2
QUEUEING BASED ON SUCCESSOR ID
ALLOCATION OF RESOURCES
RESTORATION OF CONTINUITY DATA
EXECUTION OF ACTION PROGRAM 2
NORMAL TRANSACTION TERMINATION SPECIFIED
DEALLOCATION OF RESOURCES
OUTPUT OF MESSAGE 2

Figure 3—3. Dialog Transaction, External Succession

The example in Figure 3-3 is the same as Figure 3-2 up to the termination of action program 1. The action program in Figure 3-3 specifies in the PIB the name of another action as its successor and designates the type of termination as external succession. (The name of an action is the same as the name of the first action program in that action.) The deallocation of resources occurs as in Figure 3-2 except for the continuity data area. The contents of the continuity data area are saved in the continuity data file before the area is released. When input message 2 is received, it is queued for the action designated as the external successor. Input message 2 is not tested for a transaction code. When the new action is scheduled, the saved continuity data record is read into the new continuity data area for action program 2.

How is the dialog transaction (Figure 3-3) different from two successive simple transactions like the one in Figure 3-2? First, in Figure 3-3, input message 2 does not contain a transaction code. It is immediately associated with the succeeding action designated by action program 1. In Figure 3-2, if a second simple transaction follows the one shown, the input message must contain a transaction code. Second, in Figure 3-3, a continuity data area is used to carry information from the first action to the second action. The action programs need not intervene to provide the continuity. In Figure 3-2, if a second simple transaction follows the one shown, information from the first transaction cannot be passed to the second (other than through the second input message or through a user data file).

The action program structures shown in Figures 3-2 and 3-3 should meet the requirements of most applications. For applications that require greater flexibility in the allocation of resources during an action, there are two additional types of action program succession – immediate internal succession and delayed internal succession.

3.1.2.3. Immediate Internal Succession

The processing within an action may consist of the execution of two or more action programs in sequence. Figure 3-4 illustrates the execution of two action programs in a single action. The process is the same as in Figure 3-2 except when action program 1 terminates. In this situation, the action program specifies in the program information block the name of another action program and designates the type of information as immediate internal succession. This effectively provides an overlay capability.

Action program 1 is released, but the allocated main storage areas are held and no message is output. Action program 2 is then acquired; it is given control and access to the same main storage areas (program information block, input message area, work area, output message area, continuity data area, and defined record area) as its predecessor. Action program 2 completes the processing for the action and terminates. The normal transaction termination results in the deallocation of resources and the output of the message.

Because immediate internal succession involves only one action, all files accessed by the successor program must be available when the initial program is executed. This means that they must be specified in the configurator ACTION section for that action.

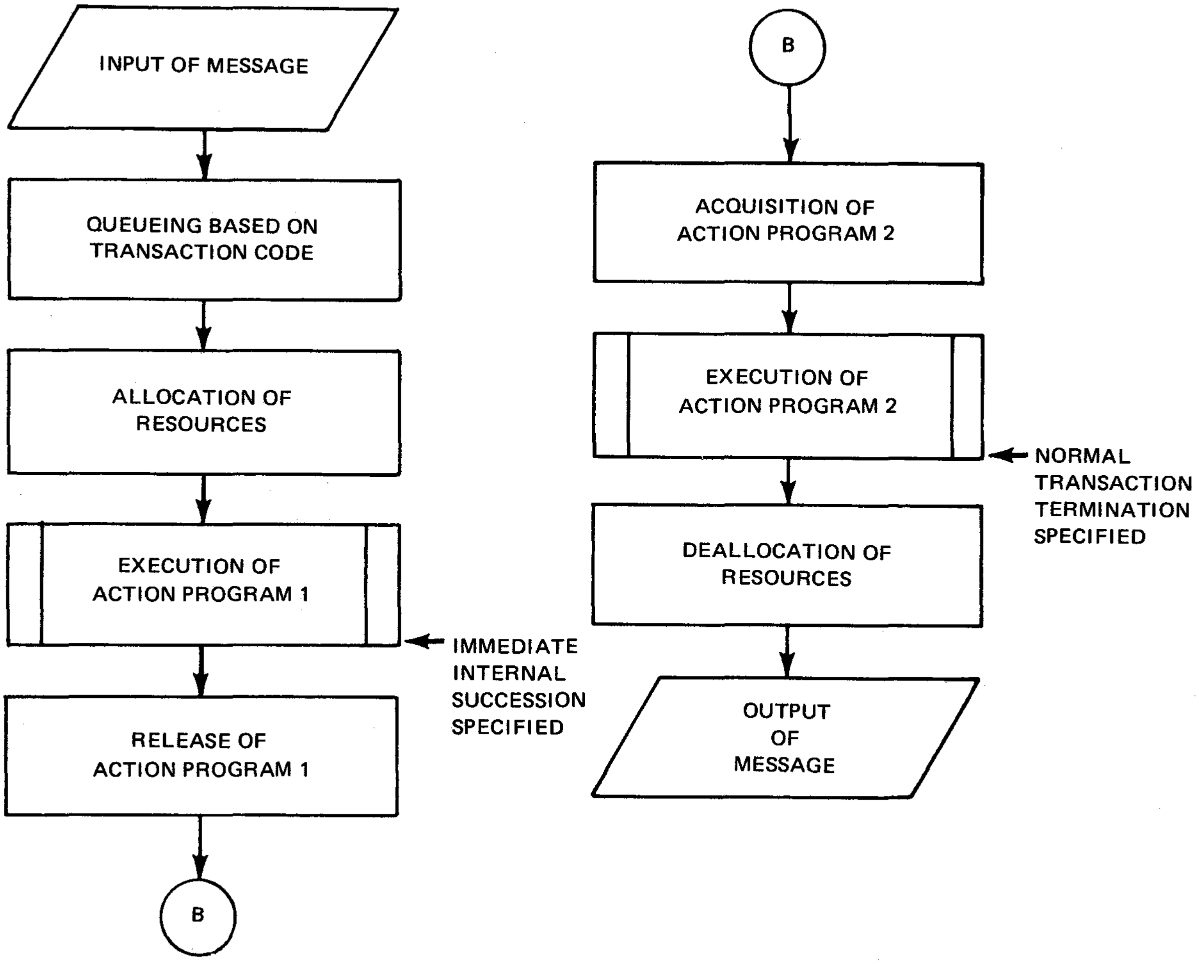

INPUT OF MESSAGE
QUEUEING BASED ON TRANSACTION CODE
ALLOCATION OF RESOURCES
EXECUTION OF ACTION PROGRAM 1
IMMEDIATE INTERNAL SUCCESSION SPECIFIED
RELEASE OF ACTION PROGRAM 1
B
B
ACQUISITION OF ACTION PROGRAM 2
EXECUTION OF ACTION PROGRAM 2
NORMAL TRANSACTION TERMINATION SPECIFIED
DEALLOCATION OF RESOURCES
OUTPUT OF MESSAGE

Figure 3—4. Immediate Internal Succession

3.1.2.4. Delayed Internal Succession

In some situations, main storage areas as well as action programs must be changed during the processing of a message. This can be the case, for example, when some exceptional condition that is encountered only infrequently in an input message requires a different program with a much larger work area to carry out the processing.

Delayed internal succession can be used to effect this dynamic change.

Delayed internal succession also can be used to minimize the I/O area requirements for an action. During the scheduling of an action, IMS 90 must dynamically allocate I/O areas for all files referenced in the action. If many files are accessed in processing a message, some frequently, some rarely, then the frequently accessed files should be specified at configuration time for one action, and all of the other files should be specified for another action.

The first action should be scheduled to process all messages from the terminal. If in processing a message it encounters an exceptional condition that requires a rarely accessed file, the first action should terminate and designate the second action as its delayed internal successor.

Delayed internal succession provides the only example of an action that *apparently* does not include both an input message and an output message. Actually, the first action builds an output message in the output message area, but this message is not sent to the originating terminal. Instead, it is queued as the input message to the second action. As a result, the second action does not require the input of a message from the terminal. In this type of structure, even though there is apparently only one input message and one output message, internally there are two separate actions, each with an input message and output message.

Note that delayed internal succession when used in single-thread IMS 90 behaves like immediate internal succession; that is, the output message is not queued as an input message to the next action but, instead, is passed immediately to the successor action program.

Figure 3-5 shows a transaction that employs delayed internal succession. This example is the same as Figure 3-2 up to the termination of action 1. The action program specifies in the PIB the name of another action as its successor and designates the type of information as delayed internal succession. The deallocation of resources occurs as in the first action in Figure 3-3. The output message of action 1, however, is not output to a terminal; it is queued as an input message to action 2.

When the scheduling occurs for action 2, the output message from the previous action is provided as the input message. In addition, the continuity data record developed in action 1 is made available in the new continuity data area for action 2.

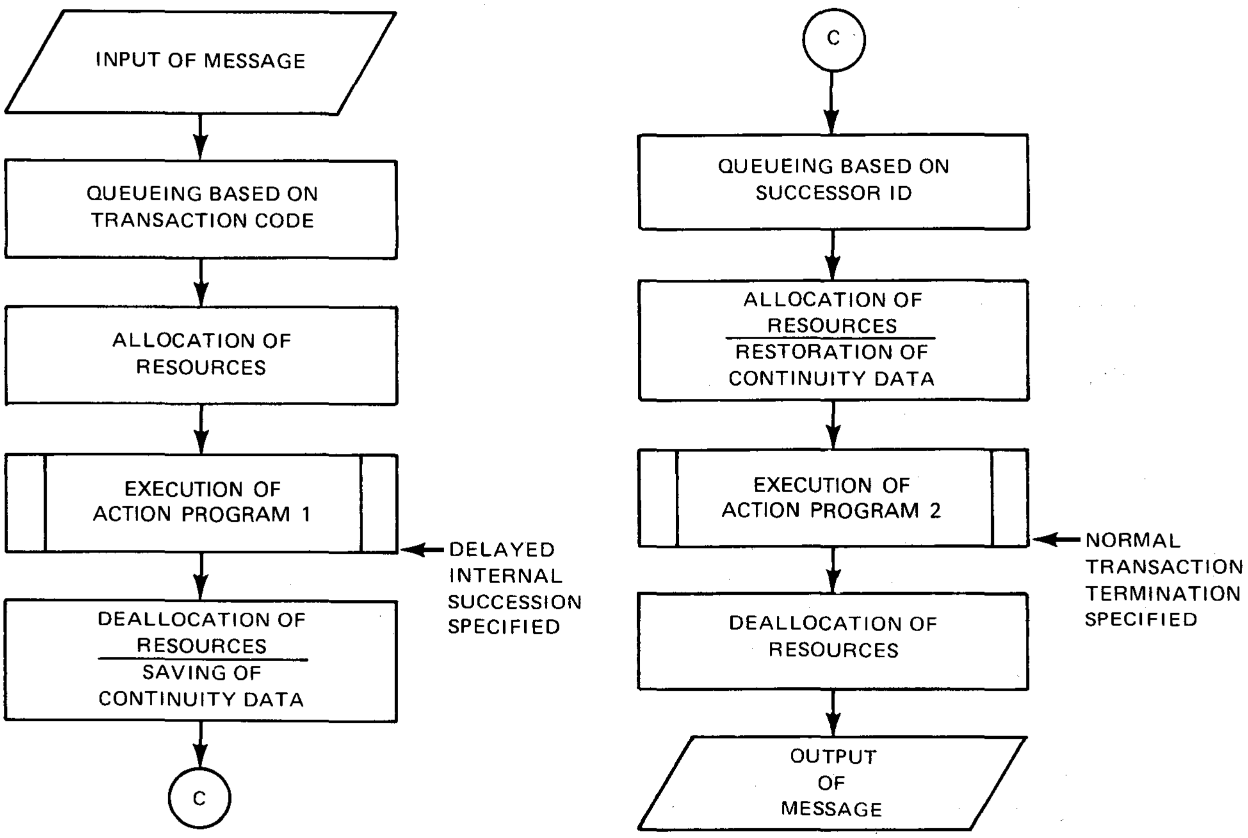
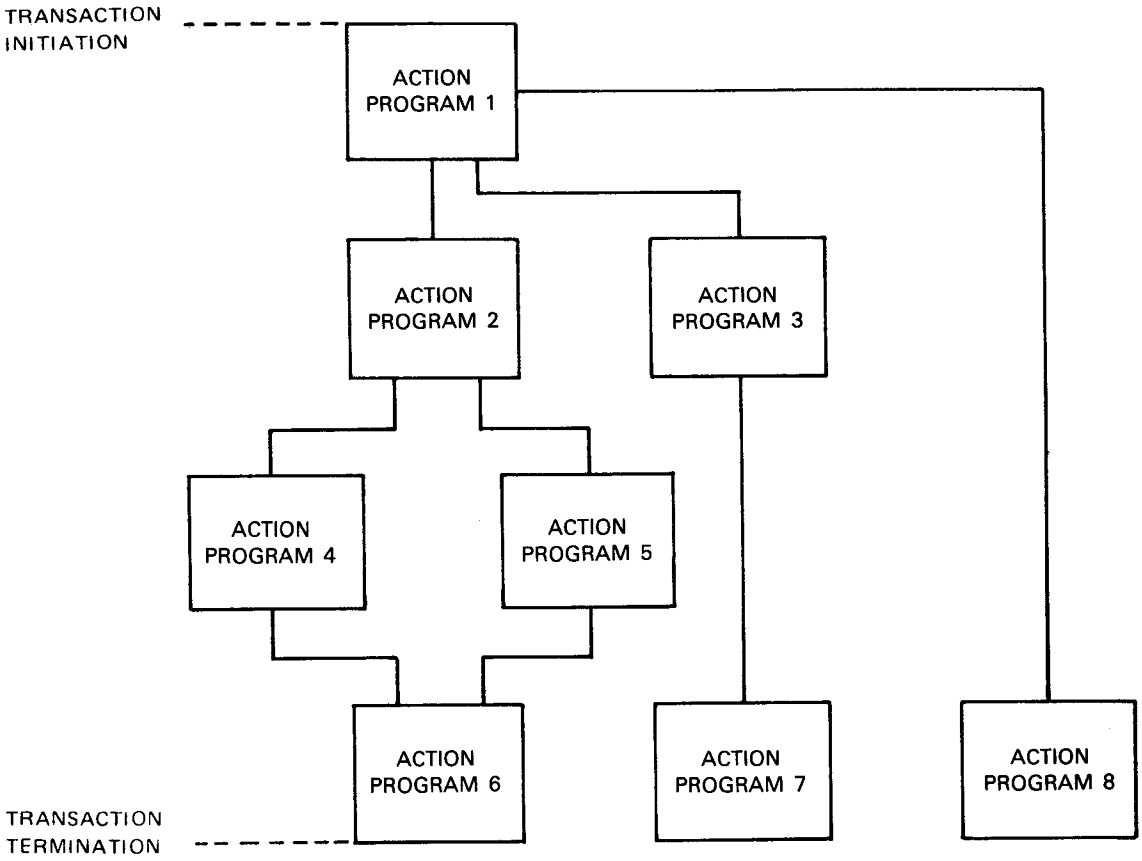

INPUT OF MESSAGE
QUEUEING BASED ON TRANSACTION CODE
ALLOCATION OF RESOURCES
EXECUTION OF ACTION PROGRAM 1
DELAYED INTERNAL SUCCESSION SPECIFIED
DEALLOCATION OF RESOURCES
SAVING OF CONTINUITY DATA
C
C
QUEUEING BASED ON SUCCESSOR ID
ALLOCATION OF RESOURCES
RESTORATION OF CONTINUITY DATA
EXECUTION OF ACTION PROGRAM 2
NORMAL TRANSACTION TERMINATION SPECIFIED
DEALLOCATION OF RESOURCES
OUTPUT OF MESSAGE

Figure 3—5. Delayed Internal Succession

3.1.2.5. Combination Structures

Combinations of the various types of action program succession leads to great flexibility in the structuring of transactions. There are basically no limitations to the ways in which types of succession can be combined. For example, immediate internal succession, delayed internal succession, and, finally, external succession all can be specified in turn.

The capability of specifying the successor identification dynamically in the PIB (versus prespecifying the same identification at configuration time) enables a program to develop a dynamic transaction structure. The structure is limited to one level of control; that is, at any given time, only one action program can process a given transaction. The processing, however, can take an arbitrary number of paths. Figure 3-6 is an example of dynamic transaction structure.



NOTE:

Connecting lines represent immediate internal, delayed internal, or external succession, or any combination of them.

Figure 3—6. Dynamic Transaction Structure

3.1.3. Action Program Reusability

Action programs can be written as serially reusable, reentrant (BAL only), or sharable (COBOL only). The type of reusability, as well as other attributes of action programs, must be defined to IMS 90 at configuration time. Programming considerations for program reusability are noted where applicable in the discussions on COBOL, RPG II, and BAL action programs. The type of reusability has little effect on processing speed in a single-thread environment; however, for effective use of multithread IMS 90, action programs should be either shared code (COBOL) or reentrant (BAL). (RPG II programs are serially reusable only.) By using shared code or reentrant action programs in a multithread environment, you can avoid deadlock. Action programs written in either shared code or reentrant code avoid the case where a serially reusable program with outstanding record or file locks cannot be scheduled for execution. (See 3.8.7.2 for a detailed discussion of deadlock.)

3.1.4. Device Independent Control Expressions (DICE)

User-written action programs can use either remote device hardware control characters or device independent control expressions (DICE) to format input and output messages via function codes that position the cursor, control carriage return, control forms, control line, feed line, and erase the screen.

ICAM automatically inserts DICE sequences into input messages. One reason for their use in input messages is to compare the input message with a previous output message for validation. For output messages, your action program must move the 4-byte DICE sequence to the output message area.

In most cases, the user configures the removal of DICE sequences from input messages by specifying EDIT=tablename or EDIT=c in the configurator ACTION section or by omitting the EDIT parameter. If EDIT=NONE is specified, the DICE sequences are not removed.

The 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, a column on the terminal. To set the function code, you choose the hexadecimal value equivalent to the operation required to format a message.

The COBOL action programmer can specify DICE sequence values in the working-storage section of his action program or, if certain DICE sequences are used frequently, he may file his description in a COBOL copy library to be called into the action program via the COPY statement in his action program's working-storage section. (See Figures 3–27 and 3–28.)

The action programmer using 1968 American National Standard COBOL must indicate the hexadecimal values for the DICE sequences by supplying their multipunch equivalents. (See Figures 3–27 and 3–28 for examples that use multipunch written to 1968 standards.) The 1974 COBOL standards permit the action programmer 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:

```
Ø1 DICE
    03 FIELD-1 PIC X.
    Ø3 FIELD-2 PIC X.
    Ø3 FIELD-3 PIC X.
    03 FIELD-4 PIC X.
MOVE =‘1Ø’ TO FIELD-1.
MOVE =‘03’ TO FIELD-2.
MOVE =‘Ø1’ TO FIELD-3.
MOVE =‘Ø1’ TO FIELD-4.
```

Example 2:

```
   Ø3 DICE PIC X(4).
MOVE =‘1ØØ3Ø1Ø1’ TO DICE.
```

Example 3:

```
77 DICE PIC X(4) VALUE =‘1ØØ3Ø1Ø1’
```

In a BAL action program you can use DICE macros provided by ICAM to generate DICE sequences inline. For an example of use of the ICAM procedure, ZO#POSC, to generate inline the DICE sequence for clearing the current line and repositioning of the cursor, see Figure 3-38.

DICE is a more convenient method of writing COBOL and BAL action programs than using the device dependent editing and control codes peculiar to specific terminals. DICE usage and DICE codes are described in Appendix E.

3.2. COBOL ACTION PROGRAMS

The basic format of a COBOL action program is:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. program-name.
(Any optional entry may be specified.)
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
OBJECT-COMPUTER.
DATA DIVISION.
WORKING-STORAGE SECTION.
(Optional. May be used only to contain constants.)
LINKAGE SECTION.
(Describes each area that is referenced in the procedure division ENTRY
  statement.)
    01 PROGRAM-INFORMATION-BLOCK
          .
          .
          .
    01 INPUT-MESSAGE-AREA
          .
          .
          .
    [01 WORK-AREA
          .
          .
          .
    [01 OUTPUT-MESSAGE-AREA
          .
          .
          .
    [01 CONTINUITY-DATA-AREA]]]
          .
          .
          .
PROCEDURE DIVISION USING PROGRAM-INFORMATION-BLOCK INPUT-MESSAGE-AREA
[WORK-AREA[OUTPUT-MESSAGE-AREA[CONTINUITY-DATA-AREA]]].
```

3.2.1. COBOL Action Program Sharability

A COBOL action program can be compiled with or without the shared code parameter of the extended COBOL compiler. The shared code parameter format for American National Standard 1968 COBOL is:

```
// PARAM OUT=(M)
```

The shared code parameter format for American National Standard 1974 COBOL is:

```
// PARAM IMSCOD=YES
```

Use of this parameter allows the compiler to check the program for conformance to IMS 90 language restrictions and issue appropriate diagnostics at compile time. Using this option in combination with the TYPE=SHR and SHRDSIZE parameters of the configurator allows the programs to be run as reentrant under multithread IMS 90.

3.2.2. COBOL Language Restrictions

Use of certain COBOL language elements is restricted in action programs. The following restrictions enable programs to be compiled as either sharable or serially reusable:

- IDENTIFICATION DIVISION

  Do not use a function key (F#nn) as the PROGRAM-ID name, because the COBOL compiler treats the # symbol as invalid. If you want to use a function key to identify the load module, 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.

- CONFIGURATION SECTION

  Special names must be omitted.

- INPUT-OUTPUT SECTION

  The input-output section must be omitted because all I/O is performed by IMS 90 file management.

- FILE SECTION

  The file section must be omitted.

- WORKING-STORAGE SECTION

  The working-storage section is optional. When present, it is used only to contain constants; that is, all elementary items must be described with a VALUE clause. When you compile your action program with the shared-code parameter, COBOL flags only procedure statements that move data to the working-storage section. If the program does change these values, it must not depend upon their original value during a subsequent execution.*

- PROCEDURE DIVISION

  A DECLARATIVE clause may not be included. Debugging language (EXHIBIT, TRACE) and segmentation (priority numbers, SEGMENT-LIMIT clause) are prohibited.

**If the action program is to be compiled only under the serially reusable option, data items within the working-storage section may be changed in value during execution. However, the execution of such a program must not depend upon values left in working storage by a previous execution.*

The following COBOL verbs, clauses, and sections are illegal in action programs. When compiling with the // PARAM OUT=(M) or // PARAM IMSCOB=YES options, the following language elements are diagnosed and deleted from the program by the compiler:

ACCEPT
ALTER
CLOSE
DECLARATIVE SECTION
DISPLAY
ENTRY
EXHIBIT
EXIT-PROGRAM
FILE SECTION
INPUT-OUTPUT SECTION
INSERT
OPEN
READ
READY TRACE
RELEASE
RESET TRACE
RETURN
REWRITE
SEEK
SEGMENT-LIMIT
SORT
STOP
SYSCHAN-t
SYSCOM
SYSCONSOLE
SYSDATE
SYSERR [-m]
SYSLST
SYSSWCH
SYSTIME
WRITE

The following verbs must not have working-storage items as receiving operands. If the shared code parameter was used upon detection of this condition, the compiler generates the statement and issues a precautionary diagnostic.

ADD
COMPUTE
DIVIDE
EXAMINE (REPLACING option)
MOVE
MULTIPLY
PERFORM (VARYING option)
SEARCH (VARYING option)
SET
SUBTRACT
TRANSFORM

Normally, execution-time errors result in a CE error message and program termination. In an action program, execution-time errors result in a program check interrupt and a snapshot dump of the action program with the address of the CE message in register 1. The action program is terminated. For further details, refer to 3.8.6.3.

Under multithread IMS 90, for the COBOL object program to be reentrant at CALL interrupts, the volatile work area used by the program must be saved and restored by the IMS 90 system. The size of the area (which varies between programs) is displayed in decimal by the printer immediately prior to the COBOL COMPILATION COMPLETE message. The message reads:

```
SHARED CODE VOLATILE DATA AREA = nnnn BYTES
```

This size is used in computing the SHRDSIZE parameter in the IMS 90 configurator. This message and the reentrant capability of the program is available only when the program is compiled with the COBOL shared-code parameter.

3.2.3. Linkage Section

The linkage section is required. It describes areas that are made available to an action program when it is called by action scheduling to begin or continue the processing of an input message (Figure 3-1). The areas are allocated by action scheduling based on the information contained in the action control table (specified at configuration time) and parameters set in the program information block by the final action program in the preceding action (if any). The contents of areas referenced through the linkage section can be modified by an action program as required in the processing of an action. For each area referenced in the USING list at the point of entry to the action program, there must be a corresponding 77 or 01 level data description in the linkage section.

The main storage areas described in the linkage section always include the program information block (PIB) and input message area (IMA) and may also include a work area (WA), an output message area (OMA), and a continuity data area (CDA). The defined record area is never described in the linkage section; it is used by defined record management (DRM) when defined files are accessed and cannot be written into by the action program. Main storage areas and their uses are discussed in 3.6.

3.2.4. Procedure Division

The procedure division header with the USING statement lists the main storage areas used by a COBOL action program. The parameters of the USING list are positional and must be coded in the prescribed order, taking care to code a dummy parameter to maintain this order where an optional parameter is omitted. For example, if the WORK-AREA and CONTINUITY-DATA-AREA parameters are not required, the procedure division starts with the following statement:

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

where:

D

Is a data name used as a dummy parameter for WORK-AREA to maintain the proper position for OUTPUT-MESSAGE-AREA. The CONTINUITY-DATA-AREA parameter is omitted entirely. The D must also have a corresponding 77 or 01 level entry in the linkage section.

A basic rule that must be followed in writing the procedure division statements of a COBOL action program is that no I/O operations are specified using standard COBOL I/O verbs. All file operations must be requested through IMS 90 file management, using the CALL statement. (See 3.7.) While message input and output are generally handled implicitly via the input and output message areas, certain message output operations can be performed by making explicit requests to internal message control. (See 3.8.) These and other requests to IMS 90 also are made by means of the CALL statement. The basic format of the CALL statement as used in COBOL action programs is:

```
CALL function-name [USING data-name...]
```

where:

`function-name`

Is the name of an IMS 90 function. The IMS 90 functions are GET, GETUP, UNLOCK, SNAP, SUBPROG, GETLOAD, SETLOAD, PUT, DELETE, INSERT, SETL, ESETL, FREE, SEND, and RETURN. Function names must be enclosed by single quotes.

`data-name`

Is the name of an item defined in the working-storage section or linkage section. The level number associated with each data name need not be 01 or 77. However, if a dataname at a level other than 01 or 77 is given in the USING list, a precautionary diagnostic message (166) is printed for the statement by the COBOL compiler. This message is ignored.

The CALL statement is preceded by an ENTER LINKAGE statement and followed by an ENTER COBOL statement. When a function is requested of IMS 90, the function is always carried to completion before control is returned to the statement following the CALL statement. Therefore, only one request can be outstanding for a given action at a given time.

Program termination is requested by means of the RETURN function, whose format is:

```
CALL 'RETURN'
```

The CALL RETURN statement must be present at least once in an action program. The execution of this function results in the return of control to action scheduling and termination of the action program. The EXIT PROGRAM or RETURN statement must not be used for this purpose. When a COBOL action program is compiled without the PARAM OUT=(M) statement, a diagnostic message is issued by the compiler because of the absence of an EXIT PROGRAM or RETURN statement. This diagnostic should be ignored.

3.3. RPG II ACTION PROGRAMS

An RPG II action program is distinguished from a usual RPG II program by the letter A in column 74 of the RPG II control card specifications form. In this way, the RPG II compiler generates an RPG II program that interfaces with IMS 90 instead of data management.

RPG II action programs operate under the application management component of IMS 90 to process input messages and generate output messages. These programs must be written so that they are serially reusable. After each execution of an action program, RPG II resets all indicators and internal switches. Therefore, the programmer must reset all fields to their original values before the action program is executed again. All file I/O requests are handled by IMS 90. RPG II action programs are compiled and linked offline. The sample RPG II action program, LSTLIM (see Appendix C), provides a detailed explanation of this entire process. Users should be familiar with the current version of the RPG II user guide, UP-8067.

3.3.1. IMS 90/RPG II Interface Areas

Four defined interface areas are used for control and communication of data between IMS 90 and RPG II action programs:

1. Input message area (IMA)
2. Program information block (PIB)
3. Output message area (OMA)
4. Continuity data area (CDA)

Unlike COBOL action programs, RPG II action programs do not use the work area. Use of the four interface areas depends upon the requirements of the RPG II action program.

If the action program needs to reference a field in any of the IMS 90 interface areas, the interface area name preceded by a single asterisk (*) must be indicated as the device name in columns 40-46 of the RPG II file description specifications form (e.g., *IMA, * OMA, etc.). (See Figure 3-7.) It is important to remember in referencing all four data interface areas to reference only those fields within the interface areas that your RPG II action program actually use; otherwise, the field will be flagged as unreferenced in your RPG II program.

3.3.1.1. Input Message Area (IMA)

The IMA consists of a 16-byte control header plus the input message received from a terminal. The size of the IMA is, therefore, the input message length plus 16 bytes for the IMS 90 header information. In an RPG II action program, the IMA is defined as an input file on the file specifications form (columns 7-14) and the fields within the IMA are described on the input format specifications form (columns 44-58). In addition, the INSIZE parameter in the ACTION section of the IMS 90 configuration must specify the total IMA size used by an RPG II action program.

Following normal RPG II input specification rules for the IMA header, you do not define fields not required by your action program; however, you must allow the first 16 bytes of the IMA for the header and start the first input message character in position 17. Table 3-1 is a summary of required entries defining the IMA on the file description specifications form.

Table 3—1. Summary of Required Entries for File Description Specifications Form

Interface Area	User-Specified File Name (Columns 7–14) See Note.	File Type (Column 15)	File Designation (Column 16)	Format (Column 19)	Record Length (Columns 24–27)	Device Name (Columns 40–46)
PIB	Any	I or U	D	F	48	*PIB
IMA	Any	I or U	P, S, or D	F	16 + message size	*IMA
OMA	Any	U or O	D or blank	F	16 + message size	*OMA
CDA	Any	I, U, or O	P, S, D, or blank	F	data size	*CDA

NOTE:

Any user file name can be specified but must begin with an alphabetic character and must not exceed seven characters.

3.3.1.2. Program Information Block (PIB)

The PIB is used to pass control information between IMS 90 and the user action program. It is a predefined 48-byte area.

Use of the PIB in an RPG II action program is optional. RPG II automatically performs the following PIB functions whether a PIB is or is not defined in an RPG II action program:

- RPG II checks the status code returned after each I/O request. When an error condition exists, RPG II sets the H0 indicator on and places the error status code into the *ERROR field. These error codes are shown in the system messages programmer/operator reference, UP-8076 (current version).

- RPG II sets the abnormal termination code 'S' in the TERMINATION-INDICATOR of the PIB if the user does not set the H0 indicator off by the end of detail calculations.

- RPG II returns default value 'N' in the TERMINATION-INDICATOR of the PIB. If no error occurs (H0–H9 indicators set off), no abnormal termination indicator 'S' is set and no succession is indicated (the E, I, or D indicators set).

The PIB is defined on the file description specifications form, and the PIB fields that are accessed or updated in the action program are described on either the input format specifications form, the output format specifications form, or both. There are two allowable specifications for the PIB on the file specifications form, columns 15 and 16:

- Input, demand file (I in column 15 and D in column 16)

- Update, demand file (U in column 15 and D in column 16)

The input demand file specification indicates that the action program requires data from the PIB but does not want to change the data. The action program accesses the PIB any number of times during detail calculations via the READ operation.

The update demand file specification indicates that the action program requires data from the PIB and wants to update the data. In this case, the action program accesses the PIB the same as an input demand file; however, it updates the PIB by issuing either the EXCPT operation in the calculations specifications or by using detail or total output specifications and letting the RPG II logic perform the output.

Only PIB fields referenced by an action program are defined on the input or output specifications forms. Any name can be specified for a PIB field referenced in your action program; however, the formats and positions of the fields referenced in the action program must agree with their definition and position in the IMS 90 PIB. (See Table 3-1 for the entries required to define the PIB on the file description specifications form.)

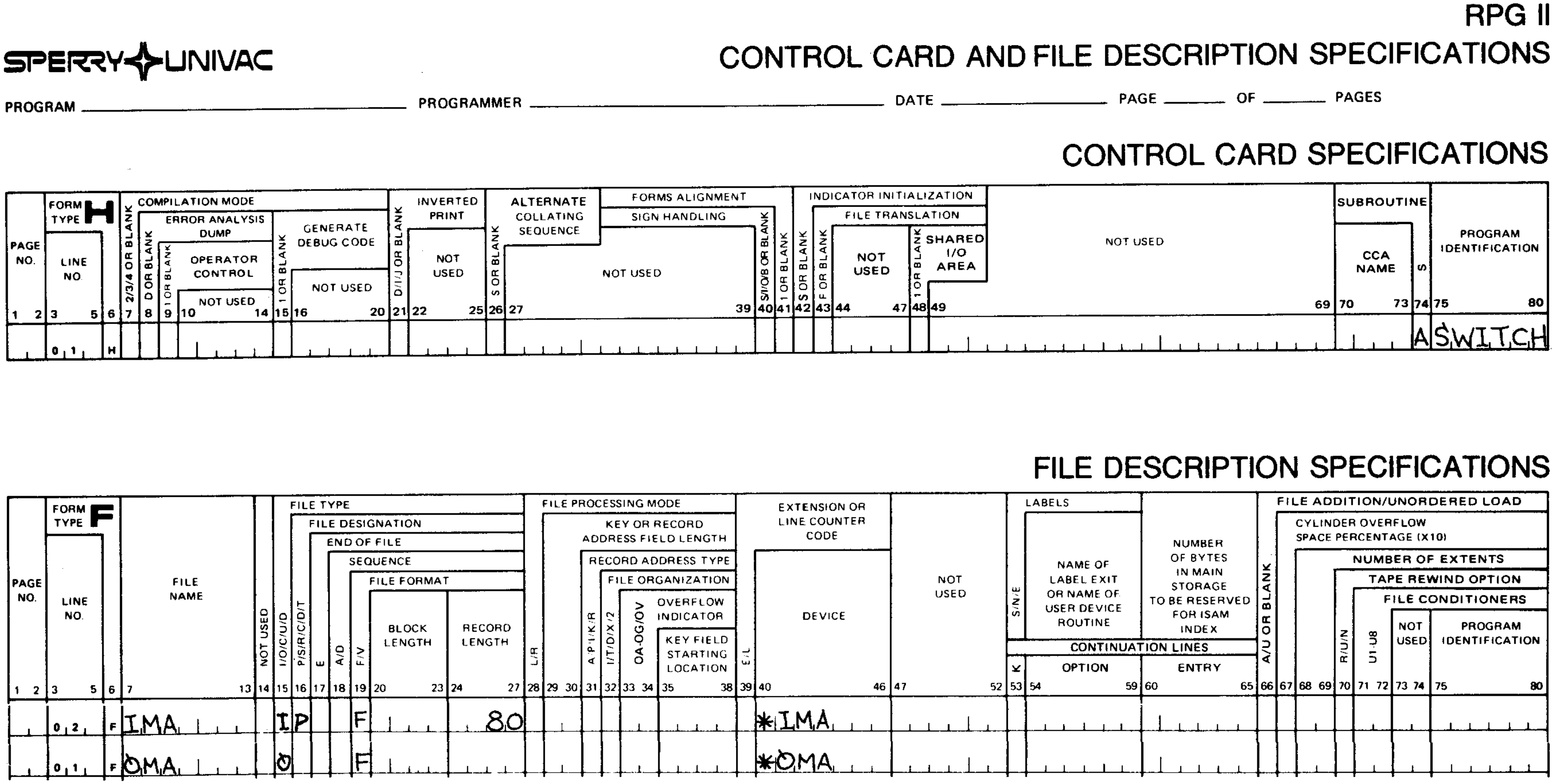
3.3.1.3. Output Message Area (OMA)

The OMA consists of a 16-byte control header plus the output message built by the action program and sent to a terminal in response to an input message. Use of the OMA in RPG II action programs is optional; however, when the OMA is used, the OUTSIZE parameter of the IMS 90 ACTION configuration section must indicate the size of the user's output message.

The OMA can be used for multiple output messages during one action program execution. Also, an output message can be transmitted to a terminal other than the initiating terminal (e.g., message switching). To perform either of these operations, disk queueing must be configured for ICAM via the DISCFILE macro, and unsolicited outpout must be configured via the UNSOL parameter.

If an error occurs on a CALL 'SEND' to unsolicited output, the *ERROR field in the dump will contain a letter 'K' and at action program termination, 'RPG036' will be displayed on the system console.

When an output message is to be transmitted to a terminal other than the initiating terminal, output specifications for the switched message must appear before those for the required action termination message. Figure 3-7 illustrates the file description input format and output format specifications for a message switching operation.



SPERRY✦UNIVAC

RPG II

CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

PROGRAM ____ PROGRAMMER ____ DATE ____ PAGE ____ OF ____ PAGES

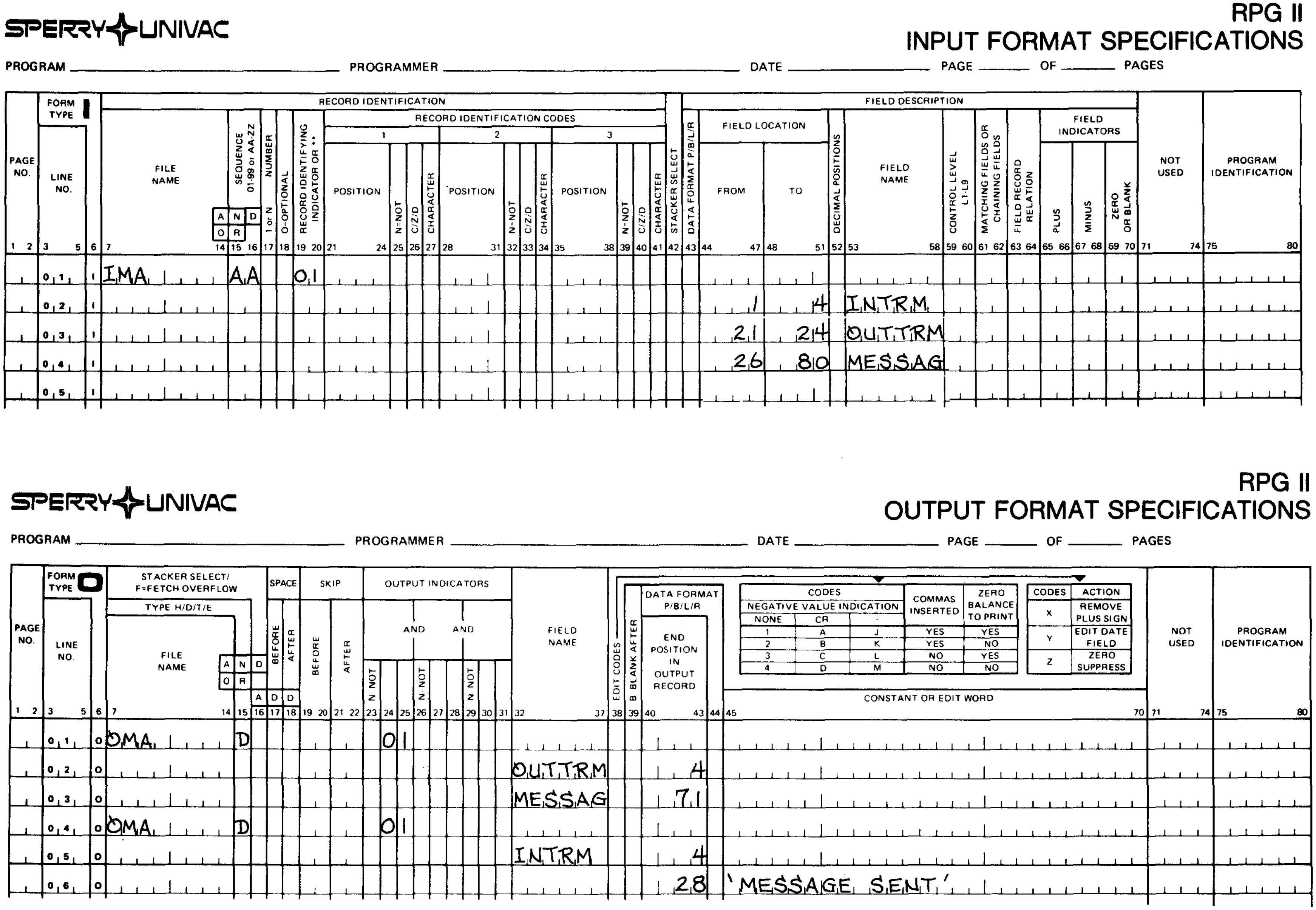
CONTROL CARD SPECIFICATIONS

PAGE NO. 1-2	FORM TYPE H LINE NO. 3-5	6	COMPILATION MODE 2/3/4 OR BLANK 7	ERROR ANALYSIS DUMP D OR BLANK 8	1 OR BLANK 9	OPERATOR CONTROL NOT USED 10-14	1 OR BLANK 15	GENERATE DEBUG CODE NOT USED 16-20	D/I/J OR BLANK 21	INVERTED PRINT NOT USED 22-25	S OR BLANK 26	ALTERNATE COLLATING SEQUENCE / FORMS ALIGNMENT / SIGN HANDLING NOT USED 27-39	S/I/O/B OR BLANK 40	1 OR BLANK 41	S OR BLANK 42	F OR BLANK 43	INDICATOR INITIALIZATION / FILE TRANSLATION NOT USED 44-47	1 OR BLANK 48	SHARED I/O AREA / NOT USED 49-69	SUBROUTINE CCA NAME 70-73	S 74	PROGRAM IDENTIFICATION 75-80
	01	H																			A	SWITCH

FILE DESCRIPTION SPECIFICATIONS

PAGE NO. 1-2	FORM TYPE F LINE NO. 3-5	6	FILE NAME 7-13	NOT USED 14	FILE TYPE I/O/C/U/D 15	FILE DESIGNATION P/S/R/C/D/T 16	END OF FILE E 17	SEQUENCE A/D 18	FILE FORMAT F/V 19	BLOCK LENGTH 20-23	RECORD LENGTH 24-27	FILE PROCESSING MODE L/R 28	KEY OR RECORD ADDRESS FIELD LENGTH 29-30	RECORD ADDRESS TYPE A/P/I/K/R 31	FILE ORGANIZATION I/T/D/X/2 32	OVERFLOW INDICATOR OA-OG/OV 33-34	KEY FIELD STARTING LOCATION 35-38	EXTENSION OR LINE COUNTER CODE E/L 39	DEVICE 40-46	NOT USED 47-52	LABELS S/N/E / CONTINUATION LINES K 53	NAME OF LABEL EXIT OR NAME OF USER DEVICE ROUTINE / OPTION 54-59	NUMBER OF BYTES IN MAIN STORAGE TO BE RESERVED FOR ISAM INDEX / ENTRY 60-65	FILE ADDITION/UNORDERED LOAD A/U OR BLANK 66	CYLINDER OVERFLOW SPACE PERCENTAGE (X10) 67	NUMBER OF EXTENTS 68-69	TAPE REWIND OPTION R/U/N 70	FILE CONDITIONERS U1-U8 71-72	NOT USED 73-74	PROGRAM IDENTIFICATION 75-80
	02	F	IMA		I	P			F		80								*IMA											
	01	F	ØMA		Ø				F										*ØMA											

Figure 3—7. Message Switching Program Specifications (Part 1 of 2)



SPERRY UNIVAC

RPG II INPUT FORMAT SPECIFICATIONS

PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE ____ OF ____ PAGES

PAGE NO. (1-2)	LINE NO. (3-5)	FORM TYPE (6)	FILE NAME (7-14)	SEQUENCE 01-99 or AA-ZZ (15-16)	NUMBER 1 or N (17)	O=OPTIONAL (18)	RECORD IDENTIFYING INDICATOR OR ** (19-20)	RECORD IDENTIFICATION CODES 1: POSITION (21-24)	N=NOT (25)	C/Z/D (26)	CHARACTER (27)	2: POSITION (28-31)	N=NOT (32)	C/Z/D (33)	CHARACTER (34)	3: POSITION (35-38)	N=NOT (39)	C/Z/D (40)	CHARACTER (41)	STACKER SELECT (42)	DATA FORMAT P/B/L/R (43)	FIELD LOCATION FROM (44-47)	TO (48-51)	DECIMAL POSITIONS (52)	FIELD NAME (53-58)	CONTROL LEVEL L1-L9 (59-60)	MATCHING FIELDS OR CHAINING FIELDS (61-62)	FIELD RECORD RELATION (63-64)	FIELD INDICATORS PLUS (65-66)	MINUS (67-68)	ZERO OR BLANK (69-70)	NOT USED (71-74)	PROGRAM IDENTIFICATION (75-80)
	01	I	IMA	AA			01																										
	02	I																				1	4		INTRM								
	03	I																				21	24		OUTTRM								
	04	I																				26	80		MESSAG								
	05	I																															

SPERRY UNIVAC

RPG II OUTPUT FORMAT SPECIFICATIONS

PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE ____ OF ____ PAGES

CODES	NEGATIVE VALUE INDICATION NONE	CR	-	COMMAS INSERTED	ZERO BALANCE TO PRINT
	1	A	J	YES	YES
	2	B	K	YES	NO
	3	C	L	NO	YES
	4	D	M	NO	NO

CODES	ACTION
X	REMOVE PLUS SIGN
Y	EDIT DATE FIELD
Z	ZERO SUPPRESS

PAGE NO. (1-2)	LINE NO. (3-5)	FORM TYPE (6)	FILE NAME (7-14)	TYPE H/D/T/E (15)	STACKER SELECT/ F=FETCH OVERFLOW (16)	SPACE BEFORE (17)	SPACE AFTER (18)	SKIP BEFORE (19-20)	SKIP AFTER (21-22)	OUTPUT INDICATORS N NOT (23)	(24-25)	AND N NOT (26)	(27-28)	AND N NOT (29)	(30-31)	FIELD NAME (32-37)	EDIT CODES (38)	B BLANK AFTER (39)	END POSITION IN OUTPUT RECORD (40-43)	DATA FORMAT P/B/L/R (44)	CONSTANT OR EDIT WORD (45-70)	NOT USED (71-74)	PROGRAM IDENTIFICATION (75-80)
	01	O	OMA	D							01												
	02	O														OUTTRM			4				
	03	O														MESSAG			71				
	04	O	OMA	D							01												
	05	O														INTRM			4				
	06	O																	28		'MESSAGE SENT'		

Figure 3—7. Message Switching Program Specifications (Part 2 of 2)

3.3.1.4. Continuity Data Area (CDA)

The CDA is used for passing data from one action program to its successor during a dialog transaction. Its size varies and its use is optional (3.6.5). If the CDA is used, its size is specified by the CDASIZE parameter of the IMS 90 configuration ACTION section.

The CDA file is described as an input, update, or output file on the file description specifications form (column 15), and the fields within the file are described on the input format specifications form or the output format specifications form according to their use in the action program. Table 3-1 lists the required file description specification entries for the CDA.

An RPG II action program can create, read, or update the CDA. If the action program is creating the CDA, it must define the CDA on the file specifications form as an output file. If it reads the CDA without changing any of its data, the action program must define the CDA as an input file on the file specifications form. When updating the CDA area and passing its data to the successor program, the action program must define the CDA as an update file on the file specifications form. When CDA is to be deleted, the action program moves zeros to the CONTINUITY-DATA-OUTPUT-LENGTH field in the PIB. Table 3-2 lists these actions.

Table 3—2. File Type Specifications for Creating, Moving, and Updating the CDA

File Type Specification (Column 15)	Operation
I (input)	Reads the CDA
U (update)	Reads and updates CDA contents and passes to successor program
O (output)	Creates the CDA

3.3.2. User Logical Files and IMS 90 Defined Files

RPG II action programs access user logical ISAM, MIRAM, SAM, and DAM files as well as IMS 90 defined files. (To access IRAM files, you must define them as MIRAM files at configuration time.) User logical files are data files you create via OS/3 data management. Defined files are files created by IMS 90 from user logical files according to user-supplied data definitions. (See Section 2.)

To be used by RPG II action programs, user logical files that are not accessed by defined files must be identified by the FILES parameter of the configuration ACTION section. The FILE section of the configuration defines all logical files. Table 3-3 summarizes the file organization, related access methods, and file types used in an RPG II action program.

In RPG II action programs, ISAM files and defined files are specified and processed in the same manner. Table 3-4 lists the allowable file description specifications for logical and defined files.

Table 3—3. Summary of File Organizations, Access Methods, and File Types Used by RPG II Action Programs

File Organization	Related Access Method	File Type
IMS 90 Files: Defined	Random Sequential	Input/Update/Output* Input/Update/Output*
User Files: ISAM	Random Sequential, by key	Input/Update/Output* Input/Update/Output*
IRAM or MIRAM	Indexed Random Nonindexed Random Sequential	Input/Update/Output* Input/Update/Output Input
SAM	Sequential	Output
DAM	Relative	Input/Update/Output

*For output files, only ADD is allowed.

Table 3—4. Allowable RPG II File Description Specifications for ISAM, IRAM, MIRAM, DAM, and Defined Files

Column Title and Numbe.	Specification
Form Type (Column 6)	F
File Name (Column 7—14)	User-defined name
File Type (Column 15)	I, U, or O
File Designation (Column 16)	S, R, C, D, or P
Format (Column 19)	F
Record Length (Column 24—27)	User's record size
Mode of Processing (Column 28)	L,R, or blank
Key Field Length (Column 29—30)	01—99 ①
Record Address Type (Column 31)	A or P ① R ② blank ③
File Organization (Column 32)	I ① D ② blank ③
Key Field Start Position (Column 35—38)	0001-9999 ①
Device (Column 40—46)	Must be disk device
File Addition (Column 66)	Blank or A

NOTES:

① ISAM, IRAM, MIRAM and defined files

② DAM files

③ Sequential processing

RPG II action programs can process tape or disk SAM output files. Table 3-5 lists the RPG II file description specifications allowed for SAM output files.

Table 3—5. Allowable RPG II File Description Specifications for SAM Output Files

Column Title and Number	Specification
Form Type (Column 6)	F
File Name (Column 7—14)	User-defined name
File Type (Column 15)	O
Format (Column 19)	F
Record Length (Column 24—27)	User's record size
Overflow Indicator (Column 33—34)	May be specified for line counter files
Line Counter (Column 39)	Blank or L
Device (Column 40—46)	Must be disk or tape device

All user files used by RPG II action programs must be defined in file description specifications and input/output specifications. Record descriptions of IMS 90 defined files used by RPG II action programs must match their descriptions in the IMS 90 configuration.

An RPG II action program can access ISAM, DAM, MIRAM, and IMS 90 defined files in random mode by defining them as chained files on the file description specifications (column 16).

Under IMS 90, the RPG II program retrieves one record at a time. Updating or deletion of the retrieved record must be done before the next record is retrieved. Records being added to or deleted from a file on which updating is being performed cannot be added or deleted between the reading and writing of a record that is being updated. The ADD or DEL specifications in columns 16-18 of the output format specifications perform add or delete functions.

3.3.3. Specifications Forms for RPG II Action Programs

3.3.3.1. Control Card Specifications Form

The RPG II control specifications form identifies the action program and specifies that the program is to be compiled as an IMS 90 action program. Column 74 requires the letter 'A' and columns 75-80 require a 1- to 6-character program name beginning with an alphabetic character. Table 3-6 lists the entries allowable on the control card specifications form. (For restrictions on control card specifications, see Table 3-7.)

Table 3—6. RPG II Control Card Specifications for RPG II Action Programs

Column Title and Number	Specification
Form Type (Column 6)	H
Compilation Mode (Column 7)	Blank, 2, 3, or 4①
Error Analysis Dump (Column 8)	Must be blank②
Operator Control (Column 9)	Must be blank②
Generate Debug Code (Column 15)	Must be blank② or 1
Inverted Print (Column 21)	Blank, D, I, or J
Alternating Collating Sequence (Column 26)	Blank or S
Sign Handling (Column 40)	Blank, S, I, O, or B
Forms Alignment (Column 41)	Must be blank②
Indicator Initialization (Column 42)	Blank or S
File Translation (Column 43)	Blank or F
IMS 90 Action Program Indicator (Column 74)	Must be A
Program Identification (Column 75—80)	Blank or program name

NOTES:

① The file access method (MIRAM, ISAM, DAM) used in the program is not affected by this entry. This is determined by the IMS 90 configurator.

② This field is an RPG II option that is not permitted in an RPG II action program.

3.3.3.2. File Description Specifications Form

The file description specifications form describes the user logical files, defined files, and IMS 90/RPG II interface areas used or referenced by an RPG II action program. Tables 3-4 and 3-5 list the allowable file description specifications for user logical and defined files. Table 3-1 shows the specifications required to define IMS 90/RPG II interface areas as files in the action program. (For restrictions on file description specifications, see Table 3-7.)

3.3.3.3. Input Format Specifications Form

The input format specifications describe the fields within user files, defined files, or IMS 90/RPG II interface areas referenced by the action program. The same rules apply for coding action program input format specifications as for normal RPG II programs. (For restrictions on input format specifications, see Table 3-7.)

Table 3—7. Restricted RPG II Language Features

Specifications Form	Column	Description
Control card specifications	8	Error analysis dump
	9	Operator control
	41	First page forms alignment
File description specifications	15	File type (C and D)
	16	Table and array file designation (T) ①
	20–23	Block length ②
	32	File organization:
		ADDROUT (D) ①
		Record address (blank) ①
		Additional I/O areas ②
		SAM tape/disk input files ①
		ISAM and MIRAM output files
	40–46	Device:
		CTLRDR
		READER
		CRP
		PUNCH
		CONSOLE
		PRINTER
	53	Labels ②
	54–59	Name of label exit option ②
	60–65	Size of ISAM index entry ②
	66	Unordered load
	67	Cylinder overflow space percentage ②
	68–69	Number of extents ②
	70	Tape rewind ②
	71–72	File conditioners (U1–U8)
Extension specifications ①	9–10	Chaining (C1–C9) tables or arrays
Input format specifications	19–20	Spread card feature (TR)
	42	Stacker select
Calculation specifications	28–32	Display operation (DSPLY)
Output format specifications	16	Stacker select
Telecommunications specifications	–	–

NOTES:

① Used only with MIRAM files; must not be used with SAM input files.

② Ignored by RPG II compiler; must be specified in IMS 90 configuration.

3.3.3.4. Calculation Specifications Form

Operations on the files described on the file description specifications form and input or output format specifications forms can be specified on a calculation specifications. See Table 3-7 for restrictions on the use of calculation specifications.

3.3.3.5. Output Format Specifications Form

The output format specifications describe the complete output message on the file description specifications form. This output description includes DICE codes (see Appendix E) for screen display positioning, display headings, and data fields required by the action program. (See Figure C-8.) Output specifications also describe IMS 90/RPG II interface area fields referenced by the action program for use in external succession; e.g., PIB or CDA fields. For restrictions on the output format specifications, see Table 3-7.

3.3.4. RPG II Action Program Restrictions

A number of the RPG II language features are restricted from use in an action program. Table 3-7 lists these features and their restrictions by column number and RPG II specification form.

3.4. BAL ACTION PROGRAMS

3.4.1. Linkage Conventions

An action program is treated as a subprogram of the IMS 90 online program and is activated by action scheduling using standard linkage conventions. When control is transferred to the action program entry point, the register contents are as follows:

- Register 1 points to a parameter list containing (in order):

  - Program information block address

  - Input message area address

  - Work area address (optional)

  - Output message area address (optional)

  - Continuity data area address (optional)

  If any of these parameters is not specified, the omission is indicated by a binary 0 in the parameter list. The position of parameters in the list is fixed.

- Register 13 contains the address of a 72-byte save area. The standard linkage conventions must be observed in the use of the save area. In particular, forward and backward save area links must be established to relate the save area given by register 13 to the save area used by the action program to make requests to IMS 90 for all functions.

- Register 15 contains the address of the action program entry point.

The action program must allocate space for a 72-byte, word-aligned save area to be used on calls to IMS 90 for all functions. This area must be properly linked to the save area provided by action scheduling for an action program. In all parameter lists that are passed to IMS 90, the sign bit must be set in the final parameter word. Standard linkage conventions must be observed.

When a continuity data area is referenced in a BAL action program, the program should not save addresses of items within the area from one action to another. Since the location of the continuity data area for a transaction can change from action to action, any saved addresses can thereby be invalidated.

3.4.2. Function Requests

An action program must not contain any direct requests to OS/3 data management. All file I/O functions must be requested through function calls in the form of CALL or ZG#CALL macros. (The difference between CALL and ZG#CALL is explained in 3.4.3.) Explicit message output, termination, and other functions also are requested in this manner. The format for CALL and ZG#CALL is:

```
[name]{CALL   }function-name[,(param-1,...,param-n)]
      {ZG#CALL}
```

File I/O and explicit message output functions are described in 3.8 and 3.9. Termination is requested via the RETURN function in this format:

```
ZG#CALL 'RETURN'
```

Execution of the RETURN function returns control to action scheduling.

3.4.3. Reentrant Programming Considerations

The following rules must be observed in coding a reentrant BAL action program:

1. Instructions must not be modified during the execution of the action program. For example, the length field of an MVC instruction must not be stored into the instruction itself. The EX instruction can be used to modify the length field or the MVC instruction can be built and executed in the activation record.

2. Data must not be embedded in the action program. All references to variable data must be made to the activation record (i.e., the program information block, input message area, work area, output message area, and continuity data area). DSECTs can be used to conveniently reference the areas in the activation record. Constant data can be defined in the action program.

3. Parameter lists containing addresses that vary from request to request or containing addresses of parameters defined in the activation record must be built in the activation record. The CALL macro with the (param-1,...,param-n) form of the parameter list can be used only to make requests for which all parameters are defined in the action program because the CALL macro generates a list of address constants in line for the parameter list.

   The macro ZG#CALL can be used to make requests for which some or all of the parameters are defined in the activation record. ZG#CALL builds a parameter list at the user-defined location PLIST. PLIST must be at least a 16-byte area beginning on a word boundary within the activation record, typically within the work area. The parameter list is built by ZG#CALL through use of a series of LA and ST instruction pairs. The ZG#CALL macro must be used for the RETURN function; CALL RETURN is not valid in an action program.

   The following examples illustrate the difference in the use of the CALL and ZG#CALL macros:

   - CALL ESETL, (INFILE)
   - INFILE is a constant defined in the action program.
   - ZG#CALL RETURN
   - No parameter list is generated.
   - ZG#CALL GET, (INFILE, RECORD, KEY)
   - INFILE is a constant defined in the action program, but RECORD and KEY are defined in the work area of the activation record.

3.5. USER-WRITTEN RESIDENT SUBPROGRAMS

BAL and COBOL action programs can call user-written resident subprograms. Thus, common functions, such as repetitive computations, need be coded only once, as subroutines, and then called by those action programs that need them. The fact that these subroutines are made resident guarantees their efficient use by not requiring that they be loaded into main storage each time they are called. They are loaded with IMS 90 during the start-up procedure.

User-written subprograms can be prepared as either serially reusable or reentrant load modules and must be resident in main storage to be called by a COBOL or BAL action program. They cannot be used with RPG II action programs and are not available in a basic IMS 90 system.

A subprogram cannot call on another subprogram; however, a subprogram can issue all the function calls supported for action programs. Information is passed to the subprogram from the calling action program via a parameter list. A subprogram can access only those files that are allocated to the calling action program.

Subprogram use must be specified in the IMS 90 configuration via the SUBPROG parameter in the OPTIONS section. In addition, the subprogram name must be specified on the program-name parameter of the PROGRAM section and SUBPROG=YES must be specified in the same section. The COBOL or BAL program must then place the subprogram name in the SUCCESSOR-ID field of the PIB before calling the resident subprogram.

In case of error in a subprogram, IMS 90 provides snapshot dumps of both the calling action program and the active subprogram.

3.5.1. Subprogram Reusability

A resident subprogram can be serially reusable or reentrant; it cannot be shared. A serially reusable subprogram can read and write into its own area, the calling action program, and the activation record. If the originating action program is reentrant, the subprogram cannot modify the calling program. Reentrant subprograms are executed as read-only and can modify only the activation record. They can modify other areas of the calling action program only if it is nonreentrant.

3.5.2. COBOL Action Program Interface

A COBOL action program calls a resident subprogram with the following sequence:

```
MOVE subprogram-name TO SUCCESSOR-ID.
ENTER LINKAGE.
CALL 'SUBPROG' [USING data-name-1...data-name-n].
ENTER COBOL.
```

where:

`data-name-1...data-name-n`
Refer to data items in the data division of the COBOL action program. No more than 12 data-names can be specified.

A subprogram written in COBOL returns control to the calling action program as follows:

```
ENTER LINKAGE
CALL 'RETURN'.
ENTER COBOL.
```

3.5.3. BAL Action Program Interface

A BAL action program or subprogram calls a resident subprogram via the following macroinstruction:

```
{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.

The calling action program must place the name of the called subprogram in the PIB at location ZA#PSID before issuing the ZG#CALL macro. The subprogram name must be left-justified and zero filled (X'F0') in a 6-byte area.

When control is transferred to the called subprogram, register 1 points to the specified parameter list. Other register contents are as follows:

- Register 13 points to a 72-byte save area supplied by the calling action program. The subprogram is responsible for saving the caller's registers, using standard save area linkages. If the subprogram requires working storage, the address of the working storage can be passed to the subprogram either in the parameter list or in a register.
- Register 14 contains the return address. A subprogram returns control to the calling program via the ZG#CALL RETURN macro.
- Register 15 contains the entry point address of the subprogram.
- Registers 2 through 12 have the same contents as the calling program.
- Register 0 is unpredictable.

3.6. ACTIVATION RECORD

The activation record is made available to the user action program after the program is loaded and given control by IMS 90. The activation record consists of the following areas. If present in a single-thread IMS 90 system, they appear in storage in the order listed:

- Program information block (PIB)
- Output message area (OMA)
- Input message area (IMA)
- Work area (WA)

- Continuity data area (CDA)
- Defined record area (DRA)

In the activation record layout for multithread IMS 90, the CDA and WA precede the IMA. (See Figures 3-16 and 3-17.)

Formats for PIB, OMA, and IMA headers are available to the COBOL user in the IMS 90 copy library under the names PIB, OMA, and IMA. For American National Standard 1974 COBOL users, these names are PIB74, OMA74, and IMA74. The BAL user accesses these formats by calling DSECTs as macros from the $Y$MAC system macro library or a user macro library.

The ZM#DPIB macro calls the ZA#DPIB DSECT, the ZM#DOMH macro calls the ZA#OMH DSECT, and the ZM#DIMH macro calls the ZA#IMH DSECT.

3.6.1. Program Information Block (PIB)

The program information block is always present in the activation record and is 136 bytes in length. It is used to communicate processing information between IMS 90 functions and the action program. The COBOL and BAL formats for the PIB are shown in Figures 3-8 and 3-9. The COBOL description of the PIB is stored in the IMS 90 copy library under the name PIB or under the name PIB74 for American National Standard 1974 COBOL. The ZM#DPIB macro generates the PIB DSECT for a BAL program. The shaded area in Figure 3-9 represents locations used only by IMS 90; a user-written BAL action program should never access these.

The COBOL format PIB fields listed in Figure 3-8 are described in 3.6.1.1 through 3.6.1.6.

```
01  PROGRAM-INFORMATION-BLOCK
    Ø2  STATUS-CODE                             PIC  9(4)   COMP-4.  ③
    02  DETAILED-STATUS-CODE                    PIC  9(4)   COMP-4.  ③
    Ø2  RECORD-TYPE REDEFINES DETAILED-STATUS-CODE.
        Ø3  PREDICTED-RECORD-TYPE               PIC  X.
        Ø3  DELIVERED-RECORD-TYPE               PIC  X.
    Ø2  SUCCESSOR-ID                            PIC  X(6).           ②
    Ø2  TERMINATION-INDICATOR                   PIC  X.              ① ②
    Ø2  LOCK-ROLLBACK-INDICATOR                 PIC  X.              ① ②
    Ø2  TRANSACTION-ID.                                              ①
        Ø3  YEAR                                PIC  9(4)   COMP-4.
        03  DAY                                 PIC  9(4)   COMP-4.  ④
        Ø3  TIME                                PIC  9(9)   COMP-4.  ⑤
    Ø2  DATA-DEF-REC-NAME                       PIC  X(7).           ① ②
    Ø2  DEFINED-FILE-NAME                       PIC  X(7).           ① ②
    Ø2  STANDARD-MSG-LINE-LENGTH                PIC  9(4)   COMP-4.  ①
    Ø2  STANDARD-MSG-NUMBER-LINES               PIC  9(4)   COMP-4.  ①
    Ø2  WORK-AREA-LENGTH                        PIC  9(4)   COMP-4.  ①
    Ø2  CONTINUITY-DATA-INPUT-LENGTH            PIC  9(4)   COMP-4.  ①
    Ø2  CONTINUITY-DATA-OUTPUT-LENGTH           PIC  9(4)   COMP-4.  ① ②
    Ø2  WORK-AREA-INC                           PIC  9(4)   COMP-4.  ②
    Ø2  CONTINUITY-DATA-AREA-INC                PIC  9(4)   COMP-4.  ②
    Ø2  SUCCESS-UNIT—ID.
        Ø3  TRANSACTION-DATE.
            Ø4  YEAR                            PIC  99.
            Ø4  MONTH                           PIC  99.
            Ø4  DAY                             PIC  99.
        Ø3  TIME-OF-DAY.
            Ø4  HOUR                            PIC  99.
            Ø4  MINUTE                          PIC  99.
            Ø4  SECOND                          PIC  99.
        Ø3  UNIQUE-SUFFIX                       PIC  999
    Ø2  SOURCE-TERMINAL-CHARS.
        Ø3  SOURCE-TERMINAL-TYPE                PIC  X.
        Ø3  SOURCE-TERM-MSG-LINE-LENGTH         PIC  9(4)   COMP-4.
        Ø3  SOURCE-TERM-MSG-NUMBER-LINES        PIC  9(4)   COMP-4.
```

NOTES:

① These fields are set by action scheduling at action initiation. All other PIB fields are set to 0 at initiation.

② These fields determine the termination procedures and may be set by the action program prior to termination mode multithread.

③ These fields also are used to return status information for I/O requests issued by the action program.

④ The name for this field in American National Standard 1974 COBOL is TODAY.

⑤ The name for this field in American National Standard 1974 COBOL is HR-MIN-SEC.

Figure 3—8. COBOL and American National Standard 1974 COBOL Format for Program Information Block

```
ZA#DPIB   DSECT PROGRAM-INFORMATION-BLOCK
ZA#PSC    DS    H                         STATUS-CODE
ZA#PDSC   DS    H                         DETAILED-STATUS-CODE
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'                      HOLDS LOCKS IND
ZA#PLRIR  EQU   C'R'                      RELEASE PENDING LOCKS IND
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
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    CLI                       TERMINAL TYPE
* * *     EQUATES  FOR  ZA#TTTYP
ZA#TFCC   EQU   C'F'                      FULL FIELD CONTROL CHARACTER DEVICE
ZA#TNON   EQU   C'V'                      VIDEO DEVICE WITHOUT FCC
ZA#TWS    EQU   C'W'                      WORKSTATION
ZA#TNOV   EQU   C'N'                      NON-VIDEO DEVICE
ZA#TMLL   DS    H                         TERMINAL-MSG-LINE-LENGTH
ZA#TMNL   DS    H                         TERMINAL-MSG-NUMBER-LINES
ZT#HSAAP  EQU   *                         ACTION PROGRAM SAVE AREA
ZA#PSAAP  EQU   *                         ACTION PROGRAM SAVE AREA
ZT#HSAIW  EQU   *+28                      CONT ACT PROGRAM AND INTERNAL
ZA#PSAVE  DS    18A                       SAVE AREA
*                                         ROS=FM ENTRY PT USED BY LNK MOD
          CNOP  0,8
ZA#PLGTH  EQU   *-ZA#DPIB
ZA#PLEN   EQU   *-ZA#DPIB                 PIB LENGTH
$SYSECT   CSECT
```

Figure 3—9. BAL Format for PIB (ZA#DPIB DSECT)

3.6.1.1. STATUS-CODE

STATUS-CODE is a half-word binary integer value returned by IMS 90 that indicates the completion status of a request. The status code values that can be returned are:

Description	Value
Successful	0
Invalid key or record number	1
End of file or unallocated optional file	2
Invalid request	3
I/O error	4
Violation of data definition	5
Internal message control error	6
Screen formatting error	7

In general, an invalid request status code is returned when an error is detected in a request by IMS 90 before the request is passed to OS/3 data management, control system, or ICAM. An I/O error status code is returned when an unrecoverable error is detected by data management, the control system, or ICAM.

An error return option can be specified for each action program at configuration time. If the option to accept errors is chosen (ERET=YES specified to the configurator), then, regardless of the value of the status code, control is returned to the action program after a function is completed. In this case, the action program must include tests for all status codes that are possible at the completion of the requested function. If the option to reject errors is chosen (or defaulted), then control is returned to the action program only if the status code equals 1 or 2. If any other status code is returned, control is not returned to the action program.

3.6.1.2. DETAILED-STATUS-CODE

DETAILED-STATUS-CODE is a half-word binary value returned by IMS 90 following a request when the status code is either an invalid request, I/O error, or internal message control error. The detailed status code supplements the status code by providing more detailed information concerning the error. When the status code is I/O error, the detailed status code contains either filenameC+2 or the error code and subcode returned by the access method. All file types except MIRAM files return a detailed status code of filenameC+2. MIRAM files return an error code (DM) and subcode. When the status code is invalid request (3) or internal message control error (6), the detailed status code is a binary integer. The possible values for the detailed status code for invalid request (status code 3) are listed in Table 3-8. Detailed status codes are interpreted differently for internal message control errors on explicit output message processing. (See 3.9.2.)

Table 3—8. Detailed Status Codes for Invalid Requests (Part 1 of 2)

Code	Description	Meaning
01_{16}	Incorrect number of parameters	The number of parameter addresses contained in a request parameter list is 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_{16}	Function code out of legal range	This error may occur when the IMS 90 link module that is linked to a serially reusable or sharable action program is inadvertently written into by the action program or control is improperly passed from an action program to IMS 90.
03_{16}	Incorrect parameter value	The parameter list address passed to IMS 90 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 DAMR file was not half-word aligned.
04_{16}	Shared record not in use by this transaction.	This code does not apply to user action program requests.
05_{16}	File not defined	A logical or defined file named in a request to IMS 90 has not been defined at configuration time or by means of the data definition processor.
06_{16}	File not open	A logical file named in a request to IMS 90 has been closed from the master terminal (ZZCLS) or a logical file has been closed by data management as the result of an unrecoverable error.
07_{16}	Function invalid for type of file	The function specified in a request to IMS 90 is not valid for the type of file named. For example, a SETL for a nonindexed file.

Table 3—8. Detailed Status Codes for Invalid Requests (Part 2 of 2)

Code	Description	Meaning
08_{16}	Record(s) not locked	An UNLOCK request has been made when no locks exist.
09_{16}	PUT or DELETE function not preceded by a GETUP function	The function sequence for an update operation is not valid.
$0A_{16}$	Illegal function requested	The requested function is not consistent with the DTF or RIB parameters in the configuration.
$0B_{16}$	File not assigned to this action	A logical or defined file named in a request to IMS 90 has not been named in the configured definition of the action making the request, or a defined file has not been dynamically named by the preceding action.
$0C_{16}$	Required module not included in configuration.	A request has been made that requires a module not included in the IMS 90 load module at configuration time.
$0D_{16}$	Capacity exceeded on INSERT function	A request has been made to insert a record into a MIRAM or ISAM file, but insufficient space exists to contain the new record.
$0E_{16}$	Insufficient space in main storage	User must allocate more main storage.
$0F_{16}$	Update not permitted in configuration	A request has been made to perform some update function, but update has been disallowed at configuration time.
10_{16}	Update suppressed for files	The requested update is not permitted because of an I/O error in AUDFILE, a file used by online recovery.
11_{16}	Trace file down	File recovery is not operational; only file displays are allowed.
12_{16}	Record has been locked by another transaction (for single-thread only)	Under single-thread, action program issued either a GETUP or INSERT function on a record, but this record was already locked by some other transaction.

The DETAILED-STATUS-CODE field has special uses under defined record management, described in 3.8.6.1. In the COBOL description of the PIB, DETAILED-STATUS-CODE is redefined as RECORD-TYPE for this purpose and the two record type bytes under the redefined STATUS-CODE are PREDICTED-RECORD-TYPE and DELIVERED-RECORD-TYPE.

- PREDICTED-RECORD-TYPE is the one-byte alphanumeric indicator that specifies to the defined record management the type record expected from a GET, GETUP, or INSERT function call. It can also specify the type of the next sequential record expected as a result of the SETL and GET function calls.

- DELIVERED-RECORD-TYPE is the one-byte alphanumeric indicator that specifies the type record actually returned by defined record management to the action program.

3.6.1.3. SUCCESSOR-ID

SUCCESSOR-ID is the identification of the action program activated after the termination of the current action program. When either external or delayed internal succession is invoked, SUCCESSOR-ID must contain the name of the successor action. The required value must be moved to this field by the current action program if a successor is desired. This is a 6-byte field and the identification must be left-justified and zero filled (i.e., X'FO'). No value need be specified on normal transaction termination. When the action program voluntarily chooses to terminate the transaction abnormally, a termination code is first moved to SUCCESSOR-ID. This termination code is useful in determining the cause of failure. If a code is specified, it is reported to the originating terminal operator (or console operator) after the abnormal termination occurs.

3.6.1.4. TERMINATION-INDICATOR

TERMINATION-INDICATOR is a 1-byte value, set by the current action program, that indicates the type of termination to take place for the program. The indicator has one of the following possible values:

N

Indicates normal transaction termination. When the current action program terminates, messages are output and all resources including the current action program are released. This is the default value for the indicator.

A

Indicates abnormal transaction termination. When the current action program terminates, IMS 90 creates and sends an error message to the originating terminal. All resources are released and files are rolled back, where applicable.

S

Indicates abnormal transaction termination with snap dump. This option is the same as the A option except that, in addition, a snap dump of the action program and its activation record is performed.

E

Indicates external succession. When the current action program terminates, messages are output, all resources including the current action program are released, and the succeeding action is scheduled when the next input message is received from the originating terminal.

I

Indicates immediate internal succession. When the current action program terminates, no message is output, the current action program is released, the succeeding action program is initiated, and all areas that were referenced by the terminating action program are passed to its successor. Note that indiscriminate use of I in a multithread environment can cause deadlock. Avoid terminating with an I when the GETUP and SETL functions are outstanding in the current action.

D

Indicates delayed internal succession. When the current action program terminates, no message is output, the output message is queued as input to the succeeding action, all resources (including the current action program) are released, and the succeeding action is initiated through normal scheduling procedures.

When you use delayed internal succession, be sure to allocate adequate space for the input staging buffer via the INBUFSIZ parameter in the configurator GENERAL section. If insufficient buffer space is available for the successor program, the transaction terminates with a status code of 6 (internal message control error) and a detailed status code of 8 (record not locked).

A summary of the types of termination by an action program is included in Table 3-9.

Table 3—9. Summary of Action Program Termination Types

	Type of Termination					
Program Information Block Data Items	**Normal Transaction Termination**	**Abnormal Transaction Termination**	**Abnormal Transaction Termination With Snap Dump**	**Action Termination With External Successor**	**Action Termination With Immediate Internal Successor**	**Action Termination With Delayed Internal Successor**
SUCCESSOR-ID	Ignored	Termination-code	Termination code	Action program name	Action program name	Action program name
TERMINATION-INDICATOR	N	A	S	E	I	D

The TERMINATION-INDICATOR is used to control voluntary action program termination; i.e., termination that occurs after the execution of the CALL 'RETURN' statement. An action program also can terminate involuntarily. Involuntary termination occurs when an abnormal condition is encountered by IMS 90 in the processing of a request issued by an action program. (See description of STATUS-CODE, 3.6.1.1). Involuntary termination also can occur when the execution of an action program causes a program check or when an execution loop within an action program continues beyond a specified time limit. When either of these conditions occurs, the standard abnormal termination message is sent to the originating user terminal and to the system console. In addition, a snap dump of the action program and its activation record is performed. (Refer to 3.12 for a description of the snap dump.)

3.6.1.5. LOCK-ROLLBACK-INDICATOR

LOCK-ROLLBACK-INDICATOR is a 1-byte value, set by the action program, that indicates to action scheduling the record lock and rollback functions that are to be performed at action termination. The indicator is set to one of the following values:

H

Causes all record locks that have been imposed in the action to be held into the subsequent action.

N

Establishes a new rollback point in the audit file for this transaction and releases all locks for this transaction. This is the default value.

O

Causes the rollback of all updates performed by this transaction to the previous rollback point, releases all locks for this transaction, and establishes a new rollback point in the audit file for this transaction.

R

Causes the release of all locks for pending updates that have been imposed in the action. A record lock is pending if GETUP is done for it but no PUT function is issued for the record.

R and H options are effective only when the termination indicator is set to E (external) or D (delayed internal). In long transactions, R and H options should be used with caution. Holding of locks across many actions in a multithread environment can cause deadlocks.

The N option is useful for long-running update transactions. The N option releases all locks when the termination indicator is set to E (external) or D (delayed internal). It also is used to establish additional rollback points, to limit the range of rollback, and to cut down the size of the audit file. By limiting the number of updates in an action or by establishing additional rollback points in a long-running transaction, you can reduce the size of the audit file and thus save disk space.

The O option is effective for N (normal), as well as E and D, and causes online file recovery to be activated to carry out the rollback. In the case of N (normal transaction termination), a new rollback point is established in the audit file.

3.6.1.6. Additional PIB Fields

The remaining COBOL format fields of the PIB contain the following information:

- TRANSACTION-ID is the date-time stamp for the first input message of a transaction. It is set by action scheduling for the use of all action programs that are activated during the processing of a transaction. The date is given in Julian form. The time is in milliseconds. The TRANSACTION-DATE and TIME-OF-DAY under SUCCESS-UNIT-ID should be used by the action program when accurate date and time of day are required.

- DATA-DEF-REC-NAME is the name of the data definition record (in the named record file) that contains the description of the defined file designated by DEFINED-FILE-NAME. This is a 7-byte item and the name must be left-justified and blank filled.

  When an action is scheduled, IMS 90 moves the record name specified by the DDRECORD parameter in the configurator ACTION section into this field and the file name specified by the DFILE parameter into the DEFINED-FILE-NAME field, unless values were left in these fields by the preceding action. If the action terminates in delayed internal or external succession and the successor action accesses a different defined file, the action program can either move the record name and file name for the successor action into these fields or can zero out both fields and let IMS 90 insert the values specified to the configurator for the successor action. If the successor action accesses only logical files, zeros also should be placed in both PIB fields. This allows the successor action to access logical records that have contributed to the defined file used by the previous action.

- DEFINED-FILE-NAME is the name of a defined file that is to be accessed. DEFINED-FILE-NAME is a 7-byte item and the name must be left-justified and blank filled. Its use is described under DATA-DEF-REC-NAME.

- STANDARD-MSG-LINE-LENGTH is a half-word binary integer set by action scheduling that specifies the configuration-defined maximum line length for a message.

- STANDARD-MSG-NUMBER-LINES is a half-word binary integer set by action scheduling that specifies the configuration-defined maximum number of lines for a message.

- WORK-AREA-LENGTH is a half-word binary integer set by action scheduling that specifies the length of the work area allocated to the action.

- CONTINUITY-DATA-INPUT-LENGTH is a half-word binary integer set by action scheduling that specifies the length of the continuity data record that was passed to this action by its predecessor. The continuity data record, if present, begins with the first byte of the continuity data area.

- CONTINUITY-DATA-OUTPUT-LENGTH is a half-word binary integer set by action scheduling that specifies the length of the continuity data area allocated to the action. This value determines the number of bytes in the continuity data area, starting with the first byte, that are to be saved when termination with external succession or delayed internal succession is requested. The value is changed in the current action prior to termination to reduce the size of the continuity data record that is saved.

- WORK-AREA-INC is a half-word binary integer set in the current action program. This value specifies the number of bytes in addition to the value specified in the action control table at configuration time that should be allocated for the successor of the current action. This is implemented in multithread only.

- CONTINUITY-DATA-AREA-INC is a half-word binary integer set in the current action and is used to increment the length of the continuity data area. This increment value is added to the length of the saved continuity data record and compared to the length specified in the action control table to determine the larger, which becomes the size of the continuity data area for the succeeding action.

- SUCCESS-UNIT-ID provides a data and time stamp to the user program at the beginning of each success unit. A success-unit is defined as an action.

- SOURCE-TERMINAL-TYPE is a 1-byte field that contains a type code for the source terminal. The values set by IMS 90 are:

Value	Description
C'F'	Full field control character (FCC) device (i.e., UTS 400)
C'V'	Video device without FCC (i.e., U100, U200)
C'W'	Workstation
C'N'	Nonvideo device (i.e., teletypewriter, batch terminal)

- SOURCE-TERM-MSG-LINE-LENGTH is a half-word binary integer that specifies the number of characters per line for the source terminal. For a nonvideo device, however, this value is the configured line length (CHRS/LIN specification in the GENERAL section of the IMS 90 configuration).

- SOURCE-TERM-MSG-NUMBER-LINES is a half-word binary integer that specifies the number of lines for the source terminal. For a nonvideo device, however, this value is the configured number of lines (LNS/MSG specification in the GENERAL section of the IMS 90 configuration).

3.6.2. Output Message Area (OMA)

The output message area is optional and has a length as specified by the OUTSIZE keyword parameter in the configurator ACTION section. The output message area is divided into a control header and an area for the message text. The message text portion is set to blanks at initiation. If an output message area is selected for an action and the action terminates normally, the message in the area is sent to the originating terminal unless otherwise specified. Figures 3–10 and 3–11 show the predefined COBOL and BAL formats for the OMA control header. The COBOL description of the OMA control header is available in the copy library under the name OMA or under the name OMA74 for American National Standard 1974 COBOL. Execution of the macro ZM#DOMH in a BAL action program generates the OMA DSECT.

DESTINATION-TERMINAL-ID is the identifier of the terminal to which the output message is to be sent. If no value is moved to this item prior to the termination of the action, the source terminal is assumed to be the destination terminal. The identifier must be left-justified and blank filled. The four bytes (FILLER) following this field are reserved for system use.

SFS-OPTIONS is used to specify information about the screen format that is being used. If the first byte of the field is set to the character I, this format is to be used for the following input. Any other value in this field means this format is not to be used for the following input.

CONTINUOUS-OUTPUT-CODE is used to identify a continuous output message, when continuous output is being generated. The continuous output option is described in 3.10.

TEXT-LENGTH is a binary half-word integer that specifies the length of the output message text. The value specified in TEXT-LENGTH must include the length of the actual text plus four bytes for the length field. This value is set to the predefined output message text length by action scheduling at action initiation and is reduced by the action program. The output message text length is specified in the configuration definition of an action. If the value is set to zero and no explicit output message has been sent by the action program, a default termination message is sent to the source terminal.

AUXILIARY-DEVICE-ID is used to specify whether the output message is to be sent to an auxiliary device and, if so, to identify the device. This field also can be used to specify additional printing options. If the output message is not to be sent to an auxiliary device, set the entire field to binary 0's; this is the original value of the field, set by IMS 90 when it generates the OMA header. For example, in a COBOL action program

```
MOVE LOW-VALUES TO AUXILIARY-DEVICE-ID.
```

states that the output message is not to be sent to an auxiliary device, but is to be displayed or listed on the primary device - the destination terminal with no special options.

On the other hand, if the output message is to be listed on an auxiliary device attached to the destination terminal, you must use each byte of the AUXILIARY-DEVICE-ID field.

```
Ø1  OUTPUT-MESSAGE-AREA.
    Ø2 DESTINATION-TERMINAL-ID      PIC X(4).
    02 SFS-OPTIONS                  PIC X(2).
    Ø2 FILLER                       PIC X(2).
    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.
```

Figure 3—10. COBOL and American National Standard 1974 COBOL Format for OMA Control Header

```
ZA#OMH   DSECT
*
*  OUTPUT MESSAGE HEADER
*
ZA#ODTID DS    CL4                    DESTINATION TERMINAL ID
ZA#OSFSO DS    CL2                    SFS OPTIONS
*
*   EQUATES FOR ZA#OSFSO
*
ZA#OSFSI EQU   C'I'                   INPUT FORMAT
*
         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
```

Figure 3—11. BAL Format for OMA Control Header (ZA#OMH DSECT) (Part 1 of 2)

```
*         SS: SPACE SUPPRESSION        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)
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 3—11. BAL Format for OMA Control Header (ZA#OMH DSECT) (Part 2 of 2)

To list the output message on the COP or TP in *print mode,* for example, set the AUX-FUNCTION byte of this field to X'FO'; to list it in *print-transparent mode,* set the AUX-FUNCTION byte to X'F4'. You must also set, in the AUX-DEVICE-NO byte of the same field, a character in the range 1–9 (that is, X'F1' through X'F9') to identify the auxiliary device. This number corresponds to the logical device number appended to the AUX keyword parameter of the TERM declarative macro in your ICAM network definition. (See the IMS 90 system support functions user guide/programmer reference, UP-8364 (current version).) For example, in a COBOL action program, the statements

```
MOVE '4' TO AUX-FUNCTION.
MOVE '1' TO AUX-DEVICE.
```

cause the output message to be sent to an auxiliary device at the primary destination terminal and listed in print-transparent mode; the device specified is the first auxiliary device configured at that terminal. Print mode, print-transparent mode, space suppression, and other print options are further described in 3.10.1.1 (in the context of printing continuous output). Table 3–16 summarizes the settings of the AUX-FUNCTION byte.

3.6.3. Input Message Area (IMA)

The input message area is required for all actions. Its size is specified at configuration with the INSIZE keyword parameter in the ACTION section. When no length is specified for the input message area at configuration time or the length specified is inadequate, an area large enough to contain the actual input message is allocated by action scheduling. This area contains the input message when the first and subsequent action programs of an action are activated. If the input entered from the terminal is a function key (5.1.5), IMS 90 creates a message in the format:

```
F#nn
```

This message is preceded by the DICE sequence:

```
10010101
```

This DICE sequence is present even if you specified DICE=OFF in your ICAM network definition. However, if you have configured input message editing (any specification except NO on the EDIT parameter in the ACTION section), the DICE sequence is stripped from the message before it is sent to your program.

The area also contains information related to the message in a control header. When the input message area is larger than the actual input message plus the control header, the balance of the area is blank filled. The COBOL and BAL formats for the IMA control header are shown in Figures 3–12 and 3–13. The COBOL description of the IMA header is available in the copy library under the name IMA or under the name IMA74 for American National Standard 1974 COBOL. Execution of the ZM#DIMH macro in a BAL action program generates the DSECT for the IMA header.

```
        Ø1   INPUT-MESSAGE-AREA.
             02 SOURCE-TERMINAL-ID          PIC X(4).
             Ø2 DATE-TIME—STAMP.
                03 YEAR                     PIC 9(4) COMP-4.
                Ø3 DAY                      PIC 9(4) COMP-4.  ①
                Ø3 TIME                     PIC 9(9) COMP-4.  ②
             02 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 American National Standard 1974 COBOL is TODAY.

② The name of this field in American National Standard 1974 COBOL is HR-MIN-SEC.

Figure 3—12. COBOL and American National Standard 1974 COBOL Format for IMA Control Header

```
ZA#IMH   DSECT
*
*  INPUT MESSAGE HEADER
*
ZA#ISTID DS    CL4              SOURCE TERMINAL ID
ZA#IDTS  DS    CL8              DATE/TIME STAMP
ZA#IMHL  EQU   *—ZA#IMH         INPUT MESSAGE AREA HEADER LENGTH
ZA#ITL   DS    H                TEXT LENGTH
ZA#IDEV  DS    CL2              AUXILIARY-DEVICE-ID
*
*   EQUATES FOR ZA#IDEV+1
*
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
```

Figure 3—13. BAL Format for IMA Control Header (ZA#IMH DSECT)

The IMA control header contains the following items:

- SOURCE-TERMINAL-ID is the identifier of the terminal that originated the input message.
- DATE-TIME-STAMP is the value of the date and time at which the input message was made available to internal message control. The date-time is provided in binary. The first half word of the field contains the year; the second half word contains the Julian day. The second word contains a sequence number unique to this input message and not usable for determining the time of day.
- TEXT-LENGTH is a binary half-word integer that specifies the length of the input message text. The two bytes (FILLER) following the TEXT-LENGTH field are reserved for system use. The value specified in TEXT-LENGTH includes the length of the actual text plus four bytes for the length field.
- AUXILIARY-DEVICE-ID is the number of the auxiliary device from which data is transmitted to the action program.

3.6.4. Work Area (WA)

The work area is optional for most programs; however, it is generally used by sharable or reentrant action programs requesting file I/O functions. The work area is used as modifiable working storage for the input and output of defined records and logical data records via IMS 90 file management. The work area also is used for the building of output messages that are explicitly sent via internal message control. The length of the area is equal to the length specified in the action control table at configuration time plus the length specified as WORK-AREA-INC in the PIB by the preceding action. WORK-AREA-INC is not supported in single-thread IMS 90. If the WORKSIZE parameter is not specified in the configurator ACTION section, no work area is generated. The work area, if generated, is set to binary 0's at initiation.

3.6.5. Continuity Data Area (CDA)

The continuity data area is optional. It is used to contain data that is passed from action to action in a dialog transaction.

If a fixed-length continuity data area is required in a dialog transaction, the length of the area is specified in the action control table record for the first action by means of the CDASIZE parameter in the configurator ACTION section. The record is then passed from action to action by action scheduling without further intervention by the action programs.

It is possible to vary the length of the continuity data area from action to action or to eliminate the area when the succession is made from one action to another. These changes are accomplished by varying the parameters that determine CDA length.

The length of the area is equal to the larger of:

- the length specified in the action control table at configuration time for the action being scheduled; or

- the length specified as CONTINUITY-DATA-INC in the PIB by the preceding action plus the actual length of the continuity data record saved at the termination of the preceding action.

The actual length of the record saved is specified in CONTINUITY-DATA-OUTPUT-LENGTH in the PIB of the preceding action. The same value is made available to the succeeding action in the CONTINUITY-DATA-INPUT-LENGTH entry of the PIB. Either the CONTINUITY-DATA-INC specified by the preceding action or the continuity data area length specified in the action control table, or both, is zero. If the value in CONTINUITY-DATA-OUTPUT-LENGTH is zero when an action terminates, then no record is saved for the succeeding action in the continuity data file.

The continuity data area is set to binary 0's at action initiation. The continuity data record, if any, is then read into the area as its starting location.

3.6.6. Defined Record Area (DRA)

The defined record area is used by defined record management. A DRA is generated if the DDRECORD parameter is specified in the ACTION section of the configuration for this user action or if DEFINED-FILE-NAME in the PIB contained a defined file name when the action that named the current action as its successor terminated. The DRA is approximately 400 bytes long plus four times the maximum logical record size plus two times the record key size.

The defined record area is not specified in the action program and cannot be written into by the action program.

3.7. LINK EDITING ACTION PROGRAMS

After you obtain a clean action program compilation, you must link it to the IMS 90 link module, ZF#LINK, in one job using the LINK jproc in the following format:

```
// LINK action-prog-name, OUT={(vol-ser-no,label)}
                              {(RES,$Y$LOD)      }
```

This LINK jproc can be used only when the action program is compiled with the PARAM OUT=(M) statement or the // PARAM IMSCOB=YES statement. This procedure produces a load module and places it in the IMS 90 load library for later reference. This library must be the same one specified on the LIBL parameter of the IMSCONF jproc at configuration. (The default value for the LIBL parameter is $Y$LOD.)

If an action program is not compiled with either of the shared code PARAM statements, an ENTER statement naming the program to be linked must be included in the link stream.

ZF#LINK has a defined entry point for each of the IMS 90 functions that an action program can request. During execution, when an action program calls an IMS 90 function, control passes to ZF#LINK. ZF#LINK, in turn, passes control to IMS 90, which executes the function. After completion of the requested service, control returns from the IMS 90 module directly to the action program at the return point designated in the function call.

3.8. FILE PROCESSING

IMS 90 file management makes records available to action programs for processing. In turn, user-written action programs issue file I/O functions to retrieve, insert, update, or delete records. IMS 90 file management supports sequential, relative, and indexed files. In addition, it supports defined files in an indexed file organization via defined record management.

An IMS 90 action program can issue one or more I/O function calls: GETUP, GET, PUT, DELETE, INSERT, SETL, and ESETL. These calls are processed by SAM, DAM, ISAM, or MIRAM access methods. (To access IRAM files, you must define them as MIRAM files at configuration time.) Table 3-10 summarizes the files supported by IMS 90.

Table 3—10. Summary of Files Supported by IMS 90 File Management

File Organization	Access Mode	Data Management Access Method	Functions Available Through IMS 90 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.

An action program may issue random or sequential I/O functions to indexed and relative files but only sequential I/O functions to sequential files. Table 3-11 lists the file I/O functions allowed with each file organization.

Table 3—11. Summary of File I/O Function CALL Statements

File Organization	Random Functions	Sequential Functions
Sequential		GET USING file-name record-area. PUT USING file-name record-area.
Relative (nonindexed)	GET USING file-name record-area record-number. GETUP USING file-name record-area record-number. PUT USING file-name record-area [record-number].[2] INSERT USING file-name record-area record-number. DELETE USING file-name record-area record-number.	SETL USING file-name position [record-number].[1] GET USING file-name record-area. ESETL USING file-name.
Indexed	GET USING file-name record-area key. GETUP USING file-name record-area key. PUT USING file-name record-area. INSERT USING file-name record-area. DELETE USING file-name record-area.	SETL USING file-name positon [key[partial-key-count]].[1] GET USING file-name record-area. ESETL USING file-name.
Indexed (Defined File)	GET USING file-name record-area key. GETUP USING file-name record-area key. PUT USING file-name record-area. INSERT USING file-name record-area key. DELETE USING file-name record-area.	SETL USING file-name position [key]. GET USING file-name record-area. ESETL USING file-name.

NOTES:

[1] Sequential functions available with MIRAM, not DAM

[2] Required for DAM relative files

3.8.1. Formats and Rules for File I/O Functions

The four following general rules apply to file I/O functions:

1. In an action program coded in extended COBOL, function requests are preceded and followed by ENTER statements:

```
ENTER LINKAGE.
CALL 'GET' USING file-name record-area key.
ENTER COBOL.
```

   Action programs coded in 1974 American National Standard COBOL do not use ENTER statements.

2. Function requests made in BAL action programs use either the CALL or the ZG#CALL macro. (Refer to 3.4.2 for details.) An example of a BAL function request that is equivalent to the preceding COBOL function call is:

```
CALL GET,(file-name,record-area,key)
```

3. The completion status of a function request is set in the program information block. Values for the status codes are listed in the description for each function. Values of detailed status codes are listed in Table 3-8.

4. The positional parameters *file-name, record-area, record-number, key,* and *position* refer to data names in the data division of a COBOL action program or labels of storage locations in a BAL action program. The locations they represent contain the following values at the time the I/O function is performed:

   - *File-name* contains the 7-character name of the file on which the specified function is to be performed. This name must be left-justified and blank filled.

   - *Record-area* is the area to or from which a logical or defined record is to be moved by file management. The size of the record area must be equal to or greater than the size of the largest logical record it is to contain. In the case of a variable-length record, the record begins with the length field. COBOL programmers do not usually have to be aware of the length field, but they must include a description of this field in their description of a variable-length record when writing an action program.

   - *Record-number* is an 8-byte field containing a right-justified binary number identifying a record in the file. The number designates the position of the record relative to the beginning of the file. The first number is 1.

   - *Key* contains the key value used to identify a record to be retrieved from a file or inserted into a file.

   - *Position* contains a 1-byte value designating the position of the file at the completion of the SETL function. Values are listed in the description of the SETL function.

3.8.2. Indexed Files

The indexed sequential, indexed random, and multiple indexed random access methods (ISAM, IRAM, 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 90 supports multiple key MIRAM files, you must use only the key identified in the configurator FILE section (KEYn parameter) to retrieve or update records. No duplicate keys are allowed on MIRAM files. Changes to alternate keys are allowed.

3.8.2.1. Random Functions for Indexed Files

The random function calls GET, GETUP, PUT, INSERT, and DELETE retrieve records with or without updating, write records back to a file, logically delete records, and overwrite an existing record or add a new record to a file.

3.8.2.1.1. GET and GETUP Functions

The random GET function only retrieves the record. The GETUP function retrieves the record for updating and temporarily locks the requested record from access by other transactions. Random GET and GETUP functions are not performed if the requested record is currently locked by a different transaction. File keys supplied as parameters in a GET or GETUP function locate the required record in a logical file. Any key required for a function need only be as long as the key field itself.

A GET function must not be followed by a PUT or DELETE function; however, 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. A value of X'FF' in the first byte of data in the logical record indicates the logical deletion of that record. When a transaction requests a record with X'FF' in the first byte, an invalid key status code is returned. To be effective, PUT or DELETE functions must immediately follow the GETUP function. If other functions intervene, the record must be retrieved again with a GETUP function before a PUT or DELETE can be performed. The key field must remain unaltered until the PUT or DELETE function is completed for ISAM files or until after the GETUP function is completed for MIRAM files. No key parameter is given on the PUT or DELETE functions issued to indexed records because the key was already given in the preceding GETUP function.

COBOL and BAL formats for the random GET and GETUP function calls and resulting status codes follow:

COBOL Format:

```
CALL{'GET'  }USING file-name record-area key.
    {'GETUP'}
```

BAL Format:

```
{CALL   }{GET  },(file-name,record-area,key)
{ZG#CALL}{GETUP}
```

Status Codes:

0 Successful
1 Invalid key
2 Unallocated optional file*
3 Invalid request
4 I/O error

**This code applies to MIRAM files only.*

3.8.2.1.2. PUT Function

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 nonkeyed data in a record must not be an X'FF'. Keys are not supplied on PUT functions issued to indexed files.

COBOL and BAL formats for the PUT function call and resulting status codes follow:

COBOL Format:

```
CALL 'PUT' USING file-name record-area.
```

BAL Format:

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

Status Codes:

0	Successful
1	Not used
2	Unallocated optional file*
3	Invalid request
4	I/O error

3.8.2.1.3. INSERT Function

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'. Any INSERT function using a previously deleted record slot not only removes the delete control character but also changes the record length field for variable-length records. Changing the length of a variable-length record is permitted for MIRAM files only. It is not allowed for ISAM files. 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 that already exist in the file. No additional 3 bytes of work space are required following the embedded key.

COBOL and BAL formats for the random GET and GETUP function calls and resulting status codes follow:

COBOL Format:

```
CALL 'INSERT' USING file-name record-area:
```

This code applies to MIRAM files only.

BAL Format:

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

Status Codes:

0	Successful
1	Not used
2	Unallocated optional file*
3	Invalid request
4	I/O error

3.8.2.1.4. DELETE Function

The DELETE function logically deletes a record that was retrieved for updating. This function must be immediately preceded by a GETUP function for a record to be deleted. If other functions intervene between the GETUP and DELETE functions, the GETUP function must be reissued before the record can be DELETED. The DELETE function changes the first byte of nonkey data in a record retrieved for update to X'FF' before the record is written to the file. The DELETE function is invalid if the first byte of nonkey data is part of an alternate key.

COBOL and BAL formats for the DELETE function call and resulting status codes follow:

COBOL Format:

```
CALL.'DELETE'.USING.file-name.record-area:
```

BAL Format:

```
{CALL...}DELETE,(file-name,record-area)
{ZG#CALL}
```

Status Codes:

0	Successful
1	Not used
2	Unallocated optional file*
3	Invalid request
4	I/O error

**This code applies to MIRAM files only.*

3.8.2.2. Sequential Functions for Indexed Files

Sequential function calls SETL, GET, and ESETL 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.

When you access an indexed file sequentially, your action program 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.

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 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.

3.8.2.2.1. SETL Function

The SETL function sets an indexed file into sequential mode and logically positions the file according to the following 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

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

You must supply a file name and choose a position value on the SETL function for indexed files. Depending upon the position chosen, you may optionally supply a key parameter. In addition, the SETL function allows for partial key search of indexed MIRAM files. To do this, you supply the optional *partial key count* parameter. The *partial key count* parameter is the symbolic address of a one-word field containing a right-justified binary number. This binary number indicates the number of key argument leading bytes used for the positioning search. Table 3-12 explains the SETL parameter choices for indexed files.

Table 3—12. SETL Parameter Choices for Indexed Files

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

NOTES:

1. The letter X means applicable to this file type.
2. Key and partial key count are not used with position value B.

The COBOL and BAL formats for the SETL function call and resulting status codes follow:

COBOL Format:

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

BAL Format:

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

Status Codes:

0	Successful
1	Invalid key
2	Not used
3	Invalid request
4	I/O error

3.8.2.2.2. GET Function

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 then retrieves the following record. *File-name* and *record-area* parameters are required on sequential GET functions for indexed files. The COBOL and BAL formats for the sequential GET function call and resulting status codes follow:

COBOL Format:

```
CALL 'GET' USING file-name record-area.
```

BAL Format:

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

Status Codes:

0	Successful
1	Not used
2	End of file or unallocated optional file
3	Invalid request
4	I/O error

3.8.2.2.3. ESETL Function

The ESETL function changes the mode of indexed files from sequential back 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 90 file management resets the file to random mode. The ESETL function always requires a *file-name* parameter. The COBOL and BAL formats for the ESETL function call and resulting status codes follow:

COBOL Format:

```
CALL 'ESETL' USING file-name.
```

BAL Format:

```
{CALL   }ESETL,(file-name)
{ZG#CALL}
```

Status Codes:

0	Successful
1 and 2	Not used
3	Invalid request
4	I/O error

3.8.3. Relative Files

The direct, indexed random, and multiple indexed random access methods (DAM, IRAM, 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 always required on sequential functions to relative files.

3.8.3.1. Random Functions for Relative Files

The random function calls GET, GETUP, PUT, INSERT, and DELETE retrieve records with or without updating, write records back to a file, logically delete records, and overwrite an existing record or add a new record to a file.

A DAM file must be preformatted offline before its initial use and must contain the maximum number of physical records to be referenced online under file management. For processing IRAM files, all volumes in a file must be mounted online.

3.8.3.1.1. GET and GETUP Functions

The random GET function only retrieves the record. The GETUP function retrieves the record for updating and temporarily locks the requested record from access by other transactions. Random GET and GETUP functions are not performed if the requested record is currently locked by a different transaction. When issued to relative files, the GET and GETUP functions must supply a record number. File relative record numbers supplied as parameters in a GET or GETUP function locate the required record in a logical file. All record number fields must be eight bytes long.

A GET function must not be followed by a PUT or DELETE function; however, 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. A value of X'FF' in the first byte of data in the logical record indicates the logical deletion of that record. When a transaction requests a record with X'FF' in the first byte, an invalid record number status code is returned.

COBOL and BAL formats for the random GET and GETUP function calls and resulting status codes follow:

COBOL Format:

```
CALL{'GET'  }USING file-name record-area record-number.
    {'GETUP'}
```

BAL Format:

```
{CALL   }{GET  },(file-name,record-area,record-number)
{ZG#CALL}{GETUP}
```

Status Codes:

0	Successful
1	Invalid record number
2	Unallocated optional file*
3	Invalid request
4	I/O error

**This code applies to MIRAM files only.*

3.8.3.1.2. PUT Function

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 nonkeyed data in a record must not be an X'FF'. A *record-number* parameter is required on the PUT function for DAM files, but is optional for other relative files. Nevertheless, if you omit *record-number* for MIRAM files, you must place the GETUP function immediately before the PUT function.

COBOL and BAL formats for the PUT function call and resulting status codes follow:

COBOL Format:

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

BAL Format:

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

Status Codes:

0	Successful
1	Invalid record number
2	Unallocated optional file*
3	Invalid request
4	I/O error

3.8.3.1.3. INSERT Function

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'. For MIRAM files only, any INSERT function using a previously deleted record slot not only removes the delete control character but also changes the record length field for variable-length records.

INSERT functions issued to a relative file must supply a *record-number* parameter. If you specify RCB=NO at configuration, 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.

COBOL Format:

```
CALL 'INSERT' USING file-name record-area record-number.
```

**This code applies to MIRAM files only.*

BAL Format:

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

Status Codes:

0 Successful
1 Invalid record number
2 Unallocated optional file*
3 Invalid request
4 I/O error

3.8.3.1.4. DELETE Function

The DELETE function logically deletes a record that was retrieved for updating. This function must be immediately preceded by a GETUP function for a record to be deleted. If other functions intervene between the GETUP and DELETE functions, the GETUP function must be reissued before the record can be DELETED.

You must supply a *record-number* parameter on the DELETE function for DAM files; however, *record-number* parameters are optional on DELETE functions for other relative files. The DELETE function changes the first byte of nonkey data in a record retrieved for update to X'FF' before the record is written to the file.

COBOL and BAL formats for the DELETE function call and resulting status codes follow:

COBOL Format:

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

BAL Format:

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

Status Codes:

0 Successful
1 Invalid record number
2 Unallocated optional file*
3 Invalid request
4 I/O error

This code applies to MIRAM files only.

3.8.3.2. Sequential Functions for Relative Files

Sequential function calls SETL, GET, and ESETL set nonindexed IRAM and MIRAM relative files into sequential mode and position them to a selected location in the files, retrieve records sequentially, and reset these relative files from sequential mode to random mode. Sequential functions cannot be processed by the direct access method (DAM).

When you access an indexed file sequentially, your action program 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.

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 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.

3.8.3.2.1. SETL Function

The SETL function sets a relative file into sequential mode and logically positions the file according to the following 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

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

You must supply a file name and choose a position value on the SETL function for relative files. Depending upon the position chosen, you may optionally supply a *record-number* parameter. Table 3-13 explains the SETL parameter choices for relative files. The SETL function is not available for DAM files.

Table 3—13. SETL Parameter Choices for Relative Files

File Type	Parameters					
	File Name	Position				
		B	G	K	H	Record Number
Nonindexed IRAM	X	X	X			X
Nonindexed MIRAM	X	X	X	X	X	X

NOTES:

1. The letter X means applicable to this file type.
2. Record number is not used with position value B.

The COBOL and BAL formats for the SETL function call and resulting status codes follow:

COBOL Format:

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

BAL Format:

```
{CALL   }SETL,(file-name,position[,record-number]
{ZG#CALL}
```

Status Codes:

0	Successful
1	Record number
2	Not used
3	Invalid request
4	I/O error

3.8.3.2.2. GET Function

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 then retrieves the following record.

File-name and *record-area* parameters are required on sequential GET functions for relative files. The COBOL and BAL formats for the sequential GET function call and resulting status codes follow:

COBOL Format:

```
CALL 'GET' USING file-name record-area.
```

BAL Format:

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

Status Codes:

0	Successful
1	Not used
2	End of file or unallocated optional file
3	Invalid request
4	I/O error

3.8.3.2.3. ESETL Function

The ESETL function changes the mode of relative files from sequential back 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 90 file management resets the file to random mode. The ESETL function always requires a *file-name* parameter. The COBOL and BAL formats for the ESETL function call and resulting status codes follow:

COBOL Format:

```
CALL 'ESETL' USING file-name.
```

BAL Format:

```
{CALL   }ESETL,(file-name)
{ZG#CALL}
```

Status Codes:

0	Successful
1 and 2	Not used
3	Invalid request
4	I/O error

3.8.4. Sequential Files

The sequential and dedicated multiple indexed random access methods (SAM and dedicated MIRAM) are the only access methods that can process function calls issued by your action program to sequential files. Only two functions, the sequential GET and PUT, can be issued to sequential files. The same SAM or dedicated sequential MIRAM file cannot be used for both input and output. (It is defined in the configurator FILE section as an input file or an output file.) Files used for input may only be accessed by the sequential GET function. For output files, only the sequential PUT function may be used.

3.8.4.1. Sequential Input GET Function

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 then retrieves the following record. The first record of a sequential file retrieved in an IMS 90 session is always the first record of the file. No record is ever read twice in a sequential file.

File-name and *record-area* parameters are required on the GET function for sequential files. The COBOL and BAL formats for the sequential GET function call and resulting status codes follow:

COBOL Format:

```
CALL 'GET' USING file-name record-area.
```

BAL Format:

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

Status Codes:

0 Successful
1 Not used
2 End of file (DAM files only) or unallocated optional file (MIRAM files only)
3 Invalid request
4 I/O error

3.8.4.2. Sequential Output PUT Function

The sequential PUT function writes fixed- or variable-length logical records to sequential files on tape or disk. *File-name* and *record-area* parameters are always required on this function. Standard labeled tapes are assumed prepped or file space allocated for disk files before you issue a sequential PUT function.

The COBOL and BAL formats for the sequential PUT function call and resulting status codes follow:

COBOL Format:

```
CALL 'PUT' USING file-name record-area.
```

BAL Format:

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

Status Codes:

0 Successful
1 Not used
2 Unallocated optional file*
3 Invalid request
4 I/O error

3.8.5. Defined Record Management

Defined record management is the part of IMS 90 file management that services requests from action programs to retrieve and update the records of defined files. An action program can call upon the random access functions INSERT, GET, GETUP, PUT, and DELETE and also the sequential access functions SETL, GET, and ESETL. In response, IMS 90 places defined records into and takes them from the record area you name in the I/O function call.

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 that is 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 3.6.1.6.)

During a given action, a transaction can access only one defined file but can also access ISAM, SAM, DAM, IRAM, or MIRAM files if they are not referenced by the defined file. You must access standard files by using the I/O function calls pertaining to them. (See 3.8.2, 3.8.3, and 3.8.4.).

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

The following are parameter definitions for defined record management I/O function calls:

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

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

  B

  Retrieves the first record of the file.

  G

  Retrieves the first record whose identifier (key) is equal to or greater than that specified by the *key* parameter.

*This code applies to MIRAM files only.

H

Retrieves the first record whose identifier is greater than that specified by the *key* parameter.

- *Key* is a data name or storage location that contains the identifier of a defined record. The identifier follows the rules specified in the data definition for the defined file *filename*. An identifier consists of one or more identifier 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.

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

3.8.5.1. Defined Record Management Returns to Action Program

In response to a function call, defined record management determines the type of defined record (as specified in the TYPE statement of the data definition) involved in the call and returns the record type to the action program in the program information block.

PREDICTED-RECORD-TYPE	DELIVERED-RECORD-TYPE

DETAILED-STATUS-CODE
Redefined as RECORD-TYPE

Before issuing any random GET, GETUP, or INSERT function request, the action program can indicate to defined record management the record type it expects to receive by placing it in the PREDICTED-RECORD-TYPE byte of the DETAILED-STATUS-CODE. If defined record management finds a value other than zero, it verifies the prediction before carrying out the retrieval or insertion. If the predicted type is not correct, the record involved is not moved; instead, a status code of 1 is returned to the calling program. If the predicted type is correct, the function is carried out, and the PREDICTED-RECORD-TYPE byte reverts to zero. The action program, therefore, can use the PREDICTED-RECORD-TYPE byte prior to 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, the programmer is strongly advised to use this feature. This assures that further processing will apply the correct defined record definition.

When you issue the sequential function calls SETL and GET, defined record management returns the record type of the next sequential record to your action program in the PREDICTED-RECORD-TYPE byte. If the delivered record type is the parent of the predicted record type and you wish to access records from a record type other than currently indicated in the PREDICTED-RECORD-TYPE byte, 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 all sets subordinate to the current delivered record type are skipped. 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.

When defined record management 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. See Figures 2-38 and 2-39 for sample data definition processor output listings showing status bytes.) These values can be tested by the action programmer (in COBOL programs for fixed-length records but not variable-length records) to check the validity of individual items in the defined record.

The value X'80' is returned by defined record management for all functions to indicate that the item has been successfully delivered.

For GET and GETUP functions, defined record management 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 X'40' is returned for one or more items along with a value of zero in the status code of the PIB, it generally means that 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 generally means that the *key* parameter points to a nonexistent primary part.

For PUT and INSERT functions, defined record management returns a value of X'20' in the item status byte, along with a value of 5 in the status code of the PIB, to indicate that the item being changed or added does not conform to conditions specified in the data definition. This 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); or
- no new value was entered for a MUST ADD item (INSERT only).

If an error occurs while accessing a file before returning control to the action program, defined record management changes the lock-rollback-indicator to "O".

3.8.5.2. Random File I/O Functions

To access defined files randomly, you may issue the GET, GETUP, PUT, DELETE, and INSERT functions calls. During random access to defined files, logical record locks are imposed on logical records involved in the GETUP and INSERT functions.

3.8.5.2.1. GET and GETUP Functions

The GET and GETUP functions retrieve the record identified by the *key* parameter from the named file and place the record into the record area. A GET function cannot be followed by a PUT or DELETE function. The GETUP function retrieves a record for update and can be followed by a PUT or DELETE function.

COBOL Format:

```
CALL{'GET'  }USING file-name record-area key.
    {'GETUP'}
```

BAL Format:

```
{CALL   }{GET  },(file-name,record-area,key)
{ZG#CALL}{GETUP}
```

Status Codes:

0	Successful
1	Invalid key
2	Not used
3	Invalid request
4	I/O error
5	Violation of data definition

3.8.5.2.2. PUT Function

The PUT function writes a record that was retrieved for update back to the file. The PUT function must immediately follow the GETUP function for the record to be updated.

COBOL Format:

```
CALL 'PUT' USING file-name record-area.
```

BAL Format:

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

Status Codes:

0	Successful
1 and 2	Not used
3	Invalid request
4	I/O error
5	Violation of data definition

3.8.5.2.3. DELETE Function

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 Format:

```
CALL 'DELETE' USING file-name record-area.
```

BAL Format:

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

Status Codes:

0	Successful
1 and 2	Not used
3	Invalid request
4	I/O error
5	Violation of data definition

3.8.5.2.4. INSERT Function

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 Format:

```
CALL 'INSERT' USING file-name record-area key.
```

BAL Format:

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

Status Codes:

0	Successful
1	Invalid key
2	Not used
3	Invalid request
4	I/O error
5	Violation of data definition

3.8.5.3. Sequential File I/O Functions

To access defined files sequentially, you may issue the SETL, sequential GET, and ESETL function calls.

3.8.5.3.1. SETL Function

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:

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

COBOL Format:

```
CALL 'SETL' USING file-name position [key].
```

BAL Format:

```
{CALL    }SETL,(file-name,position [,key])
{ZG#CALL }
```

Status Codes:

0	Successful
1	Invalid key
2	Not used
3	Invalid request
4	I/O error

NOTES:

1. *The key parameter is omitted when the value of position is B.*

2. *Invalid key means that the identifier is not within defined bounds. That is, the defined record type cannot be determined from the value of the identifier. This code is not used to indicate missing data. The successful completion (status code of 0) does not preclude the possibility that ensuing GET calls will yield only total cycles and no detail defined records.*

3.8.5.3.2. GET Function

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

COBOL Format:

```
CALL 'GET' USING file-name record-area.
```

BAL Format:

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

Status Codes:

0	Detail cycle
1	Invalid record type
2	Total cycle
3	Invalid request
4	I/O error

NOTES:

1. *If a status code is 0 (detail cycle), a new defined record is returned to your action program. The detailed status code identifies the record type. (See 3.8.4.1.)*

2. *A status code of 2 means that there are no more records in the current set. No new defined record is returned. The detailed status code 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.*

3. *After a detail record is delivered, all subordinate records are also delivered in response to subsequent GETs. 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 equal to the record type of the empty set.*

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

3.8.5.3.3. ESETL Function

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, the file is changed to random mode at termination by IMS 90 file management.

COBOL Format:

```
CALL 'ESETL' USING file-name.
```

BAL Format:

```
{CALL    }ESETL,(file-name)
{ZG#CALL }
```

Status Codes:

0	Successful
1 and 2	Not used
3	Invalid request
4	I/O error

3.8.6. Online File Recovery

The recovery facility provides for the rollback of user data file modifications (updates, inserts, and deletions) that have occurred in a transaction that abnormally terminates prior to completion or explicitly requests rollback prior to completion. When transactions terminate abnormally, IMS 90 issues messages to the source terminal and system console. These messages are documented in the OS/3 system messages programmer/operator reference, UP-8076 (current version). Each IRAM, MIRAM, ISAM, or DAM file modified in the transaction prior to the time of rollback is returned to the logical state that existed before the transaction was initiated or before the last rollback point was recorded on the audit file. Rollback occurs automatically when needed upon abnormal termination and does not require any user programming.

Rollback can be requested upon the normal termination of an action by specifying the O option of the LOCK-ROLLBACK-INDICATOR in the program information block. The range of rollback can be limited to less than an entire update transaction by specifying the N option of the LOCK-ROLLBACK-INDICATOR in the program information block upon the termination of some intermediate action in a dialog update transaction.

The current state of each record that is to be updated or deleted is recorded in the IMS 90 audit file prior to the update or deletion. In addition, the identifiers inserted into the file are recorded prior to insertion. Information also is recorded in the audit file to mark the initiation and termination of each transaction that modifies a file. When rollback points are specified, they are also recorded on the audit file. Online file recovery uses the audit file to accomplish file rollback. Table 3-14 lists the specific functions performed in the rollback of file modifications.

Table 3—14. 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

3.8.6.1. File I/O Error Returns

When unrecoverable I/O errors occur in the audit file, the source terminal operator is notified and a message is sent to the print file. Rollback is attempted for all existing transactions that have been logged in the audit file. IMS 90 prohibits any additional update requests (unless the LOCK=UP parameter is specified in the configurator FILE section) and returns an invalid request code (3) in the STATUS-CODE field of the PIB, as well as a binary value in the DETAILED-STATUS-CODE field. (Detailed status codes for file I/O functions are listed in Table 3-8.)

3.8.6.2. Prefix Area Format

If an I/O error occurs on a user data file during the rollback of a file modification (updates, inserts, deletions), a snapshot dump is taken of the prefix area of the record being rolled back. Upon completion of the snapshot dump, the rollback procedure continues until rollbacks of all modifications made to user data files for a transaction are attempted.

If an error occurs on the AUDCONF continuity data and audit file or the AUDFILE during the rollback of updates made by a transaction, the current action program name (contained in the prefix area of the record being rolled back) will be ZU#ROL.

The format of the prefix area is shown in Figure 3-14. Table 3-15 describes the content of each field.

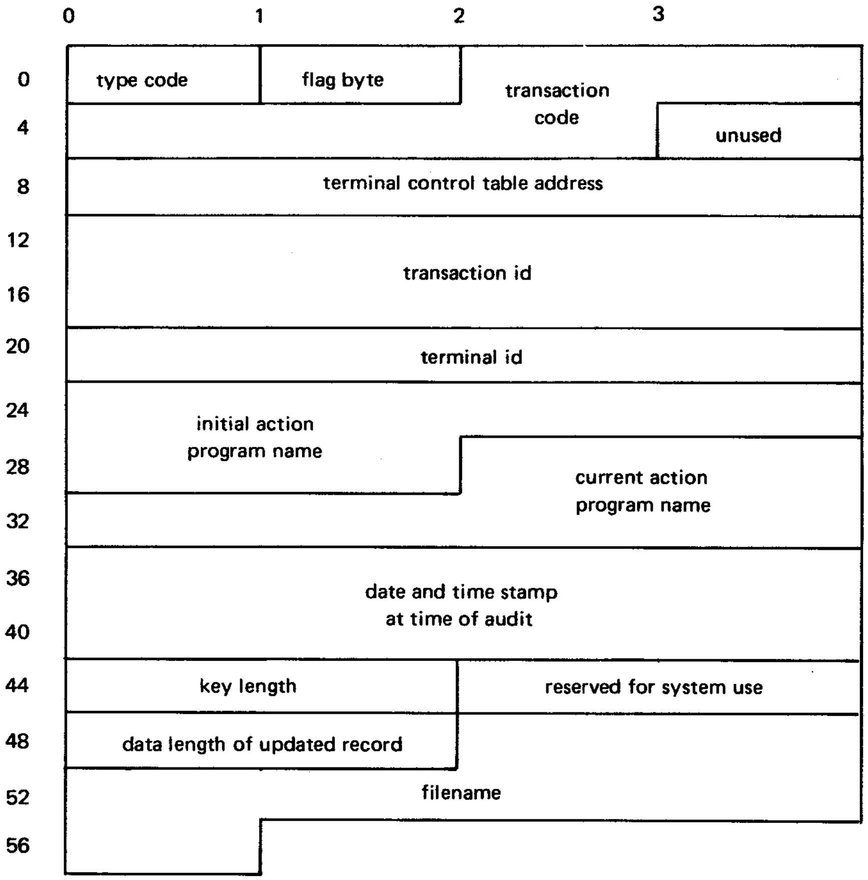

0
1
2
3
0
4
8
12
16
20
24
28
32
36
40
44
48
52
56
type code
flag byte
transaction code
unused
terminal control table address
transaction id
terminal id
initial action program name
current action program name
date and time stamp at time of audit
key length
reserved for system use
data length of updated record
filename

Figure 3—14. Format of Prefix Area of Records in the Audit File (Online Recovery)

Table 3—15. Content of Prefix Area for Records in the Audit File (Online Recovery)

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
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 terminal 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.

3.8.6.3. COBOL Action Program Error Messages

The COBOL error routine (CO@BJERR) records data in a message buffer that corresponds to errors contained in the canned message file. The IMS 90 user locates the message buffer in his dump and inspects it to find the cause of the error. The 4-byte message buffer, located at the address contained in general register 1 in the register save area of the program dump listing, contains the following:

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. For the text of the message, its meaning, and suggested corrective action, refer to the OS/3 system messages programmer/operator reference, UP-8076 (current version).

Bytes 1—2 Content	*COBOL Message*
043A	*CE03*
043B	*CE04*
043C	*CE05*

3.8.7. Logical Record Lock Facility

Logical record locks are used to protect records that are in the process of being updated by one transaction from concurrent updating by other transactions. Either of two logical lock options, lock-for-update or lock-for-transaction, is selected for a DAM, ISAM, IRAM, or MIRAM file via the LOCK parameter of the FILE section at configuration time.

3.8.7.1. Lock for Update

The lock-for-update option causes a lock to be imposed on a logical record when the record is retrieved from the file via the GETUP function. The lock prohibits access to the record by other transactions until it is unlocked. It does not prohibit further access in the same transaction to the same record. The record is unlocked when one of the following events occurs in the transaction that imposed the lock:

- The record is updated by means of the PUT or DELETE function.
- The action in which the lock was imposed or a subsequent action terminates with the termination-indicator set to N (normal transaction termination), A (abnormal transaction termination, voluntary), or the transaction abnormally terminates involuntarily.

- The action in which the lock was imposed, or a subsequent action, terminates with the termination-indicator set to E (external successor) or D (delayed internal successor) and the lock-rollback-indicator set to R (release all locks for pending updates), N (establish a new rollback point), or O (return to an old rollback point).

- An UNLOCK request is made for the file containing the record.

3.8.7.2. Lock for Transaction

The lock-for-transaction option causes a lock to be imposed on a logical record when it is retrieved from the file via the GETUP function or when it is inserted into the file via the INSERT function. The lock prohibits access to the record by other transactions until it is unlocked. It does not prohibit further access in the same transaction to the same record. The record is unlocked when one of the following events occurs in the transaction that imposed the lock:

- The action in which the lock was imposed or a subsequent action terminates with the termination-indicator set to N (normal transaction termination), A (abnormal transaction termination, voluntary), S (abnormal transaction termination, voluntary, with snap dump), or the transaction is involuntary and abnormally terminates.

- The action in which the lock was imposed, or a subsequent action, terminates with the termination-indicator set to E (external successor) or D (delayed internal successor) and the lock-rollback-indicator set to R (release all locks for pending updates). In this case, only those locks imposed as the result of GETUP requests and for which subsequent PUT or DELETE requests have not been issued are unlocked. Locks-for-transactions that fall into this category are called pending locks. If the lock-rollback-indicator is set to N or O, all locks are released.

- An UNLOCK request is made for the file containing the record.

In single-thread configurations, when a transaction requests access to a locked record, the record is not accessed. Control returns immediately to the requesting action program with a status code of 3 (invalid request) and a detailed status code of 18 indicating that the record being accessed was locked by another transaction.

In multithread configurations, the request for a locked record is queued until the record is unlocked. The record is then accessed before control returns to the requesting action program.

The use of the lock-for-transaction option is closely related to the online file recovery feature of IMS 90. Consider an update transaction in which two records, A and B, are to be updated. Record A is in one file and B is in another, and, for both files, the lock-for-transaction option is specified.

When record A is retrieved for updating (GETUP), a lock is applied. When updating is carried out (PUT), the lock is not released because LOCK=TR was specified at configuration time. Another transaction then attempts to access record A, but its request is queued to wait for the lock to be released. The first transaction then comes to an abnormal termination because it is unable to successfully update record B.

Online file recovery rolls back record A to its original state and then releases the lock. The second transaction then accesses the correct original version of record A. Record B is not updated.

In general, lock-for-transaction in combination with online file recovery is used to ensure that all records that are to be updated in a transaction are either successfully updated or rolled back to their original state before any other transaction can access the records in the same order. Only files for which lock-for-transaction (i.e., LOCK=TR) or no locking (i.e., RECLOCK=NO) is specified can be rolled back by online file recovery.

Since the capability exists for a transaction to lock multiple logical records and since a transaction must wait if it attempts to access a record that is already locked by another transaction, deadlock can occur in IMS 90. Deadlock can be avoided by choices made in the design of action programs.

- First, deadlock does not occur if only a single record is updated in a transaction.

- Second, deadlock does not occur if records are updated serially in a transaction (e.g., GETUP record A, PUT record A, GETUP record B, PUT record B, etc.) and all files for which updates are performed have the lock-for-update option specified.

- Third, deadlock does not occur if all transactions that update a set of records update those records in the same order regardless of lock option.

If deadlock occurs among two or more transactions as the result of improper file or action program design, no output other than the periodic status message will be sent back to the terminals that originated those transactions. A terminal operator, recognizing that output has not been returned in a reasonable period time, should cancel the transaction he has initiated. The cancel will free up all resources allocated to the transaction and may allow the deadlock to be broken. A sufficient number of cancellations also break the deadlock. This procedure is not desirable for most applications. To avoid it, the design rules must be followed or the user must have a clear understanding that action programs he designs must not reference the files in such a way as to cause deadlock.

3.8.7.3. UNLOCK Function

The UNLOCK function causes the release of record locks that have been imposed by file management for a transaction on a specific file. The UNLOCK function also makes ISAM, MIRAM, and IRAM files, held for a transaction pending the completion of an update, available for processing.

COBOL Format:

```
CALL 'UNLOCK' USING file-name.
```

BAL Format:

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

The UNLOCK function applies to both lock-for-update and lock-for-transaction imposed on DAM, IRAM, MIRAM, or ISAM files within the IMS 90 job. Normally, locks are released implicitly within IMS 90 as the result of action or transaction termination. The UNLOCK function should be requested only when the normal implicit release of locks does not meet the requirements of the application.

Only the locks associated with pending updates for this transaction are released. An update is pending after a GETUP function but before either a PUT or DELETE function changes the record or some other event occurrs to indicate that the update will not be performed.

If the lock-for-update or lock-for-transaction option is specified for a DAM, IRAM, MIRAM, or ISAM file and an update of a record in that file is currently pending for a transaction, then an UNLOCK request from the transaction aborts the update by releasing the lock on the record. In the case of an ISAM file, the UNLOCK request makes the file, as well as the individual record, accessible for the processing of requests from other transactions. In the case of a DAM file, the UNLOCK request unlocks only the individual record. The file as a whole still remains accessible to other transactions.

3.8.8. General File Processing Considerations

3.8.8.1. Opening and Closing of Files

All files defined in the file control table at configuration time are opened normally as a function of IMS 90 start-up and closed as a function of shutdown. Files also can be opened and closed from the master terminal via the master terminal commands, ZZOPN and ZZCLS. These master terminal commands cause IMS 90 file management to issue calls to data management to perform open and close functions. No capability exists to open and close files from an action program. For a detailed description of the ZZCLS and ZZOPN master terminal commands, see 5.2.2.6 and 5.2.2.7.

3.8.8.2. Serial Use of File Descriptors

Each file accessible to IMS 90 has a single file descriptor entry in the file control table. File management queues requests for each file and provides for servicing of the requests one at a time.

3.8.8.3. Dynamic Allocation of I/O Areas

I/O areas for user data files are preallocated in multithread IMS 90. In single-thread IMS 90, I/O areas are allocated when required. Action scheduling acquires I/O areas, when needed, from main storage management as a function of action initiation. At most, one I/O area is allocated to a file at a given time. Once allocated, an I/O area can be used to support a number of file functions for a number of different transactions. When no potential or actual requests are outstanding for a file, the I/O area for the file is released by action scheduling to main storage management.

3.8.8.4. File Sharing

All data files allocated to IMS 90 are accessible to all terminals in the configured network. An ISAM or MIRAM file (sequential mode) is allocated exclusively to an action for a series of sequential file operations when the SETL function is performed. It is deallocated either explicitly by the action by means of the ESETL function or implicitly at action termination by file management.

Because of ISAM file updating procedures, such a file is exclusively allocated to a given transaction during the time that transaction has an update pending for the file. This exclusive allocation begins with the successful completion of a GETUP function and ends with the completion of a PUT or DELETE function to carry out the update in the same action or with the termination of the action, whichever comes first.

Exclusive use is forced to end at action termination to increase the sharability of the file (that is, to allow queued requests to be serviced). Although exclusive use of an ISAM file for update purposes must end with action termination, the lock on the record being updated can be held from one action to another.

If a record is retrieved with a GETUP in one action and the update is to be completed in the following action, the following action must again retrieve the record with a GETUP before updating with a PUT or DELETE. This means that two retrievals of the record are necessary in a multiaction update. It is more efficient to carry out updates of ISAM or MIRAM files in a single action.

3.8.8.5. Work and Record Areas for DAM File Access

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. Moreover, to achieve device independence across disk subsystems, so that your program can access a DAM file on any disk used under OS/3, the same point holds - I/O areas should be multiples of 256 bytes in length.

To ensure device independence in a BAL or COBOL action program that accesses DAM files, you should do the following:

- If you are using a continuity data area (CDA) for record I/O, specify its length as a 256-byte multiple via the CDASIZE configuration parameter in the ACTION section.

- If your program varies the length of the CDA when succession is made from one action to another, the increment or the new output length you set in the PIB should be a 256-byte multiple.

- If the work area is used for record I/O, specify its length as a 256-byte multiple via the WORKSIZE configuration parameter in the ACTION section.

- If you specify a different work-area size for a succeeding action that is to access a DAM file, the increment placed into the WORK-AREA-INC field of the PIB should be a multiple of 256 bytes. The work-area increment is applicable only in multithread IMS 90.

- The *record-area* parameter of any IMS 90 function call (GET, GETUP, PUT, DELETE, or INSERT) that your action program issues to process a DAM file must refer to an area whose reserved length is some multiple of 256 bytes on a half-word boundary.

You have 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 the data management programmer reference, UP-8159.

3.8.8.6. Test Mode

When a terminal has been put in the test mode (via the ZZTMD terminal command), any request to file management to change the contents of a file are only simulated. Specifically, no update, delete, insert, or append functions are performed when requested. Control is simply returned to the requesting transaction with a successful completion status code.

A terminal can be put in the test mode after completion of a transaction, i.e, when not in an interactive mode. To revert to normal mode, the ZZNRM terminal command is used. Test mode is used in the training of new terminal operators in the use of 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 used in the testing of newly written or modified action programs that perform file modifications.

3.8.9. Common Storage Area Files

You can increase file processing efficiency by making frequently accessed ISAM 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. It is not available for basic IMS 90, and the CSA is not accessible through UNIQUE.

If you specify CAFILE=YES in the configurator FILE section, the designated ISAM file is loaded into the common storage area at start-up time. When you issue GET, GETUP, and PUT function calls, IMS 90 retrieves records from and makes updates to the CSA file, thereby reducing disk access. If you specify CUPDATE=YES to the configurator, updates are made to the disk as well as to the resident file. This saves disk accesses on reads but not on writes. However, if you omit CUPDATE or specify CUPDATE=NO, the disk file is not updated until IMS 90 shutdown, when the entire CSA file is written to disk. File locking and recovery functions are the same for the CSA file as for a disk file.

You can access a CSA file only in random mode. You use GET, GETUP, and PUT function calls the same way for any ISAM file, but INSERT and DELETE functions are not valid. UNIQUE can access a CSA file through defined record management, but only through a supplement definition; you must not specify ASSUMES CONTROLLED ROLE IN UPDATE in the supplement definition.

3.9. IMPLICIT AND EXPLICIT MESSAGE OUTPUT

Normally, a message is sent to the designated logical destination from the output message area at action termination (CALL 'RETURN'). This is *implicit* output. Implicit output takes several forms. It can be:

- displayed or listed on the terminal initiating the transaction (source terminal) or the terminal designated by the DESTINATION-TERMINAL-ID field of the OMA header;
- listed on an auxiliary device attached to the source terminal or destination terminal; or
- printed as continuous output at the source terminal or on an auxiliary device attached to the source terminal.

If more than one message must be sent during an action, or if a transaction is to be initiated at a terminal other than a source terminal, this *explicit* output must be transmitted by means of the SEND function.

3.9.1. Transmitting Messages via SEND Function

The SEND function is typically used to send messages to terminals other than the originating terminal. This usage applies under both single and multithread IMS 90. It can also be used to initiate a transaction, typically a continuous output print transaction, at a distant terminal via output-for-input queueing (described in 3.10.2). 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 originating terminal is not able to handle the error.

COBOL Format:

```
CALL 'SEND' USING output-message-area [master].
```

BAL Format:

```
{CALL    }SEND,(output-message-area[,master])
{ZG#CALL }
```

where:

output-message-area
: Refers to a data-name (COBOL) or storage area (BAL) containing an output message header (in the format shown in Figures 3-10 and 3-11) and text. The output message area does not necessarily have to be the same as the OMA that is predefined for the action in the action control table. An output message can, for example, be explicitly sent from the work area allocated to an action. This area must be aligned on a full-word boundary.

master
: Refers to a data-name or storage location that contains the value "M" indicating that this message is to be sent to the master terminal.

If the *master* parameter is specified, and the data name referenced in the action program contains the value "M", the message is sent to the master terminal. If this parameter is omitted, the message is sent to the terminal specified in the DESTINATION-TERMINAL-ID field of the OMA. If the data name referenced does not contain the value "M", an error is returned to the action program. This error is indicated by the value 3 (invalid request) in the STATUS-CODE field of the PIB and by the value 3 (incorrect parameter value) in the DETAILED-STATUS-CODE field of the PIB. The master parameter option is used with the output-for-input queueing process by placing the value "I" in the AUX-FUNCTION field of the OMA header. This results in the generated message being queued as input for the master terminal regardless of the DESTINATION-TERMINAL-ID in the OMA header.

An output message is not sent to the designated terminal by internal message control until the successful termination of the current action. If the transaction is terminated abnormally or canceled in the current action, all output messages generated in the action are deleted from the output message queue and no messages are delivered. A message indicating the reason for termination is sent to the originating terminal. After the output message is moved from the output message area and written to the output message queue, control is returned to the statement following the CALL statement.

If the SEND function is used frequently, disk queueing should be specified in the communications network definition. Refer to the IMS 90 system support functions user guide/programmer reference, UP-8364 (current version). If you use the SEND function, you must specify the UNSOL=YES parameter in the OPTIONS section of the configurator. To use output-for-input queueing, the CONTOUT=YES also must be configured. (See 3.10.2.)

3.9.2. Returns from SEND Function

Following execution of the SEND function, IMS 90 notifies the action program of the success or failure of the request by placing binary values in the STATUS-CODE and DETAILED-STATUS-CODE fields of the program information block. If you specified ERET=YES to the configurator for this action program, the action program regains control at the instruction after the SEND function call and must interrogate these status bytes. If you did not specify ERET=YES, the program does not regain control in the event of unsuccessful completion of the SEND function and is abnormally terminated by IMS 90. A 3-line transaction termination message is sent to the system console. Transaction termination messages are documented in OS/3 system messages programmer/operator reference, UP-8076 (current version).

IMS 90 returns the following values to the STATUS-CODE field of the PIB following the execution of the SEND function when ERET=YES is configured:

Status Codes:

0	Successful
1 and 2	Not used
3	Invalid request
4 and 5	Not used
6	Internal message control error

When the return made to the STATUS-CODE field of the PIB is 3 (invalid request), the DETAILED-STATUS-CODE field contains either 2 or 3. The value 2 indicates that unsolicited output or continuous output was not configured or that no process files were generated in your ICAM CCA. The value 3 indicates a parameter error has occurred.

When the return made to the STATUS-CODE field of the PIB is 6 (internal message control error), the return made to the DETAILED-STATUS-CODE field is one of the following:

2	Destination terminal is busy or on hold.
3	Destination terminal is down; message is queued.
4	Invalid destination terminal or auxiliary device specification
5	No network buffer is available in the ICAM load module.
6	Disk error
7	Invalid length specification

IMS 90 returns the value 2 (destination terminal busy or on hold) to the DETAILED-STATUS-CODE field only when the SEND function is issued to generate *output-for-input queueing* and one of the following conditions exists:

- Destination terminal is in interactive mode.
- Destination terminal already has an input message on queue.
- A ZZHLD terminal command has been entered at the destination terminal or a ZZDWN command for this terminal has been entered at the master terminal; therefore, it is logically down to IMS 90.
- The destination terminal is marked physically down to ICAM.

These are not permanent conditions; the output message header content is not invalid, and the same message may be retransmitted successfully at some later time.

IMS 90 returns a DETAILED-STATUS-CODE of 3 when an action program attempts to send a message (via CALL SEND) to a destination terminal that is physically or logically down. The message is still queued for that terminal and is automatically transmitted when the terminal comes back up. If the destination terminal is down and is alternated (via the ZZALT command) to another terminal in the IMS network, no error is returned to the action program. The message is sent to the alternate terminal. If the alternate terminal is also down, a DETAILED-STATUS-CODE of 3 is returned to the action program.

When IMS 90 returns the DETAILED-STATUS-CODE value 4 (invalid destination terminal or auxiliary specification) after the SEND function, the content of the OMA header is not valid, and efforts to retransmit the output message will continue to be unsuccessful. One of the following errors is likely to have been made in your action program:

- The DESTINATION-TERMINAL-ID specified in the OMA header is not a valid configured terminal identification.
- The AUXILIARY-DEVICE-ID specification is invalid.
- The AUX-FUNCTION byte contains the hexadecimal value C3, F3, or F7, indicating that continuous output is requested. Continuous output must be the *implicit* output from an action program requesting it; it can not be transmitted via the SEND function.

3.10. PRINT TRANSACTIONS USING CONTINUOUS OUTPUT

When your installation applications include a need for printing at one of your interactive network terminals, you can easily implement print transactions by using the optional continuous output feature. You can direct the printing to be done at the terminal that originates the print transaction or cause the printing to be initiated at a different terminal - which can be an unattended terminal, or even a receive-only or output-only interactive device.

Continuous output is an option you select at configuration time via the CONTOUT keyword parameter in the OPTIONS section and is available for both multithread and single-thread operations (but not for basic IMS 90). It is not limited to normal interactive processing, but can be used in the batch processing mode – online for production or offline for testing. When you specify the use of continuous output at configuration time, an additional internal message control component is included in your IMS 90 load module by the configurator.

The CONTOUT specification also causes the automatic inclusion of IMS 90 support for unsolicited output. The module giving this capability is used in continuous output to support the ability to initiate a print transaction at a destination terminal other than the originating terminal. For this, you use the output-for-input-queueing feature via the SEND function. Use of the continuous output feature also requires a resident ICAM.

The terminals and auxiliary devices to which continuous output can be directed by your printing transactions are:

- DCT 500 Data Communications Terminals operating in teletypewriter mode
- DCT 1000 Data Communications Terminals
- TELETYPE* Modules 28, 33, 35, 37, and 38

*Trademark of Teletype Corporation

- Auxiliary devices at the UNISCOPE 100 or 200 Display Terminal or Universal Terminal System 400 (UTS 400). These are:
  - Communications Output Printers (COPs)
  - Terminal Printers (TPs)
  - Model 610 Tape Cassette System (TCS)
  - 8406 Diskette Subsystem

Note that these are the interactive devices supported by ICAM; the DCT 1000 operating as a batch (card-oriented) terminal or any other batch terminal is not usable for continuous output.

In addition, ICAM supports a UTS 400 feature called screen bypass, which is addresssable only for output (3.10.3).

The following paragraphs describe how you can use the continuous output option as an action programmer; sample action programs are presented in 3.16 to further illustrate the points developed here.

3.10.1. Generating Continuous Output

A print transaction must direct all its continuous output to the same terminal, i.e., the one specified in the SOURCE-TERMINAL-ID field (ZA#ISTID) of the input message area header. When you do not desire continuous output at the terminal initiating the transaction, you direct a print transaction to be initiated at another terminal, using the *output-for-input-queueing* feature explained in 3.10.2.

Whether you are transmitting continuous output at a primary device or at an auxiliary attached to it, you generate only one continuous output message per action; this message must be your implicit output and cannot be generated via the SEND function. You are not restricted from using the SEND function for other unsolicited output, however, nor are you restricted from sending other output to the primary device via the SEND function.

3.10.1.1. Output Message Header Fields for Continuous Output

You specify that an output message is for continuous output by the value you send in the AUXILIARY-FUNCTION field (the first byte of the AUXILIARY-DEVICE-ID field) in the OMA header. When your continuous output is to be transmitted to a COP, TP, tape cassette, or diskette attached to the primary device at which your program is initiated, you must also set the second byte of this field (the AUX-DEVICE-NO) to specify which of the configured auxiliary devices is to be used for data transmission; otherwise, set the AUX-DEVICE-NO byte to binary 0. Table 3-16 summarizes the settings of the AUX-FUNCTION byte for normal and for continuous output.

When you are transmitting to a COP, TP, cassette, or diskette auxiliary device, you can specify print-transparent mode. In this mode, although your output goes through the logic of the primary device, its format is independent of the format on the screen. Whatever DICE sequences you then include in your message apply to these auxiliary devices and the cursor return characters normally inserted by the logic of the terminal are not transmitted to the auxiliary interface. Thus, the length of lines written to the auxiliary device is independent of the line length of the screen.

When using print-transparent mode with a UNISCOPE 100 display terminal, you should make sure that your output message does not exceed the capacity of the screen. If it does, the excess lines wrap around and overlay the first few lines originally at the top of the display. Since the message on the screen is the message sent to the auxiliary device, your transmitted result will begin with the excess lines in place of the original lines that are lost. The same consideration also applies to the UNISCOPE 200 and UTS 400. However, their larger screen capacity makes wraparound a less likely occurrence.

In print mode you apply your DICE sequences for the screen, and the message printed on the auxiliary device will have the same format as the screen. For further details on print mode and print-transparent mode, refer to the current versions of the UNISCOPE programmer reference, UP-7807 and the UTS 400 programmer reference, UP-8359.

Table 3—16. Settings for Auxiliary Function Byte of Output Message Header (Part 1 of 2)

Devices		Input/Output Options			Contents of AUX-FUNCTION Field			
Primary	Auxiliary	Name	Space Suppression	Inhibit Space Suppression	Continuous Output		No Continuous Output	
					Hex	Character	Hex	Character
X					C3	C	00	
	X	Print Mode	X		F3	3	F0	0
				X	F5	5	F2	2
		Print Transparent	X		F7	7	F4	4
				X	F9	9	F6	6
		Print Form (ESC H)	X		C1	A	D1	J
				X	C6	F	D6	O
		Transfer All (ESCG)	X		C2	B	D2	K
				X	C7	G	D7	P
		Transfer Variable (ESC F)	X		C4	D	D4	M
				X	C8	H	D8	Q
		Transfer Changed (ESC E)	X		C5	E	D5	N
				X	E8	Y	F8	8

Table 3—16. Settings for Auxiliary Function Byte of Output Message Header (Part 2 of 2)

Devices		Input/Output Options			Contents of AUX-FUNCTION Field			
Primary	Auxiliary	Name	Space Suppression	Inhibit Space Suppression	Continuous Output		No Continuous Output	
					Hex	Character	Hex	Character
	X	Read			D9	R		
		Read Transparent			E2	S		
		Search and Read			E3	T		
		Search and Read Transparent			E5	V		
		Report Address			E6	W		
		Backward One Block			D3	L	E7	X
		Search and Position			E9	Z	E4	U

When you choose print or transfer options, you may either allow or inhibit space suppression in output messages. If you do not specify the inhibit space suppression option, the remote device handler (RDH) suppresses nonsignificant spaces. If you do specify the inhibit space suppression option, the RDH changes all spaces to DC3 characters making it necessary to strap the COP or TP devices to space when they receive a DC3 character in the text data.

In addition to print and print transparent options, the following print options directed to the UTS 400 terminal printers, cassette, or diskettes can be specified with or without the inhibit space suppression option. Unless the inhibit space suppression option is specified with each of these print options, nonsignificant spaces are suppressed.

- Print form (ESC H) sends to the terminal printer, cassette, or diskette all of the unprotected characters and protected characters from the start-of-entry (SOE or home position) to the cursor. Spaces are substituted for protected data. Field control characters (FCC) are suppressed.

- Transfer all (ESC G) sends to the terminal printer, cassette, or diskette all characters from SOE to cursor including FCC sequences.

- Transfer variable (ESC F) sends to the terminal printer, cassette, or diskette only the variable (unprotected) characters between the SOE and cursor including FCC sequences.

- Transfer changed (ESC E) sends to the terminal printer, cassette, or diskette only the changed characters (or altered fields) between the SOE and the cursor including FCC sequences.

Several options are available to read and write, search, or position data on cassette and diskette auxiliary devices. To use these options, your action program must move both the desired option character code and the auxiliary device number to the AUX-FUNCTION and AUX-DEVICE-NO fields of the output message header respectively. Your action program can then issue either a SEND or RETURN function to transmit an output message to the auxiliary device, i.e., write a block of data to the cassette or diskette. In this way, your action program may also read a block of data from the cassette or diskette, display it on the UNISCOPE screen or position a cassette/diskette at a specific block. Note that the SEND function cannot be used to transmit continuous output options.

In some cases, input from the auxiliary device may be expected as a result of output from your action program. Cassette/diskette input then appears on the terminal screen but is not transmitted to your program unless you press the transmit key at the terminal. Otherwise, you can set the AUTO TRANSMIT switch on the tape cassette or diskette system before data is transmitted from the cassette/diskette to the terminal and the input data is automatically supplied to your action program. When your action program receives input resulting from a previous output to the auxiliary device, the auxiliary device number from the OMA is returned to the IMA. Figure 3-15 illustrates the data flow between your action program and the cassette/diskette auxiliary devices.

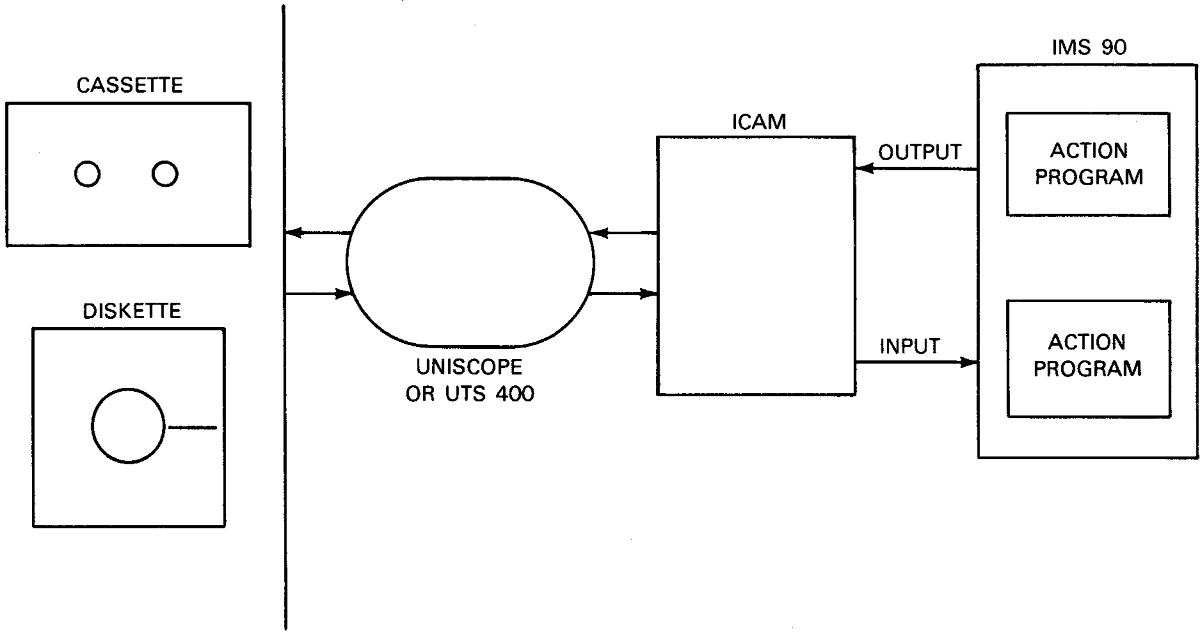


Figure 3—15. Action Program Interface with Cassette/Diskette

The output options print mode, print transparent, print form, transfer all, transfer variable, and transfer changed can be used with cassette/diskette; however, the print form, transfer all, transfer variable, and transfer changed may be used only when the primary device is a UTS 400 terminal.

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:

- The read option reads a block of data from the cassette/diskette to the terminal screen. When you set this option, your action program should not contain text in the OMA and the text length field should contain 4.

- 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. If the data is transmitted to your action program, the IMA message text will then be identical to the OMA text originally written to the cassette/diskette.

- 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 OMA 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. (See Table 3-17 for their formats.) When you use the search and read options, the only contents of the OMA message text should be the search argument in the mode you choose.

- 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.

Two other options available for cassette/diskette on continuous and noncontinuous output are the search-and-position and backward-one block options.

- The search-and-position option positions the cassette/diskette to the block requested in the search argument that your action program supplies in the output message text of the OMA. (See Table 3-18 for formats used in describing the search argument.) Your OMA message text may not contain any other entries.

- The backward-one-block option repositions the cassette/diskette one block in reverse. The AUXILIARY-DEVICE-ID field must be set and the TEXT-LENGTH field in the OMA must be 4.

Table 3—17. 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.

Table 3—18. User Message Text for Search and Positioning

Search Argument Format	Search Type
@taaaa or 0taaaa 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.

One additional option, the report address, displays the address of the cassette/diskette device on the terminal screen. Because input is expected after this action ends, the report address option cannot be used with the continuous output feature. The OMA TEXT-LENGTH field must contain the value 4.

In addition to making the required settings of the AUXILIARY-DEVICE-ID field of the OMA header, you also can insert into the 4-byte CONTINUOUS-OUTPUT-CODE field an optional alphanumeric character string that identifies the continuous output message you have generated. This optional code is returned by IMS 90 in the first four bytes of a 5-byte input message created for processing by the external successor designated by this action program. If you do not specify a code, the first four bytes of the input message generated by IMS 90 for your external successor contain binary zeros.

The CONTINUOUS-OUTPUT-CODE field of the OMA assumes special importance when you use any of the four input options or the report address option. When you use these options, a delivery notice is returned only if it represents a delivery notice error; otherwise, there is no input to the successor action program until a message is transmitted from the screen or, in the case of a screen bypass terminal, until the auto-transmit feature is used to transmit the message from the cassette/diskette device.

When using a screen bypass terminal, you must first set the control page for that terminal to take advantage of the auto-transmit 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 will be "hung" in interactive mode waiting for input that is never sent to it.

Because a successor action program may receive as input either a delivery notice error or an input message, the CONTINUOUS-OUTPUT-CODE used in the preceding continuous output message should be distinguishable from the first four characters of any possible input message being read from the cassette or diskette. In this way the successor action program determines what type of input message it receives (i.e. delivery notice error or input text) and processes it accordingly. In either case, the successor action program must be capable of handling both delivery notices and standard input messages.

3.10.1.2. Terminating Print Transactions

To continue printing, your action program terminates in external succession, naming itself or another action program in the SUCCESSOR-ID field of the PIB and passing (via the continuity data area) such information as is required to prepare the next of the continuous series of output messages. A program that creates a message for continuous output can terminate with only external succession. Immediate internal succession is not invalid but does not cause the output message to be sent. Any other form of succession causes IMS 90 to abnormally terminate the transaction, and the message generated is not transmitted.

On the other hand, the routines of your program that do not create continuous output are not restricted to terminating in external succession. They can use external succession or any other form of succession that is appropriate and typically they are routines that receive control as a result of your encountering errors or conditions relating to interruption or resumption of the continuous printing transaction. Normal termination is appropriate when you have completed all printing, for example.

3.10.1.3. Delivery Notice Scheduling

A program named as the successor to a program that creates continuous output also must be prepared to accept and process the 5-byte input message created by IMS 90. The 5-byte message indicates that ICAM has delivered your output message to its destination for continuous printing and has received an acknowledgement (the delivery notice) from the destination terminal.

The first four bytes of this input message contain either an optional 4-byte code you moved into the CONTINUOUS-OUTPUT-CODE field of the OMA header before you terminated, or binary zeros. If you have specified lowercase-to-uppercase translation or editing for input messages for the action receiving this input (via the ACTION section of your configurator input), this 4-byte field can vary from the CONTINUOUS-OUTPUT-CODE field as generated in the previous OMA.

The fifth byte of the new input message contains a value indicating the completion status of the output message. If the last output message was successfully delivered, you proceed to generate anocher or to terminate normally if you have no more to produce – possibly with a message output to the primary device to notify the terminal operator of the completion of the continuous print transaction.

However, if you are printing continuous output on preformatted forms – paychecks, for example – and do not want any printable characters output after normal termination, your last output message should comprise null characters. The point here is that if you do not put out *some* message when you terminate normally, then IMS 90 issues one of its canned messages by default (ACTION COMPLETE, or TRANSACTION COMPLETE) to the primary device after you call the RETURN function. If your printing is not being performed on a COP or TP, the canned IMS 90 message is printed on your form. (You may also need to program similar interceptions of the error messages that IMS 90 generates under certain conditions.)

When the completion status code in the fifth byte of the input message indicates unsuccessful output, you have a number of recovery options, depending on your application, that are discussed further in the following paragraphs. In planning recovery, however, it is important to realize the difference between polled and unpolled devices with respect to unsuccessful delivery notices.

Only the DCT 1000, UNISCOPE 100 and 200, and UTS 400 terminals are polled devices that actually transmit an acknowledgment to ICAM. The nonpolled devices (the TELETYPE and DCT 500 terminals) do not do so; unless a line-down condition prevents it, a delivery notice is generated automatically for messages sent to these latter devices, and it always indicates successful output completion, regardless of the true condition of the output message. Only a line-down condition causes an unsuccessful completion status.

IMS 90 always receives a successful completion status from ICAM when a message has been delivered to nonpolled devices and, therefore, reschedules your program with a successful delivery notice code in the fifth byte of the input message that it creates for you. For these reasons, when completing your continuous output or recovery in cases of nondelivery in a critical part of your printing application, you should avoid using nonpolled devices.

3.10.1.4. Recovery Considerations with Delivery Notice Scheduling

Recovery and restart processing are the responsibility of your action program; IMS 90 merely gives your printing transaction the means to perform recovery by notifying you of the success or failure of each continuous output message. For example, your successor action program keeps track of your continuous output in case any restarts are necessary. When unsuccessful completion status is returned, you may note these occurrences and continue as if successful - or terminate the transaction, to be restarted at some later time at the appropriate point. If your continuous output is directed to an auxiliary device, you send a regular output message to the primary device, indicating the need for assistance. You might terminate with external succession, awaiting an input message from the terminal to trigger the continuation of your print transaction. In any case, your action program must be prepared to accept actual input from the destination terminal as well as the delivery notice input message generated by IMS 90. Your terminal operators should be appropriately trained in these recovery procedures.

Both operator-entered input and delivery notice input can cause attempts to schedule your action program. If operator-entered input exists, IMS 90 processes that input and eliminates the delivery notice from the previous continuous output message. You should, therefore, code your action program to handle keyboard input that can end, temporarily break, and resume the continuous output transaction. The best way to interrupt continuous output is to use function keys as keyboard input. Function keys are faster to use because they are never locked even when all remaining keys on the terminal are locked.

When a message is unsuccessfully sent to an auxiliary device, only that device is marked down; output can still be addressed to the primary device. Your successor action program, which receives the error notification, can send an error message to the primary device and allow input from that terminal to cause continuation of the print transaction. Your recovery procedure should take into account the fact that IMS 90 has canceled the output message from the queue.

On the other hand, if an error occurs while your continuous output message is being sent to a polled primary device, it also is likely that an error message will be unsuccessfully delivered, and more likely that you will need help from the terminal operator or a technican to get the terminal in working order. In this situation, your action program might terminate the print transaction altogether, recording the point at which the error was encountered. A later initiation of this print transaction could start the printing at the point recorded.

An alternative in this situation is to send an unsolicited output message to another terminal, indicating the error and requesting assistance. (This could be the master terminal.) The action program could then terminate with a null message. When the out-of-order terminal is back in operation, the operator can enter an input message from the reinstated terminal to activate the successor action. The operator should not enter a terminal command. (This is also a recommended procedure when printing at a screen bypass terminal.)

Table 3—19. Output Delivery Notice Status Codes

Condition	Primary Devices Addressed				Corresponding Labels in TCS DSECT ①	Hexadecimal Value
	Polled		Nonpolled			
	UNISCOPE and UTS 400	DCT 1000	DCT 500	TTY		
Successful output completion	Yes	Yes	Yes, regardless of delivery	Yes, regardless of delivery	TM#TDNEM	81 ②
Line down or disconnected. Message deleted by IMS 90.	Yes	Yes	No	No	TM#TDLNO	11
Terminal marked down. Message deleted by IMS 90. ③	Yes	Yes	No	No	TM#TDDNA	12
Auxiliary device down. Message deleted by IMS 90. Output may be addressed to the primary device.	Yes	No	No	No	TM#TDNAX	40④
Missing or invalid destination or auxiliary spec. in header	Yes	Yes	Yes	Yes	TM#TEDST	84②
No ICAM network buffer available ⑤	Yes	Yes	Yes	Yes	TM#TENBA	85②
Disk error	Yes	Yes	Yes	Yes	TM#TEDER	86②
Invalid output buffer length	Yes	Yes	Yes	Yes	TM#TEILG	87②

NOTES:

① A BAL action program should access the labels in the TCS DSECT instead of testing the hexadecimal values in the input message directly. The hexadecimal values shown in the table can change in future releases, but the DSECT labels will remain the same.

② The hexadecimal value 81, indicating successful output completion, is translated to the character A if the lowercase-to-uppercase translate option is specified for messages input to the successor action. Similarly, the hexadecimal values 84 through 87, indicating error conditions, are translated to the characters D through G if the translate option is specified.

③ When a terminal is marked down, input solicitation (polling) by ICAM continues automatically. When ICAM receives input from the down terminal, that terminal is marked up, and the input is scheduled for IMS 90.

④ Refer to Table 3-15 for UNISCOPE and UTS 400 auxiliary device condition codes that are ORed with TM#TDNAX.

⑤ If this condition exists, a user action program can try to resend the last continuous output message.

In the fifth byte of the input message, IMS 90 provides you with the output delivery notice status code it receives from ICAM. The hexadecimal values of the code are listed in Table 3-19 or 3-20; these tables also list the corresponding labels in the ICAM *transaction control section* (TCS) DSECT, TM#TCS. If yours is a COBOL action program, you must test for this hexadecimal value in the fifth byte of the input message, as shown in the sample program PRINT in 3.15.3.

However, a BAL action program should be coded to access the labels in a TCS DSECT it has generated inline for this purpose, instead of testing for the hexadecimal value directly in the input message. The reason for this is that these hexadecimal values (which are *equate* (EQU) values for each DSECT label) can change in future OS/3 releases (but the ICAM DSECT labels always remain the same). If you access the labels, you have only to reassemble your BAL action program with each new release to be sure that your DSECT is current; otherwise you must change your code *and* reassemble.

Note also that if you specify TRANSLAT=YES for the action, ICAM DSECTs may not be used to evaluate delivery notice status codes because status codes will be changed by the translate routine. (See note 2 , Table 3-19.)

You call the ICAM procedure TM#DSECT, using the operand TCS, to generate the TCS DSECT inline when your BAL program is assembled. Figure 3-16 shows the delivery notification error code labels contained in the DSECT.

By using output delivery status codes, continuous output users can apply recovery procedures when output message errors are detected at message queueing time as well as delivery time. Errors with hexadecimal values 84 through 87 (Table 3-19) are discovered at the time the output is queued. All others are detected at the time the output is delivered to the terminal. Reasons for output message errors are:

- 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; or
- a disk error occurred.

If the no-ICAM-network-buffer-available status exists, a user action program can try to resend the last continuous output message.

```
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 EQU  X'87' . INVALID OUTPUT BUFFER LENGTH
```

NOTE:

The contents listed with each label in the DSECT indicate that the message is being held by ICAM. However, IMS 90 deletes these messages from the queue.

Figure 3—16. Portion of ICAM TCS DSECT in BAL Action Program Showing Delivery Notification Error Codes

3.10.2. Output-for-Input Queueing via the SEND Function

When you want to print continuous output at a terminal other than the one initiating the transaction, you do so by means of the output-for-input-queueing option, which causes an output message transmitted via the SEND function to be queued as an input message at the destination terminal instead of being output there. You do not transmit continuous output by this method but cause a transaction to be initiated at the destination terminal, which generates and prints continuous output. For example, the transaction you initiate at the originating terminal can create the records you want to be printed and write them to an indexed sequential file. The last stage of this transaction generates an output message for input queueing at the terminal where the printing is to be done; the transaction initiated by this input message reads the ISAM file sequentially and prints the messages as continuous output, using delivery notice scheduling for recovery (3.10.1.4).

Table 3—20. UNISCOPE and UTS 400 Auxiliary Device Condition Codes

Auxiliary Device Condition	Label ①	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 3-19.)

This output message is queued as input from the destination terminal. Its text must begin with the *transaction code* that causes activation of the print transaction. The name of the file to be read by this transaction at the destination terminal must be specified to the configurator via the FILES or DFILE parameter as specified for the corresponding file in the ACTION section for the originating transaction, which creates the records.

Your action issuing the SEND function directs that its output be queued as an input message from the destination terminal by setting the hexadecimal value C9 in the AUX-FUNCTION byte of the AUXILIARY-DEVICE-ID field of the OMA header; it does not use the CONTINUOUS-OUTPUT-CODE field. You must supply the configured identification of the terminal addressed in the DESTINATION-TERMINAL-ID field.

If the destination terminal is in interactive mode when your SEND function is executed, or if it already has an outstanding input to be scheduled for it, the output message you are sending cannot cause input scheduling. In this event, your action program issuing the SEND function receives an unsuccessful STATUS CODE in the PIB on return from the SEND. (Refer to 3.9.2.)

When you generate an output message and request that it be queued as input from another terminal, IMS 90 validates your header and the status of the destination terminal. Any errors encountered at this point are indicated to your originating action program by the returns made to the PIB after execution of the SEND function. However, any errors encountered in the text of the message (such as an invalid transaction code or some error found in the text by the print transaction scheduled to process the message at its destination) are reported by output to the destination terminal. The action program creating the output message is not informed of these errors by IMS 90. If your application requires such feedback to the originating terminal, you must provide for it specifically by instructions to the destination terminal operator or in your coding of the print transaction.

The message generated is queued immediately to the destination terminal. Therefore, if your action program that directs its output to be queued as input to another terminal via the SEND function terminates abnormally after issuing the SEND function, the output still generates a new transaction. In single-thread IMS 90, only one output message is created in any one action. Also, an action program running under single-thread IMS 90 and generating this type of output message must not terminate in delayed internal succession; to do so causes IMS 90 to abnormally terminate the action program. External succession can be appropriate in certain circumstances and is allowed; however, normal termination is generally considered adequate.

For a sample COBOL action program that illustrates output-for-input queueing, refer to 3.15.4.

3.10.3. Addressing a Screen Bypass Device

A UTS 400 screen bypass device is defined to ICAM as a logical terminal, but since it is physically a separate buffer that can have a telecommunications printer attached to it, it has no input medium. It is uniquely addressable for output and cannot be used to enter input. Thus, in the IMS 90 environment, the only way to access a screen bypass is to use the output-for-input queueing feature. That is, another terminal in the IMS 90 network can generate an output message via CALL SEND to be processed as input on behalf of the screen bypass device. The transaction initiated by that message can then be a simple transaction or a print transaction using continuous output. Neither interactive transactions or switched messages can be processed by a screen bypass device because both require input.

3.11. DISCONNECTING A LINE FROM AN ACTION PROGRAM

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, you must include the continuous output capability in your configuration by specifying CONTOUT=YES in the OPTIONS section. This feature is available only in a dedicated ICAM network, not a global network.

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 (ZA#PSIND field) in the PIB to C'E' and the SUCCESSOR-ID (ZA#PSID field) to C'HANGUP'.

HANGUP is an action program supplied by IMS 90 that terminates with a special code causing IMS 90 to issue a line release/line request sequence to ICAM to disconnect the line.

After the output message is sent, no further input is required from the terminal operator. IMS 90 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.

3.12. SNAPSHOT DUMP PROCESSING

IMS 90 provides a snapshot dump under four conditions:

- An action program voluntarily terminates by moving an 'S' into the 1-byte TERMINATION-INDICATOR field of the program information block (See 3.6.1.4.)

- An action program terminates due to program check

- An action program abnormally terminates due to timer-check (time-out due to loop in action program)

- An action program or subprogram issues a SNAP function call; i.e. a CALL 'SNAP' statement in a COBOL action program or the ZG#CALL macro in a BAL action program.

IMS 90 places the snapshot dump in the spool file, from which it can be printed immediately without terminating the IMS 90 job.

3.12.1. Voluntary and Abnormal Snaps

Every IMS snap dump contains header conformation regarding the terminated action program and why it failed.

The next section of the snap dump contains registers. Here you'll find one or two sets of registers depending on the reason for the snap dump.

If you voluntarily terminated your action program by moving S to the TERMINATION-INDICATOR field of the program information block, the snap dump contains one set of registers. These are IMS registers. They are of little use to an action programmer. To find the registers belonging to your action program, you must go to relative location PIB + 40, where there is a full word forward pointer. This word is the address of the SAVE area that contains your action program's registers. Go to this address and advance three full words. The next full word is register 14, then 15, then registers 0–12.

If, on the other hand, IMS terminated your action program due to a program check or time-out, the snap dump contains two sets of registers, IMS and user action program registers. The user registers are labeled so they are easily identifiable. In addition, a duplicate set of user registers can be found at location PIB + 44 (subscript 16). At this location in the program information block you'll find the 16-byte program status word (PSW) indicating the address of the instruction immediately following the one that caused the abnormal termination. Also, right after the PSW are the action program's 16 registers (0–F).

Your action program must never issue the SNAP or SNAPF supervisor macros; to do so causes IMS 90 to terminate.

3.12.2. Call Snaps

The SNAP function allows an action program to display up to six noncontiguous main storage areas in hexadecimal. Output is to the printer.

COBOL Format:

```
CALL 'SNAP' USING item-1 next-item-1[...item-6 next-item-6].
```

where:

`item-1...item-6`
Is the data name of the beginning of the area to be displayed.

`next-item-1...next-item-6`
Is the data name of the end of the area to be displayed.

BAL Format:

```
ZG#CALL SNAP (start-addr-1,end-addr-1[,...start-addr-6,end-addr-6]
```

where:

`start-addr-1...start-addr-6`
Is the start address of the area to be displayed.

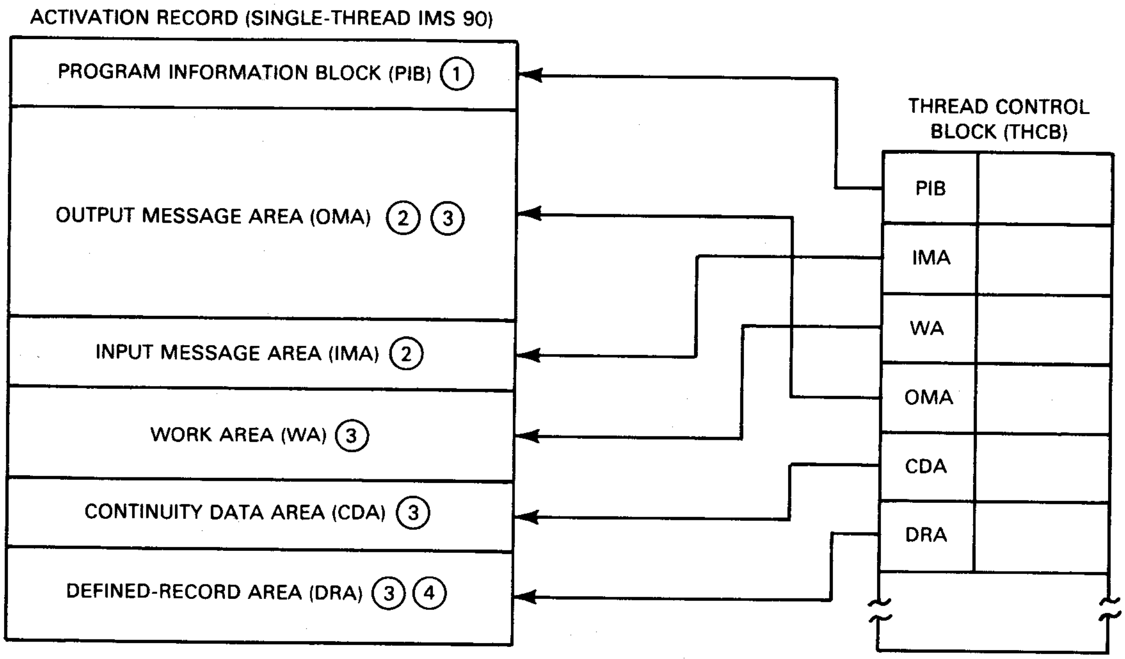
`end-addr-1...end-addr-6`
Is the end address of the area to be displayed.

Example 1 (COBOL):

```
1      8   12
------------------------------------------------
           CALL 'SNAP' USING DICE-CODES END-WS
           PIB END-PIB IMA END-IMA OMA END-OMA
```

Example 2 (BAL):

```
1        10     16
---------------------------------------
         ZG#CALL SNAP,(WSTAT1,WEND)
```


ACTIVATION RECORD (SINGLE-THREAD IMS 90)
PROGRAM INFORMATION BLOCK (PIB) ①
OUTPUT MESSAGE AREA (OMA) ② ③
INPUT MESSAGE AREA (IMA) ②
WORK AREA (WA) ③
CONTINUITY DATA AREA (CDA) ③
DEFINED-RECORD AREA (DRA) ③ ④
THREAD CONTROL BLOCK (THCB)
PIB
IMA
WA
OMA
CDA
DRA

NOTES:

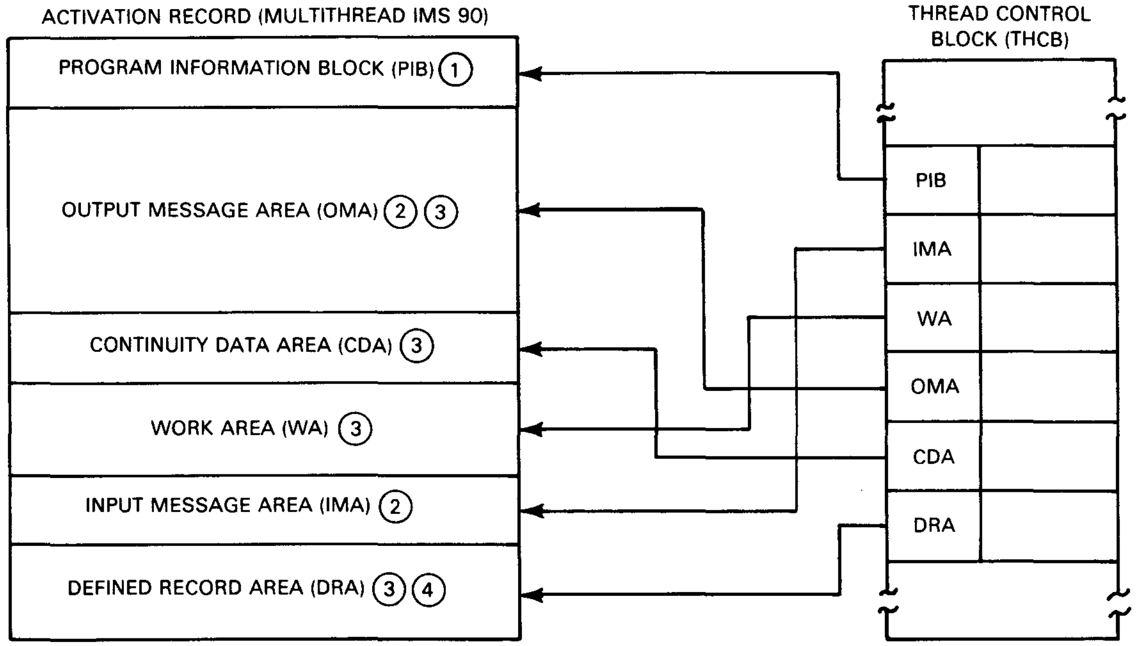
① Fixed-length area

② Contains 16-byte header in front of text

③ Optional area

④ Used only by defined record management

Figure 3—17. Single-Thread IMS 90 Activation Record Layout


ACTIVATION RECORD (MULTITHREAD IMS 90)
PROGRAM INFORMATION BLOCK (PIB) ①
OUTPUT MESSAGE AREA (OMA) ② ③
CONTINUITY DATA AREA (CDA) ③
WORK AREA (WA) ③
INPUT MESSAGE AREA (IMA) ②
DEFINED RECORD AREA (DRA) ③ ④
THREAD CONTROL BLOCK (THCB)
PIB
IMA
WA
OMA
CDA
DRA

NOTES:

① Fixed-length area

② Contains 16-byte header in front of text

③ Optional area

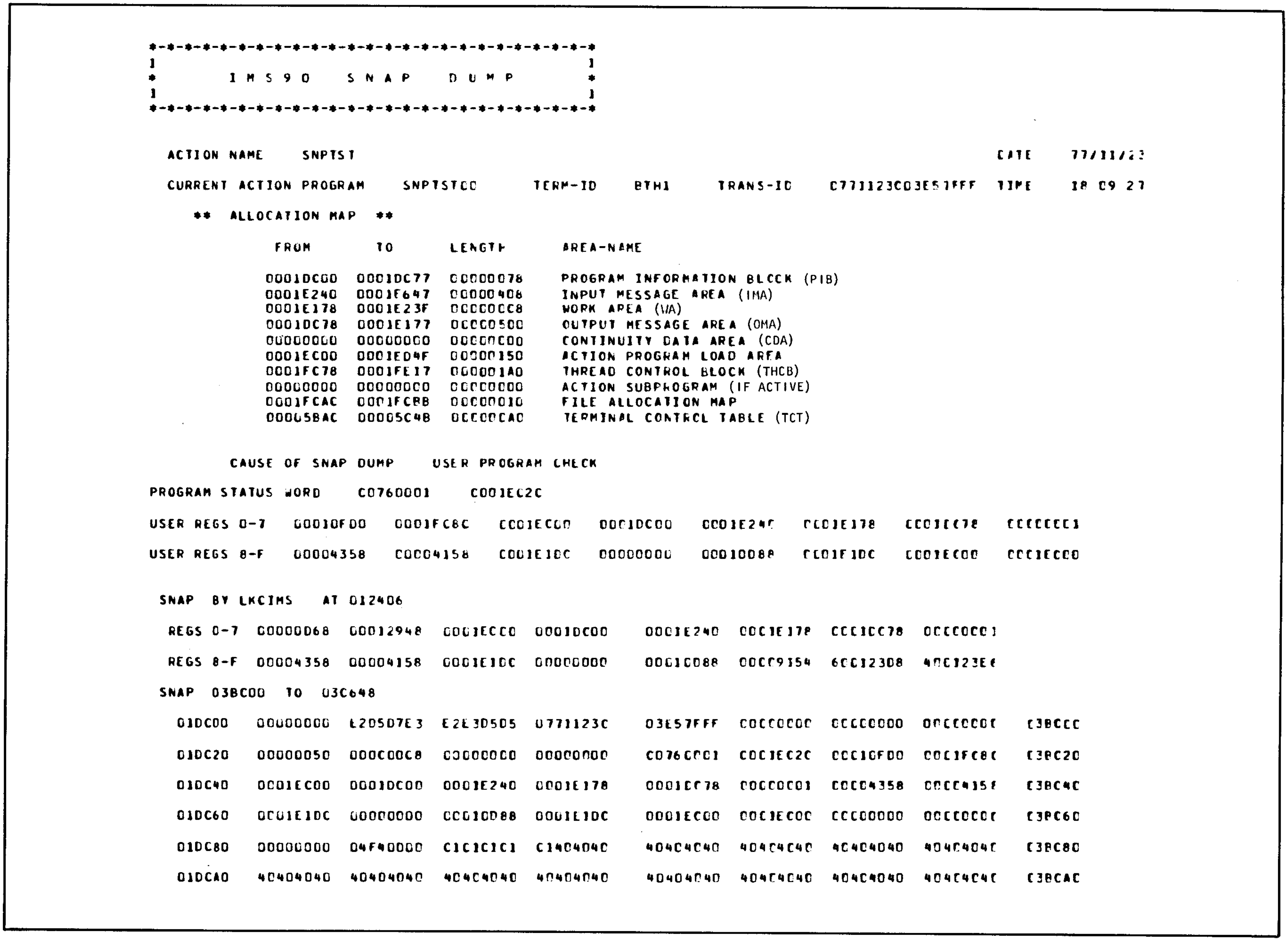
④ Used only by defined record management

Figure 3—18. Multithread IMS 90 Activation Record Layout

3.12.3. Edited Directory for Snapshot Dumps

To obtain an edited directory for snapshot dumps, multithread IMS 90 users must indicate SNAPED=YES in the OPTIONS section of the configuration. The configurator then includes the module ZG#SNAPM in the input stream to the linkage editor in creating the IMS 90 load module for online execution. If you omit the SNAPED parameter or indicate SNAPED=NO, the module ZG#SNAPM is not included in the IMS 90 load module and snapshot dumps are printed without the edited directory.

Single-thread IMS 90 users receive as a standard feature the edited directory on all snapshot dumps. Figure 3-19 illustrates a portion of a snapshot dump with an edited directory. If BASIC=YES is specified, the termination dump is not edited.



```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
1                                             1
*        I M S 9 0     S N A P     D U M P    *
1                                             1
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

  ACTION NAME    SNPTST                                                          CATE    77/11/23

  CURRENT ACTION PROGRAM    SNPTSTCC      TERM-ID    BTH1      TRANS-IC    C771123C03E57FFF  TIME    18 C9 27

    **  ALLOCATION MAP  **

           FROM        TO        LENGTH       AREA-NAME

         0001DC00  0001DC77  00000078    PROGRAM INFORMATION BLCCK (PIB)
         0001E240  0001F647  00000408    INPUT MESSAGE AREA (IMA)
         0001E178  0001E23F  000000C8    WORK APEA (WA)
         0001DC78  0001E177  00000500    OUTPUT MESSAGE AREA (OMA)
         00000000  00000000  00000C00    CONTINUITY DATA AREA (CDA)
         0001EC00  0001ED4F  00000150    ACTION PROGRAM LOAD AREA
         0001FC78  0001FE17  000001A0    THREAD CONTROL BLOCK (THCB)
         00000000  000000C0  00000000    ACTION SUBPROGRAM (IF ACTIVE)
         0001FCAC  0001FCBB  00000010    FILE ALLOCATION MAP
         00005BAC  00005C4B  000000A0    TERMINAL CONTRCL TABLE (TCT)

       CAUSE OF SNAP DUMP    USER PROGRAM CHECK

PROGRAM STATUS WORD    C0760001    C001EC2C

USER REGS 0-7    00010F00    0001FC8C    C001EC00    0001DC00    0001E240    0001E178    C001EC78    00000001

USER REGS 8-F    00004358    00004158    C001E1DC    00000000    0001008B    0C01F1DC    C001EC00    C001EC00

 SNAP  BY LKCIMS   AT 012406

  REGS 0-7  C0000D68  00012948  0001ECC0  0001DC00    0001E240  00C1E178  CCC1CC78  0CCC0CC1

  REGS 8-F  00004358  00004158  0001E1DC  00000000    0001C088  00CC9154  6CC12308  40C123E6

 SNAP  03BC00  TO  03C648

   01DC00    00000000  E205D7E3  E2E3D5D5  0771123C    03E57FFF  C0CC0C0C  CCCC0000  00CCCC0C    C3BCCC

   01DC20    00000050  000C00C8  C3000000  00000000    C076C0C1  C0C1EC2C  CCC10F00  C0C1FC8C    C3BC20

   01DC40    0C01EC00  0001DC00  0001E240  0001E178    0001EC78  C0CC0C01  C0C04358  CCCC415F    C3BC4C

   01DC60    0C01E1DC  00000000  CC010D88  0001E1DC    0001EC00  C0C1EC0C  CCC00000  0CCC0C0C    C3BC60

   01DC80    00000000  04F40000  C1C1C1C1  C140404C    404C4C40  404C4C40  4C404040  404C404C    C3BC80

   01DCA0    40404040  40404040  404C4040  40404040    40404040  404C4040  404C4040  404C4C4C    C3BCAC
```

Figure 3—19. Portion of Snapshot Dump with Edited Directory

3.13. UTS 400 DOWNLINE LOAD CAPABILITY

The Universal Terminal System 400 (UTS 400), in an IMS 90 communications environment, can perform certain functions in its own processor. The UTS 400 is particularly useful in editing and validating IMS 90 input messages, via user-written COBOL, MAC 80, or PLM programs. If any errors occur in input editing or validation, they can be handled directly at the UTS 400, without transmitting the message to the host computer.

Your UTS 400 programs must be stored in the IMS 90 load library (the library containing your online IMS 90 load module and action programs) from where they are downline loaded to the UTS 400 at your direction during online processing. You have two ways of downline loading the UTS 400:

- Enter the transaction code DLOAD to activate the IMS 90 downline load action program ZUKLOD (5.3.3).

- Write your own downline load action program (3.13.1).

Two types of UTS 400 loads may be accomplished: a load for an immediate execution that is loaded directly into the UTS 400 main storage, and a load to the UTS 400 auxiliary storage device (a cassette or a diskette).

To use this downline loading feature, you must generate a resident ICAM and must specify DLLOAD=YES in the OPTIONS section of configurator input.

The UTS 400 terminal accepting a downline load must be a master or primary UTS 400 station, never a slave station.

To use the downline loading feature, you should be familiar with the description of the UTS 400 terminal found in the fundamentals of ICAM user guide, UP-8194 (current version), and with the Universal Terminal System 400 programmer reference, UP-8359 (current version).

3.13.1. User-written Downline Load Action Programs

As an alternative to using the ZUKLOD downline load action program, you can write your own downline load action program to read blocks of UTS 400 program code from the IMS 90 load library to a UTS 400 terminal. The user-written downline load action program must contain the following:

- An 8-byte field defined for the UTS 400 load-module-name. The module-name must be moved into this field in the OMA that is being downline loaded before the SETLOAD function is called.

- One SETLOAD function call for each downline load is required. The SETLOAD function must be issued before any GETLOAD function call because initialization must occur before a block of code can be read from a UTS 400 load module.

- GETLOAD function calls issued to read blocks of code from the UTS 400 load module into the data buffer in the OMA.

- A 400-byte area defined on the word boundary in the CONTINUITY-DATA-AREA. This area is used as a work area by the SETLOAD and GETLOAD function calls.

- The data-buffer and 2-byte field indicating SIZE defined in the GETLOAD function call. The data-buffer contains a block of code read from the load module.

  Before the GETLOAD function is called, the size field should have the length of the buffer area in binary format. After the return from the GETLOAD call, the size field will have the number of bytes actually moved into the buffer area. This number will also be in the binary format.

After the GETLOAD function call, the user downline load program must:

- Check for end-of-file (02) in the STATUS-CODE field of the PIB.

- Process the status code in the PIB for successful completion of the GETLOAD function call.

If the GETLOAD function is successful, the user downline load program should:

- Move 'C' to the AUX-FUNCTION field (the first byte of the AUXILIARY-DEVICE-ID field) of the output message header.

- Prefix the data block received from the GETLOAD function call with a proper heading to load this block either directly into the UTS 400 main storage or to an auxiliary storage device. This prefixed data block becomes the text in the downline load program's OMA whose length can be calculated using the length returned in the SIZE parameter of the GETLOAD function call.

  An example of the OMA and the prefixing operations required to format the text part of the OMA for immediate execution might look as follows:

```
01  OUTPUT-MESSAGE-AREA COPY OMA.

    02  DOWNLINE-LOAD-MESSAGE.

        03  DOWNLINE-LOAD-HEADER PIC X(6).

        03  DOWNLINE-LOAD-TEXT PIC X(1000).
```

  The user downline load action program should move the 6-byte prefix, X'1B0E30323130', into DOWNLINE-LOAD-HEADER to provide the header information for loading the UTS 400 main storage.

  If the downline load is intended for the auxiliary storage device, the user action program should instead move X'1313nnnnnnnn' into DOWNLINE-LOAD-HEADER. Here 'nnnnnnnn' is a 4-character sequence naming the UTS 400 load program.

- Send the message from the user-written downline load action program OMA to the UTS 400 terminal using the continuous output feature.
- Terminate the downline load action program with external succession (i.e., place 'E' in the TERMINATION-INDICATOR of the PIB) 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 as described in 3.10.1.4. This includes testing the delivery notice for error and if an error occurs, moving an error message to the OMA before terminating the program normally (TERMINATION-INDICATOR=N in the PIB).

If the SETLOAD or GETLOAD function is unsuccessful and ERET=YES is included in the PROGRAM section of the configurator, the user downline load action program receives control with error indications set in the STATUS-CODE field of the PIB. For status code settings in this case, see status codes 3 and 4 in 3.13.1.1 and 3.13.1.2. The action program should then send an appropriate error message to the terminal.

If the SETLOAD or GETLOAD function is unsuccessful and ERET=YES is not configured, IMS 90 cancels the transaction and sends the following message to the terminal:

```
DOWN LINE LOAD ERROR.
```

If the GETLOAD function returns an end-of-file condition (STATUS-CODE set to X'02' in the PIB), the buffer area will contain the transfer record. This is the last block that should be sent to the UTS 400; thus, no more GETLOAD functions should be issued for this load module. If the blocks of code are sent to the main storage for the immediate execution of the program, then when the UTS 400 terminal receives a transfer record it automatically transmits a response (input message) indicating whether or not the downline load was successful. Therefore, the user-written 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 will then receive in its IMA the 24-byte message header from the UTS 400 in the following formats:

24 bytes

Terminal-id	Date/time Stamp	Text-length	unused	10	01	01	01	39	30	30	30
4	8	2	2	DICE							

Successful load

24 bytes

Terminal-id	Date/time Stamp	Text-length	unused	10	01	01	01	39	30	34	*
4	8	2	2	DICE							

Unsuccessful load

Table 3-21 defines the various error bit configurations (*) that can be returned in the last byte of the message from the UTS 400.

Table 3—21. Rejected Load Error Byte Definition

Bit Number*	Error Type	Probable Cause/Recovery
7	Never set	
6	Always set	
5	LOADFL contents (AA_{16}) prevents loading	This value (AA_{16}) was not cleared by a UTS 400 program which had been executed previously. The UTS400 operator should initiate a 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 400 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	The OS/3 downline load procs should prevent this occurrence
2	Block overflowed available/ assigned main storage	If main storage is available, the UTS 400 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	
0	Addresses A and B not equal	The OS/3 downline load procs should prevent this occurrence

*Numbered from right to left; i.e. bit 7 is the most significant bit, bit 0 is the right-most or least significant bit.

NOTE:

If the user-written downline load action program is configured with EDIT=tablename or EDIT=c in its ACTION section (or if the EDIT parameter is omitted), the DICE characters X'10010101' are stripped from the message before it is sent to the action program.

An example of the IMA might look as follows:

```
01  INPUT-MESSAGE-AREA COPY IMA.
    02  UTS400-RESPONSE-MESSAGE.
        03  UTS400-RESPONSE-DICE         PIC X(4).
        03  UTS400-RESPONSE              PIC X(4).
```

The user-written downline load action program should:

- Interrogate the response message and send an appropriate output message to the terminal indicating the success or failure of the downline load.
- Terminate with normal termination; i.e., place 'N' in the TERMINATION-INDICATOR of the PIB.

If the action program downline loads a program to a local storage device, the UTS 400 terminal will not generate a response message after the last block of code is sent to the device. Therefore, the status of the downline load will not be known until the program is read from the local storage device into main storage.

3.13.1.1. Downline Load Initialization

The first function called by a downline load action program is the SETLOAD function:

COBOL Format:

```
CALL 'SETLOAD' USING {module-name} .
                     {work-area  }
```

BAL Format:

```
{CALL    } SETLOAD,(module-name,work-area)
{ZG#CALL }
```

where:

module-name
: Is an 8-byte field containing the name of the UTS 400 program load module to be downline loaded. The UTS 400 program must be in the same load library as your action programs and the online IMS load module.

work-area
: Is a 400-byte area defined in the CONTINUITY-DATA-AREA. This area must be word-aligned.

Status Codes	Detailed Status Codes	
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 transfer record is received from the GETLOAD call.

3.13.1.2. Downline Load Processing

The GETLOAD function is called by the downline load program immediately after the SETLOAD function. It should be called repeatedly until end-of-file is reached for the UTS 400 load module.

COBOL Format:

```
CALL 'GETLOAD' USING(work-area  ).
                    {buffer-area}
                    (size       )
```

BAL Format:

```
{CALL    }GETLOAD,(work-area,buffer-area,size)
{ZG#CALL }
```

where:

work-area
Is the area previously defined in the SETLOAD function.

buffer-area
Is the data-buffer in the OMA where the user expects a block of code from his load module.

size
Is a 2-byte field where the length (size) of the buffer-area is stored.

Status Codes	Detailed Status Codes	
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; work-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.

3.14. SCREEN FORMATTING SERVICES

Screen formatting services is a method of displaying predefined formatted screens at a terminal. These predefined screens simplify the action programmer's output formatting job and can also aid the terminal operator in his data entry procedure. IMS 90 places the predefined screen formats in your action programs via CALL functions in BAL and COBOL programs or via output format specifications in RPG II programs. You can display screen formats on the following devices:

- UNISCOPE 100 (must have protection feature)
- UNISCOPE 200 (must have protection feature)
- UTS 400 (if operating in native mode, must have PROTECT/FCC switch set to FCC and control page set to XMIT VAR)
- UTS 20
- Workstations

You indicate use of screen format services by including the SFS parameter in the OPTIONS section of your IMS 90 configuration. (See IMS system support functions user guide/programmer reference, UP-8364 (current version).)

You predefine your screen formats offline from IMS 90 via the OS/3 screen format generator. (Refer to screen format services concepts and facilities user guide/programmer reference, UP-8802 (current version).) In your screen format description, you assign a format name to each screen format and the screen format generator (SFG) places it in the format library $Y$FMT. $Y$FMT is a MIRAM disk file where all the screen formats used by your action program reside. Only one screen format file is supported per execution of IMS 90.

When you use screen formats, your IMS 90 job control stream must include a device assignment set for each device type using screen formatting. These device assignments define where the formats used by your action programs reside. You must use an LFD name of TC01FMTF for the first terminal class, TC02FMTF for the second, etc. In addition, this job control stream must include a //OPTION OFT=+n statement, where n specifies the number of device types. This number should agree with the number specified on the SFS configurator parameter. (See the IMS system support functions user guide/programmer reference, UP-8364 (current version).)

Note that action programs using screen formatting services may not run in a batch (online or offline) environment; however, the IMS screen format services interface supports the writing of formatted screens to auxiliary devices on applicable terminals.

You must put the AUX-DEVICE-NO field and AUXILIARY-FUNCTION field in the output message area header before building the screen format. You cannot specify the AUX-DEVICE-NO field with an error screen because it causes unpredictable output on the auxiliary device.

Table 3-22 illustrates screen format services support of auxiliary devices.

Table 3—22. Screen Format Services Support of Auxiliary Devices

Input/Output Options			Contents of AUX-FUNCTION Field				Auxiliary Devices			
Name	Suppression	Inhibit Space Suppression	Continuous Output		No Continuous Output		UTS 400		UNISCOPE 100/200	
			Hex	Character	Hex	Character	Supported	Not Supported	Supported	Not Supported
Print Mode	X		F3	3	F0	0	X (recommended)①③		X (recommended)①	
		X	F5	5	F2	2	X (recommended)①③			X (unpredictable output at screen and auxiliary device)
Print Transparent	X		F7	7	F4	4	X ②③		X ②	
		X	F9	9	F6	6	X ②③			X (unpredictable output at screen and auxiliary device)
Print Form (ESC H)	X		C1	A	D1	J	X ④			X ⑥
		X	C6	F	D6	O	X ④			X ⑥
Transfer All (ESCG)	X		C2	B	D2	K	X (recommended)⑤			X ⑥
		X	C7	G	D7	P	X ⑤			X ⑥
Transfer Variable (ESC F)	X		C4	D	D4	M	X ④			X ⑥
		X	C8	H	D8	Q	X ④			X ⑥
Transfer Changed (ESC E)	X		C5	E	D5	N		X (field control characters not supported)		X ⑥
		X	E8	Y	F8	8		X (field control characters not supported)		X ⑥

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

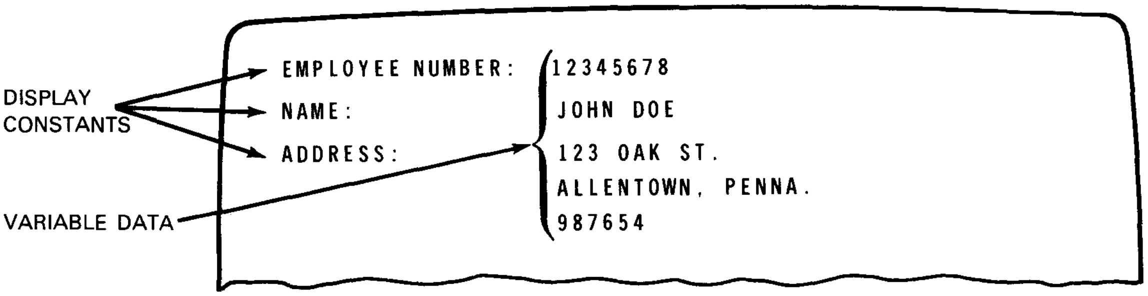
⑤ cassette/diskette - same format as screen; records all fields and all field control characters

⑥ cassette/diskette - not available

3.14.1. How IMS 90 Handles Screen Formatted Messages

When your action program requests a screen format, IMS 90 retrieves the screen format you name and places it in an action program-specified buffer (usually the output message area (OMA)). You must reserve the first 16 bytes of your buffer area for an output message header.

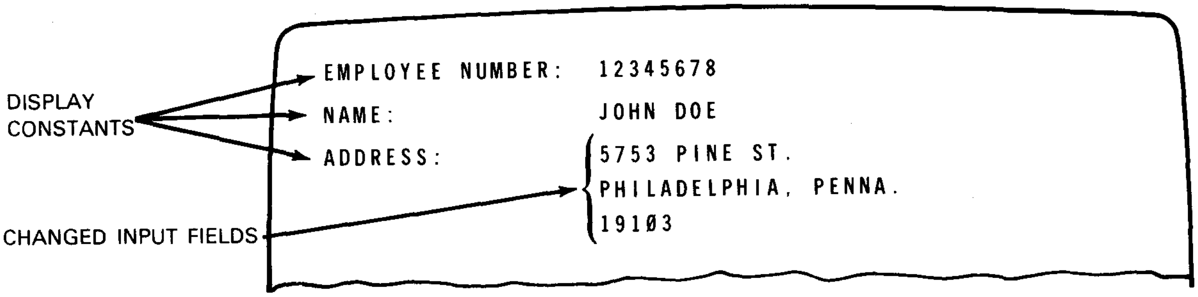
The screen format coordinator places the output display constants of the format into their respective locations within the screen buffer. These constants are always protected. IMS 90 inserts into the format buffer the variable data you supply in your action program. Variable data is all the data to be displayed on the terminal except the display constants. Figure 3-19A shows an output screen containing display constants and variable data.



```
EMPLOYEE NUMBER:  12345678
NAME:             JOHN DOE
ADDRESS:          123 OAK ST.
                  ALLENTOWN, PENNA.
                  987654
```

Figure 3—19A. Output Screen Format with Display Constants and Variable Data

Any field you define as input or both input and output in your action program is an unprotected field. This means that the terminal operator is free to change that field when making data entries on the screen format. When building your screen buffer, if you define a variable data field as output only, it is protected. Figure 3-19B shows an input screen containing the input fields (address) that the terminal operator has changed.



```
EMPLOYEE NUMBER:  12345678
NAME:             JOHN DOE
ADDRESS:          5753 PINE ST.
                  PHILADELPHIA, PENNA.
                  191Ø3
```

Figure 3—19B. Input Screen Format with Display Constants and Changed Input Fields

When your action program terminates with delayed internal succession or continuous output, IMS 90 forces the format to be output only. Also, you must use an output only format for any formatted output message being switched to a terminal other than the source terminal. A message wait light entered to retrieve a switched message cancels any screen format currently effective at the terminal. Also, function keys passed to the action program cancel any screen format currently effective at the terminal. Your action program may send multiple formatted messages to the originating terminal; however, only the last format may be used for the subsequent input format.

When the terminal operator fills in input data on the screen format, the data entered is validated before IMS 90 passes control to your action program. IMS 90 first checks the message for terminal command input before requesting the screen format coordinator to validate the input message. If the input message contains a terminal command other than ZZRSD, IMS 90 processes it accordingly and cancels any screen format currently effective at the terminal. ZZRSD causes the last output message to be resent, thus retaining the current screen format.

To schedule an action program to receive the formatted input data, you must terminate the CALL BUILD action program with external succession (E). If you terminate the CALL BUILD action program with normal succession, the first input field must contain a valid transaction code.

If an input message contains a transaction code, in the first five bytes, IMS 90 verifies the code and if it is invalid, sends the input message back to the terminal and causes the transaction code to blink. This action does not cancel the effective screen format.

When the input message does nt contain a terminal command or invalid transaction code message, IMS 90 requests the format coordinator to validate the message. If the input data filled in by the terminal operator is valid, IMS 90 places only that data into your successor action program's IMA. IMS 90 does not perform any other editing (simple editing or expanded input editing) on this input even if configured for the related action. Your action program can continue processing, possibly writing the new input to a data file. If there are input errors, IMS 90 allows the terminal operator to correct the inputuntil the retry count specified at screen format generation time is exhausted. (See screen formatting concepts and facilities user guide/programmer reference, UP-8802.)

Once the retry count is exhausted, the successor action program receives control. At that time, the PIB contains a status code of 7 and a detailed status code of 0. (See 3.14.2.1.)

NOTE:

If you want to terminate with normal succession after you output an input/output screen format, you must enter a valid transaction code in the first input field.

3.14.2. Processing Screen Formatted Messages with COBOL and BAL Action Programs

When you are ready to display a screen format on a terminal, your action program must first move the destination terminal-id into the first 4 bytes of the output message header and the output area length into the text length field of the output message header before issuing a BUILD function call statement. The output area length must be large enough to hold the format constructed by the screen format coordinator. To determine this value, see the formula described on the OUTSIZE parameter of the configurator ACTION Section in the system support functions user guide/programmer reference, UP-8364 (current version).

IMS 90 passes the format name you supply on the BUILD function to the screen format coordinator, which retrieves that screen format from the format file ($Y$FMT). IMS 90 then places the screen format into the output area supplied by your action program. Your action program may add variable data to the screen format by supplying additional parameters on the CALL BUILD function. Issuing a CALL RETURN or CALL SEND then sends that screen message to the terminal. The terminal operator then enters data that is verified and stored in your successor action program's input message area (IMA). Your successor action program may do further validation before processing the input data. Figure 3-20 illustrates message processing using the screen format services.

NOTE:

The LOW-VALUES figurative in COBOL is not valid output data for variable output fields in a CALL BUILD function. It translates to binary zeros. The HIGH-VALUES figurative constant, however, causes an input field to blink in a CALL REBUILD function because it translates to binary ones.

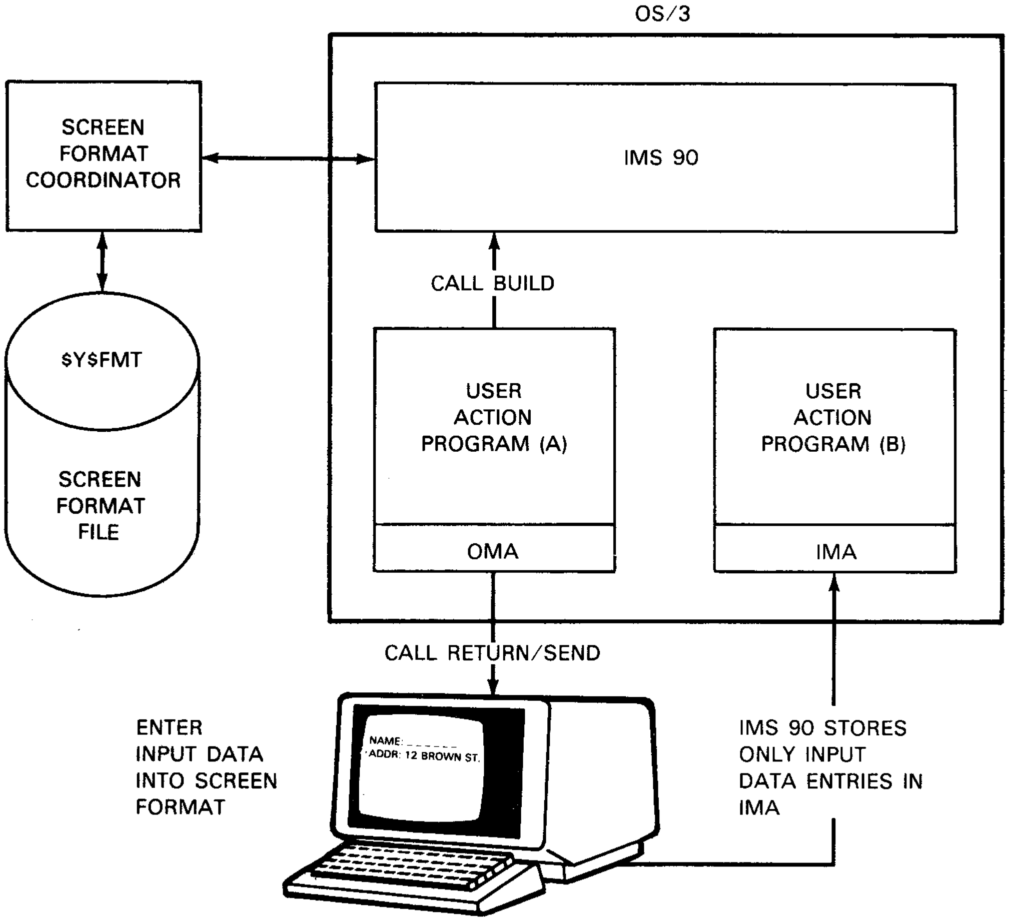


Figure 3—20. Processing Screen Formatted Messages with COBOL or BAL Action Programs

If you include a variable data area on the BUILD function, you may also include an output-status area large enough to hold one status byte for each variable data field. The screen format coordinator uses this area, when specified, to fill in error codes when it finds errors in the output validation of the variable data. When a validation error occurs, the screen format coordinator places X'FF' in the error field in your variable data area and one of the following field status error codes into the status byte for the invalid field.

Output Validation Error Codes	Cause of Error
01	Nonnumeric keyin to a numeric field or subfield
02	Nonalphabetic keyin to an alphabetic field or subfield
05	Range check failure
06	Field not in packed decimal format

After your action program issues the BUILD function, the program information block (PIB) status and detailed status code fields reflect the status of the BUILD function. (See 3.14.2.1 for status code explanations.) A status code of zero means the function call was successful and no output validation errors occurred; therefore, the format area (OMA) will contain a constructed output screen buffer.

Once the action program has issued the BUILD function, do not change the contents of the output message area that contains the output header, screen format, and variable data. Modification of this output message area may cause unpredictable results since:

- The information contained in the output header is used by screen format services to construct an appropriate screen format.
- Screen format services rely on the format structure being the same when received at input time as it was when presented to the action program at BUILD function time.

To send the screen buffer contents (i.e., format and optional variable data) to the terminal, your action program issues a SEND or RETURN function. If this screen buffer terminal display contains at least one fill-in field for data input (i.e., the format is input or input/output), this screen format remains in effect at the terminal until the following input. If, however, you are not expecting input from the formatted screen, i.e., the terminal operator clears the screen before entering the next input message, you must clear the SFS-OPTIONS field in the COBOL output message header or the ZA#OSFSO field in the BAL output message header. Use the following statement in COBOL to clear the SFS-OPTIONS field:

```
MOVE LOW-VALUES TO SFS-OPTIONS.
```

In basic assembly language, the following statement does the same thing:

```
1        10    16                                                        72
--------------------------------------------------------------------------
         MVI   ZA#OSFSO,X'00'
```

(See Figures 3-10 and 3-11.)

Suppose the terminal operator wants to enter data from the terminal. If the data he enters is valid, that data is placed in the successor action program's IMA. If he enters invalid data, however, he is allowed to correct his entry until the retry count specified at screen format generation time is exhausted. In this case, IMS 90 automatically blinks the invalid field entered at the terminal.

Once the retry count has been exhausted, IMS 90 schedules your successor action program whose IMA contains the input data entered from the terminal. The input data is followed by one status byte for each input field, indicating that the field is valid. When an input validation error is detected, one of the following field status error codes is entered in the status byte for the invalid field in your successor action program's IMA:

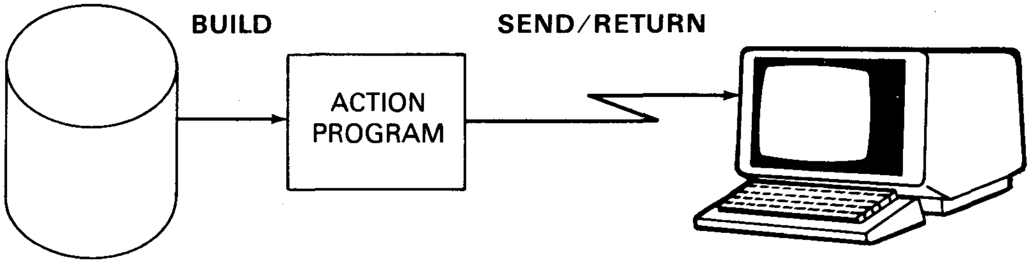
Input Validation Error Codes	Cause of Error
01	Nonnumeric keyin to a numeric field or subfield
02	Nonalphabetic keyin to an alphabetic field or subfield
03	Correct number of characters not entered
04	Decimal point alignment error
05	Range check failure

In addition, the screen format coordinator places hexadecimal F's into the invalid data fields in your IMA. Note that the length field in the input message header indicates only the length of the input data items and does not include their status bytes. Once IMS 90 has indicated invalid fields, you may want to send a general error message to the terminal operator and terminate the transaction.

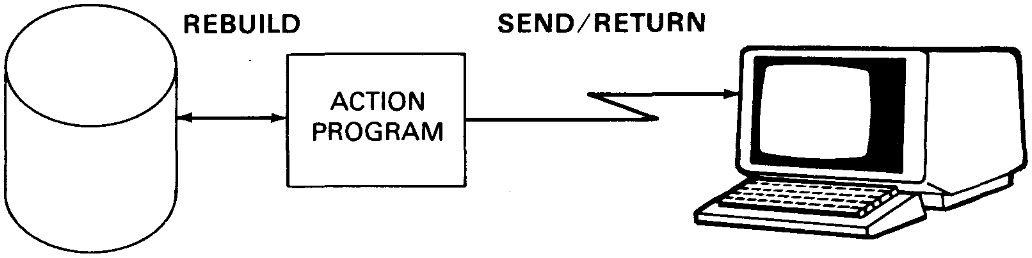
Your action program can validate input data on a more detailed level than the screen format coordinator. When the terminal operator enters input data that your program determines is valid, you can issue the REBUILD function to construct an error screen format. Before you issue a REBUILD function call, however, your program must place the output area length into the output message header and replace invalid input data with hexadecimal F's.

When you issue the REBUILD function, the screen format coordinator replaces any fields containing hexadecimal F's with the appropriate blink characters as it constructs the error screen format in the OMA. Finally, when your action program issues a subsequent RETURN or SEND function, the field in error blinks on the screen format at the terminal and all other fields remain unchanged.

In summary, your BAL or COBOL action program issues two function calls to send a formatted output message to the terminal. The BUILD and the SEND/RETURN functions construct the screen buffer contents and transmit them to the terminal.



The REBUILD and SEND/RETURN functions reconstruct erroneous screen buffer contents for transmission to the terminal.



3.14.2.1. Building a Screen Buffer (BUILD)

Your action program must issue a BUILD function call to construct a screen buffer in your action program's OMA. On return, the screen buffer always contains the display constants placed there by the screen format coordinator and may optionally contain variable data merged with the screen format. The format of the BUILD function call follows:

COBOL Format:

```
CALL 'BUILD' USING buffer-address format-name
  [variable-data data-size [output-status]].
```

BAL Format:

```
{CALL    } BUILD, ( buffer-address,format-name[variable-data,data-size )
{ZG#CALL }        (    [,output-status]]                               )
```

where:

buffer-address
Is the address of an output area (usually the OMA) into which the constructed screen is placed. This area must be full-word aligned and begin with a 16-byte output message header with the destination terminal-id and output area length fields set as indicated on the OUTSIZE parameter. (See 3.14.2.)

format-name
Is the address of an area containing the 8-byte format name that identifies the desired format.

variable-data
Is the address of an area containing a string of variable data to be merged with called format. When this parameter is not specified, only the screen format constants are sent to the terminal as an aid to the operator entering input data. If you use this paramter and do not describe output fields in your action program (this is an input-only format), the variable-data parameter is invalid and IMS 90 returns an error status in the PIB. There are no default values that can be generated by the screen format generator for output fields that can be used at CALL BUILD function time. Therefore, you must supply variable data for all of the output fields.

If you want to have default values for some or all of your input fields, generate the fields as being both input and output fields, and supply output in the variable output data area at CALL BUILD function time.

data-size
Is the address of a half word containing the length of the variable data area. This parameter is required if you specify a variable data area.

output-status

Is the address of the area into which the screen format coordinator places status errors found in the output validation of variable data. This parameter is valid only if the variable-data parameter is used. If omitted, output validation is not performed.

If the BUILD function call is unsuccessful, no screen buffer is constructed and IMS 90 sends one of the following status and detailed status codes to the PIB of your action program.

Status Code	Detailed Status Code	Reason for Error
1	–	Supplied format name could not be found
3	1	Incorrect number of parameters
3	3	Invalid parameter value (e.g., an address not within action program's activation record)
3	12	SFS not configured
6	4	Invalid terminal name
7	0	Validation error; all error fields within variable data area are replaced by hexadecimal F's and affected field error statuses are set in the output-status area.
7	1	Format area (output buffer) not large enough*
7	2	Variable data area not large enough
7	3	Insufficient number of terminal classes
7	4	Variable data specified for input format
7	5	Format width greater than screen width
7	6	Fatal error (e.g., I/O error)

3.14.2.2. Creating an Error Formatted Screen (REBUILD)

You use the REBUILD function to construct an error screen format. The REBUILD function is used more effectively when your action program needs to provide more detailed input data validation than the screen format coordinator provides via the input validation error codes to the IMA (3.14.2). The format of the REBUILD function call follows:

**When IMS 90 returns this error status, the length field in the output message header portion of your format area will contain the actual length required for this format.*

COBOL Format:

```
CALL 'REBUILD' USING buffer-address variable-data.
```

BAL Format:

```
{CALL   } REBUILD,(buffer-address,variable-data)
{ZG#CALL}
```

where:

`buffer-address`
: Is the address of an output area (usually the OMA) into which the constructed screen is placed. This area must be full-word aligned and begin with a 16-byte output message header as specified for the CALL BUILD function.

`variable-data`
: Is the address of an area containing a string of fields including error fields into which the screen format coordinator or action program has placed hexadecimal F's. This is usually part of the IMA and contains all of the input data keyed in by the terminal operator.

Before issuing a REBUILD function call to construct the error screen format, you must place the output area length in the output message area. In addition, your program must replace invalid input data fields with hexadecimal F's in the IMA.

When you issue the REBUILD function call, the screen coordinator replaces erroneous input data fields (hexadecimal F's) with blink characters and constructs an error screen format in the OMA. A subsequent RETURN or SEND function call sends the error screen format to the terminal operator indicating the fields that need correction.

If the REBUILD function call is unsuccessful, no error format is constructed and IMS 90 sends one of the following status and detailed status codes to the PIB of your action program:

Status Code	Detailed Status Code	Reason for Error
1	–	Supplied format name could not be found
7	1	Format area not large enough
7	5	Format width greater than screen width
7	6	Fatal error (e.g., I/O error)
7	7	Rebuild not allowed
7	8	Invalid field
7	9	No error field detected

3.14.3. Processing Screen Formatted Messages in RPG II Action Programs

To use screen formatting services in your RPG II action program, you follow the same procedure in coding the control card and file description specification forms as you do for any other RPG II action program.) For example, code the letter A in column 74 of the control card specifications to designate an action program and identify the program as usual in columns 75 through 80. On the file description specification you name and describe the user and IMS 90 files. The input format specifications describe the IMS 90 IMA file and user variable record fields associated with terminal input to a successor action program. When you enter data from the terminal and pass it to a successor action program, you must describe these input fields on the input format specifications. The calculation specifications retrieve the user record for inclusion in the screen format buffer (your action program's OMA). Finally, the output format specifications name the screen formats your action program uses and any variable data required by the screen format. Only one screen format is allowed for each output record. You identify the screen format on the first field description for each output record and follow it with the description of any variable data required by each named screen format.

Although RPG II action programs normally do not use a work area, when you use screen formats you must specify a WORKSIZE keyword parameter in the ACTION section of the configuration equal to the size of the variable output data.

When the terminal operator keys in the transaction code for your action program, IMS 90 loads your RPG II action program, which moves variable data to the OMA. (You must allow 16 bytes in the OMA for the header that immediately precedes the variable output data.) Subsequently, IMS 90 transmits the entire screen format including your variable data to the terminal.

The terminal operator may then enter data, which is verified and stored in your successor action program's input message area (IMA). Figure 3-20A illustrates the processing of screen formatted messages for RPG II action programs.

Once your action program builds the screen format, do not change the contents of the output message area that contains the output header, screen format, and variable data. Modification of this output message area may cause unpredictable results since:

- Screen format services uses the information contained in the output header to construct an appropriate screen format.

- Screen format services relies on the format structure being the same when received at input time as it was when presented to the action program.

If IMS 90 cannot build the screen buffer, it sends one of the following status and detailed status codes to the PIB of your action program, which you can examine when you get a snap dump.

Status Code	Detailed Status Code	Reason for Error
1	-	Supplied format name could not be found
3	12	SFS not configured

Status Code	Detailed Status Code	Reason for Error
6	4	Invalid terminal name
7	0	Validation error; all error fields within variable data area are replaced by hexadecimal F's.
7	1	Format area (output buffer) not large enough
7	2	Variable data area not large enough
7	3	Insufficient number of terminal classes
7	4	Variable data specified for input format
7	5	Format width greater than screen width
7	6	Fatal error (e.g., I/O error)

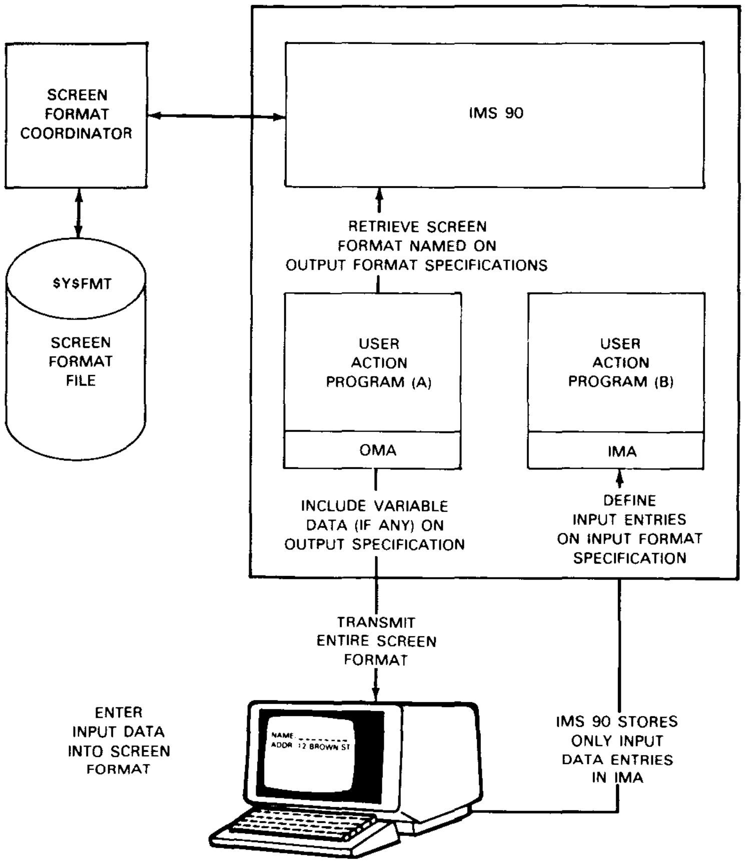
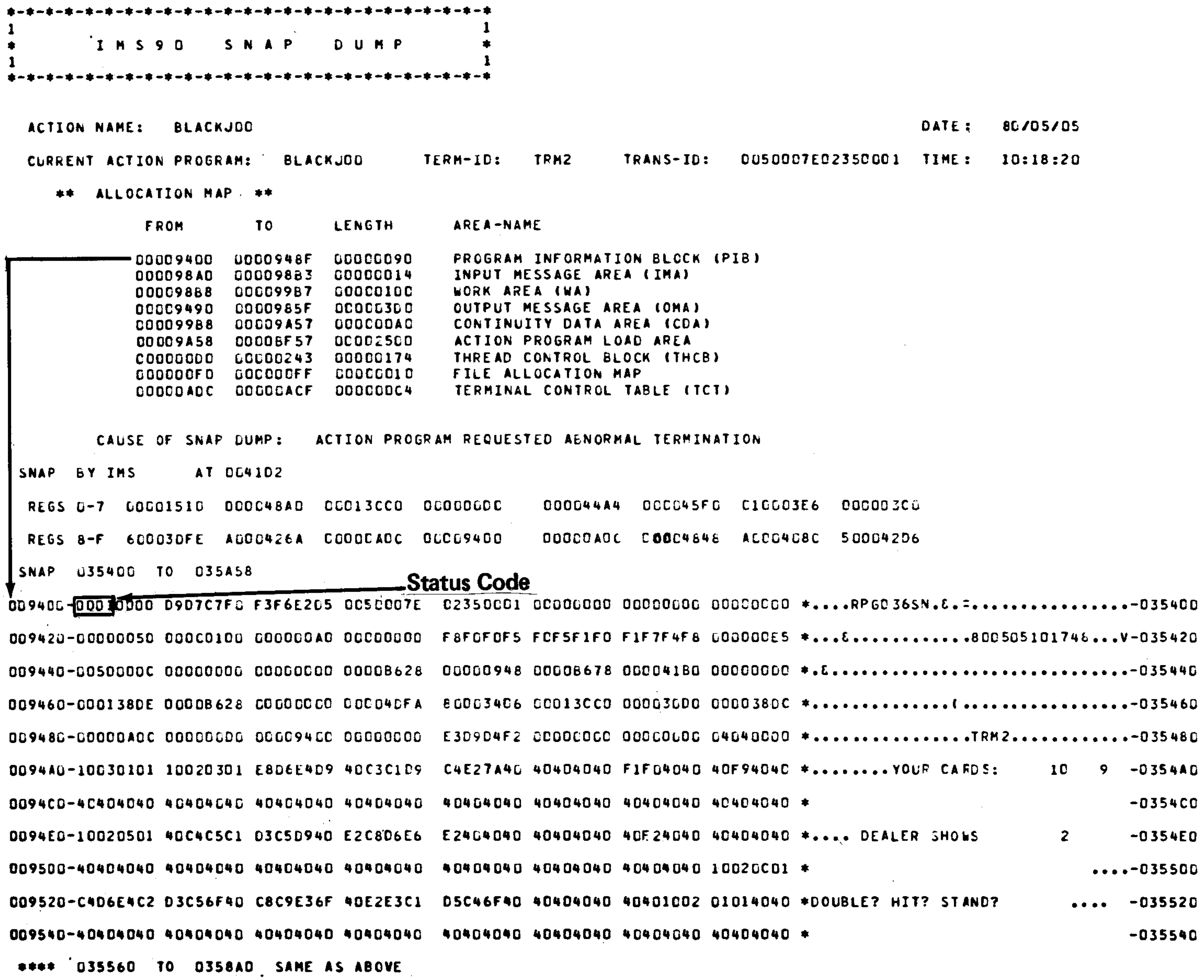

SCREEN FORMAT COORDINATOR
IMS 90
YFMT
SCREEN FORMAT FILE
RETRIEVE SCREEN FORMAT NAMED ON OUTPUT FORMAT SPECIFICATIONS
USER ACTION PROGRAM (A)
USER ACTION PROGRAM (B)
OMA
IMA
INCLUDE VARIABLE DATA (IF ANY) ON OUTPUT SPECIFICATION
DEFINE INPUT ENTRIES ON INPUT FORMAT SPECIFICATION
TRANSMIT ENTIRE SCREEN FORMAT
ENTER INPUT DATA INTO SCREEN FORMAT
NAME
ADDR 12 BROWN ST
IMS 90 STORES ONLY INPUT DATA ENTRIES IN IMA

Figure 3—20A. Processing Screen Formatted Messages with RPG II Action Programs

Figure 3-20B shows a sample snap dump for an RPG II action program that requested an invalid screen format name. The first 2 bytes of the PIB indicate a status code of 01_{16}, i.e., supplied format name was not found.



```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
1                                                   1
*       I M S 9 0    S N A P    D U M P             *
1                                                   1
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*


  ACTION NAME:   BLACKJ00                                                        DATE:    80/05/05

  CURRENT ACTION PROGRAM:   BLACKJ00      TERM-ID:   TRM2     TRANS-ID:   0050007E02350001  TIME:    10:18:20

     **  ALLOCATION MAP  **

             FROM        TO        LENGTH      AREA-NAME

            00009400  0000948F  00000090    PROGRAM INFORMATION BLOCK (PIB)
            000098A0  000098B3  00000014    INPUT MESSAGE AREA (IMA)
            000098B8  000099B7  00000100    WORK AREA (WA)
            00009490  0000985F  000003D0    OUTPUT MESSAGE AREA (OMA)
            000099B8  00009A57  000000A0    CONTINUITY DATA AREA (CDA)
            00009A58  0000BF57  00002500    ACTION PROGRAM LOAD AREA
            00000000  00000243  00000174    THREAD CONTROL BLOCK (THCB)
            000000F0  000000FF  00000010    FILE ALLOCATION MAP
            00000ADC  00000ACF  000000C4    TERMINAL CONTROL TABLE (TCT)


        CAUSE OF SNAP DUMP:   ACTION PROGRAM REQUESTED ABNORMAL TERMINATION

 SNAP  BY IMS      AT 004102

  REGS 0-7  00001510  000048A0  00013CC0  0000000C    000044A4  000045F0  C10003E6  000003C0

  REGS 8-F  60003DFE  A000426A  00000ADC  00009400    00000ADC  C0004848  A0004C8C  500042D6

 SNAP  035400  TO  035A58
009400-00010000 D9D7C7F0 F3F6E2D5 0050007E  02350001 00000000 00000000 00000000 *....RPG036SN.&.=................-035400

009420-00000050 000C0100 000000A0 00000000  F8F0F0F5 F0F5F1F0 F1F7F4F8 000000E5 *...&............800505101748...V-035420

009440-0050000C 00000000 00000000 0000B628  00000948 0000B678 000041B0 00000000 *.&..............................-035440

009460-000138DE 0000B628 00000000 00004DFA  80003406 00013CC0 00003D00 0000380C *.............(..................-035460

009480-00000A0C 00000000 000094C0 00000000  E3D9D4F2 00000000 00000000 04040000 *................TRM2............-035480

0094A0-10030101 10020301 E8D6E4D9 40C3C1D9  C4E27A40 40404040 F1F04040 40F9404C *........YOUR CARDS:     10   9  -0354A0

0094C0-40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040 *                                -0354C0

0094E0-10020501 40C4C5C1 D3C5D940 E2C8D6E6  E2404040 40404040 40F24040 40404040 *.... DEALER SHOWS         2      -0354E0

009500-40404040 40404040 40404040 40404040  40404040 40404040 40404040 10020C01 *                            ....-035500

009520-C4D6E4C2 D3C56F40 C8C9E36F 40E2E3C1  D5C46F40 40404040 40401002 01014040 *DOUBLE? HIT? STAND?       ....  -035520

009540-40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040 *                                -035540
 ****  035560  TO  0358A0  SAME AS ABOVE
```

Status Code

Figure 3—20B. Snapshot Dump with PIB Status Code 01 (Screen Format Not Found)

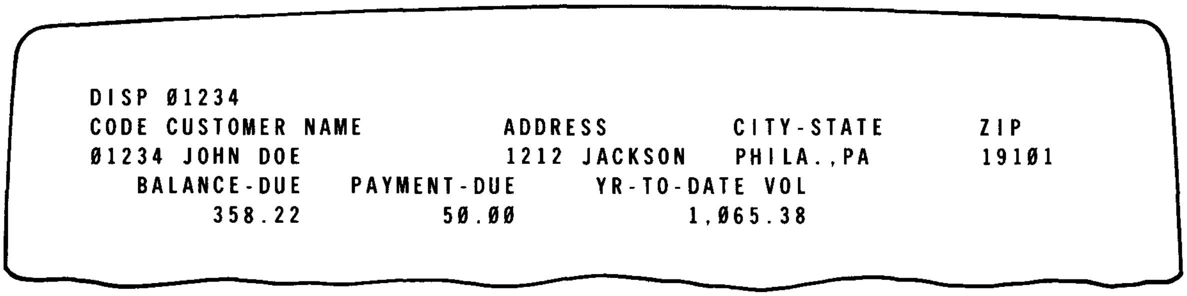
You can construct an error screen format to notify the terminal operator when a user record cannot be found. To do this, you define your error screen display constants at screen format generation time and indicate the name of the error screen format and any variable fields on your output specifications format. For a good example of what is generated on an error screen, see Figure 3-39E.

3.15. SAMPLE COBOL ACTION PROGRAMS

Four transactions using COBOL action programs are described in 3.14.1 through 3.14.4 and Figures 3-21 through 3-30. Program DISP (Figure 3-22) illustrates the use of previously coded DICE sequences stored in a COPY library. The DICE sequence coding is shown in Figure 3-23. Programs ACT1 and ACT2 (Figures 3-27 and 3-28) illustrate a dialog transaction, with ACT1 naming ACT2 as external successor. Figures 3-29 and 3-30 illustrate the use of the continuous output option. PRINT (Figure 3-29) creates continuous output to be printed at the originating terminal and uses delivery notice scheduling for control and recovery. BEGIN1 (Figure 3-30) provides an example of initiating a print transaction via the SEND function to perform continuous output at a different terminal from the one originating the processing: it uses output-for-input queueing. These two programs are not complementary; the print transaction initiated by BEGIN1 is not described, but it would necessarily be quite different in design from PRINT.

3.15.1. Sample COBOL Program Using Previously Coded DICE Sequences

The sample COBOL program in Figure 3-22 retrieves a record from the customer file (CUSTFIL) and displays it at the terminal. The program is called by the transaction code DISP, which also is the name of the program, and the 5-digit numeric key of the record desired. Figure 3-21 shows a sample input message and the corresponding output display.



```
DISP Ø1234
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 3—21. Sample Transaction Displaying Customer Record

In case of an invalid record key in the input message, or any error condition detected by IMS 90, the program moves an error message to the output message area and terminates the transaction.

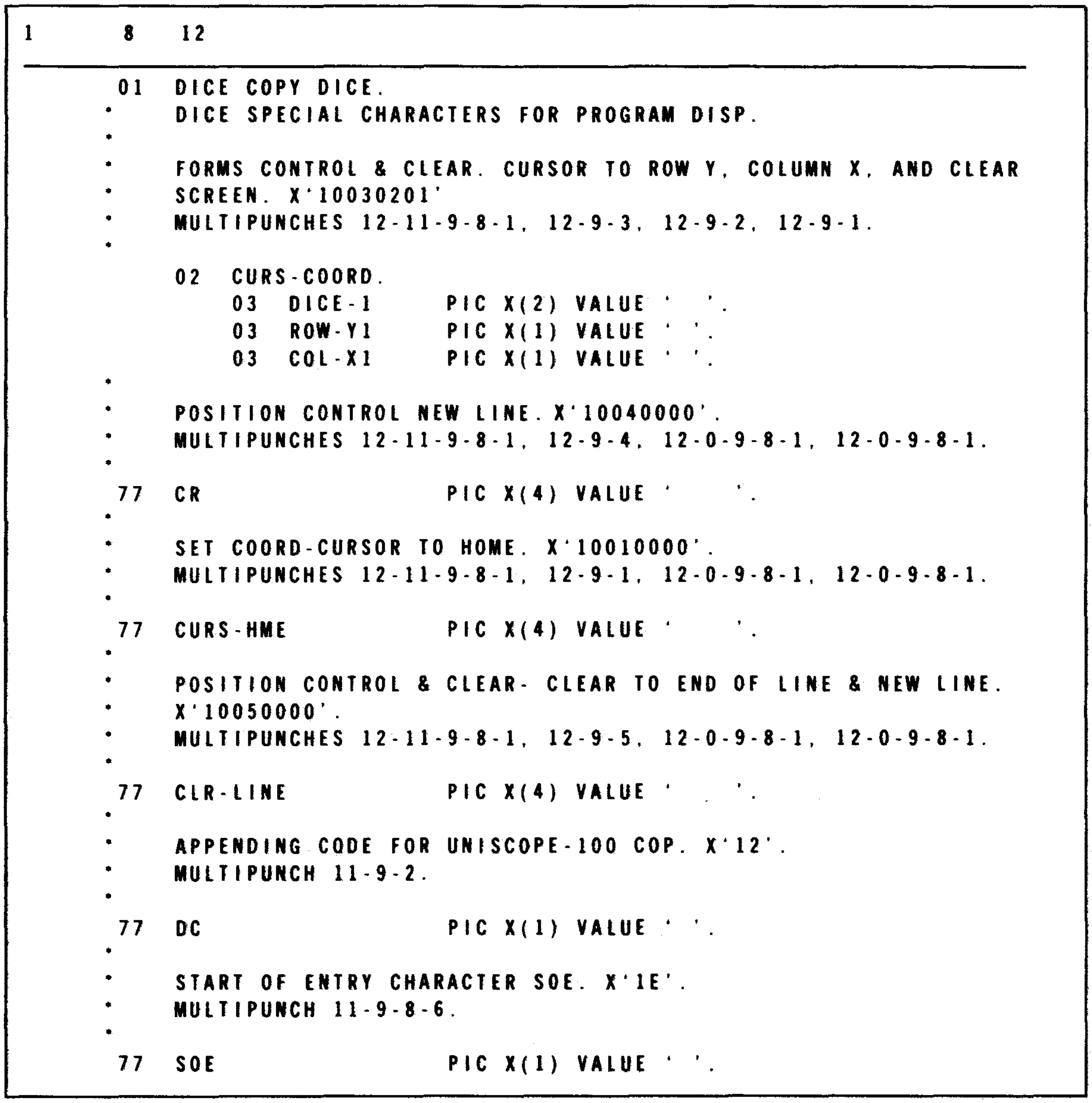
Note that the action program DISP uses DICE, previously coded and filed in a copy library, for homing the cursor, clearing the screen, and repositioning the cursor to a new line. Figure 3-23 presents an example of the coding by which the appropriate DICE sequences might be prepared for entry into a specified copy library, where they would be available for use in an action program referencing them as DISP does (via the first 01 level COPY statement in its working-storage section).

```
1      8   12
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DISP.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. UNIVAC-9030.
       OBJECT-COMPUTER. UNIVAC-9030.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  CUSTFIL  PIC X(7) VALUE 'CUSTFIL'.
       77  TEXT-1   PIC X(32) VALUE 'PROCESSING ERROR,STATUS CODE = '.
       77  TEXT-2   PIC X(23) VALUE 'DETAILED STATUS CODE = '.
       01  DICE COPY DICE.
       01  CUSHDR1.
           02  CUSHD1   PIC A(6)  VALUE ' CODE '.
           02  CUSHD2   PIC A(20) VALUE 'CUSTOMER NAME       '.
           02  CUSHD3   PIC A(15) VALUE 'ADDRESS        '.
           02  CUSHD4   PIC A(15) VALUE 'CITY-STATE     '.
           02  CUSHD5   PIC A(5)  VALUE 'ZIP  '.
       01  CUSHDR2.
           02  CUSHD6   PIC A(15) VALUE '   BALANCE-DUE '.
           02  CUSHD7   PIC A(15) VALUE '  PAYMENT-DUE  '.
           02  CUSHD8   PIC A(15) VALUE ' YR-TO-DATE VOL'.
       LINKAGE SECTION.
       01  PROGRAM-INFORMATION-BLOCK COPY PIB.
       01  INPUT-MESSAGE-AREA COPY IMA.
           02  TRANSAC-CDE   PIC X(4).
           02  FILLER        PIC X.
           02  REC-KEY       PIC X(5).
           02  REC-NO REDEFINES REC-KEY  PIC 9(5).
       01  WORK-AREA.
           02  CUS-REC.
               03  CDE            PIC X(5).
               03  NAME           PIC X(20).
               03  ADDR           PIC X(15).
               03  CTY-STE        PIC X(15).
               03  ZIP            PIC 9(5).
               03  BLNCE-DUE      PIC S9(9)V99  COMP-3.
               03  DUE-IN         PIC S9(9)V99  COMP-3.
               03  YTD-VOL        PIC 9(6)V99.
           02  ERROR-MSGE.
               03  TXT-1          PIC X(32).
               03  STAT           PIC 9(4).
               03  TXT-2          PIC X(23).
               03  DSTAT          PIC 9(4).
```

Figure 3—22. Sample COBOL Action Program DISP (Part 1 of 2)

```
1      8   12
       01  OUTPUT-MESSAGE-AREA COPY OMA.
           02  LINE-0       PIC X(4).
           02  LINE-1       PIC X(64).
           02  CR-1         PIC X(4).
           02  LINE-2.
               03  CDE      PIC X(5).
               03  FILLER   PIC X.
               03  NAME     PIC X(20).
               03  ADDR     PIC X(15).
               03  CTY-STE  PIC X(15).
               03  ZIP      PIC X(5).
           02  CR-2         PIC X(4).
           02  LINE-3       PIC X(45).
           02  CR-3         PIC X(4).
           02  LINE-4.
               03  FILLER   PIC X.
               03  OUT-BAL  PIC ZZZ,ZZZ,ZZ9.99.
               03  FILLER   PIC X(5).
               03  OUT-DUE  PIC ZZZ,ZZZ,ZZZ.99.
               03  FILLER   PIC X(5).
               03  OUT-VOL  PIC ZZZ,ZZZ.99.
           02  CR-4         PIC X(4).
           02  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
           MOVE CURS-HME TO LINE-13.
           MOVE CR TO CR-1, CR-2, CR—3, CR-4.
       CUSTOMR-FILE-SECT.
           ENTER LINKAGE.
           CALL 'GET' USING CUSTFIL CUS-REC REC-KEY.
           ENTER COBOL.
           IF STATUS-CODE IS NOT = 0 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 3—22. Sample COBOL Action Program DISP (Part 2 of 2)



```
1     8   12
      ------------------------------------------------------------
       01  DICE COPY DICE.
      *    DICE SPECIAL CHARACTERS FOR PROGRAM DISP.
      *
      *    FORMS CONTROL & CLEAR. CURSOR TO ROW Y, COLUMN X, AND CLEAR
      *    SCREEN. X'10030201'
      *    MULTIPUNCHES 12-11-9-8-1, 12-9-3, 12-9-2, 12-9-1.
      *
           02  CURS-COORD.
               03  DICE-1      PIC X(2) VALUE '  '.
               03  ROW-Y1      PIC X(1) VALUE ' '.
               03  COL-X1      PIC X(1) VALUE ' '.
      *
      *    POSITION CONTROL NEW LINE. X'10040000'.
      *    MULTIPUNCHES 12-11-9-8-1, 12-9-4, 12-0-9-8-1, 12-0-9-8-1.
      *
       77  CR                  PIC X(4) VALUE '    '.
      *
      *    SET COORD-CURSOR TO HOME. X'10010000'.
      *    MULTIPUNCHES 12-11-9-8-1, 12-9-1, 12-0-9-8-1, 12-0-9-8-1.
      *
       77  CURS-HME            PIC X(4) VALUE '    '.
      *
      *    POSITION CONTROL & CLEAR- CLEAR TO END OF LINE & NEW LINE.
      *    X'10050000'.
      *    MULTIPUNCHES 12-11-9-8-1, 12-9-5, 12-0-9-8-1, 12-0-9-8-1.
      *
       77  CLR-LINE            PIC X(4) VALUE '    '.
      *
      *    APPENDING CODE FOR UNISCOPE-100 COP. X'12'.
      *    MULTIPUNCH 11-9-2.
      *
       77  DC                  PIC X(1) VALUE ' '.
      *
      *    START OF ENTRY CHARACTER SOE. X'1E'.
      *    MULTIPUNCH 11-9-8-6.
      *
       77  SOE                 PIC X(1) VALUE ' '.
```

Figure 3—23. Example of DICE Sequences Filed in a COPY Library

3.15.2. Sample COBOL Programs Performing Dialog Transaction

The two action programs ACT1 and ACT2 (Figures 3-27 and 3-28) perform a dialog inquiry transaction. The initial input message and various possible input and output messages of the dialog transaction are shown in Figures 3-24, 3-25, and 3-26. Use of the UNISCOPE 100 display terminal is assumed. This transaction references two indexed files named STATE and CITY. The STATE file contains a record for each state. In each record is the state name, state population, and the name of the capital city. The CITY file contains a record for each city. In each record is the city name, city population, and state name. City names in the CITY file are assumed to be unique for the purposes of this example.

The transaction is designed to provide information about a state. The operator enters the name of a state. In response, the state name, population, and capital name are given. Either the population of the capital city or termination of the transaction can then be requested.

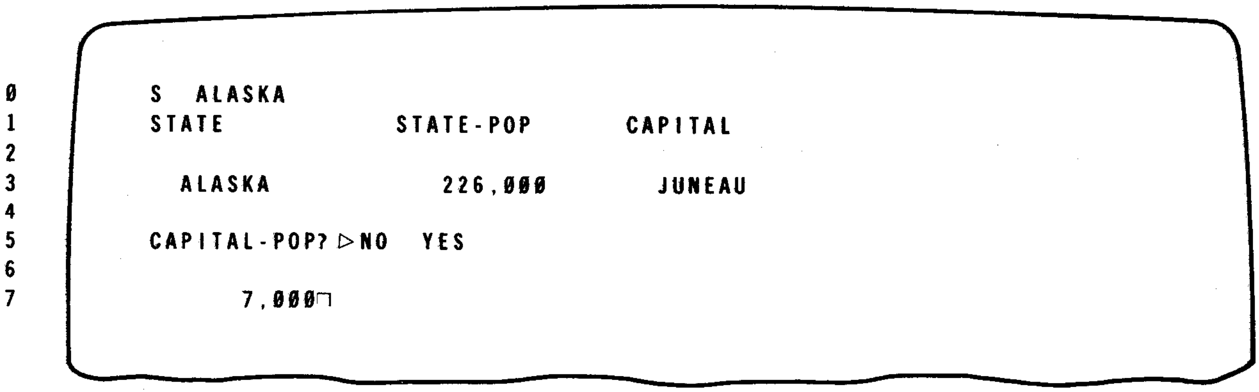
The input message (line 0) is the same for Figures 3-24, 3-25, and 3-26. S is a transaction code that uniquely identifies the state transaction to IMS 90. The S is followed by a single place to delimit it from the remaining text of the message. The name of a state is entered then and the message is transmitted to IMS 90. The symbol ¬ is the cursor and is supplied by the hardware.

IMS 90 associates the transaction code S with the action program ACT1. This relationship is established at IMS 90 configuration time. When ACT1 is initiated by action scheduling, the input message text is in the input message area (IMA) which is described in the linkage section. This area consists of a control header and a text area. The description of the control header is copied from the IMS 90 copy library; the descriptions of the output message area (OMA) control header, and the program information block (PIB) also are copied.

ACT1 uses the name of the state given in the input message to obtain a record from the STATE file. If the record exists, the name of the state, state population, and capital name are placed in the OMA. The output message headings and control characters are moved to the OMA from the working-storage section.

ACT1 saves the name of the capital city in the continuity data area. The contents of this area are automatically passed to the succeeding action program, ACT2 by IMS 90. ACT1 designates ACT2 as its external successor. The termination code E, for external succession, is moved to the TERMINATION-INDICATOR field in the PIB. The identification of the program, ACT2, is moved to the SUCCESSOR-ID field of the PIB. ACT1 terminates by means of a CALL 'RETURN' statement.

The output message built by ACT1 in its OMA is sent to the originating terminal by IMS 90 after ACT1 is terminated (lines 1-5 in Figure 3-24).

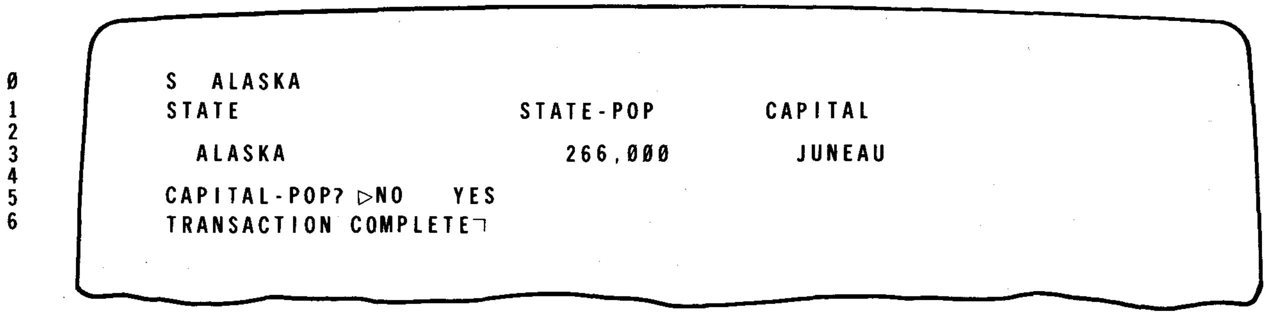


```
0      S  ALASKA
1      STATE           STATE-POP       CAPITAL
2
3        ALASKA            226,000        JUNEAU
4
5      CAPITAL-POP? ▷NO   YES
6
7           7,000¬
```

NOTE:

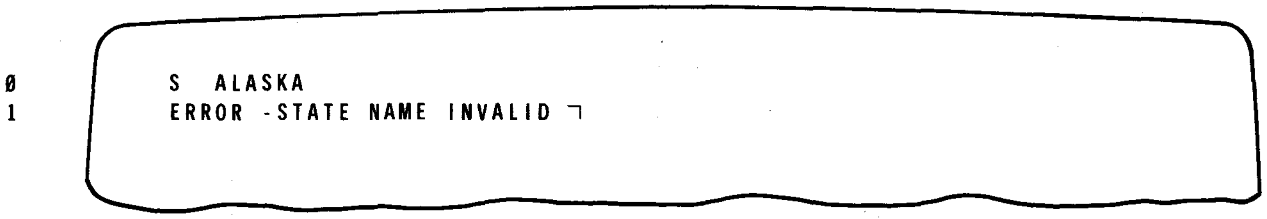
The cursor (¬) may appear at only one location on the screen at any one time. In this example, it also would have appeared after ALASKA when the operator entered the initial input message (line 0) and after NO upon transmission of the first output response built by ACT 1 (line 5). The start-of-entry character (▷) may appear at multiple locations.

Figure 3—24. Sample Dialog Transaction with Option Taken



```
Ø   S  ALASKA
1   STATE                     STATE-POP       CAPITAL
2
3     ALASKA                     266,ØØØ         JUNEAU
4
5   CAPITAL-POP? ▷NO    YES
6   TRANSACTION COMPLETE¬
```

Figure 3—25. Sample Dialog Transaction with Option Not Taken



```
Ø   S  ALASKA
1   ERROR -STATE NAME INVALID ¬
```

Figure 3—26. Sample Dialog Transaction with Error Message

The output message (line 5, Figure 3-24) gives the terminal operator the option to request the population for the capital city of the state or to terminate the transaction. The start-of-entry and cursor characters are positioned in the output message so that:

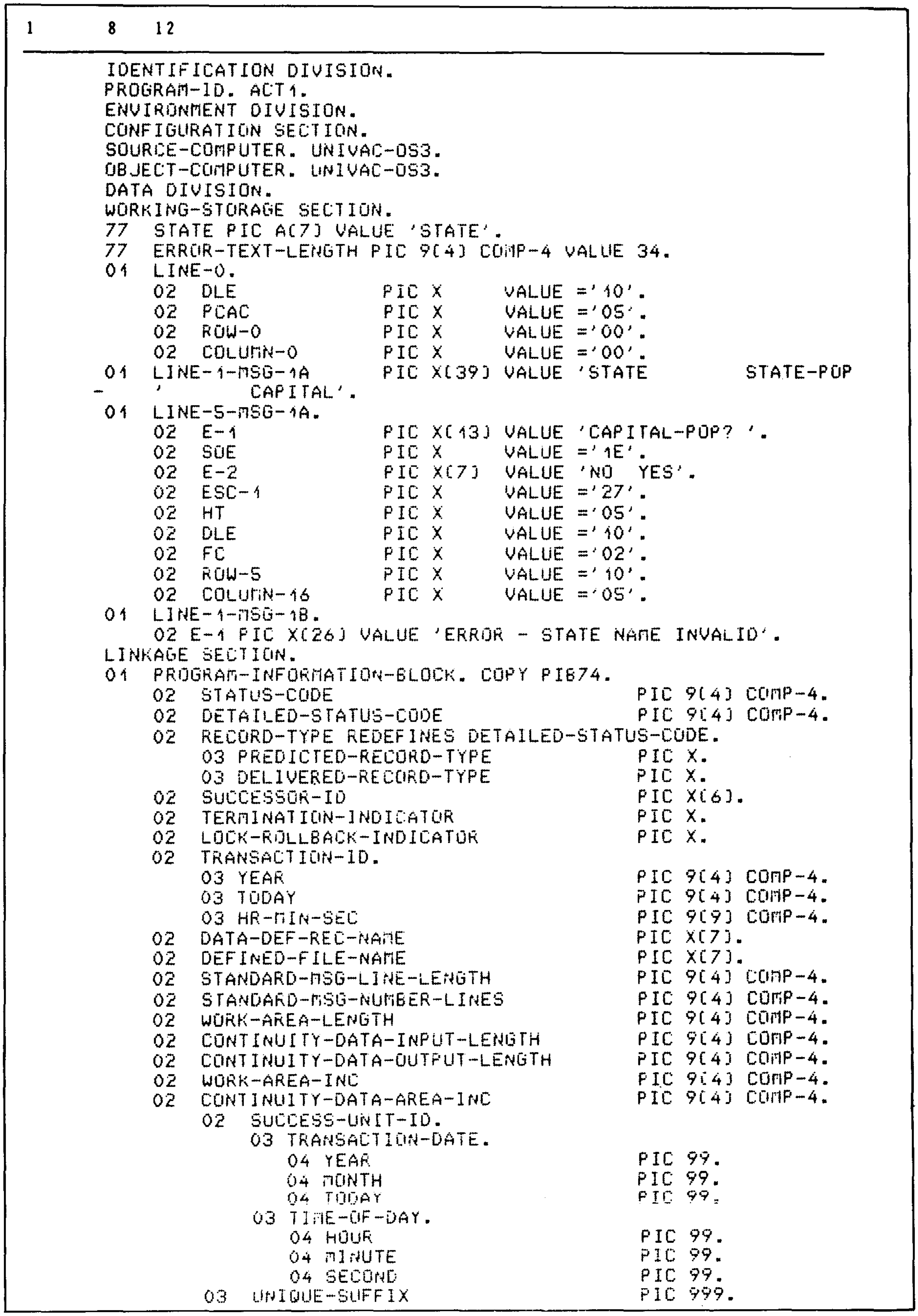
1. If the operator wants to terminate the transaction without seeing the capital population, he only needs to press TRANSMIT.

2. If the operator wants to see the capital population, he must press TAB and then TRANSMIT.

The option on line 5 in Figure 3-24 is the second input message, which always results in the scheduling of ACT2 (Figure 3-26). When the YES option is selected by the terminal operator, ACT2 obtains the CITY record for the capital city named in the continuity data area, builds an output message containing the capital population (line 7, Figure 3-24) and then terminates normally.

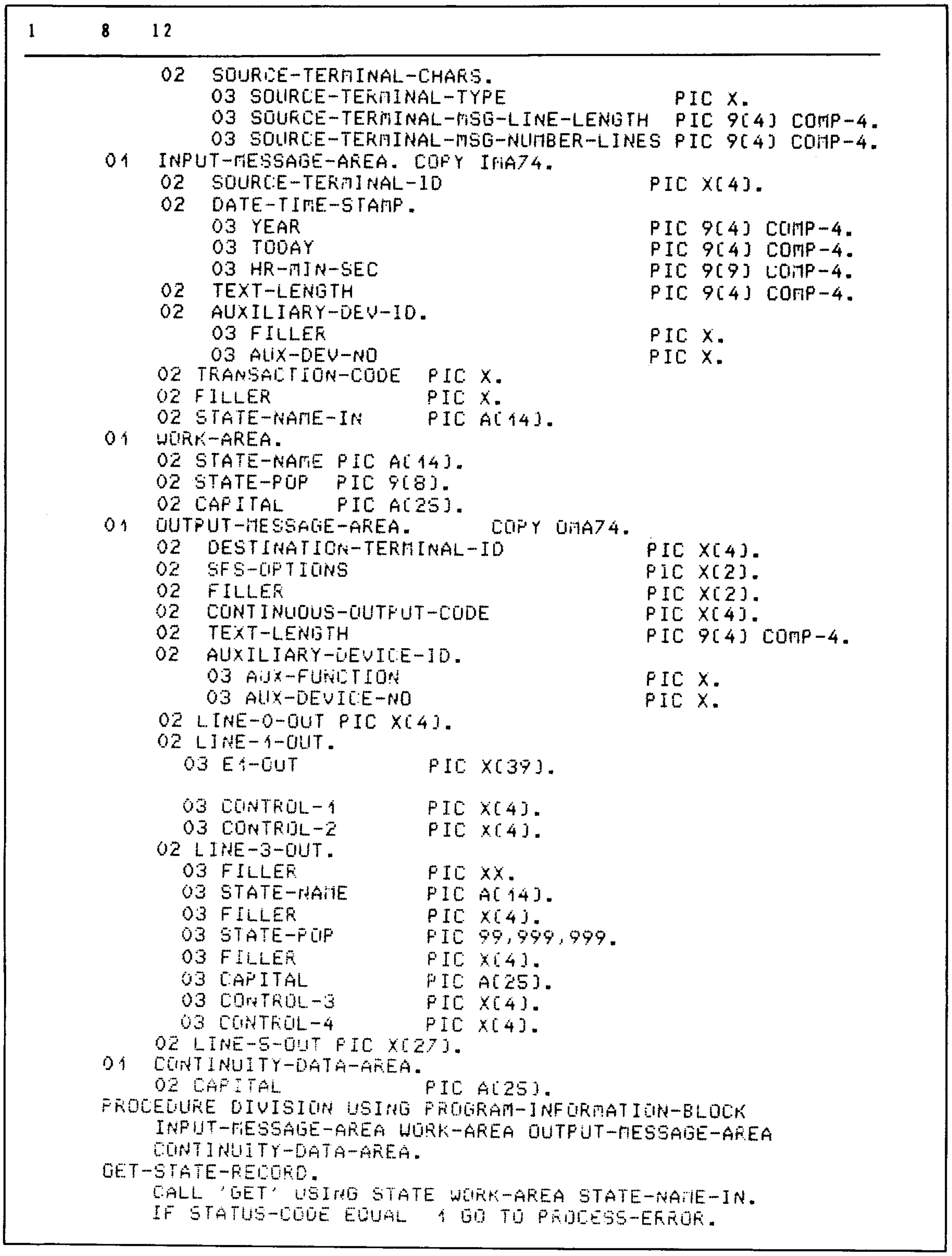
Lines 0 through 5 in Figure 3-25 are the same as lines 0 through 5 in Figure 3-24; however, the response to the option is different. The NO option is selected, and ACT2 moves zero to the TEXT-LENGTH field in the output message area control header before coming to a normal termination. Since no output message text is provided by ACT2, IMS 90 returns a standard transaction termination message to the originating terminal, as shown in line 6 in Figure 3-25.

In Figure 3-26, line 0 remains the same as it was in Figures 3-24 and 3-25. However, in the example, no such record can be found and the state name is assumed to be invalid. An error message is built in the output message area. The text length of the error message is moved to the TEXT-LENGTH field of the output message area control header to override the prespecified text length. The transaction terminates with a CALL 'RETURN' statement. The contents of the output message area are sent by IMS 90 to the originating terminal, as shown on line 1 in Figure 3-26.



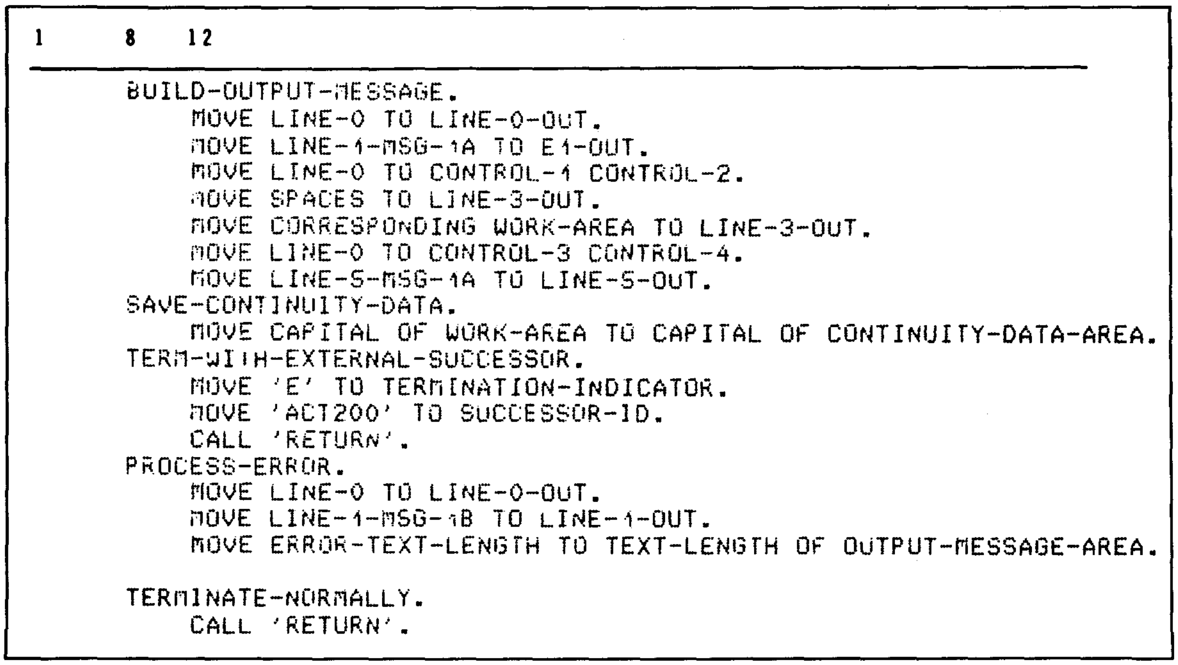
```
1      8   12
       IDENTIFICATION DIVISION.
       PROGRAM-ID. ACT1.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. UNIVAC-OS3.
       OBJECT-COMPUTER. UNIVAC-OS3.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  STATE PIC A(7) VALUE 'STATE'.
       77  ERROR-TEXT-LENGTH PIC 9(4) COMP-4 VALUE 34.
       01  LINE-0.
           02  DLE              PIC X      VALUE ='10'.
           02  PCAC             PIC X      VALUE ='05'.
           02  ROW-0            PIC X      VALUE ='00'.
           02  COLUMN-0         PIC X      VALUE ='00'.
       01  LINE-1-MSG-1A        PIC X(39)  VALUE 'STATE        STATE-POP
      -    '        CAPITAL'.
       01  LINE-5-MSG-1A.
           02  E-1              PIC X(13)  VALUE 'CAPITAL-POP? '.
           02  SOE              PIC X      VALUE ='1E'.
           02  E-2              PIC X(7)   VALUE 'NO  YES'.
           02  ESC-1            PIC X      VALUE ='27'.
           02  HT               PIC X      VALUE ='05'.
           02  DLE              PIC X      VALUE ='10'.
           02  FC               PIC X      VALUE ='02'.
           02  ROW-5            PIC X      VALUE ='10'.
           02  COLUMN-16        PIC X      VALUE ='05'.
       01  LINE-1-MSG-1B.
           02 E-1 PIC X(26) VALUE 'ERROR - STATE NAME INVALID'.
       LINKAGE SECTION.
       01  PROGRAM-INFORMATION-BLOCK. COPY PI674.
           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.
           02  CONTINUITY-DATA-OUTPUT-LENGTH        PIC 9(4) COMP-4.
           02  WORK-AREA-INC                        PIC 9(4) COMP-4.
           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  UNIQUE-SUFFIX                    PIC 999.
```

Figure 3—27. Sample COBOL Action Program ACT1 (Part 1 of 3)



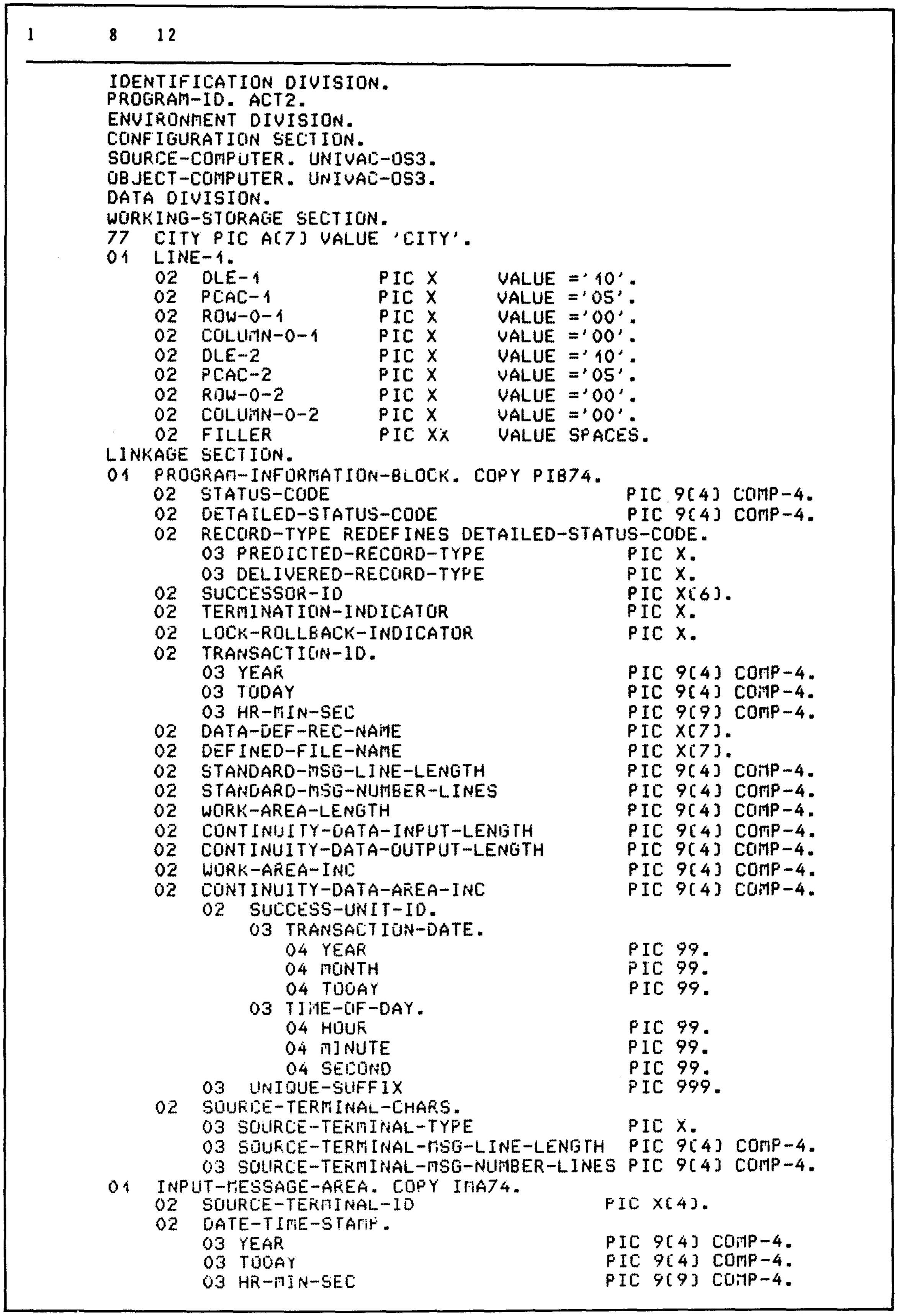
```
1      8    12
            02  SOURCE-TERMINAL-CHARS.
                03 SOURCE-TERMINAL-TYPE              PIC X.
                03 SOURCE-TERMINAL-MSG-LINE-LENGTH  PIC 9(4) COMP-4.
                03 SOURCE-TERMINAL-MSG-NUMBER-LINES PIC 9(4) COMP-4.
       01  INPUT-MESSAGE-AREA. COPY IMA74.
            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 TRANSACTION-CODE  PIC X.
            02 FILLER            PIC X.
            02 STATE-NAME-IN     PIC A(14).
       01  WORK-AREA.
            02 STATE-NAME PIC A(14).
            02 STATE-POP  PIC 9(8).
            02 CAPITAL    PIC A(25).
       01  OUTPUT-MESSAGE-AREA.       COPY OMA74.
            02  DESTINATION-TERMINAL-ID            PIC X(4).
            02  SFS-OPTIONS                        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 LINE-0-OUT PIC X(4).
            02 LINE-1-OUT.
              03 E1-OUT           PIC X(39).

              03 CONTROL-1        PIC X(4).
              03 CONTROL-2        PIC X(4).
            02 LINE-3-OUT.
              03 FILLER           PIC XX.
              03 STATE-NAME       PIC A(14).
              03 FILLER           PIC X(4).
              03 STATE-POP        PIC 99,999,999.
              03 FILLER           PIC X(4).
              03 CAPITAL          PIC A(25).
              03 CONTROL-3        PIC X(4).
              03 CONTROL-4        PIC X(4).
            02 LINE-S-OUT PIC X(27).
       01  CONTINUITY-DATA-AREA.
            02 CAPITAL            PIC A(25).
       PROCEDURE DIVISION USING PROGRAM-INFORMATION-BLOCK
            INPUT-MESSAGE-AREA WORK-AREA OUTPUT-MESSAGE-AREA
            CONTINUITY-DATA-AREA.
       GET-STATE-RECORD.
            CALL 'GET' USING STATE WORK-AREA STATE-NAME-IN.
            IF STATUS-CODE EQUAL  1 GO TO PROCESS-ERROR.
```

Figure 3—27. Sample COBOL Action Program ACT1 (Part 2 of 3)



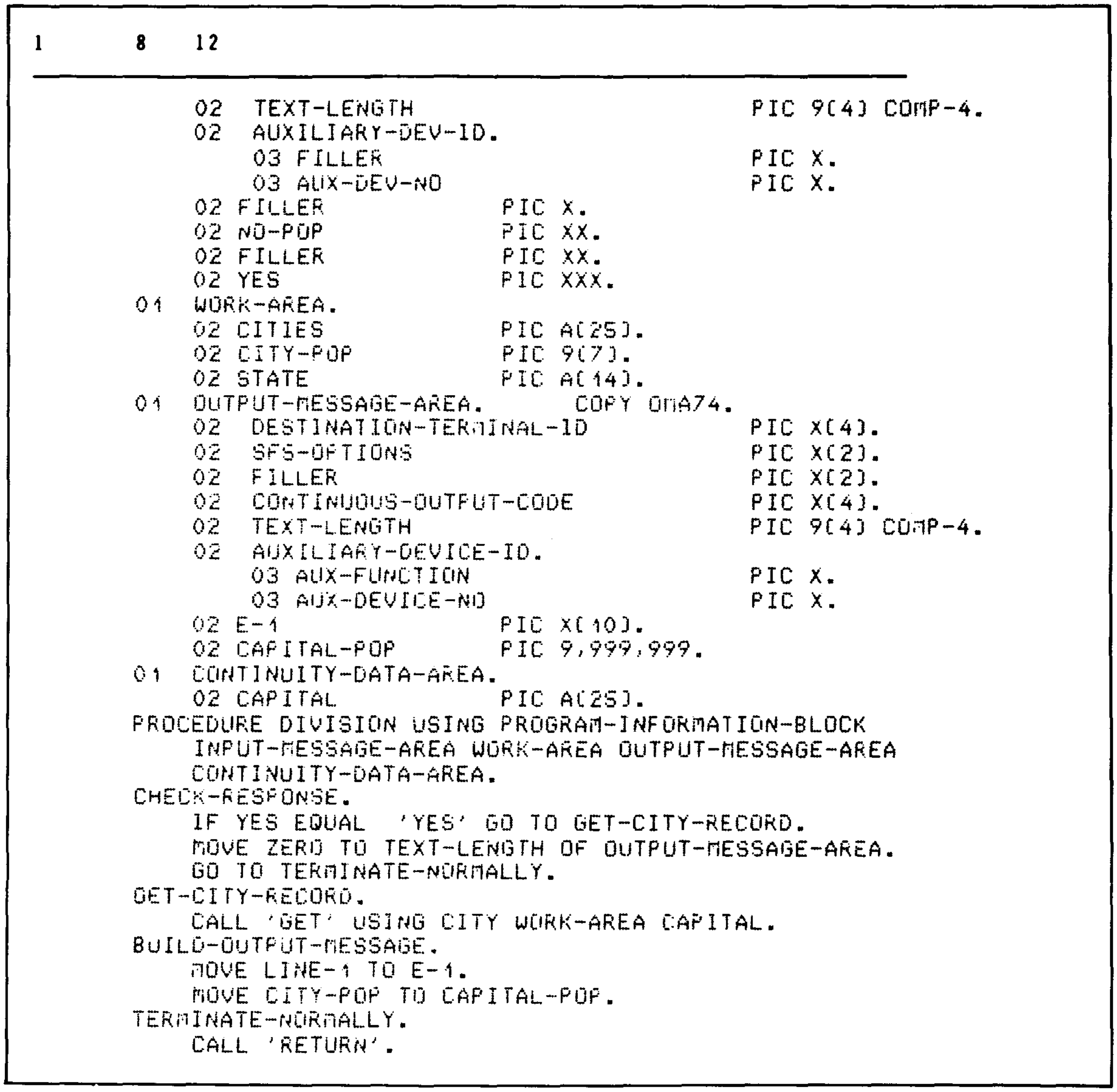
```
1      8   12
       BUILD-OUTPUT-MESSAGE.
           MOVE LINE-0 TO LINE-0-OUT.
           MOVE LINE-1-MSG-1A TO E1-OUT.
           MOVE LINE-0 TO CONTROL-1 CONTROL-2.
           MOVE SPACES TO LINE-3-OUT.
           MOVE CORRESPONDING WORK-AREA TO LINE-3-OUT.
           MOVE LINE-0 TO CONTROL-3 CONTROL-4.
           MOVE LINE-5-MSG-1A TO LINE-5-OUT.
       SAVE-CONTINUITY-DATA.
           MOVE CAPITAL OF WORK-AREA TO CAPITAL OF CONTINUITY-DATA-AREA.
       TERM-WITH-EXTERNAL-SUCCESSOR.
           MOVE 'E' TO TERMINATION-INDICATOR.
           MOVE 'ACT200' TO SUCCESSOR-ID.
           CALL 'RETURN'.
       PROCESS-ERROR.
           MOVE LINE-0 TO LINE-0-OUT.
           MOVE LINE-1-MSG-1B TO LINE-1-OUT.
           MOVE ERROR-TEXT-LENGTH TO TEXT-LENGTH OF OUTPUT-MESSAGE-AREA.

       TERMINATE-NORMALLY.
           CALL 'RETURN'.
```

Figure 3—27. Sample COBOL Action Program ACT1 (Part 3 of 3)



```
1      8   12
_____________________________________________________________
       IDENTIFICATION DIVISION.
       PROGRAM-ID. ACT2.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. UNIVAC-OS3.
       OBJECT-COMPUTER. UNIVAC-OS3.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  CITY PIC A(7) VALUE 'CITY'.
       01  LINE-1.
           02  DLE-1          PIC X     VALUE ='10'.
           02  PCAC-1         PIC X     VALUE ='05'.
           02  ROW-0-1        PIC X     VALUE ='00'.
           02  COLUMN-0-1     PIC X     VALUE ='00'.
           02  DLE-2          PIC X     VALUE ='10'.
           02  PCAC-2         PIC X     VALUE ='05'.
           02  ROW-0-2        PIC X     VALUE ='00'.
           02  COLUMN-0-2     PIC X     VALUE ='00'.
           02  FILLER         PIC XX    VALUE SPACES.
       LINKAGE SECTION.
       01  PROGRAM-INFORMATION-BLOCK. 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.
           02  CONTINUITY-DATA-OUTPUT-LENGTH        PIC 9(4) COMP-4.
           02  WORK-AREA-INC                        PIC 9(4) COMP-4.
           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  UNIQUE-SUFFIX                    PIC 999.
           02  SOURCE-TERMINAL-CHARS.
               03 SOURCE-TERMINAL-TYPE              PIC X.
               03 SOURCE-TERMINAL-MSG-LINE-LENGTH  PIC 9(4) COMP-4.
               03 SOURCE-TERMINAL-MSG-NUMBER-LINES PIC 9(4) COMP-4.
       01  INPUT-MESSAGE-AREA. 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.
```

Figure 3—28. Sample COBOL Action Program ACT2 (Part 1 of 2)



```
1      8   12

           02  TEXT-LENGTH                      PIC 9[4] COMP-4.
           02  AUXILIARY-DEV-ID.
               03 FILLER                        PIC X.
               03 AUX-DEV-NO                    PIC X.
           02 FILLER           PIC X.
           02 NO-POP           PIC XX.
           02 FILLER           PIC XX.
           02 YES              PIC XXX.
       01  WORK-AREA.
           02 CITIES           PIC A[25].
           02 CITY-POP         PIC 9[7].
           02 STATE            PIC A[14].
       01  OUTPUT-MESSAGE-AREA.      COPY OMA74.
           02  DESTINATION-TERMINAL-ID          PIC X[4].
           02  SFS-OPTIONS                      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 E-1              PIC X[10].
           02 CAPITAL-POP      PIC 9,999,999.
       01  CONTINUITY-DATA-AREA.
           02 CAPITAL          PIC A[25].
       PROCEDURE DIVISION USING PROGRAM-INFORMATION-BLOCK
           INPUT-MESSAGE-AREA WORK-AREA OUTPUT-MESSAGE-AREA
           CONTINUITY-DATA-AREA.
       CHECK-RESPONSE.
           IF YES EQUAL  'YES' GO TO GET-CITY-RECORD.
           MOVE ZERO TO TEXT-LENGTH OF OUTPUT-MESSAGE-AREA.
           GO TO TERMINATE-NORMALLY.
       GET-CITY-RECORD.
           CALL 'GET' USING CITY WORK-AREA CAPITAL.
       BUILD-OUTPUT-MESSAGE.
           MOVE LINE-1 TO E-1.
           MOVE CITY-POP TO CAPITAL-POP.
       TERMINATE-NORMALLY.
           CALL 'RETURN'.
```

Figure 3—28. Sample COBOL Action Program ACT2 (Part 2 of 2)

3.15.3. Continuous Output Example Using Delivery Notice Scheduling

Figure 3-29 reproduces a compiler listing of a sample COBOL action program, PRINT, written to illustrate a number of points relating to the use of the continuous output option. The primary function of PRINT is to prepare three types of output messages by processing customer order information entered at the terminal against an indexed file, and to cause these messages to be listed as continuous output at the originating terminal. A parameter on the initial input message can cause the printing to be sent to a COP. The routine performing these functions uses delivery notice scheduling to determine whether output is to continue or error processing is to be invoked after delivery notice of each message is received from IMS 90. So long as the output continues successfully, PRINT terminates in external succession, having named 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.

If an unsuccessful delivery notice is received, PRINT 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 reaction. He may react by breaking off, resuming, or terminating the transaction normally.

When it is first activated by action scheduling as the result of the terminal operator's initial keyin, PRINT expects to process an input message in the following form:

```
PRINT t-file-name t-order-number init-terminal[COP]
```

where:

`PRINT`
Is the *transaction code* that causes PRINT to be scheduled.

`t-file-name`
Is the name of the conventional file to be accessed. As stated, the file is an indexed file; PRINT expects to process a file named 'ORDRFIL' and validates the *t-file-name* keyed in (line 268, Figure 3-29).

t-order-number

Is an order number used as a key search argument in positioning the file for retrieval (line 272).

init-terminal

Is a configured identification to the originating terminal, used in the switching of output error messages to the operator (line 355).

COP

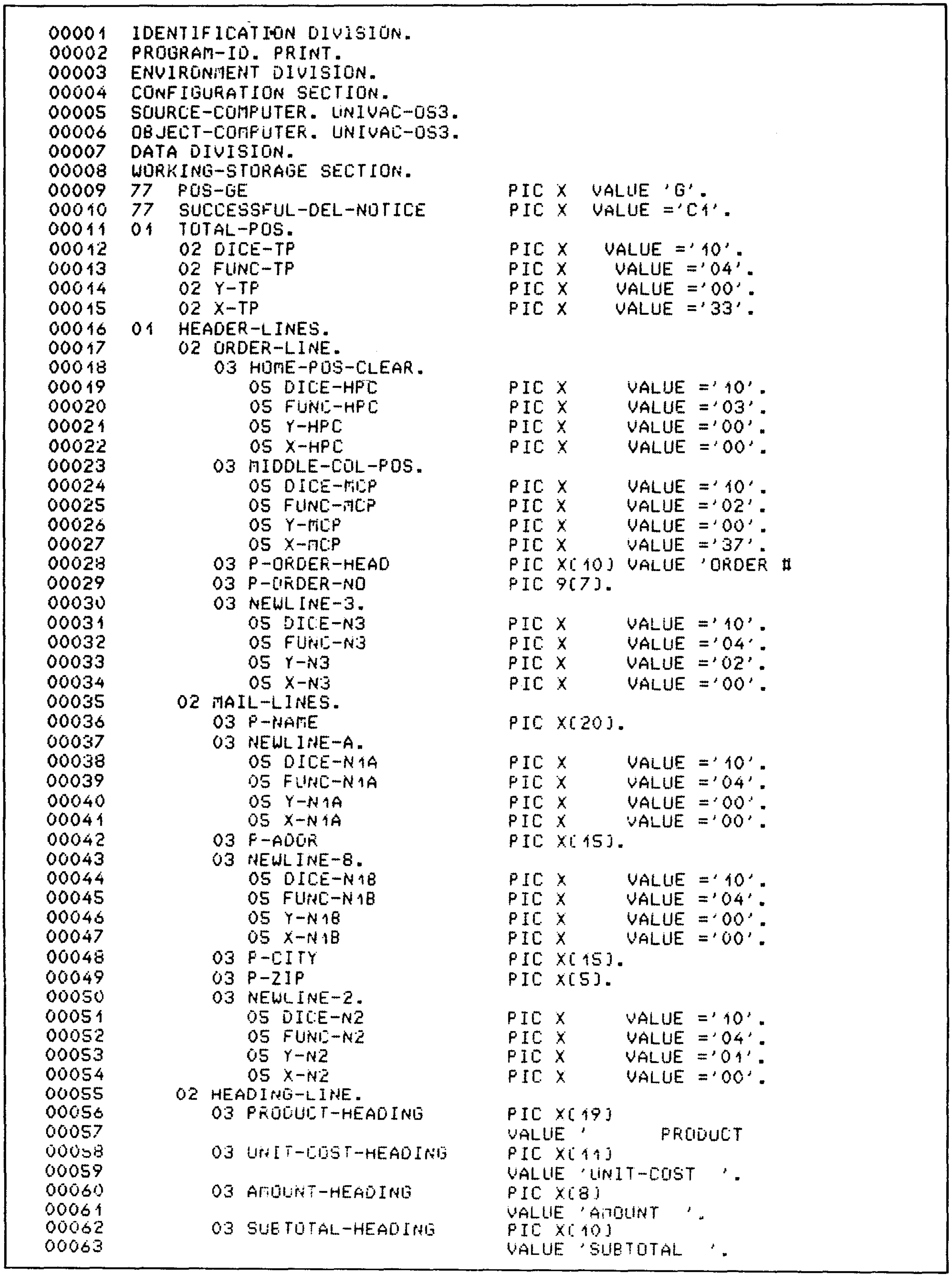
Is the 3-character code input by the terminal operator to designate that what is to be output is to be printed on the COP. Notice its use in line 302.

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. BEGIN-TRANS positions the file for sequential processing and, retrieving a record, processes it and the input message. It forms a customer record, a product record, or a total record (according to the path that control follows) in the output message area; control then passes to the CREATE-CONTINUOUS-OUTPUT routine (line 301).

Here, if the terminal operator has not keyed in COP to cause the output message to be directed to a communications output printer, the routine moves the hexadecimal value C3 to the AUX-FUNCTION byte of the AUXILIARY-DEVICE-ID field of the OMA header (line 303). This causes the output message to be written as continuous output on the screen of the primary device. Otherwise, line 304 moves the hexadecimal value F7 to this byte, to cause print-transparent continuous output on a COP, and line 305 moves a 1 to the AUX-DEVICE-NO byte of the AUXILIARY-DEVICE-ID field of the OMA header to specify the COP relative number as defined in the ICAM generation. Line 306 moves into the CONTINUOUS-OUTPUT-CODE field of the OMA header a 4-character value, developed during processing, that will identify this output message when received in the 5-byte input message that IMS 90 creates for the next activation of PRINT after attempting to deliver the message as specified.

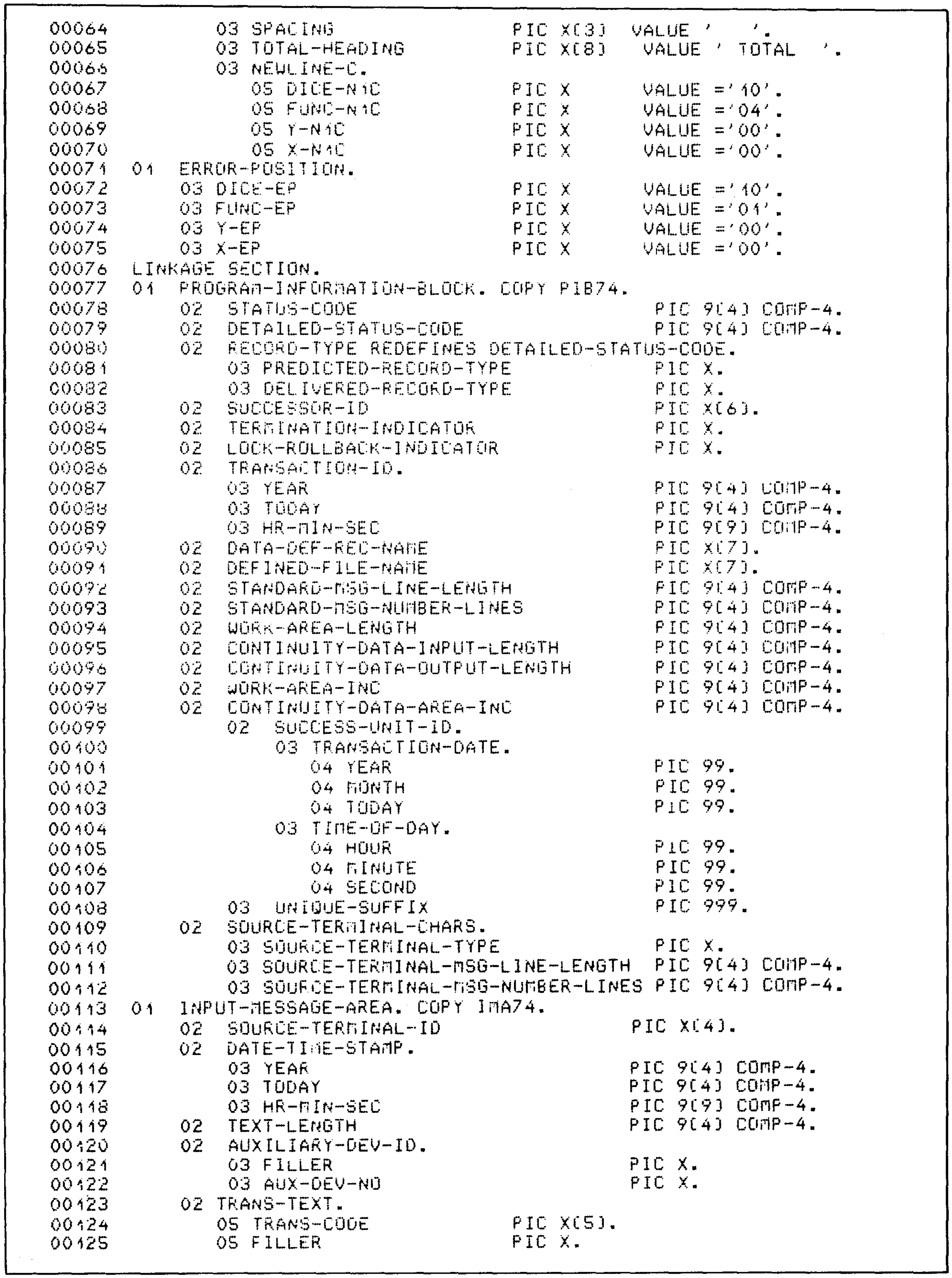
After specifying external succession (line 308) and moving its own program name into the SUCCESSOR-ID field of the PIB (line 309), PRINT terminates to await reactivation by action scheduling.

On receiving the 5-byte input message from internal message control, action scheduling reactivates PRINT. On examining this input message, PRINT checks the 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 90 by ICAM. If delivery was unsuccessful, PRINT does not attempt to determine the reason but proceeds to send an error message to the terminal operator. If an initiating terminal is specified, error messages are switched to that terminal. On successful delivery, it resumes processing.



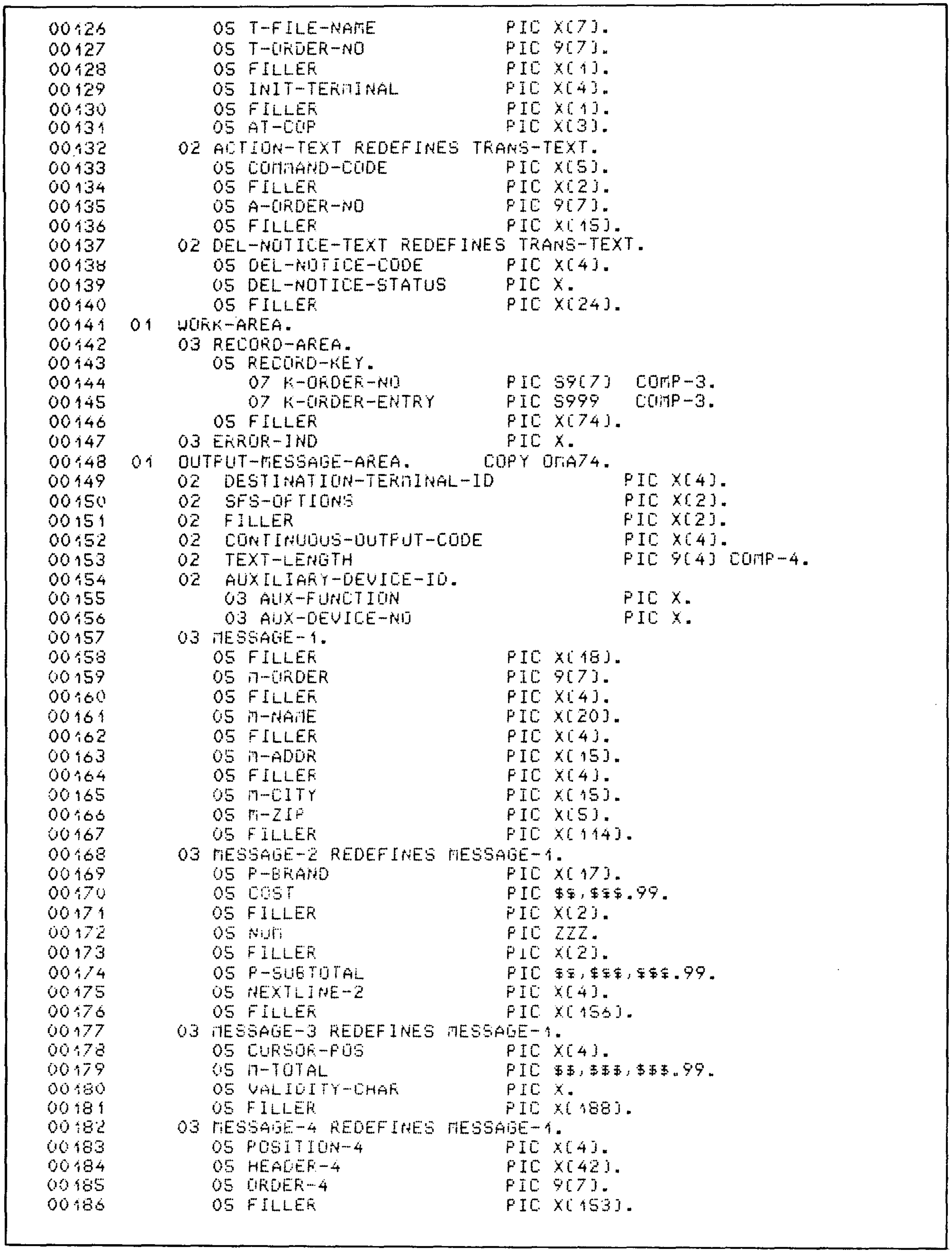
```
IDENTIFICATION DIVISION.
PROGRAM-ID. PRINT.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. UNIVAC-OS3.
OBJECT-COMPUTER. UNIVAC-OS3.
DATA DIVISION.
WORKING-STORAGE SECTION.
77  POS-GE                       PIC X  VALUE 'G'.
77  SUCCESSFUL-DEL-NOTICE        PIC X  VALUE ='C1'.
01  TOTAL-POS.
    02 DICE-TP                   PIC X   VALUE ='10'.
    02 FUNC-TP                   PIC X    VALUE ='04'.
    02 Y-TP                      PIC X    VALUE ='00'.
    02 X-TP                      PIC X    VALUE ='33'.
01  HEADER-LINES.
    02 ORDER-LINE.
       03 HOME-POS-CLEAR.
          05 DICE-HPC            PIC X     VALUE ='10'.
          05 FUNC-HPC            PIC X     VALUE ='03'.
          05 Y-HPC               PIC X     VALUE ='00'.
          05 X-HPC               PIC X     VALUE ='00'.
       03 MIDDLE-COL-POS.
          05 DICE-MCP            PIC X     VALUE ='10'.
          05 FUNC-MCP            PIC X     VALUE ='02'.
          05 Y-MCP               PIC X     VALUE ='00'.
          05 X-MCP               PIC X     VALUE ='37'.
       03 P-ORDER-HEAD           PIC X(10) VALUE 'ORDER #
       03 P-ORDER-NO             PIC 9(7).
       03 NEWLINE-3.
          05 DICE-N3             PIC X     VALUE ='10'.
          05 FUNC-N3             PIC X     VALUE ='04'.
          05 Y-N3                PIC X     VALUE ='02'.
          05 X-N3                PIC X     VALUE ='00'.
    02 MAIL-LINES.
       03 P-NAME                 PIC X(20).
       03 NEWLINE-A.
          05 DICE-N1A            PIC X     VALUE ='10'.
          05 FUNC-N1A            PIC X     VALUE ='04'.
          05 Y-N1A               PIC X     VALUE ='00'.
          05 X-N1A               PIC X     VALUE ='00'.
       03 P-ADDR                 PIC X(15).
       03 NEWLINE-B.
          05 DICE-N1B            PIC X     VALUE ='10'.
          05 FUNC-N1B            PIC X     VALUE ='04'.
          05 Y-N1B               PIC X     VALUE ='00'.
          05 X-N1B               PIC X     VALUE ='00'.
       03 P-CITY                 PIC X(15).
       03 P-ZIP                  PIC X(5).
       03 NEWLINE-2.
          05 DICE-N2             PIC X     VALUE ='10'.
          05 FUNC-N2             PIC X     VALUE ='04'.
          05 Y-N2                PIC X     VALUE ='01'.
          05 X-N2                PIC X     VALUE ='00'.
    02 HEADING-LINE.
       03 PRODUCT-HEADING        PIC X(19)
                                 VALUE '       PRODUCT
       03 UNIT-COST-HEADING      PIC X(11)
                                 VALUE 'UNIT-COST  '.
       03 AMOUNT-HEADING         PIC X(8)
                                 VALUE 'AMOUNT  '.
       03 SUBTOTAL-HEADING       PIC X(10)
                                 VALUE 'SUBTOTAL  '.
```

Figure 3—29. Sample COBOL Action Program Performing Continuous Output (Part 1 of 7)



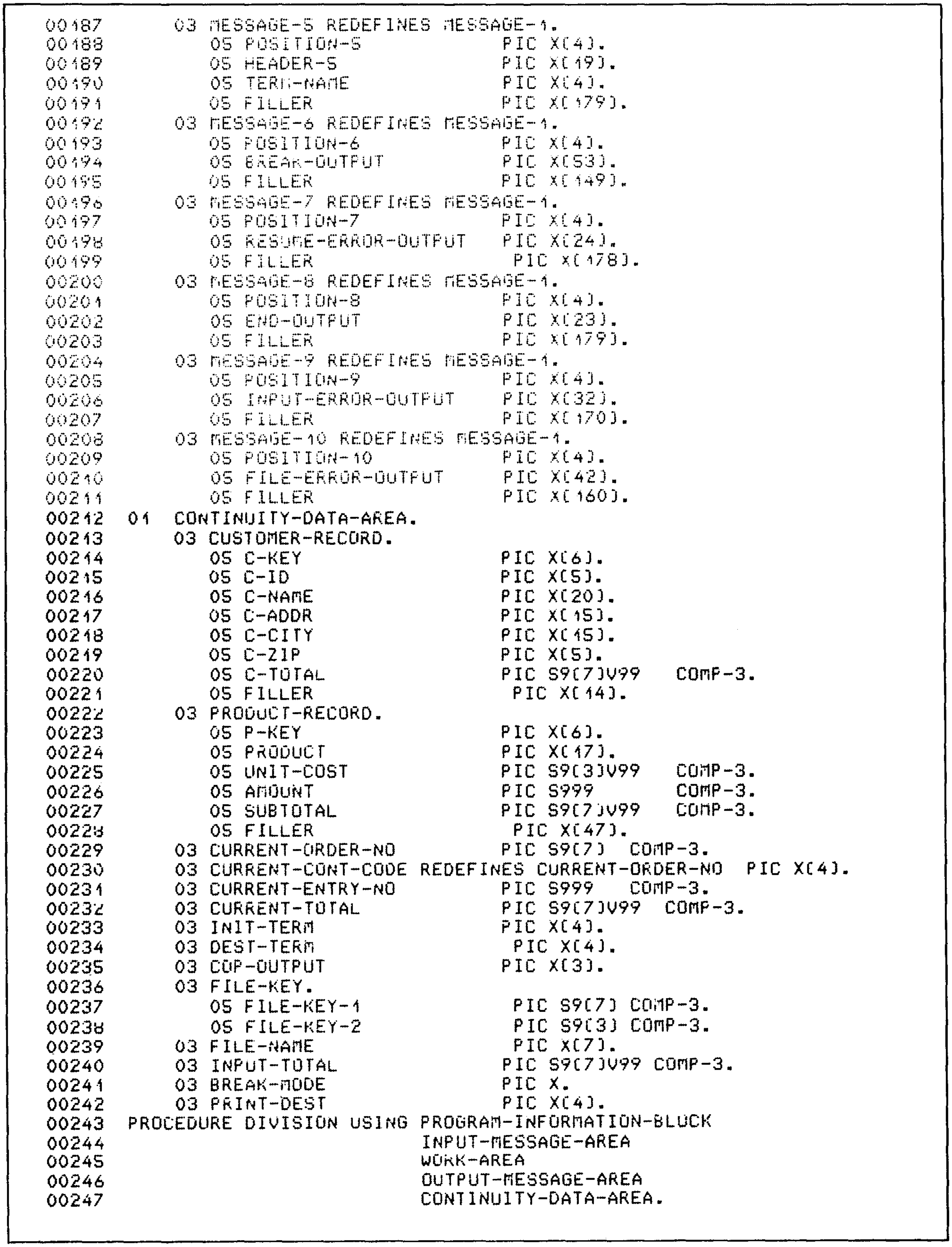
```
       03 SPACING               PIC X(3)  VALUE '   '.
       03 TOTAL-HEADING         PIC X(8)   VALUE ' TOTAL  '.
       03 NEWLINE-C.
          05 DICE-N1C           PIC X      VALUE ='10'.
          05 FUNC-N1C           PIC X      VALUE ='04'.
          05 Y-N1C              PIC X      VALUE ='00'.
          05 X-N1C              PIC X      VALUE ='00'.
01  ERROR-POSITION.
    03 DICE-EP                  PIC X      VALUE ='10'.
    03 FUNC-EP                  PIC X      VALUE ='01'.
    03 Y-EP                     PIC X      VALUE ='00'.
    03 X-EP                     PIC X      VALUE ='00'.
LINKAGE SECTION.
01  PROGRAM-INFORMATION-BLOCK. 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.
    02  CONTINUITY-DATA-OUTPUT-LENGTH       PIC 9(4) COMP-4.
    02  WORK-AREA-INC                       PIC 9(4) COMP-4.
    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  UNIQUE-SUFFIX                   PIC 999.
    02  SOURCE-TERMINAL-CHARS.
        03 SOURCE-TERMINAL-TYPE             PIC X.
        03 SOURCE-TERMINAL-MSG-LINE-LENGTH  PIC 9(4) COMP-4.
        03 SOURCE-TERMINAL-MSG-NUMBER-LINES PIC 9(4) COMP-4.
01  INPUT-MESSAGE-AREA. 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-TEXT.
       05 TRANS-CODE            PIC X(5).
       05 FILLER                PIC X.
```

Figure 3—29. Sample COBOL Action Program Performing Continuous Output (Part 2 of 7)



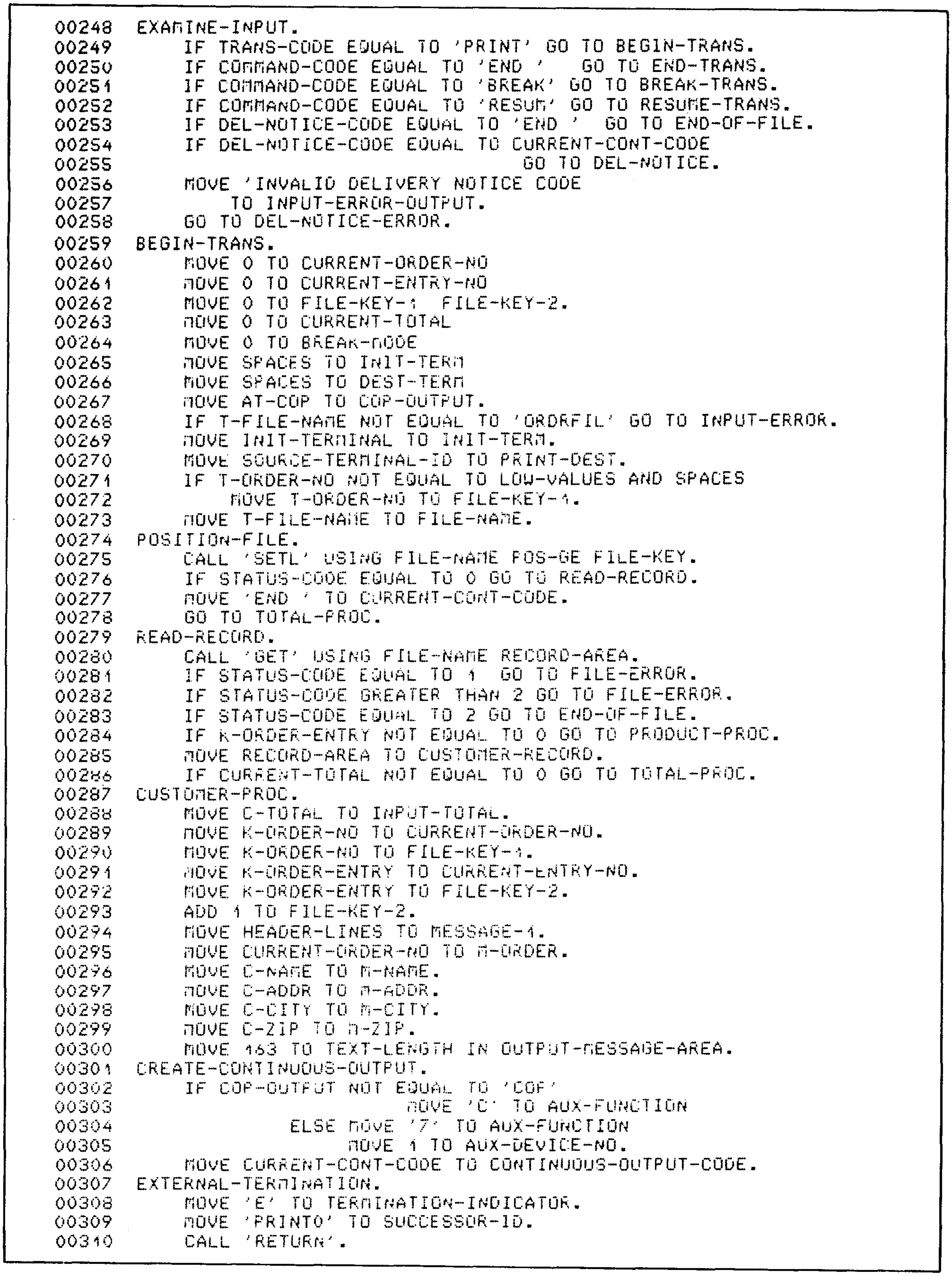
```
       05 T-FILE-NAME           PIC X(7).
       05 T-ORDER-NO            PIC 9(7).
       05 FILLER                PIC X(1).
       05 INIT-TERMINAL         PIC X(4).
       05 FILLER                PIC X(1).
       05 AT-COP                PIC X(3).
    02 ACTION-TEXT REDEFINES TRANS-TEXT.
       05 COMMAND-CODE          PIC X(5).
       05 FILLER                PIC X(2).
       05 A-ORDER-NO            PIC 9(7).
       05 FILLER                PIC X(15).
    02 DEL-NOTICE-TEXT REDEFINES TRANS-TEXT.
       05 DEL-NOTICE-CODE       PIC X(4).
       05 DEL-NOTICE-STATUS     PIC X.
       05 FILLER                PIC X(24).
01  WORK-AREA.
    03 RECORD-AREA.
       05 RECORD-KEY.
          07 K-ORDER-NO         PIC S9(7)  COMP-3.
          07 K-ORDER-ENTRY      PIC S999   COMP-3.
       05 FILLER                PIC X(74).
    03 ERROR-IND                PIC X.
01  OUTPUT-MESSAGE-AREA.      COPY OMA74.
    02  DESTINATION-TERMINAL-ID          PIC X(4).
    02  SFS-OPTIONS                      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.
    03 MESSAGE-1.
       05 FILLER                PIC X(18).
       05 M-ORDER               PIC 9(7).
       05 FILLER                PIC X(4).
       05 M-NAME                PIC X(20).
       05 FILLER                PIC X(4).
       05 M-ADDR                PIC X(15).
       05 FILLER                PIC X(4).
       05 M-CITY                PIC X(15).
       05 M-ZIP                 PIC X(5).
       05 FILLER                PIC X(114).
    03 MESSAGE-2 REDEFINES MESSAGE-1.
       05 P-BRAND               PIC X(17).
       05 COST                  PIC $$,$$$.99.
       05 FILLER                PIC X(2).
       05 NUM                   PIC ZZZ.
       05 FILLER                PIC X(2).
       05 P-SUBTOTAL            PIC $$,$$$,$$$.99.
       05 NEXTLINE-2            PIC X(4).
       05 FILLER                PIC X(156).
    03 MESSAGE-3 REDEFINES MESSAGE-1.
       05 CURSOR-POS            PIC X(4).
       05 M-TOTAL               PIC $$,$$$,$$$.99.
       05 VALIDITY-CHAR         PIC X.
       05 FILLER                PIC X(188).
    03 MESSAGE-4 REDEFINES MESSAGE-1.
       05 POSITION-4            PIC X(4).
       05 HEADER-4              PIC X(42).
       05 ORDER-4               PIC 9(7).
       05 FILLER                PIC X(153).
```

Figure 3—29. Sample COBOL Action Program Performing Continuous Output (Part 3 of 7)



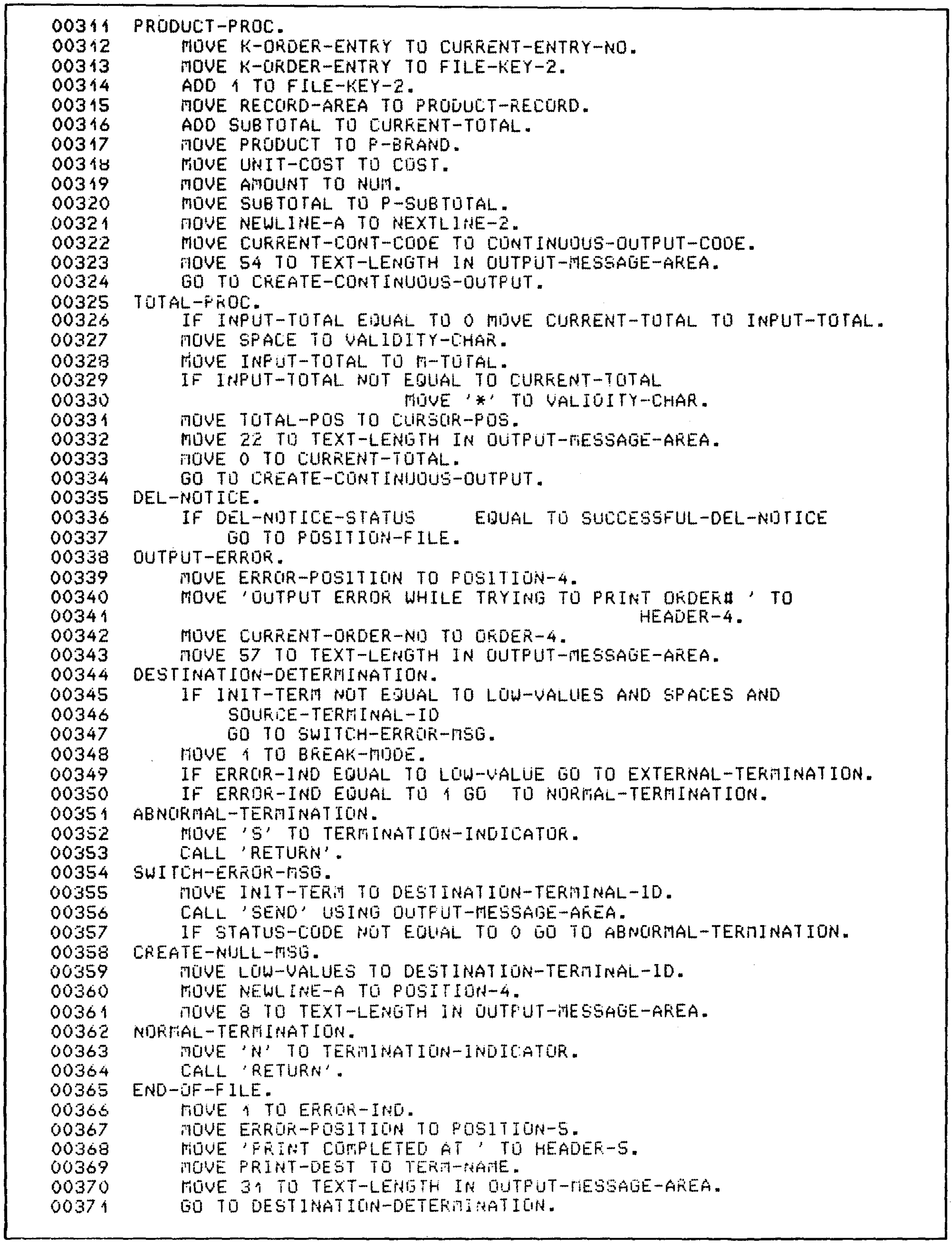
```
    03 MESSAGE-5 REDEFINES MESSAGE-1.
       05 POSITION-5            PIC X(4).
       05 HEADER-5              PIC X(19).
       05 TERM-NAME             PIC X(4).
       05 FILLER                PIC X(179).
    03 MESSAGE-6 REDEFINES MESSAGE-1.
       05 POSITION-6            PIC X(4).
       05 BREAK-OUTPUT          PIC X(53).
       05 FILLER                PIC X(149).
    03 MESSAGE-7 REDEFINES MESSAGE-1.
       05 POSITION-7            PIC X(4).
       05 RESUME-ERROR-OUTPUT   PIC X(24).
       05 FILLER                 PIC X(178).
    03 MESSAGE-8 REDEFINES MESSAGE-1.
       05 POSITION-8            PIC X(4).
       05 END-OUTPUT            PIC X(23).
       05 FILLER                PIC X(179).
    03 MESSAGE-9 REDEFINES MESSAGE-1.
       05 POSITION-9            PIC X(4).
       05 INPUT-ERROR-OUTPUT    PIC X(32).
       05 FILLER                PIC X(170).
    03 MESSAGE-10 REDEFINES MESSAGE-1.
       05 POSITION-10           PIC X(4).
       05 FILE-ERROR-OUTPUT     PIC X(42).
       05 FILLER                PIC X(160).
01  CONTINUITY-DATA-AREA.
    03 CUSTOMER-RECORD.
       05 C-KEY                 PIC X(6).
       05 C-ID                  PIC X(5).
       05 C-NAME                PIC X(20).
       05 C-ADDR                PIC X(15).
       05 C-CITY                PIC X(15).
       05 C-ZIP                 PIC X(5).
       05 C-TOTAL               PIC S9(7)V99   COMP-3.
       05 FILLER                 PIC X(14).
    03 PRODUCT-RECORD.
       05 P-KEY                 PIC X(6).
       05 PRODUCT               PIC X(17).
       05 UNIT-COST             PIC S9(3)V99   COMP-3.
       05 AMOUNT                PIC S999       COMP-3.
       05 SUBTOTAL              PIC S9(7)V99   COMP-3.
       05 FILLER                 PIC X(47).
    03 CURRENT-ORDER-NO         PIC S9(7)  COMP-3.
    03 CURRENT-CONT-CODE REDEFINES CURRENT-ORDER-NO  PIC X(4).
    03 CURRENT-ENTRY-NO         PIC S999   COMP-3.
    03 CURRENT-TOTAL            PIC S9(7)V99  COMP-3.
    03 INIT-TERM                PIC X(4).
    03 DEST-TERM                 PIC X(4).
    03 COP-OUTPUT               PIC X(3).
    03 FILE-KEY.
       05 FILE-KEY-1             PIC S9(7) COMP-3.
       05 FILE-KEY-2             PIC S9(3) COMP-3.
    03 FILE-NAME                 PIC X(7).
    03 INPUT-TOTAL              PIC S9(7)V99 COMP-3.
    03 BREAK-MODE               PIC X.
    03 PRINT-DEST               PIC X(4).
PROCEDURE DIVISION USING PROGRAM-INFORMATION-BLOCK
                         INPUT-MESSAGE-AREA
                         WORK-AREA
                         OUTPUT-MESSAGE-AREA
                         CONTINUITY-DATA-AREA.
```

Figure 3—29. Sample COBOL Action Program Performing Continuous Output (Part 4 of 7)



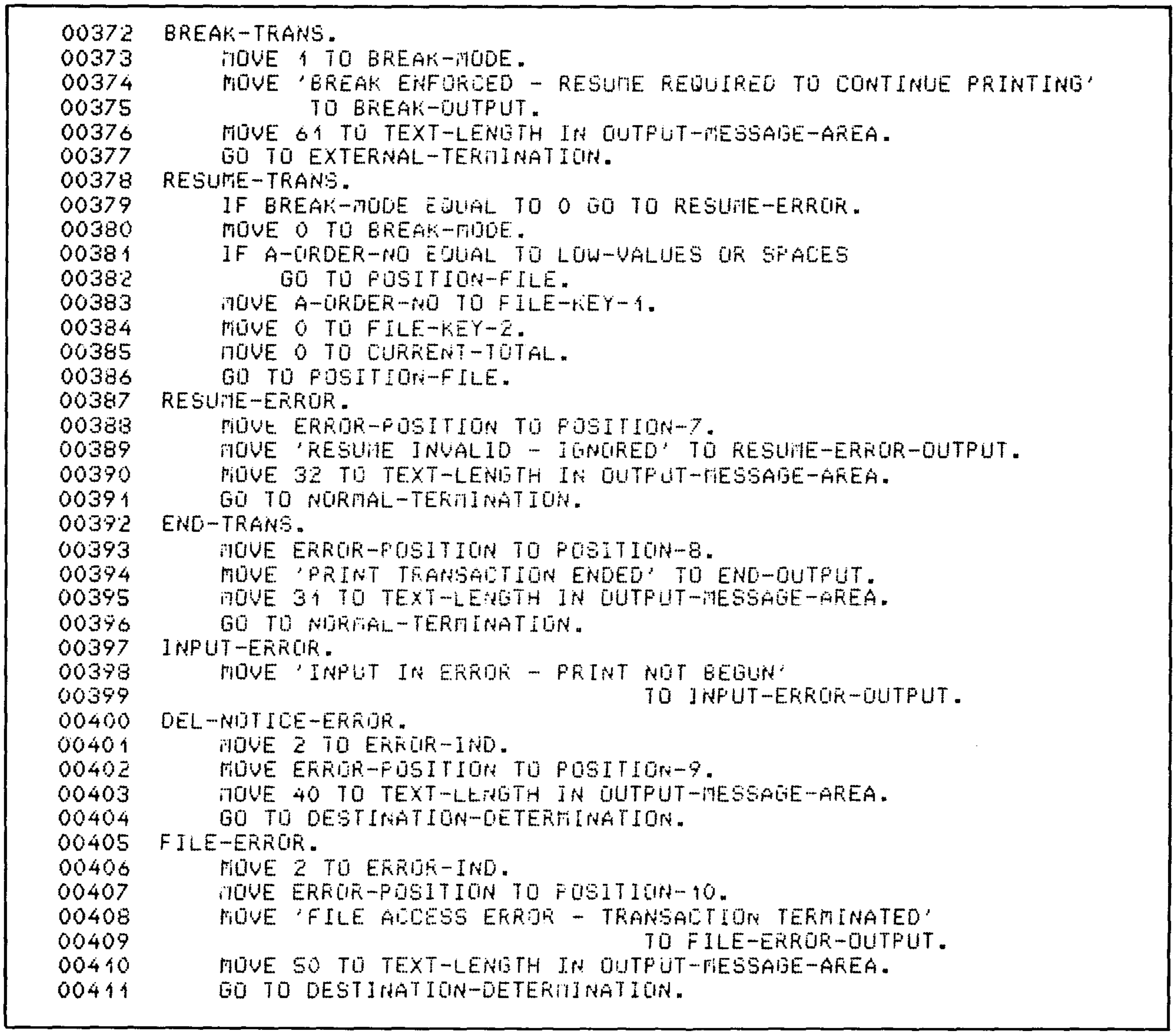
```
EXAMINE-INPUT.
    IF TRANS-CODE EQUAL TO 'PRINT' GO TO BEGIN-TRANS.
    IF COMMAND-CODE EQUAL TO 'END '   GO TO END-TRANS.
    IF COMMAND-CODE EQUAL TO 'BREAK' GO TO BREAK-TRANS.
    IF COMMAND-CODE EQUAL TO 'RESUM' GO TO RESUME-TRANS.
    IF DEL-NOTICE-CODE EQUAL TO 'END '  GO TO END-OF-FILE.
    IF DEL-NOTICE-CODE EQUAL TO CURRENT-CONT-CODE
                                 GO TO DEL-NOTICE.
    MOVE 'INVALID DELIVERY NOTICE CODE
        TO INPUT-ERROR-OUTPUT.
    GO TO DEL-NOTICE-ERROR.
BEGIN-TRANS.
    MOVE 0 TO CURRENT-ORDER-NO
    MOVE 0 TO CURRENT-ENTRY-NO
    MOVE 0 TO FILE-KEY-1  FILE-KEY-2.
    MOVE 0 TO CURRENT-TOTAL
    MOVE 0 TO BREAK-MODE
    MOVE SPACES TO INIT-TERM
    MOVE SPACES TO DEST-TERM
    MOVE AT-COP TO COP-OUTPUT.
    IF T-FILE-NAME NOT EQUAL TO 'ORDRFIL' GO TO INPUT-ERROR.
    MOVE INIT-TERMINAL TO INIT-TERM.
    MOVE SOURCE-TERMINAL-ID TO PRINT-DEST.
    IF T-ORDER-NO NOT EQUAL TO LOW-VALUES AND SPACES
        MOVE T-ORDER-NO TO FILE-KEY-1.
    MOVE T-FILE-NAME TO FILE-NAME.
POSITION-FILE.
    CALL 'SETL' USING FILE-NAME POS-GE FILE-KEY.
    IF STATUS-CODE EQUAL TO 0 GO TO READ-RECORD.
    MOVE 'END ' TO CURRENT-CONT-CODE.
    GO TO TOTAL-PROC.
READ-RECORD.
    CALL 'GET' USING FILE-NAME RECORD-AREA.
    IF STATUS-CODE EQUAL TO 1  GO TO FILE-ERROR.
    IF STATUS-CODE GREATER THAN 2 GO TO FILE-ERROR.
    IF STATUS-CODE EQUAL TO 2 GO TO END-OF-FILE.
    IF K-ORDER-ENTRY NOT EQUAL TO 0 GO TO PRODUCT-PROC.
    MOVE RECORD-AREA TO CUSTOMER-RECORD.
    IF CURRENT-TOTAL NOT EQUAL TO 0 GO TO TOTAL-PROC.
CUSTOMER-PROC.
    MOVE C-TOTAL TO INPUT-TOTAL.
    MOVE K-ORDER-NO TO CURRENT-ORDER-NO.
    MOVE K-ORDER-NO TO FILE-KEY-1.
    MOVE K-ORDER-ENTRY TO CURRENT-ENTRY-NO.
    MOVE K-ORDER-ENTRY TO FILE-KEY-2.
    ADD 1 TO FILE-KEY-2.
    MOVE HEADER-LINES TO MESSAGE-1.
    MOVE CURRENT-ORDER-NO TO M-ORDER.
    MOVE C-NAME TO M-NAME.
    MOVE C-ADDR TO M-ADDR.
    MOVE C-CITY TO M-CITY.
    MOVE C-ZIP TO M-ZIP.
    MOVE 163 TO TEXT-LENGTH IN OUTPUT-MESSAGE-AREA.
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.
EXTERNAL-TERMINATION.
    MOVE 'E' TO TERMINATION-INDICATOR.
    MOVE 'PRINTO' TO SUCCESSOR-ID.
    CALL 'RETURN'.
```

Figure 3—29. Sample COBOL Action Program Performing Continuous Output (Part 5 of 7)



```
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 VALIOITY-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.
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.
```

Figure 3—29. Sample COBOL Action Program Performing Continuous Output (Part 6 of 7)



```
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 3—29. Sample COBOL Action Program Performing Continuous Output (Part 7 of 7)

PRINT terminates in external succession after output to the operator of a message informing him of unsuccessful delivery of the last continuous output message (from 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 succession. 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).

PRINT voluntarily terminates abnormally, with a SNAP dump, whenever:

- it receives an unexpected input message on activation (line 258);
- an attempt to access some file other than 'ORDRFIL' (line 268) after having output a message to the operator (line 397);
- an unsuccessful return has been made to the STATUS-CODE field of the PIB after issuing the GET function to ORDRFIL (lines 275, 280) – again, after having notified the operator (line 405); or
- any of its error or warning messages switched to the terminal operator have not been successfully sent (line 357).

3.15.4. Output-for-Input Queueing Example

Figure 3-30 reproduces a compiler listing of a sample COBOL action program, BEGIN1, that illustrates a method of directing a print transaction using continuous output to be initiated at another terminal, using output-for-input queueing. BEGIN1 also provides for notifying the operator of the originating terminal whether the print transaction has been successfully initiated at its destination.

When BEGIN1 is activated at the originating terminal, it expects an input message in the following format (lines 61–65):

```
BEGIN dest-terminal text
```

where:

`BEGIN`
Is the 5-character transaction code entered by the terminal operator to activate BEGIN1 (specified to the configurator in the TRANSACT section).

`dest-terminal`
Is the 4-character *terminal-id* of the destination terminal, at which the continuous output print transaction is to be initiated. (This must have been assigned in the ICAM network definition.)

text

Is the alphanumeric text input by the originating terminal operator. This text is the input message expected by the print transaction that is to perform continuous output at the destination terminal and must begin with the transaction code that will cause scheduling to initiate it.

On activation of BEGIN1 the MOVE-MESSAGE procedure forms an output message to be queued as input for the destination terminal:

1. Line 94 places the *dest-terminal* input in the OMA header.

2. Line 95 specifies the length of the output message, including four bytes for the TEXT-LENGTH and AUXILIARY-DEVICE-ID fields.

3. Line 97 sets the AUXILIARY–FUNCTION field of the OMA header to the value (X'C9') that directs IMS 90 to queue the output message as input for the destination terminal (3.10.2).

4. Line 99 transmits the explicit output message to the destination terminal via the SEND function.

If there are no errors encountered by IMS 90 in executing the SEND function, the operator of the originating terminal receives a message indicating that the print transaction has been successfully queued at the destination terminal:

1. Lines 101 and 102 provide the text of the message output for the originating operator.

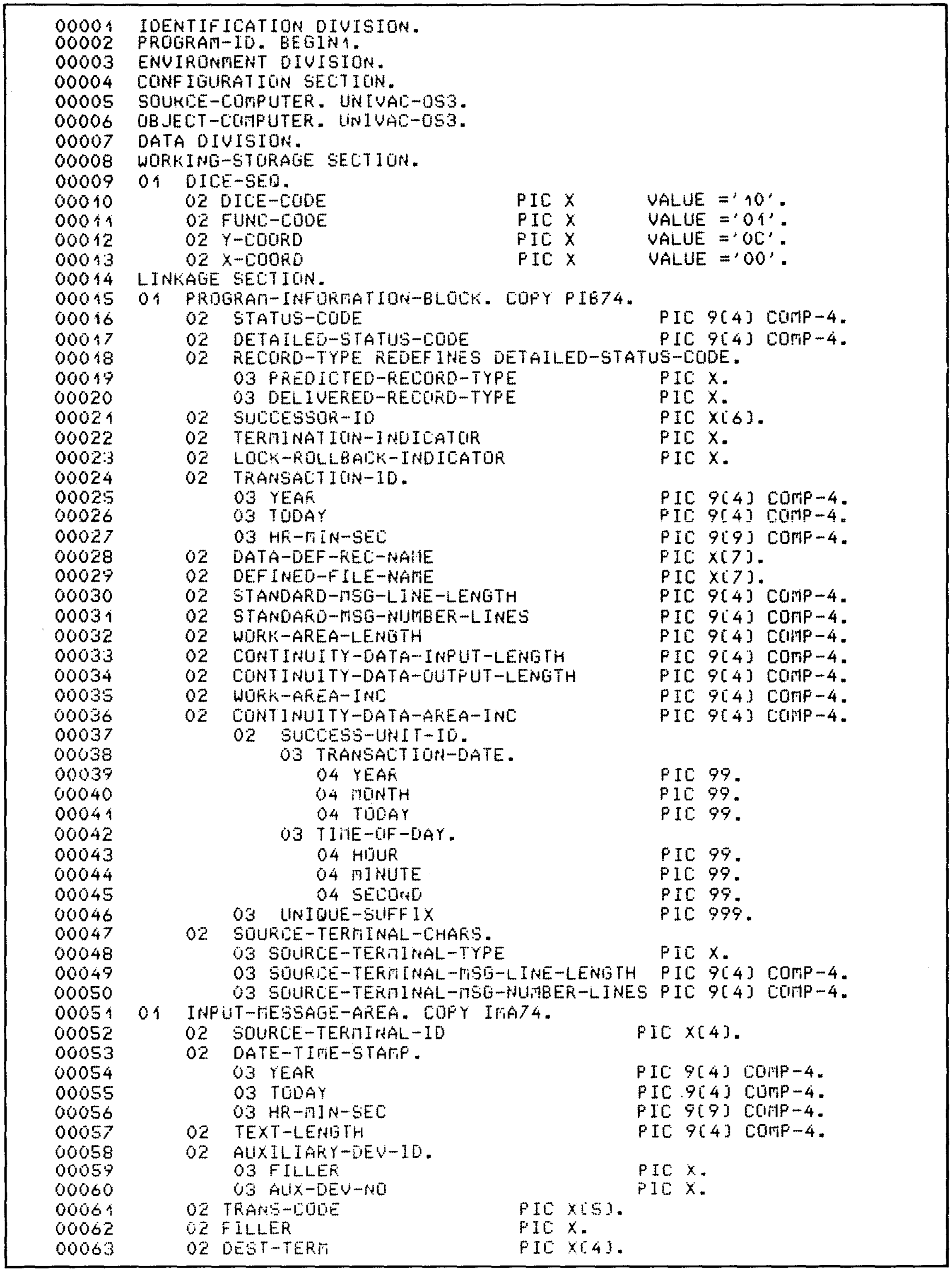
2. Line 106 sets the DESTINATION-TERMINAL-ID field of the OMA header to binary 0 and thus ensures that this message is output at the originating terminal (assumed to be a UNISCOPE device).

3. Line 107 ensures that this message is output on the UNISCOPE screen instead of on the COP.

BEGIN1 terminates normally, without succession. The originating terminal is now free for other interactive use.

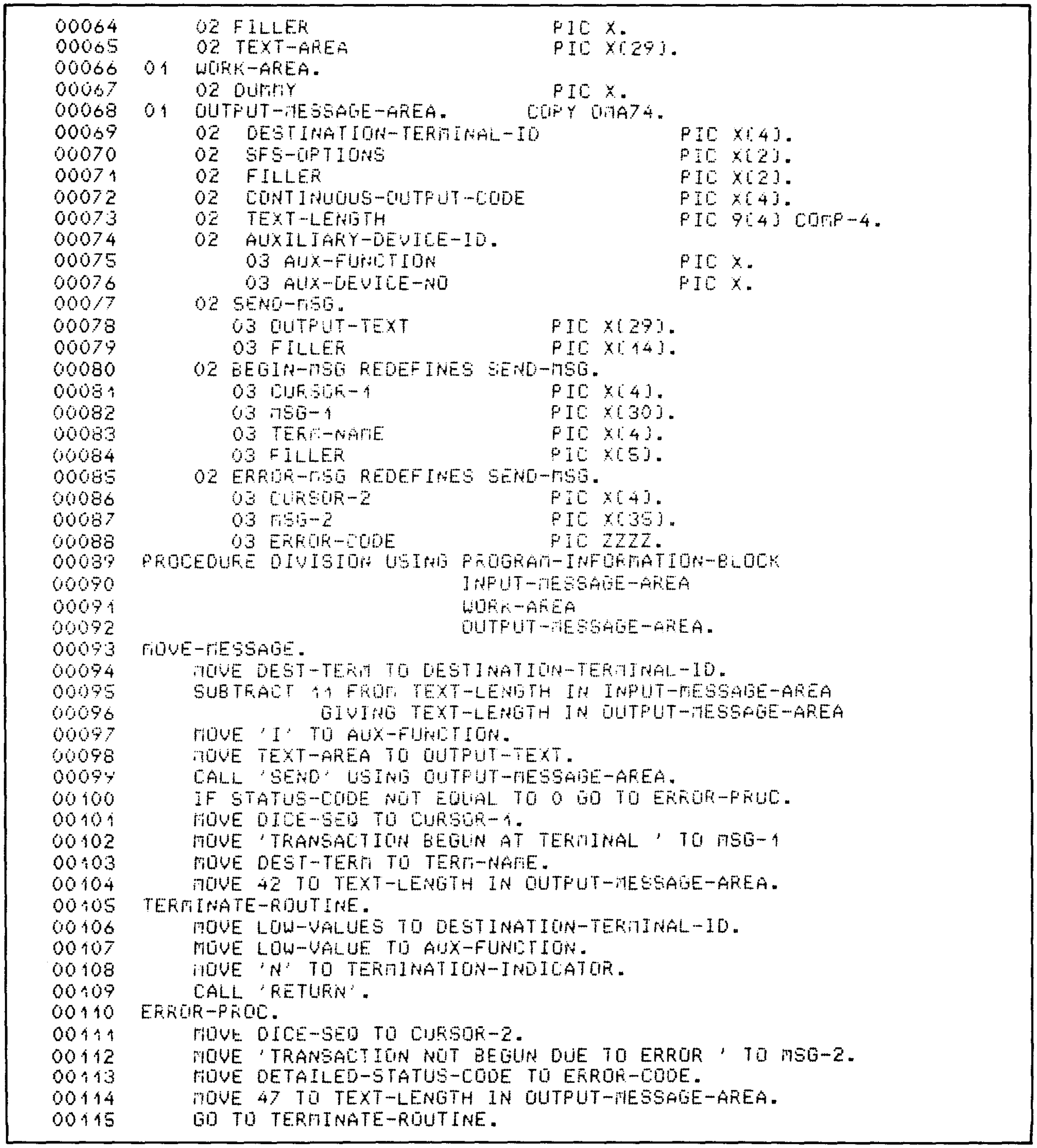
On the other hand, if IMS 90 encounters an error in queueing the message output by BEGIN1 as input to the destination terminal, the ERROR-PROC procedure formats an error message for output to the originating operator, and BEGIN1 terminates normally (lines 100 and 110–115). The output message is dequeued. The operator, depending on the nature of the error, may reenter the original input message.

Although the text of the message output to the originating terminal on successful return from the SEND function (line 102) states that the transaction has begun at the destination terminal, this may not be true. All that has actually occurred is that the output message has been 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. Such occurrences are not reported to the originating action program by IMS 90, but to the destination terminal.



```
IDENTIFICATION DIVISION.
PROGRAM-ID. BEGIN1.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. UNIVAC-OS3.
OBJECT-COMPUTER. UNIVAC-OS3.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  DICE-SEQ.
     02 DICE-CODE                  PIC X       VALUE ='10'.
     02 FUNC-CODE                  PIC X       VALUE ='01'.
     02 Y-COORD                    PIC X       VALUE ='0C'.
     02 X-COORD                    PIC X       VALUE ='00'.
LINKAGE SECTION.
01  PROGRAM-INFORMATION-BLOCK. 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.
    02  CONTINUITY-DATA-OUTPUT-LENGTH              PIC 9(4) COMP-4.
    02  WORK-AREA-INC                              PIC 9(4) COMP-4.
    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  UNIQUE-SUFFIX                          PIC 999.
    02  SOURCE-TERMINAL-CHARS.
        03 SOURCE-TERMINAL-TYPE                    PIC X.
        03 SOURCE-TERMINAL-MSG-LINE-LENGTH         PIC 9(4) COMP-4.
        03 SOURCE-TERMINAL-MSG-NUMBER-LINES PIC 9(4) COMP-4.
01  INPUT-MESSAGE-AREA. 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-CODE                  PIC X(5).
    02 FILLER                      PIC X.
    02 DEST-TERM                   PIC X(4).
```

Figure 3—30. Sample COBOL Action Program, Directing Print Transaction at Another Terminal (Part 1 of 2)



```
   02 FILLER                     PIC X.
   02 TEXT-AREA                  PIC X(29).
01 WORK-AREA.
   02 DUMMY                      PIC X.
01 OUTPUT-MESSAGE-AREA.        COPY OMA74.
   02  DESTINATION-TERMINAL-ID           PIC X(4).
   02  SFS-OPTIONS                       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 SEND-MSG.
      03 OUTPUT-TEXT             PIC X(29).
      03 FILLER                  PIC X(14).
   02 BEGIN-MSG REDEFINES SEND-MSG.
      03 CURSOR-1                PIC X(4).
      03 MSG-1                   PIC X(30).
      03 TERM-NAME               PIC X(4).
      03 FILLER                  PIC X(5).
   02 ERROR-MSG REDEFINES SEND-MSG.
      03 CURSOR-2                PIC X(4).
      03 MSG-2                   PIC X(35).
      03 ERROR-CODE              PIC ZZZZ.
PROCEDURE DIVISION USING PROGRAM-INFORMATION-BLOCK
                         INPUT-MESSAGE-AREA
                         WORK-AREA
                         OUTPUT-MESSAGE-AREA.
MOVE-MESSAGE.
   MOVE DEST-TERM TO DESTINATION-TERMINAL-ID.
   SUBTRACT 11 FROM TEXT-LENGTH IN INPUT-MESSAGE-AREA
           GIVING TEXT-LENGTH IN OUTPUT-MESSAGE-AREA
   MOVE 'I' TO AUX-FUNCTION.
   MOVE TEXT-AREA TO OUTPUT-TEXT.
   CALL 'SEND' USING OUTPUT-MESSAGE-AREA.
   IF STATUS-CODE NOT EQUAL TO 0 GO TO ERROR-PROC.
   MOVE DICE-SEQ TO CURSOR-1.
   MOVE 'TRANSACTION BEGUN AT TERMINAL ' TO MSG-1
   MOVE DEST-TERM TO TERM-NAME.
   MOVE 42 TO TEXT-LENGTH IN OUTPUT-MESSAGE-AREA.
TERMINATE-ROUTINE.
   MOVE LOW-VALUES TO DESTINATION-TERMINAL-ID.
   MOVE LOW-VALUE TO AUX-FUNCTION.
   MOVE 'N' TO TERMINATION-INDICATOR.
   CALL 'RETURN'.
ERROR-PROC.
   MOVE DICE-SEQ TO CURSOR-2.
   MOVE 'TRANSACTION NOT BEGUN DUE TO ERROR ' TO MSG-2.
   MOVE DETAILED-STATUS-CODE TO ERROR-CODE.
   MOVE 47 TO TEXT-LENGTH IN OUTPUT-MESSAGE-AREA.
   GO TO TERMINATE-ROUTINE.
```

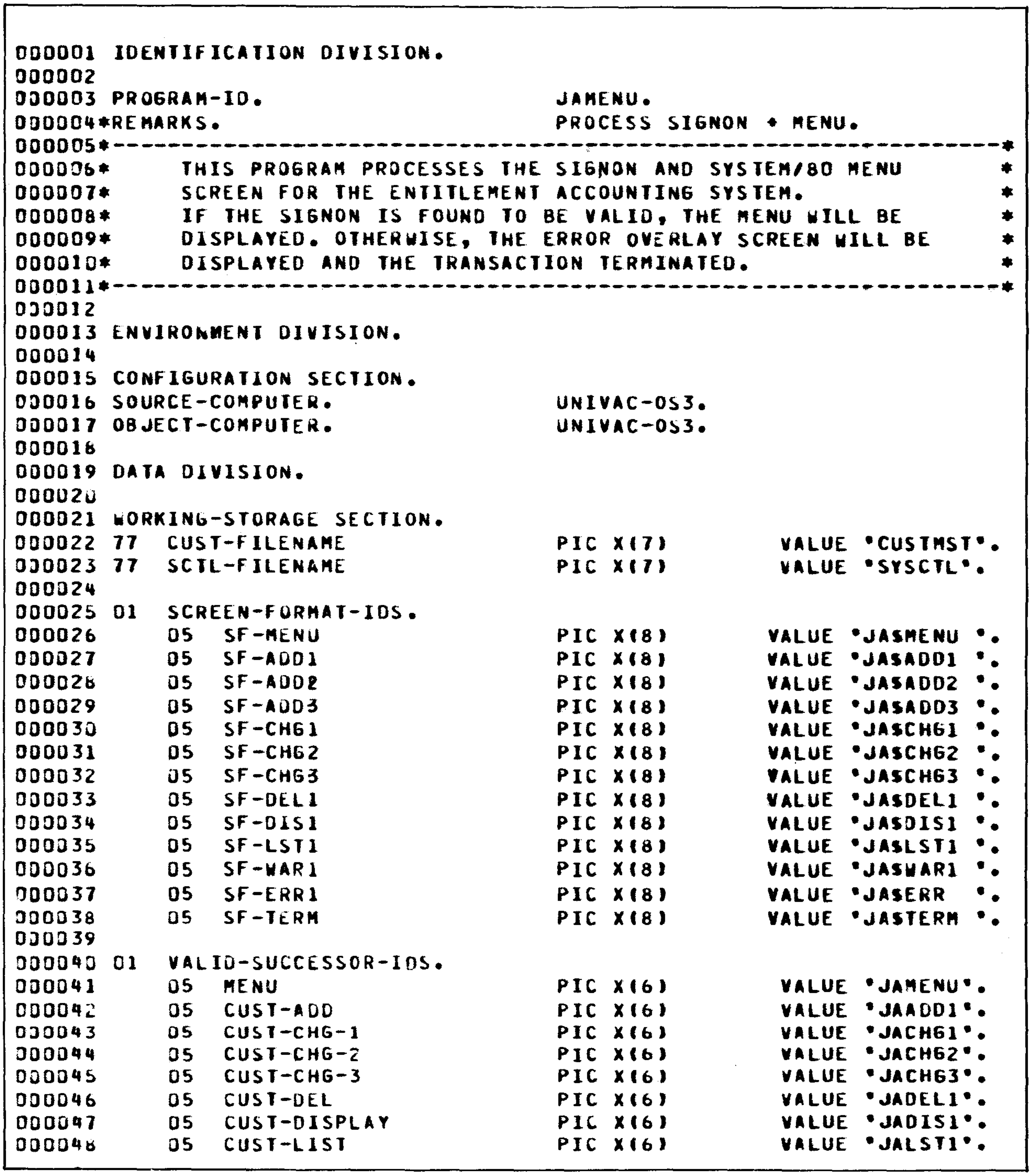
Figure 3—30. Sample COBOL Action Program, Directing Print Transaction at Another Terminal (Part 2 of 2)

3.15.5. Sample COBOL Program Using Screen Format Services

Figure 3-31 is a compiler listing of the action program, JAMENU. This program is the first in a series of programs that make up an entitlement accounting system. JAMENU processes a password entered as input from the terminal. If the input is valid, JAMENU displays a menu screen. If the input isn't valid, JAMENU displays an error screen and terminates.

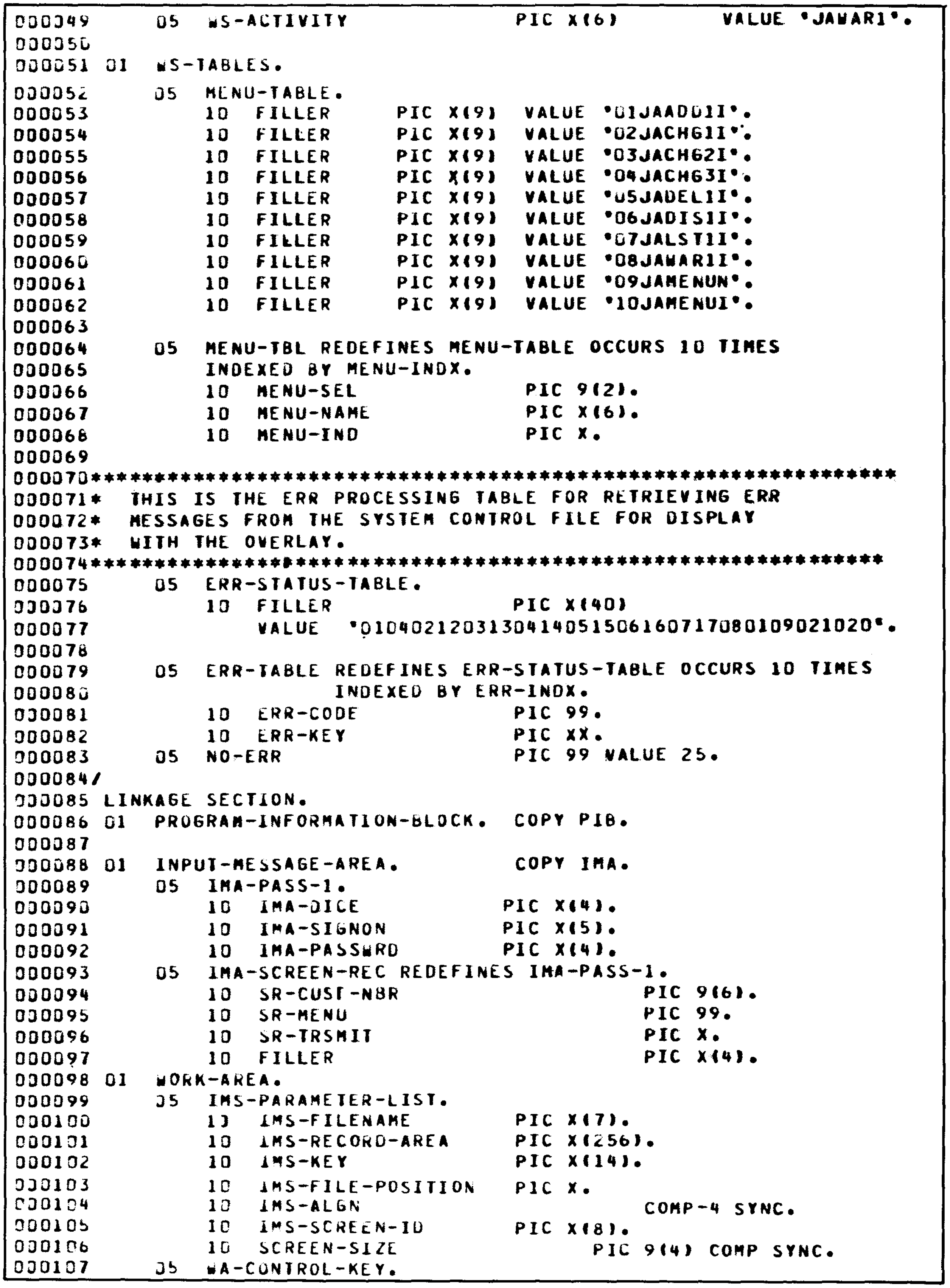
Remember, before using screen formats in your action program, you must:

- create the screen formats and store them in the screen format file ($Y$FMT)
- specify the SFS=n parameter in the OPTIONS section of the configuration
- specify OFT=+n in the IMS start-up job stream



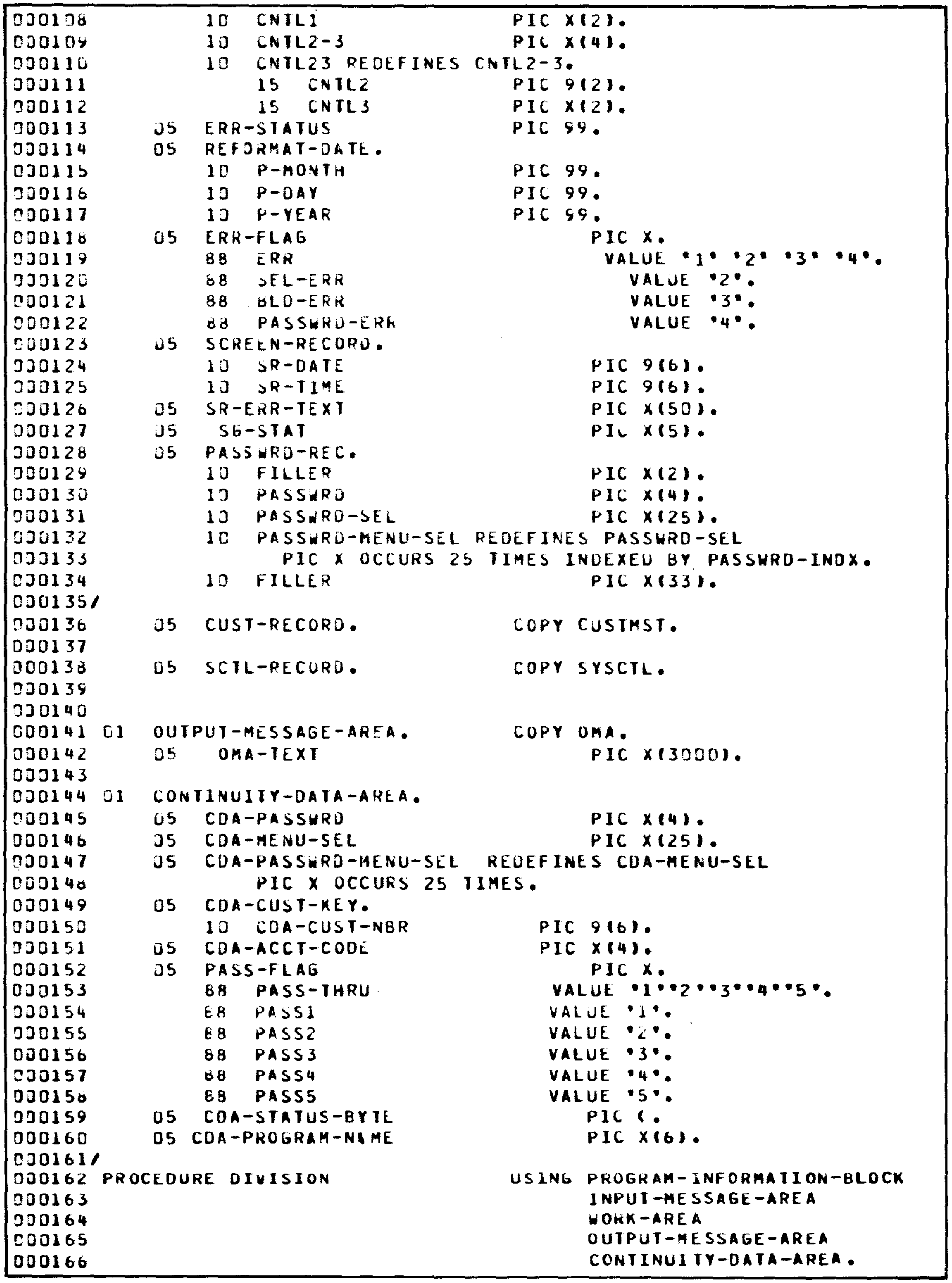
```
000001 IDENTIFICATION DIVISION.
000002
000003 PROGRAM-ID.                          JAMENU.
000004*REMARKS.                             PROCESS SIGNON + MENU.
000005*-----------------------------------------------------------------*
000006*     THIS PROGRAM PROCESSES THE SIGNON AND SYSTEM/80 MENU        *
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'.
```

Figure 3—31. Sample COBOL Program Using Screen Formats (Part 1 of 8)



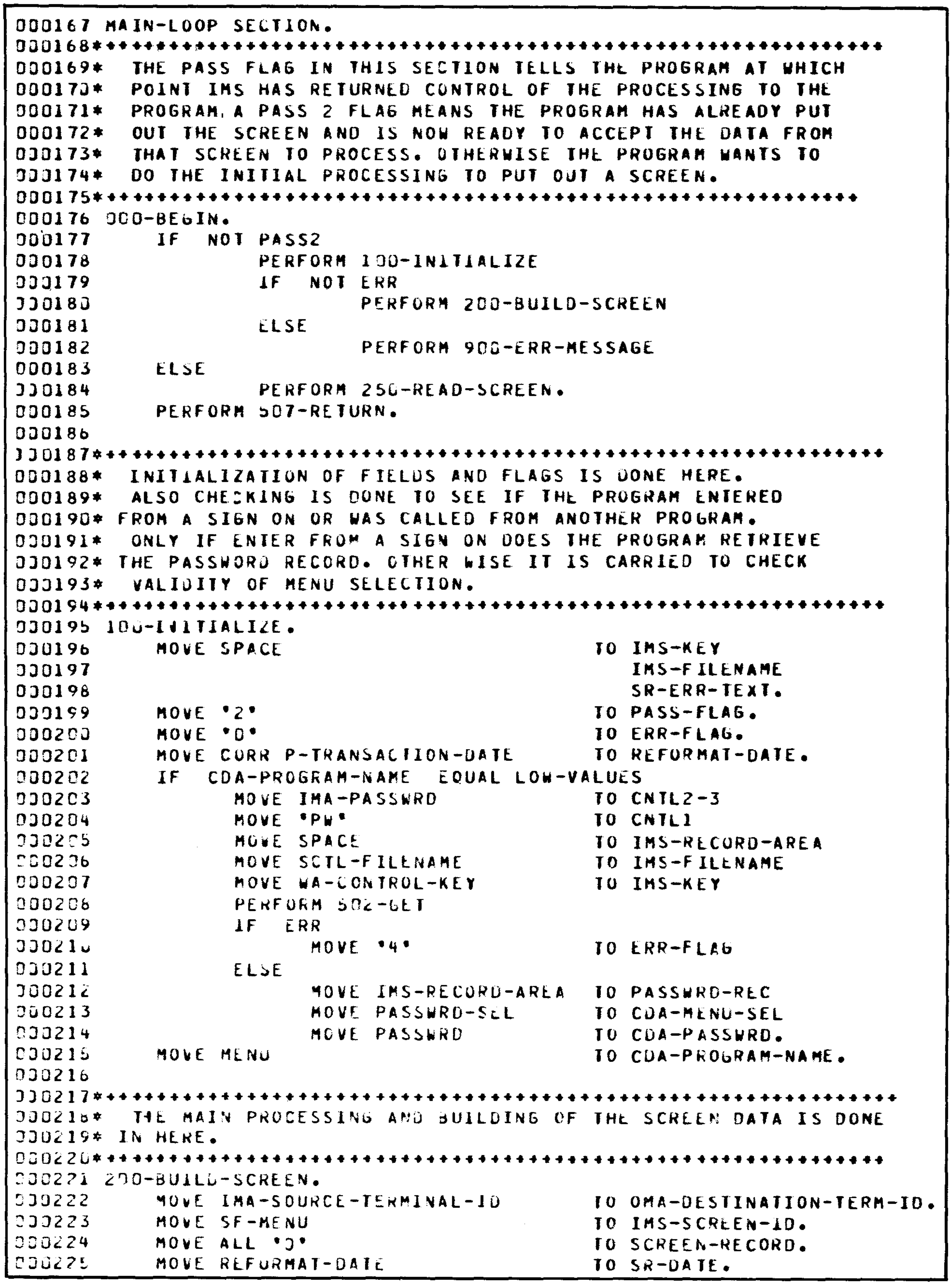
```
000049      05  WS-ACTIVITY                 PIC X(6)        VALUE 'JAWAR1'.
000050
000051 01   WS-TABLES.
000052      05  MENU-TABLE.
000053          10  FILLER      PIC X(9)  VALUE '01JAADD1I'.
000054          10  FILLER      PIC X(9)  VALUE '02JACHG1I'.
000055          10  FILLER      PIC X(9)  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******************************************************************
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.
```

Figure 3—31. Sample COBOL Program Using Screen Formats (Part 2 of 8)



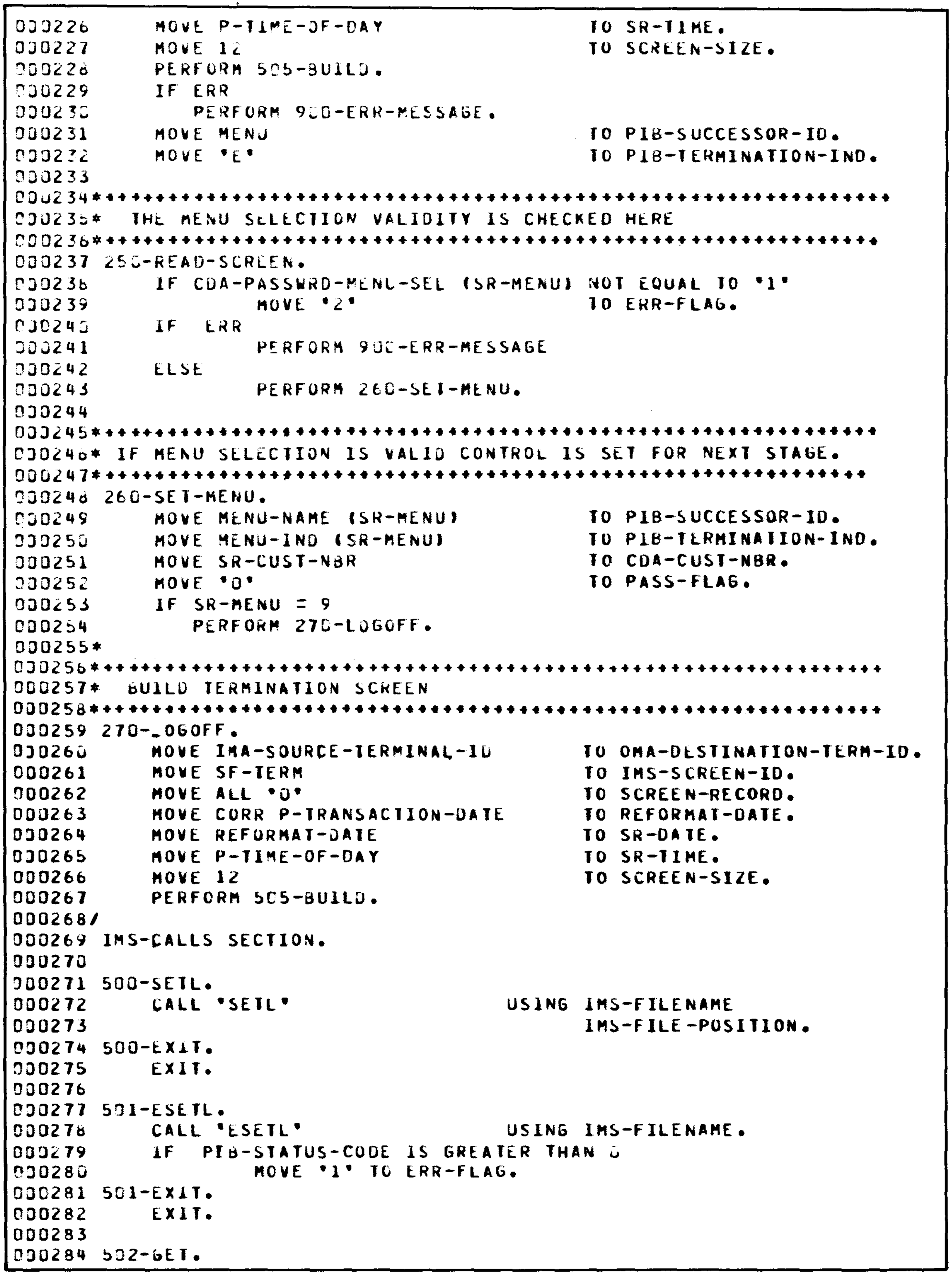
```
000108         10  CNTL1               PIC X(2).
000109         10  CNTL2-3             PIC X(4).
000110         10  CNTL23 REDEFINES CNTL2-3.
000111             15  CNTL2           PIC 9(2).
000112             15  CNTL3           PIC X(2).
000113     05  ERR-STATUS              PIC 99.
000114     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.
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 (.
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.
```

Figure 3—31. Sample COBOL Program Using Screen Formats (Part 3 of 8)



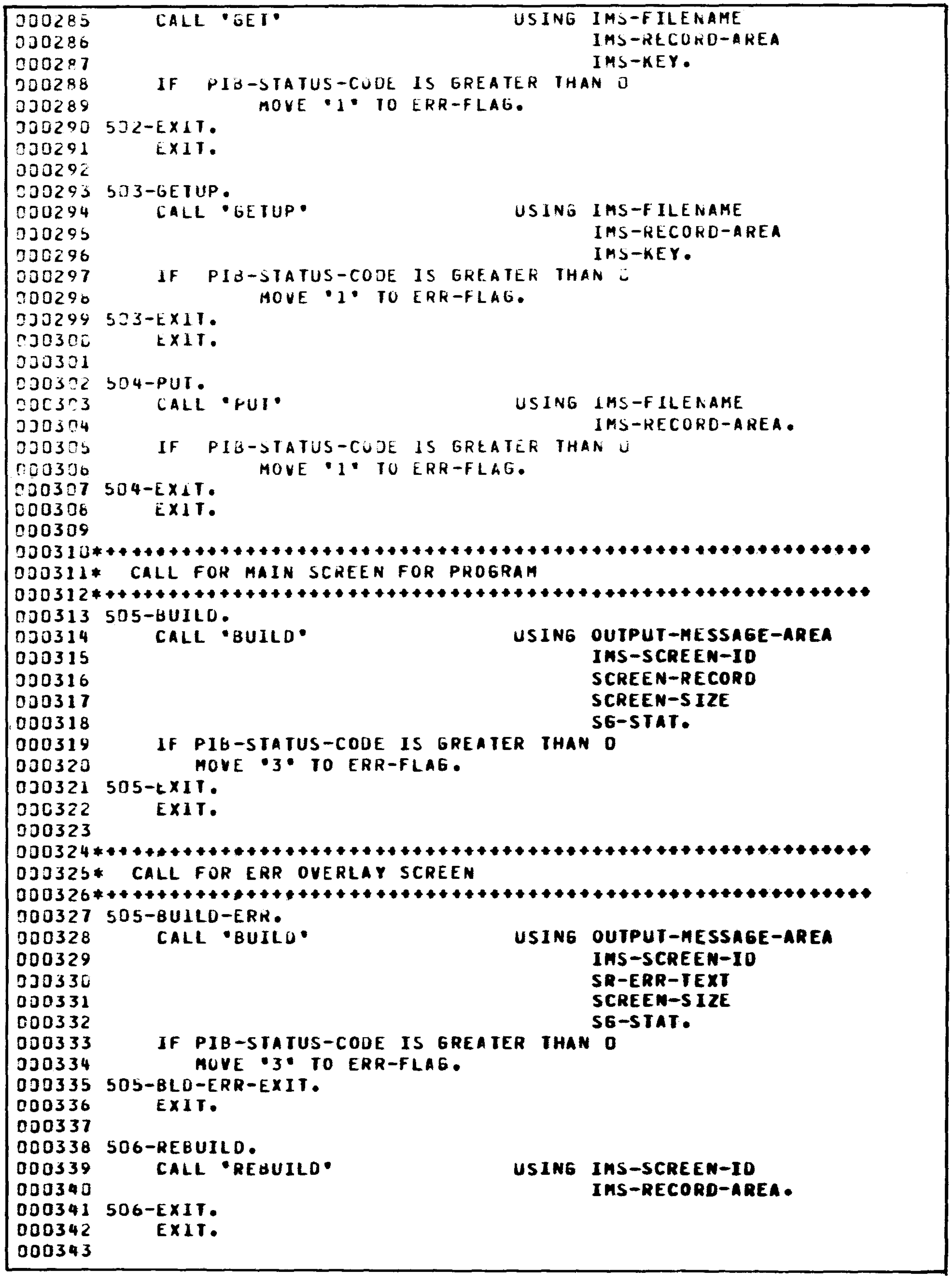
```
000167 MAIN-LOOP SECTION.
000168**************************************************************
000169*  THE PASS FLAG IN THIS SECTION TELLS THE PROGRAM AT WHICH
000170*  POINT IMS HAS RETURNED CONTROL OF THE PROCESSING TO THE
000171*  PROGRAM, A PASS 2 FLAG MEANS THE PROGRAM HAS ALREADY PUT
000172*  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
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.
```

Figure 3—31. Sample COBOL Program Using Screen Formats (Part 4 of 8)



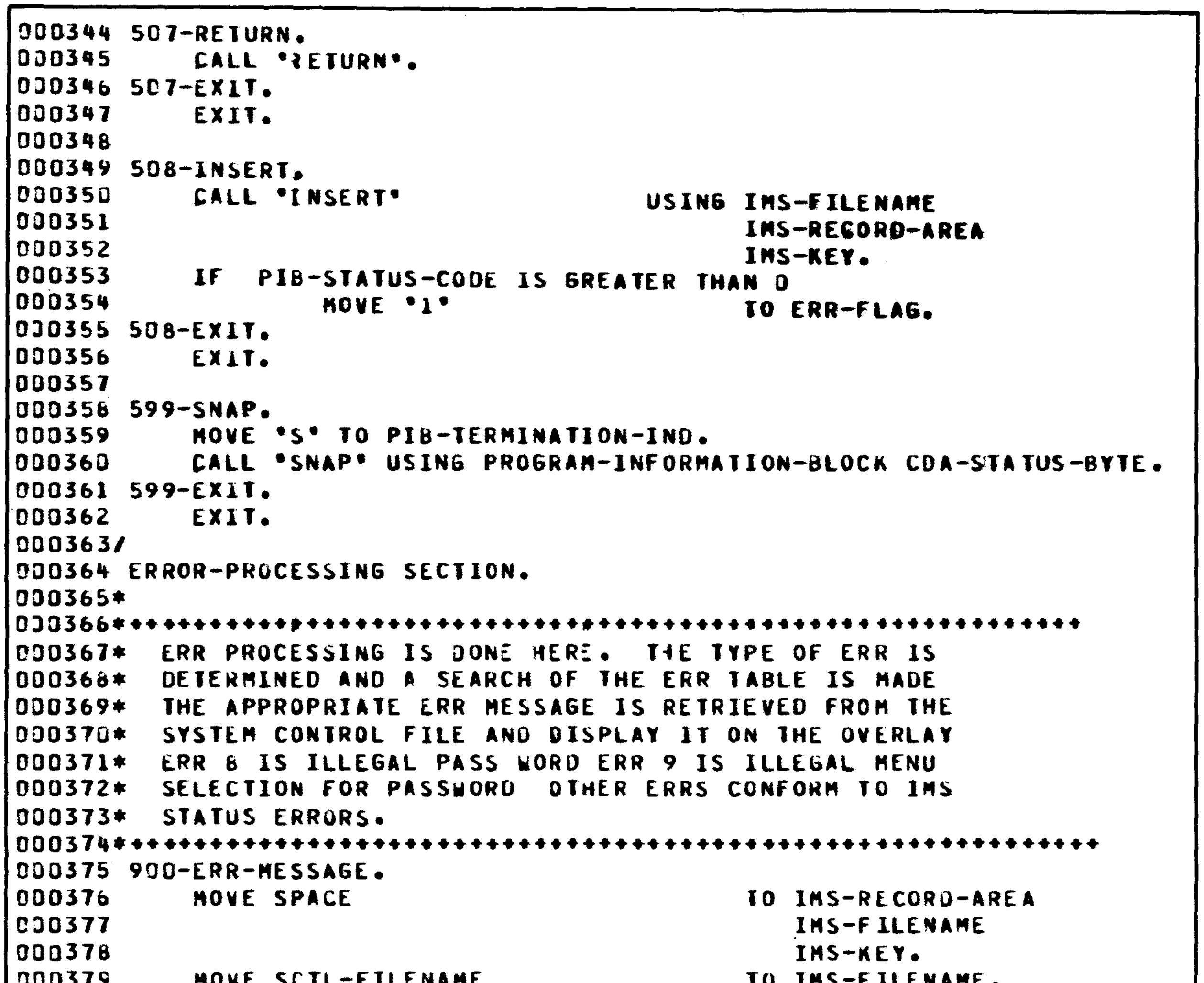
```
000226     MOVE P-TIME-OF-DAY                 TO SR-TIME.
000227     MOVE 12                            TO SCREEN-SIZE.
000228     PERFORM 505-BUILD.
000229     IF ERR
000230        PERFORM 900-ERR-MESSAGE.
000231     MOVE MENU                          TO PIB-SUCCESSOR-ID.
000232     MOVE 'E'                           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.
```

Figure 3—31. Sample COBOL Program Using Screen Formats (Part 5 of 8)



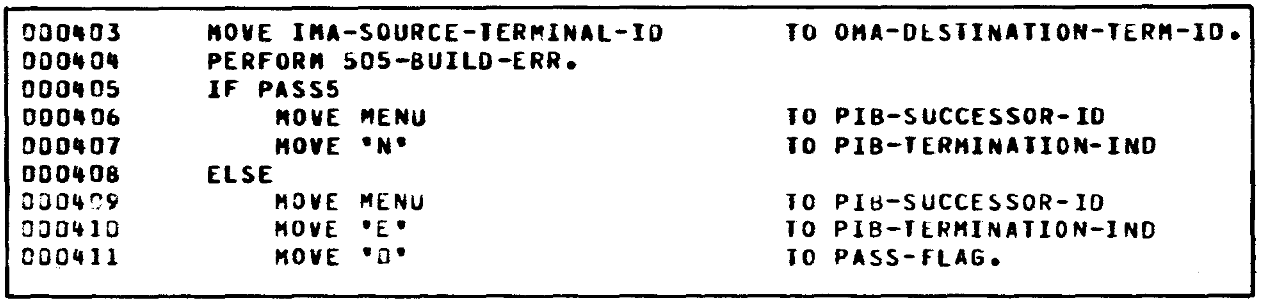
```
000285     CALL 'GET'                  USING IMS-FILENAME
000286                                       IMS-RECORD-AREA
000287                                       IMS-KEY.
000288     IF  PIB-STATUS-CODE IS GREATER THAN 0
000289             MOVE '1' TO ERR-FLAG.
000290 502-EXIT.
000291     EXIT.
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
```

Figure 3—31. Sample COBOL Program Using Screen Formats (Part 6 of 8)



```
 507-RETURN.
     CALL 'RETURN'.
 507-EXIT.
     EXIT.

 508-INSERT.
     CALL 'INSERT'               USING IMS-FILENAME
                                       IMS-RECORD-AREA
                                       IMS-KEY.
     IF  PIB-STATUS-CODE IS GREATER THAN 0
             MOVE '1'                  TO ERR-FLAG.
 508-EXIT.
     EXIT.

 599-SNAP.
     MOVE 'S' TO PIB-TERMINATION-IND.
     CALL 'SNAP' USING PROGRAM-INFORMATION-BLOCK CDA-STATUS-BYTE.
 599-EXIT.
     EXIT.
/
 ERROR-PROCESSING SECTION.
*
************************************************************
*  ERR PROCESSING IS DONE HERE.  THE TYPE OF ERR IS
*  DETERMINED AND A SEARCH OF THE ERR TABLE IS MADE
*  THE APPROPRIATE ERR MESSAGE IS RETRIEVED FROM THE
*  SYSTEM CONTROL FILE AND DISPLAY IT ON THE OVERLAY
*  ERR 8 IS ILLEGAL PASS WORD ERR 9 IS ILLEGAL MENU
*  SELECTION FOR PASSWORD  OTHER ERRS CONFORM TO IMS
*  STATUS ERRORS.
*************************************************************
 900-ERR-MESSAGE.
     MOVE SPACE                        TO IMS-RECORD-AREA
                                          IMS-FILENAME
                                          IMS-KEY.
     MOVE SCTL-FILENAME                TO IMS-FILENAME.
     IF PASSWRD-ERR
        MOVE '5'                       TO PASS-FLAG
        MOVE 8                         TO ERR-STATUS
     ELSE
     IF  SEL-ERR
             MOVE 9                    TO ERR-STATUS
     ELSE
         MOVE PIB-STATUS-CODE          TO ERR-STATUS.
     MOVE 'EM'                         TO CNTL1.
     MOVE SPACE                        TO CNTL3.
     SET ERR-INDX                      TO 1.
     SEARCH ERR-TABLE
         AT END
             MOVE NO-ERR               TO CNTL2
         WHEN ERR-CODE (ERR-INDX) IS EQUAL TO ERR-STATUS
             MOVE ERR-KEY (ERR-INDX)   TO CNTL2.
     MOVE WA-CONTROL-KEY               TO IMS-KEY.
     PERFORM 502-GET.
     MOVE IMS-RECORD-AREA              TO SCTL-RECORD.
     MOVE ALL '0' TO SR-ERR-TEXT.
     MOVE SCERR-TEXT                   TO SR-ERR-TEXT.
     MOVE SF-ERR1                      TO IMS-SCREEN-ID.
     MOVE 50          TO SCREEN-SIZE.
```

Figure 3—31. Sample COBOL Program Using Screen Formats (Part 7 of 8)

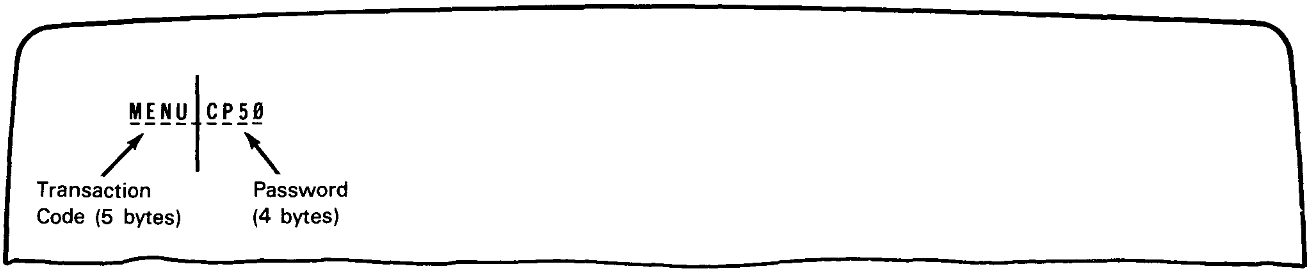


```
000403     MOVE IMA-SOURCE-TERMINAL-ID          TO OMA-DESTINATION-TERM-ID.
000404     PERFORM 505-BUILD-ERR.
000405     IF PASS5
000406         MOVE MENU                        TO PIB-SUCCESSOR-ID
000407         MOVE 'N'                         TO PIB-TERMINATION-IND
000408     ELSE
000409         MOVE MENU                        TO PIB-SUCCESSOR-ID
000410         MOVE 'E'                         TO PIB-TERMINATION-IND
000411         MOVE '0'                         TO PASS-FLAG.
```

Figure 3—31. Sample COBOL Program Using Screen Formats (Part 8 of 8)

The following discussion of the JAMENU action program assumes that you have already created a menu screen format called JA$MENU and filed it on the $Y$FMT screen format file.

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.



JAMENU uses two files (lines 22 and 23):

1. CUSTMST file

2. SYSCTL 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 130.) JAMENU accesses the SYSCTL file to validate passwords and retrieve error messages for display on the error message screen.

JAMENU performs five types of routines. It:

1. validates passwords

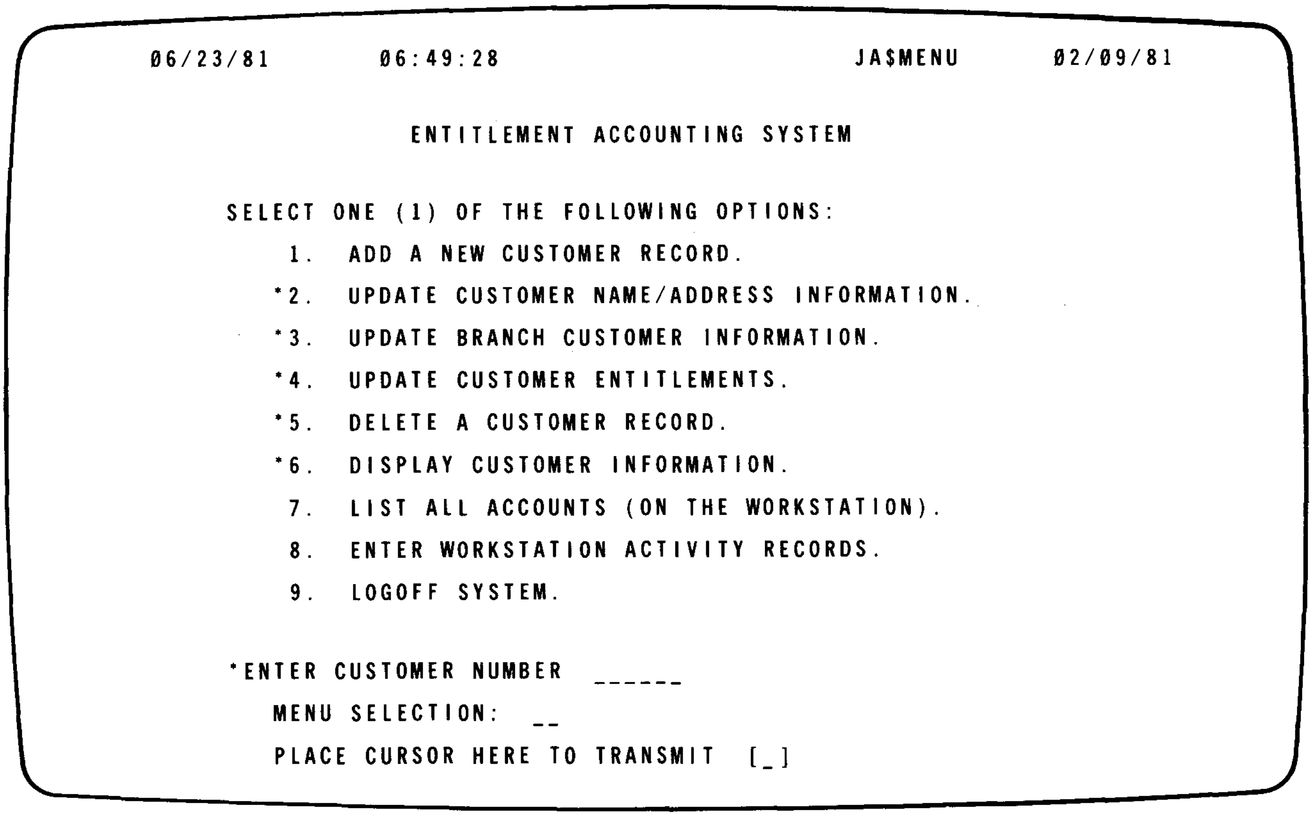
2. builds menu screen

3. validates menu selections

4. builds error screen

5. builds termination screen

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.



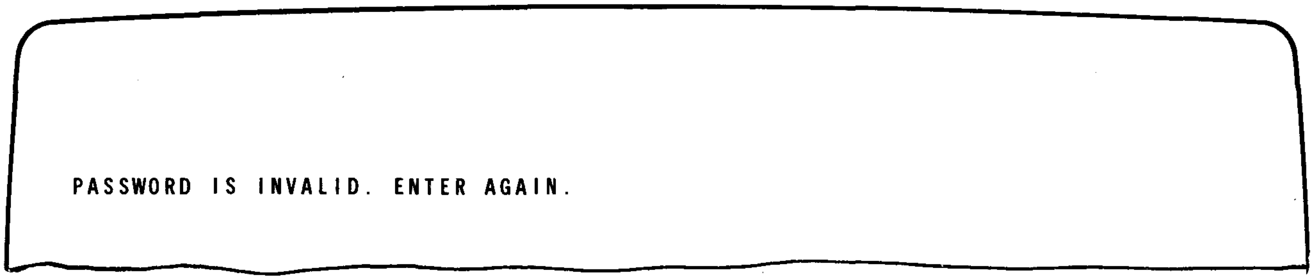
```
     06/23/81        06:49:28                        JA$MENU       02/09/81

                         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  [_]
```

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 line 105); the variable data as SCREEN-RECORD (line 316, defined 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.

If the BUILD function is unsuccessful, an error code 3 is moved to the ERR-FLAG (lines 118 and 121) indicating a build error.

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 IS INVALID. ENTER AGAIN.
```

On the second pass through JAMENU, the program tests the menu selection made, seeing 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 to the successor-id the correct program name to process the menu selection and an I to the termination-indicator. If the menu selection was invalid, JAMENU moves the 2 indicating selection error to ERR-FLAG, builds the error message screen (lines 375-411), and succeeds externally to itself.

If the menu selection (customer number where applicable and menu number) was 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. 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.

3.16. SAMPLE RPG II ACTION PROGRAMS

The two RPG II action program examples described in this discussion implement a dialog inquiry transaction with external succession and use all the IMS 90 interface areas (IMA, PIB, OMA, and CDA). RPG II action programs do not use the work area. For an example of an RPG II action program that implements a simple transaction structure with only the IMA and OMA IMS 90 interface areas and the entire procedure for getting the action program online, see Appendix C.

The RPG II action programs in Figures 3-32 through 3-39 reference the same user files (STATE and CITY files) as the preceding COBOL action program examples (ACT1 and ACT2). The STATE file contains a record for each state; each state record contains the state name, state population, and capital city name. The CITY file contains a record for each city; each city record contains the city name, city population, and state name.

To activate the transaction, the UNISCOPE 100 operator enters the transaction code S and the name of a state. If the requested state record exists, the RPG II action program (ACT1) places the state name, population, and capital name in the OMA with the output message heading and control characters. Figure 3-24 shows the results at the UNISCOPE 100.

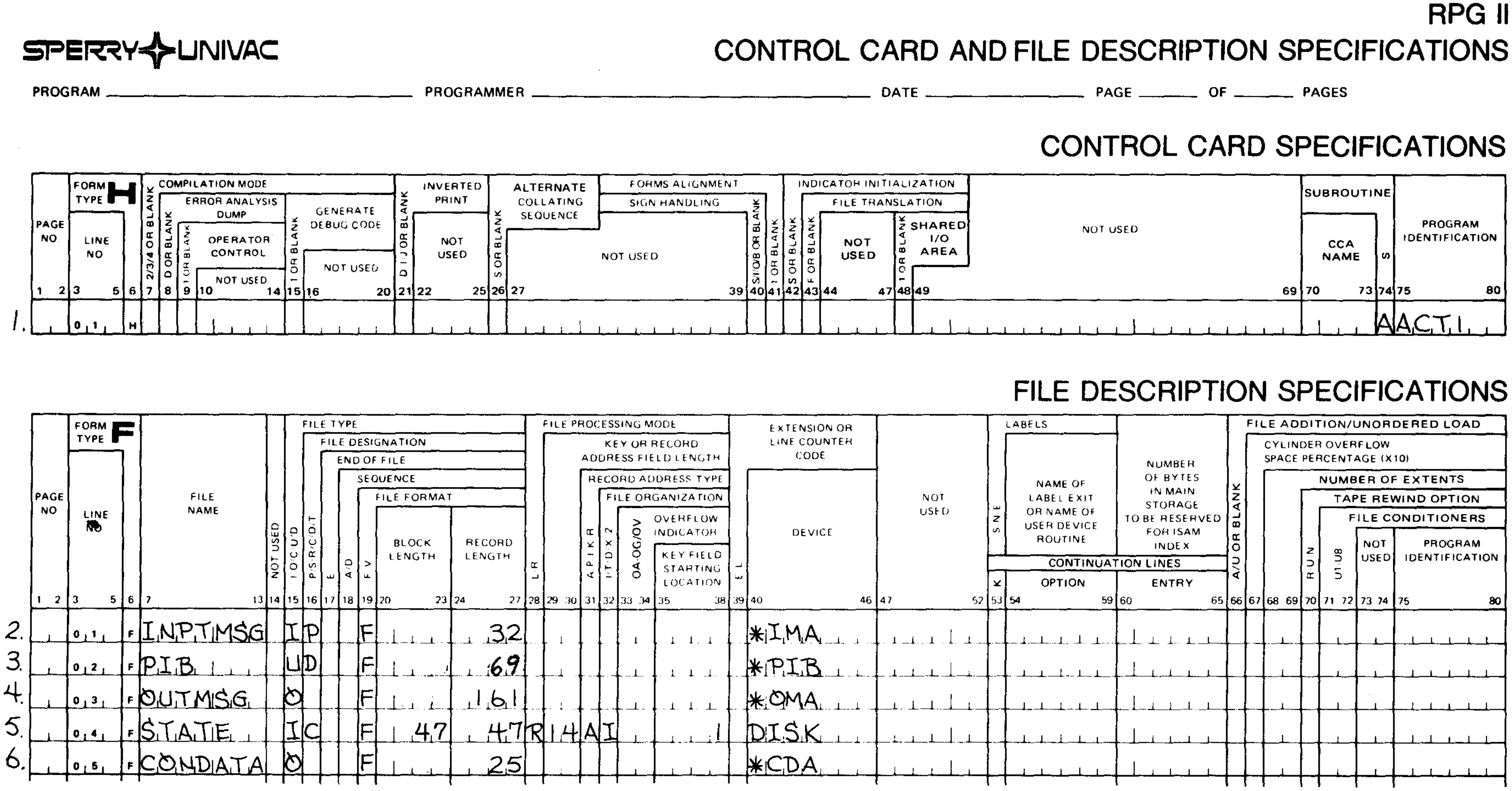
ACT1 saves the name of the capital city in the continuity data area (CDA). When ACT1 designates ACT2 as its external successor and moves the program-id (ACT2) and termination code E to the PIB, the capital city name is automatically passed to the successor action program, which retrieves capital population from the CITY file.

Following the display of requested information, the output message CAPITAL-POP? ▷NO YES (described in the OMA) allows the user to request either the capital city population by pressing the TAB and then TRANSMIT keys or a termination of the transaction by pressing only the TRANSMIT key.

Figure 3-24 shows the results when the capital city population is requested. The second action program (ACT2) obtains the CITY record for the capital city named in the CDA, builds an output message containing the capital population, and then terminates normally. Figure 3-25 shows the results when termination of the transaction is requested; i.e., no capital city population is requested (NO).

3.16.1. ACT1 Discussion

Figures 3-32 through 3-35 illustrate the RPG II specifications forms required to code ACT1, the first of the two action programs needed to provide information about a state from the STATE file. A description of each line of coding is provided.



SPERRY UNIVAC

RPG II

CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE ____ OF ____ PAGES

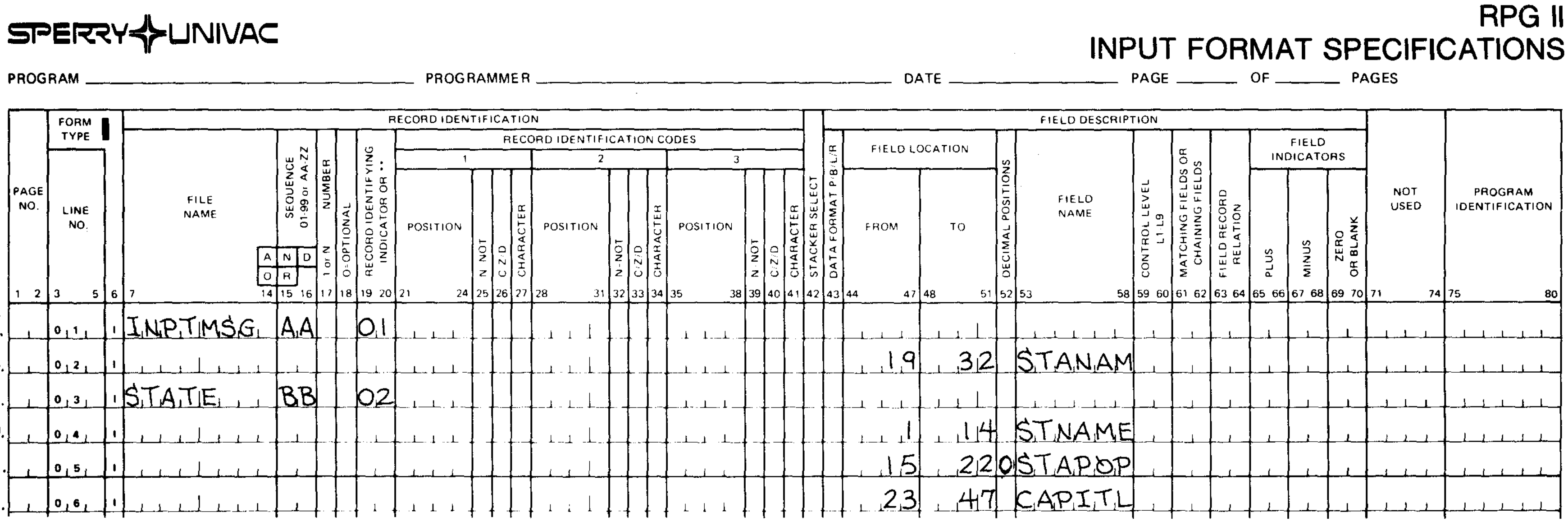
CONTROL CARD SPECIFICATIONS

	Page No (1-2)	Line No (3-5)	Form Type H (6)	Compilation Mode / Error Analysis Dump / Operator Control (7-20)	Inverted Print (21-25)	Alternate Collating Sequence / Forms Alignment / Sign Handling (26-40)	Indicator Initialization / File Translation / Shared I/O Area (41-48)	Not Used (49-69)	Subroutine CCA Name (70-73)	S (74)	Program Identification (75-80)
1.		01	H							A	ACT1

FILE DESCRIPTION SPECIFICATIONS

	Page No (1-2)	Line No (3-5)	Form Type F (6)	File Name (7-13)	I/O/C/U/D (15)	P/S/R/C/D/T (16)	E (17)	A/D (18)	F/V (19)	Block Length (20-23)	Record Length (24-27)	L/R (28)	Key or Record Address Field Length (29-30)	Record Address Type A/P/I/K (31)	File Organization I/T/D/X/2 (32)	Overflow Indicator (33-34)	Key Field Starting Location (35-38)	E/L (39)	Device (40-46)	Not Used (47-52)	Labels (53-65)	File Addition/Unordered Load (66-80)
2.		01	F	INPTMSG	I	P			F		32								*IMA			
3.		02	F	PIB	U	D			F		69								*PIB			
4.		03	F	OUTMSG	O				F		161								*OMA			
5.		04	F	STATE	I	C			F	47	47	R	14	A	I		1		DISK			
6.		05	F	CONDATA	O				F		25								*CDA			

Figure 3—32. ACT1 Control Card and File Description Specifications Forms



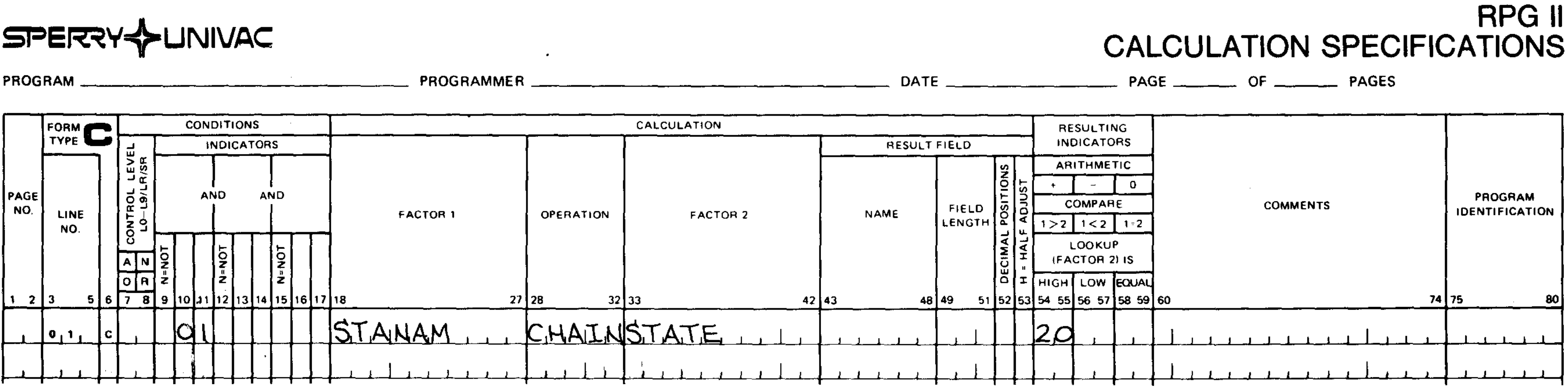
SPERRY✦UNIVAC

RPG II
INPUT FORMAT SPECIFICATIONS

PROGRAM ________ PROGRAMMER ________ DATE ________ PAGE ____ OF ____ PAGES

	PAGE NO. (1–2)	LINE NO. (3–5)	FORM TYPE (6)	FILE NAME (7–14)	SEQUENCE 01-99 or AA-ZZ (15–16)	NUMBER 1 or N (17)	O-OPTIONAL (18)	RECORD IDENTIFYING INDICATOR OR ** (19–20)	RECORD IDENTIFICATION CODES 1, 2, 3 (21–41)	STACKER SELECT (42)	DATA FORMAT P-B-L-R (43)	FIELD LOCATION FROM (44–47)	FIELD LOCATION TO (48–51)	DECIMAL POSITIONS (52)	FIELD NAME (53–58)	CONTROL LEVEL L1-L9 (59–60)	MATCHING FIELDS OR CHAINING FIELDS (61–62)	FIELD RECORD RELATION (63–64)	FIELD INDICATORS PLUS (65–66)	MINUS (67–68)	ZERO OR BLANK (69–70)	NOT USED (71–74)	PROGRAM IDENTIFICATION (75–80)
7.		01	I	INPTMSG	AA			01															
8.		02	I									19	32		STANAM								
9.		03	I	STATE	BB			02															
10.		04	I									1	14		STNAME								
11.		05	I									15	22	0	STAPOP								
12.		06	I									23	47		CAPITL								

Figure 3—33. ACT1 Input Format Specifications Form



SPERRY✦UNIVAC

RPG II
CALCULATION SPECIFICATIONS

PROGRAM ________ PROGRAMMER ________ DATE ________ PAGE ____ OF ____ PAGES

	PAGE NO. (1–2)	LINE NO. (3–5)	FORM TYPE (6)	CONTROL LEVEL L0-L9/LR/SR (7–8)	INDICATORS (9–11)	AND (12–14)	AND (15–17)	FACTOR 1 (18–27)	OPERATION (28–32)	FACTOR 2 (33–42)	RESULT FIELD NAME (43–48)	FIELD LENGTH (49–51)	DECIMAL POSITIONS (52)	H = HALF ADJUST (53)	RESULTING INDICATORS HIGH (54–55)	LOW (56–57)	EQUAL (58–59)	COMMENTS (60–74)	PROGRAM IDENTIFICATION (75–80)
13.		01	C		01			STANAM	CHAIN	STATE					20				

Figure 3—34. ACT1 Calculation Specifications Form

Explanation of Figure 3-32:

Line Explanation

1 In column 74, the letter A indicates that this RPG II program is an action program. In columns 75-80, ACT1 is the name of the action program.

2 The primary input file is one fixed-length record - the IMS 90 IMA.

3 The PIB is specified as an update file.

4 The OMA is specified as an output file.

5 The logical file (STATE) is specified as an indexed chained disk file containing fixed-length records (47 bytes) processed randomly by alphanumeric keys 14 bytes long beginning in position 1 of the record.

6 The CDA is specified as an output file and its contents, capital city name (CAPITL), are passed to the successor program, ACT2.

Explanation of Figure 3-33:

Line Explanation

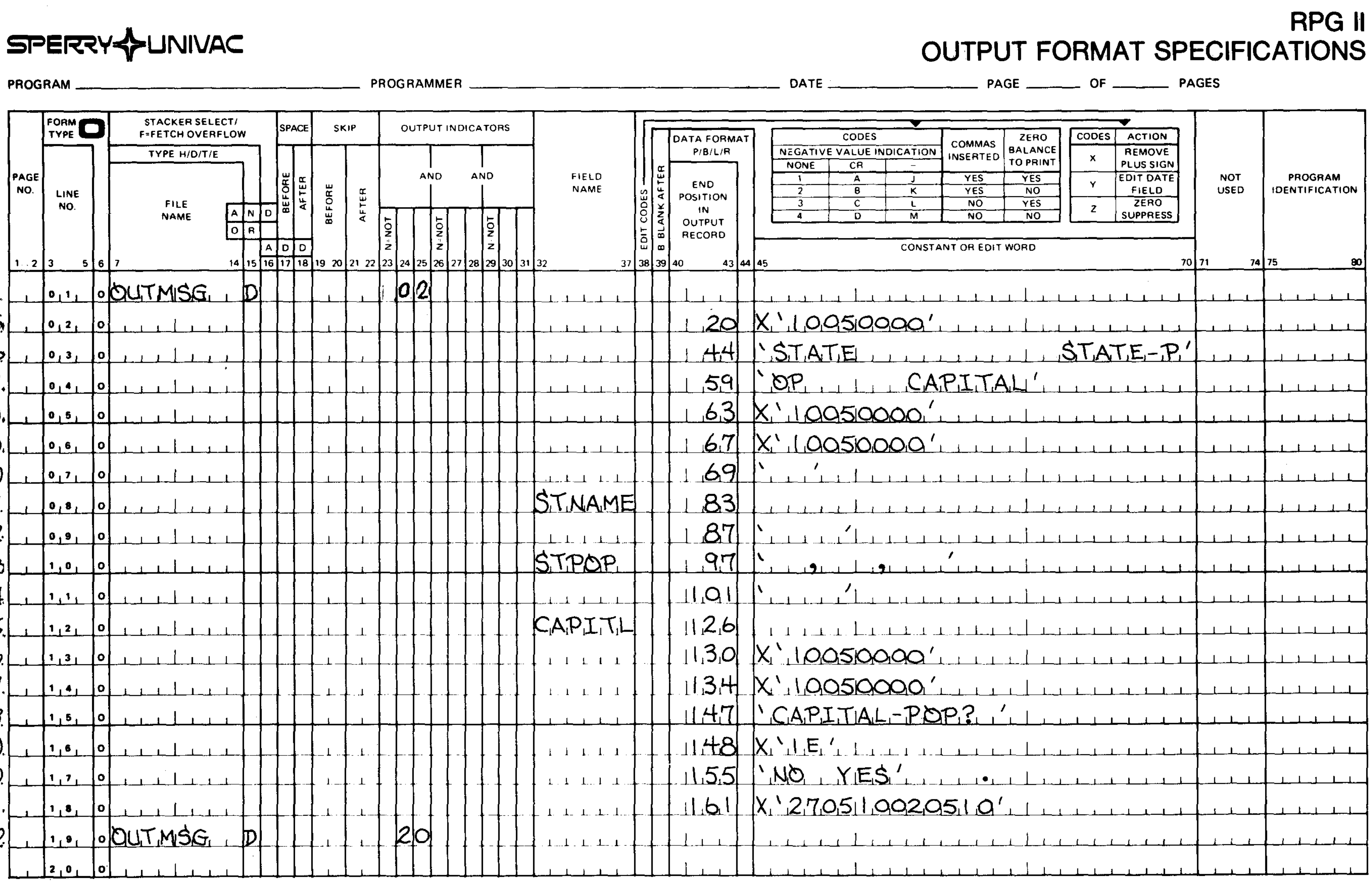
7, 8 The IMA contains the state name in positions 19 through 32. The header information in positions 1 through 16 of the record is not referenced nor is the transaction code in position 17; however, space must be allotted for them in the IMA. Because header information and transaction code fields are not referenced, they are not specified on the input format specifications form. This avoids the occurrence of a flag for an unreferenced field. The transaction code (S) is specified in the TRANSACT section of the IMS 90 configurator and is used at the UNISCOPE 100 to load the action program for use in the transaction.

9-12 The records of the STATE disk file contain the state name (1-14), state population (15-22), and state capital (23-47).

Explanation of Figure 3-33:

Line Explanation

13 The state name placed in the IMA is used as the key for the CHAIN operation to access the STATE disk file. Indicator 20 is set on if the state name is not found in the STATE file.



SPERRY✦UNIVAC

RPG II
OUTPUT FORMAT SPECIFICATIONS

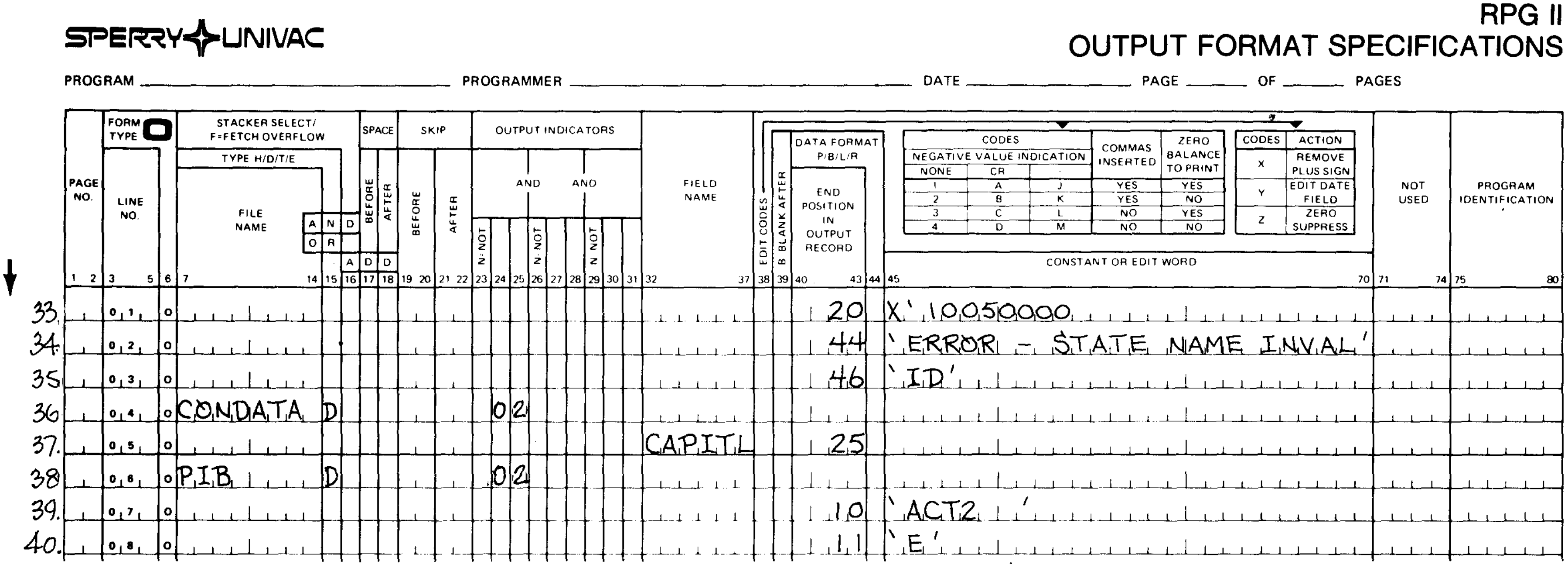
PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE ____ OF ____ PAGES

	PAGE NO. (1–2)	LINE NO. (3–5)	FORM TYPE O (6)	FILE NAME (7–14)	TYPE H/D/T/E (15)	SPACE BEFORE/AFTER (17–18)	SKIP BEFORE/AFTER (19–22)	OUTPUT INDICATORS (23–31)	FIELD NAME (32–37)	EDIT CODES (38)	B BLANK AFTER (39)	END POSITION IN OUTPUT RECORD (40–43)	DATA FORMAT P/B/L/R (44)	CONSTANT OR EDIT WORD (45–70)	NOT USED (71–74)	PROGRAM IDENTIFICATION (75–80)
14.		01	O	OUTMSG	D			02								
15		02	O									20		X'10050000'		
16		03	O									44		'STATE STATE-P'		
17.		04	O									59		'OP CAPITAL'		
18.		05	O									63		X'10050000'		
19.		06	O									67		X'10050000'		
20		07	O									69		' '		
21.		08	O						STNAME			83				
22		09	O									87		' '		
23		10	O						STPOP			97		' , , '		
24		11	O									101		' '		
25.		12	O						CAPITL			126				
26.		13	O									130		X'10050000'		
27.		14	O									134		X'10050000'		
28.		15	O									147		'CAPITAL-POP? '		
29.		16	O									148		X'1E'		
30.		17	O									155		'NO YES'		
31.		18	O									161		X'270510020510'		
32		19	O	OUTMSG	D			20								
		20	O													

CODES	NEGATIVE VALUE INDICATION NONE	CR	–	COMMAS INSERTED	ZERO BALANCE TO PRINT
	1	A	J	YES	YES
	2	B	K	YES	NO
	3	C	L	NO	YES
	4	D	M	NO	NO

CODES	ACTION
X	REMOVE PLUS SIGN
Y	EDIT DATE FIELD
Z	ZERO SUPPRESS

Figure 3—35. ACT1 Output Format Specifications Form (Part 1 of 2)



SPERRY UNIVAC

RPG II
OUTPUT FORMAT SPECIFICATIONS

PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE ____ OF ____ PAGES

CODES NEGATIVE VALUE INDICATION NONE	CR	-	COMMAS INSERTED	ZERO BALANCE TO PRINT
1	A	J	YES	YES
2	B	K	YES	NO
3	C	L	NO	YES
4	D	M	NO	NO

CODES	ACTION
X	REMOVE PLUS SIGN
Y	EDIT DATE FIELD
Z	ZERO SUPPRESS

	PAGE NO. 1-2	LINE NO. 3-5	FORM TYPE 6	FILE NAME 7-14	TYPE H/D/T/E 15	OUTPUT INDICATORS 23-31	FIELD NAME 32-37	EDIT CODES 38	B BLANK AFTER 39	END POSITION IN OUTPUT RECORD 40-43	DATA FORMAT P/B/L/R 44	CONSTANT OR EDIT WORD 45-70	NOT USED 71-74	PROGRAM IDENTIFICATION 75-80
33.		01	O							20		X'10050000		
34.		02	O							44		'ERROR - STATE NAME INVAL'		
35		03	O							46		'ID'		
36		04	O	CONDATA	D	02								
37.		05	O				CAPITL			25				
38		06	O	PIB	D	02								
39.		07	O							10		'ACT2 '		
40.		08	O							11		'E'		

Figure 3—35. ACT1 Output Format Specifications Form (Part 2 of 2)

Explanation of Figure 3-35:

Line Explanation

14–31 If the state name is found, it is displayed, as well as the state population and capital. Lines 14–25 clear the UNISCOPE 100 screen, set up the header titles, and provide for the data requested. After line positioning with DICE output function codes (lines 26 and 27), lines 28–31 set up the output message, CAPITAL-POP? ΔNO YES, and set the tab control characters after the message. In addition, the program requests the terminal operator to indicate whether the capital city population is to be displayed. If capital city population is requested, a preformatted answer (YES) is transmitted. This appears in the IMA of the successor program (Figure 3–36, line 7 of ACT2) and is compared with the literal 'YES' on the calculation specifications form of ACT2. The hexadecimal values shown in this example are ICAM DICE functions or communications control characters. (See the fundamentals of ICAM user guide, UP-8194, current version.)

32–35 If the state name is not found, an error message is displayed. These lines set up the DICE control characters and the error message.

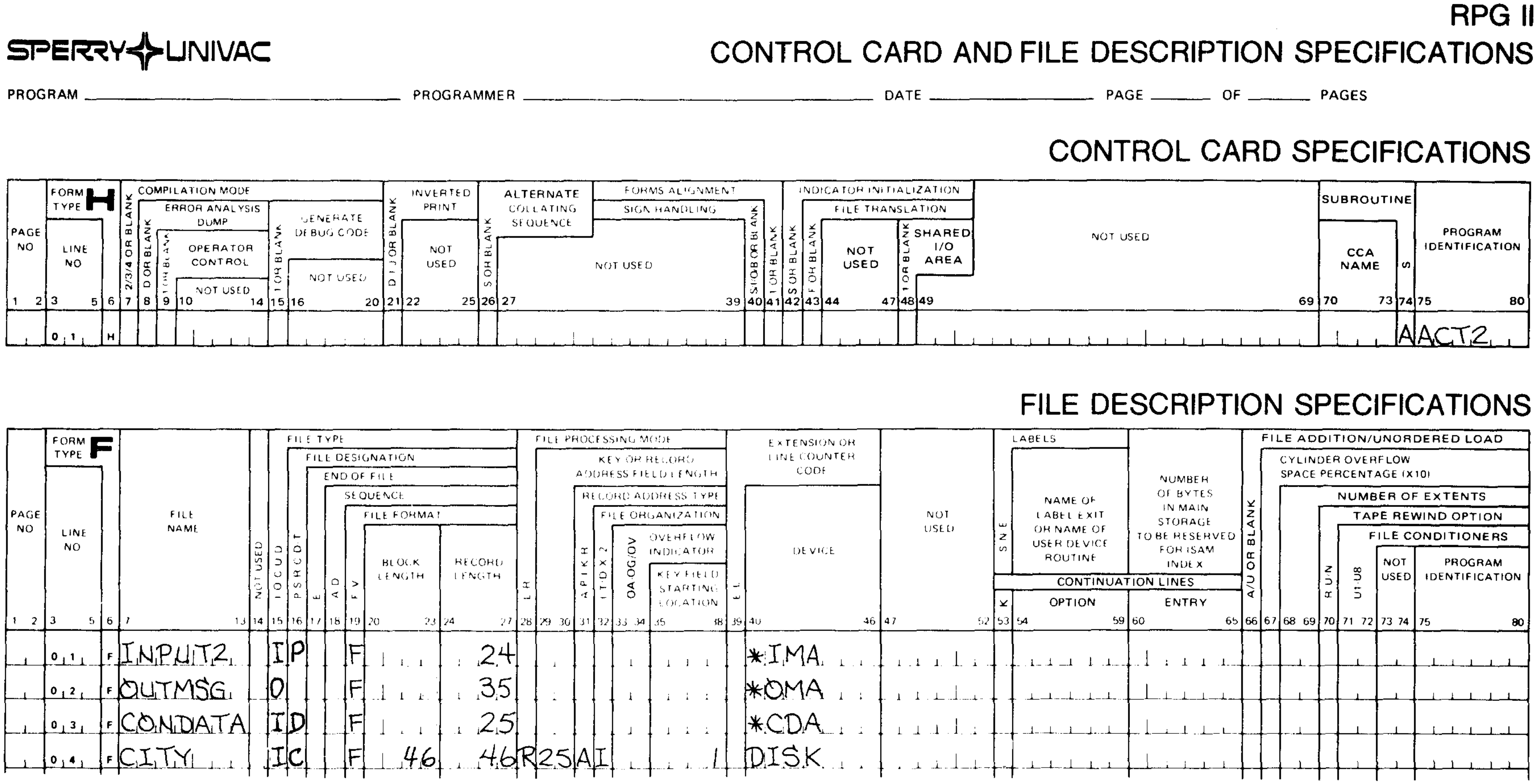
36–37 If the state name is found, the capital city name is passed to the successor program (ACT2) in the CDA where it is read and used to find capital city population from the CITY file.

38–40 If the state name is found, the successor name (ACT2) is moved to the PIB and an E is moved into the PIB to indicate external succession.

3.16.2. ACT2 Discussion

ACT2 examines the response made by the UNISCOPE 100 terminal operator. If the operator requests that the state capital population be displayed, a CHAIN operation is executed on the CITY file (Figure 3–38, line 14). The capital city name contained in the CDA is used as the key (Figure 3–37, line 11). When the capital city name is matched in the CITY file, the capital city population is displayed on the UNISCOPE 100 (Figure 3–39, line 20). If the terminal operator does not request the capital population, the output message length is set to zero (Figure 3–39, line 16). In either case, termination is normal.

Figures 3–36 through 3–39 illustrate the specifications forms required to code ACT2, the second action program, which accesses the CITY file for capital population. A line-by-line description is provided.



SPERRY UNIVAC

RPG II
CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE ____ OF ____ PAGES

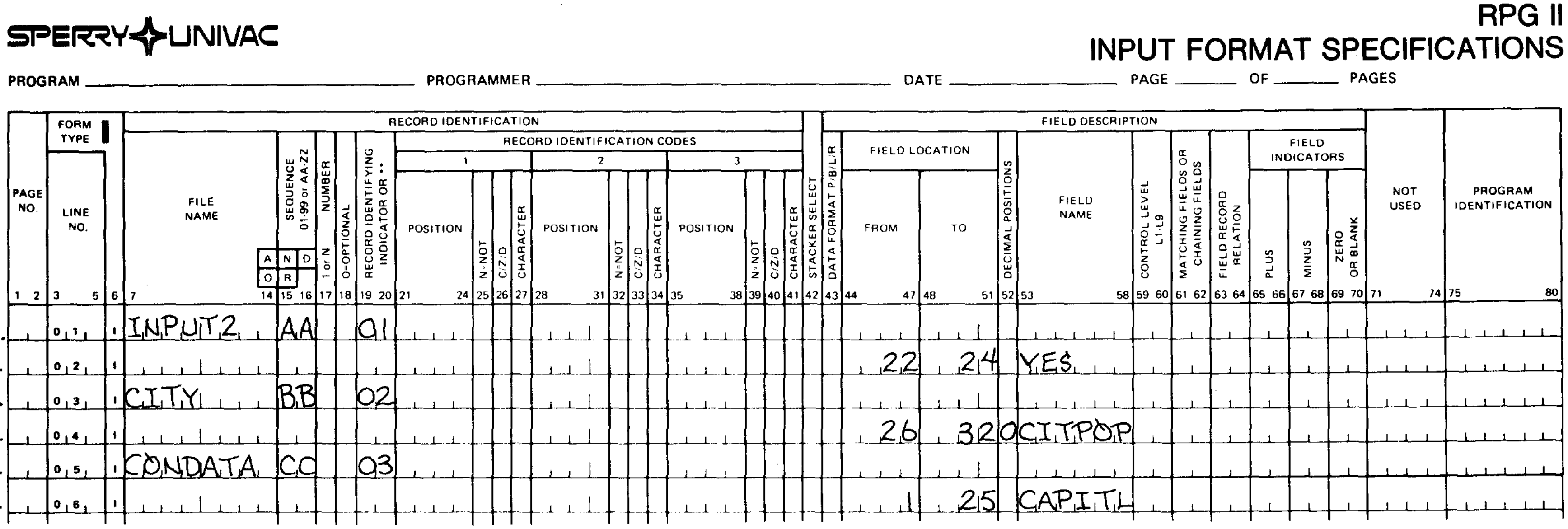
CONTROL CARD SPECIFICATIONS

	PAGE NO (1–2)	LINE NO (3–5)	FORM TYPE H (6)	COMPILATION MODE (7) / ERROR ANALYSIS DUMP (8) / OPERATOR CONTROL (9) / NOT USED (10–14)	GENERATE DEBUG CODE (15) / NOT USED (16–20)	INVERTED PRINT (21) / NOT USED (22–25)	ALTERNATE COLLATING SEQUENCE (26) / FORMS ALIGNMENT / SIGN HANDLING / NOT USED (27–39)	INDICATOR INITIALIZATION / FILE TRANSLATION (40–43) / NOT USED (44–47)	SHARED I/O AREA (48)	NOT USED (49–69)	SUBROUTINE CCA NAME (70–73)	S (74)	PROGRAM IDENTIFICATION (75–80)
1.		01	H									A	ACT2

FILE DESCRIPTION SPECIFICATIONS

	PAGE NO (1–2)	LINE NO (3–5)	FORM TYPE F (6)	FILE NAME (7–13)	NOT USED (14)	FILE TYPE I O U C D (15)	FILE DESIGNATION P S R C D T (16)	END OF FILE E (17)	SEQUENCE A D (18)	FILE FORMAT F V (19)	BLOCK LENGTH (20–23)	RECORD LENGTH (24–27)	FILE PROCESSING MODE L R (28)	KEY OR RECORD ADDRESS FIELD LENGTH (29–30)	RECORD ADDRESS TYPE A P I K (31)	FILE ORGANIZATION I T D X 2 (32)	OVERFLOW INDICATOR OA OG OV (33–34)	KEY FIELD STARTING LOCATION (35–38)	EXTENSION OR LINE COUNTER CODE E L (39)	DEVICE (40–46)	NOT USED (47–52)	LABELS S N E / K (53)	NAME OF LABEL EXIT OR NAME OF USER DEVICE ROUTINE / CONTINUATION LINES OPTION (54–59)	NUMBER OF BYTES IN MAIN STORAGE TO BE RESERVED FOR ISAM INDEX / ENTRY (60–65)	A/U OR BLANK (66)	FILE ADDITION/UNORDERED LOAD (67)	CYLINDER OVERFLOW SPACE PERCENTAGE (X10) (68–69)	NUMBER OF EXTENTS / TAPE REWIND OPTION R U N (70)	FILE CONDITIONERS U1-U8 (71–72)	NOT USED (73–74)	PROGRAM IDENTIFICATION (75–80)
2.		01	F	INPUT2		I	P			F		24								*IMA											
3.		02	F	OUTMSG		O				F		35								*OMA											
4.		03	F	CONDATA		I	D			F		25								*CDA											
5.		04	F	CITY		I	C			F	46	46	R	25	A	I		1		DISK											

Figure 3—36. ACT2 Control Card and File Description Specifications Forms



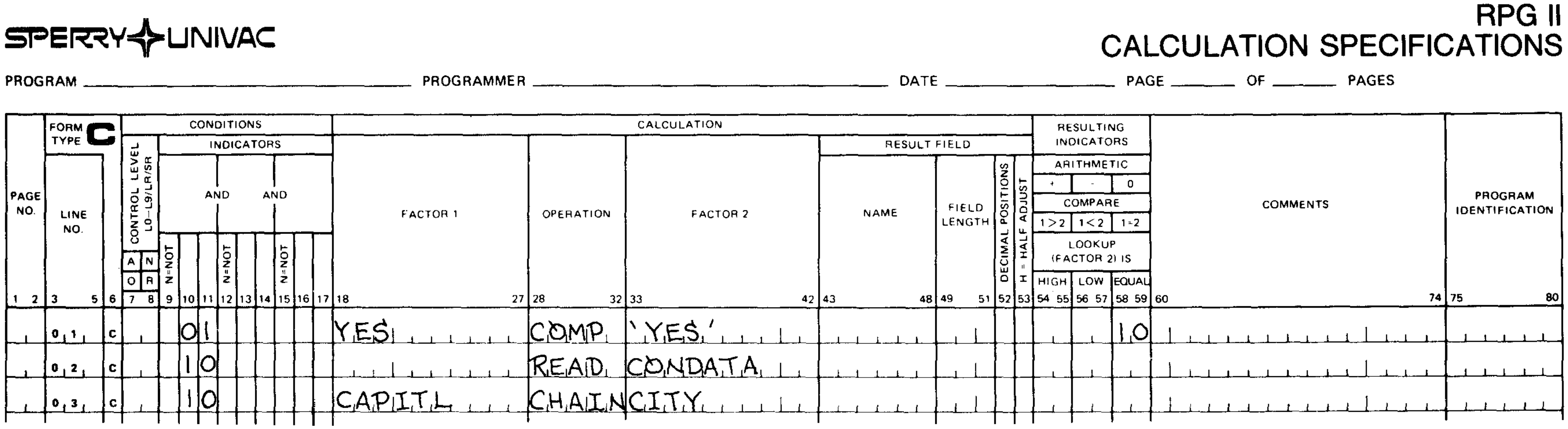
SPERRY UNIVAC

RPG II
INPUT FORMAT SPECIFICATIONS

PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE ____ OF ____ PAGES

	PAGE NO. (1-2)	LINE NO. (3-5)	FORM TYPE I (6)	FILE NAME (7-14)	SEQUENCE 01-99 or AA-ZZ (15-16)	NUMBER 1 or N (17)	O=OPTIONAL (18)	RECORD IDENTIFYING INDICATOR OR ** (19-20)	1 POSITION (21-24)	1 N=NOT (25)	1 C/Z/D (26)	1 CHARACTER (27)	2 POSITION (28-31)	2 N=NOT (32)	2 C/Z/D (33)	2 CHARACTER (34)	3 POSITION (35-38)	3 N=NOT (39)	3 C/Z/D (40)	3 CHARACTER (41)	STACKER SELECT (42)	DATA FORMAT P/B/L/R (43)	FIELD LOCATION FROM (44-47)	FIELD LOCATION TO (48-51)	DECIMAL POSITIONS (52)	FIELD NAME (53-58)	CONTROL LEVEL L1-L9 (59-60)	MATCHING FIELDS OR CHAINING FIELDS (61-62)	FIELD RECORD RELATION (63-64)	FIELD INDICATORS PLUS (65-66)	FIELD INDICATORS MINUS (67-68)	FIELD INDICATORS ZERO OR BLANK (69-70)	NOT USED (71-74)	PROGRAM IDENTIFICATION (75-80)
6.		01	I	INPUT2	AA			01																										
7.		02	I																				22	24		YES								
8.		03	I	CITY	BB			02																										
9.		04	I																				26	32	0	CITPOP								
10.		05	I	CONDATA	CC			03																										
11.		06	I																				1	25		CAPITL								

Figure 3—37. ACT2 Input Format Specifications Form



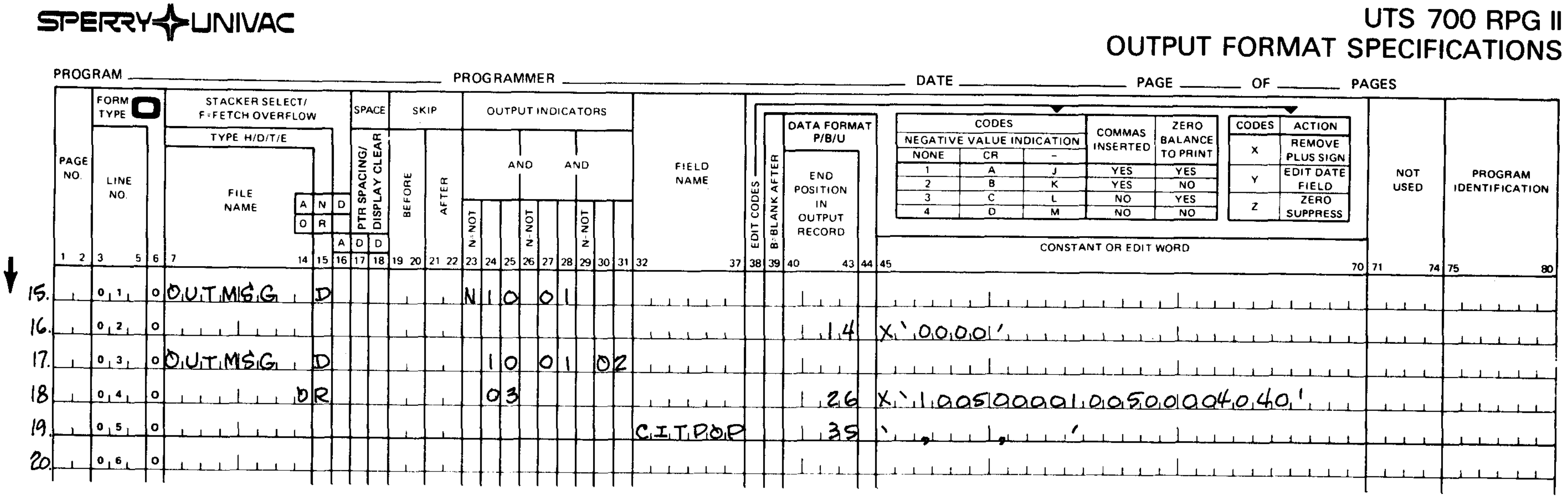
SPERRY UNIVAC

RPG II
CALCULATION SPECIFICATIONS

PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE ____ OF ____ PAGES

	PAGE NO. (1-2)	LINE NO. (3-5)	FORM TYPE C (6)	CONTROL LEVEL L0-L9/LR/SR (7-8)	INDICATORS N=NOT (9)	INDICATORS (10-11)	AND N=NOT (12)	AND (13-14)	AND N=NOT (15)	AND (16-17)	FACTOR 1 (18-27)	OPERATION (28-32)	FACTOR 2 (33-42)	RESULT FIELD NAME (43-48)	FIELD LENGTH (49-51)	DECIMAL POSITIONS (52)	H = HALF ADJUST (53)	RESULTING INDICATORS + / COMPARE 1>2 / LOOKUP HIGH (54-55)	- / 1<2 / LOW (56-57)	0 / 1=2 / EQUAL (58-59)	COMMENTS (60-74)	PROGRAM IDENTIFICATION (75-80)
12.		01	C			01					YES	COMP	'YES'							10		
13.		02	C			10						READ	CONDATA									
14.		03	C			10					CAPITL	CHAIN	CITY									

Figure 3—38. ACT2 Calculation Specifications Form



SPERRY UNIVAC

UTS 700 RPG II
OUTPUT FORMAT SPECIFICATIONS

PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE ______ OF ______ PAGES

CODES NEGATIVE VALUE INDICATION NONE	CR	–	COMMAS INSERTED	ZERO BALANCE TO PRINT
1	A	J	YES	YES
2	B	K	YES	NO
3	C	L	NO	YES
4	D	M	NO	NO

CODES	ACTION
X	REMOVE PLUS SIGN
Y	EDIT DATE FIELD
Z	ZERO SUPPRESS

	PAGE NO. 1-2	LINE NO. 3-5	FORM TYPE O 6	FILE NAME 7-14	TYPE H/D/T/E 15	STACKER SELECT/ F=FETCH OVERFLOW 16	SPACE 17	18	SKIP BEFORE 19-20	SKIP AFTER 21-22	OUTPUT INDICATORS 23-25	AND 26-28	AND 29-31	FIELD NAME 32-37	EDIT CODES 38	B-BLANK AFTER 39	END POSITION IN OUTPUT RECORD 40-43	DATA FORMAT P/B/U 44	CONSTANT OR EDIT WORD 45-70	NOT USED 71-74	PROGRAM IDENTIFICATION 75-80
15.		01	O	OUTMSG	D						N10	01									
16.		02	O														14		X'0000'		
17.		03	O	OUTMSG	D						10	01	02								
18		04	O	OR							03						26		X'1005000010050000040 40'		
19.		05	O											CITPOP			35		' , , '		
20.		06	O																		

Figure 3—39. ACT2 Output Format Specifications Form

Explanation of Figure 3-36:

<u>Line</u> <u>Explanation</u>

1 ACT2 is the name of the action program (columns 75-80). The letter A in column 74 indicates that this is an action program.

2 The primary input file is one record – the IMS 90 IMA.

3 The OMA is specified as an output file.

4 The CDA is specified as a demand input file.

5 The logical file (CITY) is specified as an ISAM chained disk file containing fixed-length records (46 bytes) processed randomly by alphanumeric keys 25 bytes long beginning in position 1 of the record. Labels are standard.

Explanation of Figure 3-37:

<u>Line</u> <u>Explanation</u>

6, 7 Only one field of the IMA is used. The header information (1-16) is not used.

8, 9 The records of the CITY file contain the capital population in positions 26-32.

10, 11 The CDA contains, in positions 1-25, the state capital which was passed to ACT2 by ACT1.

Explanation of Figure 3-38:

<u>Line</u> <u>Explanation</u>

12 The terminal operator reply is checked to see if it was YES.

13 If the reply is YES, the CDA containing the capital city name is read.

14 The capital population is then retrieved from the CITY file.

Explanation of Figure 3-39:

Line	Explanation
15, 16	If the reply is not YES, the output message length is set to zero.
17, 20	If the reply is YES, the capital population is displayed. Line 19 sets up the DICE control characters.

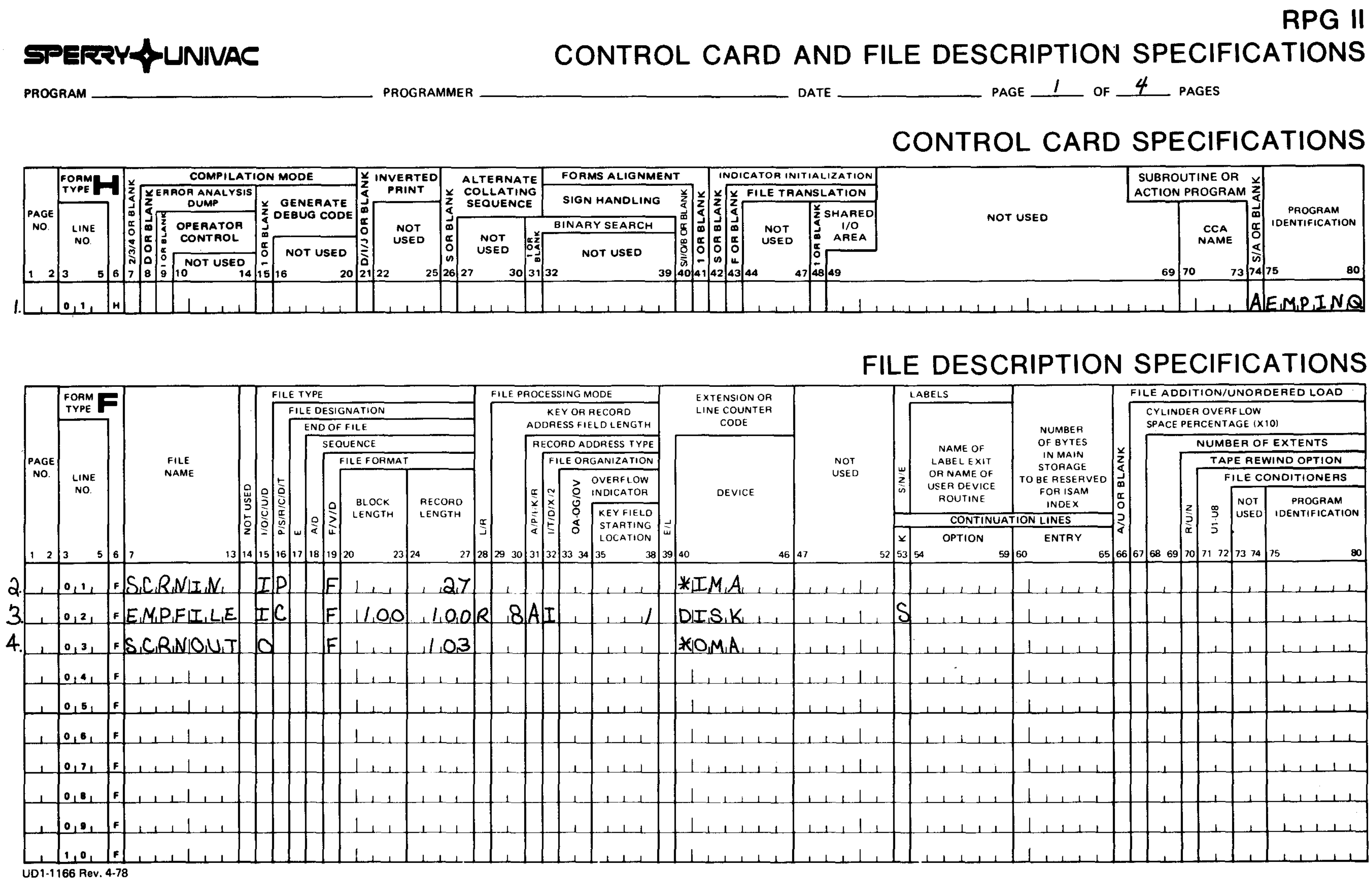
ACT1 and ACT2 illustrate passing capital city name in the STATE logical file from one action program to another as a key to retrieve capital city population requested from the CITY logical file.

3.16.3. Screen Formatting Example

The following program, EMPINQ, is an action program that processes screen formatted messages. When the terminal operator keys in the IMS 90 transaction code followed by an 8-digit employee number, the program retrieves the employee's record from a disk file and displays the name and address on a predefined screen format. When an employee number is not found in the file, another screen format displays an error message. Figures 3-39A through 3-39D show the specification forms required to code action program EMPINQ.

Here is an explanation of Figures 3-39A through 39D:

Line	Explanation
1	Columns 74-80 indicate that this program is an action program named EMPINQ.
2	Line 2 defines the IMS 90 IMA file named SCRNIN. SCRNIN is the input primary file and contains fixed-length records. The record length of 27 bytes consists of 16 bytes for the IMA header, 2 bytes for the transaction code, 1 blank byte, and 8 bytes for the employee number.
3	Line 3 defines the user disk file, EMPFILE, which contains employee records as an input chained file (IC In columns 15 and 16), meaning that records are processed randomly from an indexed sequential file (I in column 32). EMPFILE records are fixed in length at 100 bytes and blocked 100 bytes. Column 30 indicates an 8-byte key field (employee number) is used to access this file randomly (column 28), using a record address (A in column 31). The 8-byte key starts in byte 1 of the record (column 38). The file is stored on a disk device (column 40-46) and has standard labels (S in column 53).
4	Line 4 defined the IMS 90 OMA file name SCRNOUT as an output file (O in column 15) with fixed-length records (F in column 19). The record length of 103 bytes consists of 16 bytes for the OMA header, 8 bytes for the employee number, and the remaining 79 bytes for the name and address of the employee.



SPERRY✦UNIVAC

RPG II

CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

PROGRAM ________ PROGRAMMER ________ DATE ________ PAGE 1 OF 4 PAGES

CONTROL CARD SPECIFICATIONS

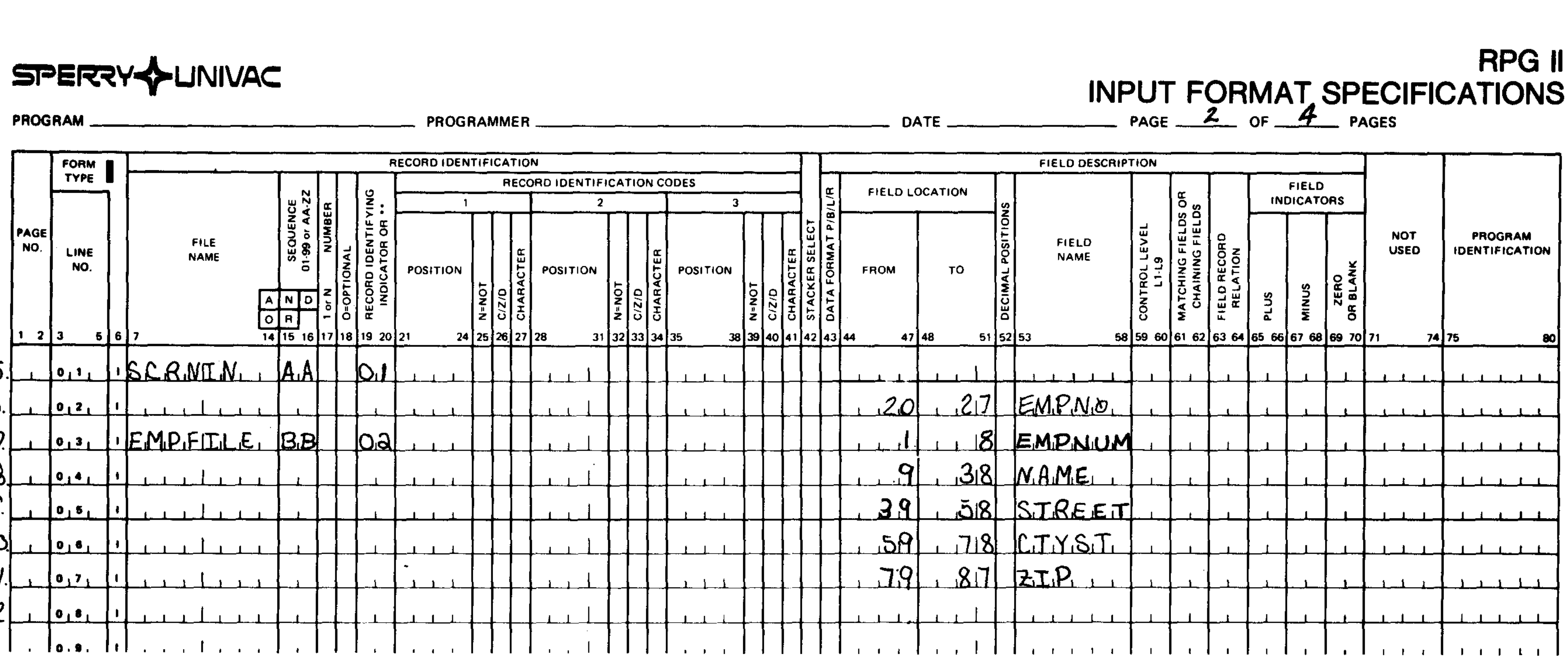
	Page No. 1-2	Form Type H: Line No. 3-5	6	2/3/4 or blank 7	Compilation Mode: D or blank 8	Error Analysis Dump: 1 or blank 9	Operator Control: Not used 10-14	1 or blank 15	Generate Debug Code: Not used 16-20	D/I/J or blank 21	Inverted Print: Not used 22-25	S or blank 26	Alternate Collating Sequence: Not used 27-30	1 or blank 31	Forms Alignment / Sign Handling / Binary Search: Not used 32-39	S/I/O/B or blank 40	1 or blank 41	Indicator Initialization: S or blank 42	File Translation: F or blank 43	Not used 44-47	1 or blank 48	Shared I/O Area / Not used 49-69	Subroutine or Action Program: CCA Name 70-73	S/A or blank 74	Program Identification 75-80
1.		01	H																					A	EMPINQ

FILE DESCRIPTION SPECIFICATIONS

	Page No. 1-2	Form Type F: Line No. 3-5	6	File Name 7-13	Not used 14	File Type: I/O/C/U/D 15	File Designation: P/S/R/C/D/T 16	End of File: E 17	Sequence: A/D 18	File Format: F/V/D 19	Block Length 20-23	Record Length 24-27	File Processing Mode: L/R 28	Key or Record Address Field Length 29-30	Record Address Type: A/P/I/K/R 31	File Organization: I/T/D/X/2 32	Overflow Indicator: OA-OG/OV 33-34	Key Field Starting Location 35-38	E/L 39	Extension or Line Counter Code: Device 40-46	Not used 47-52	Labels: S/N/E K 53	Name of Label Exit or Name of User Device Routine Continuation Lines: Option 54-59	Number of Bytes in Main Storage to be Reserved for ISAM Index Entry 60-65	A/U or blank 66	File Addition/Unordered Load 67	Cylinder Overflow Space Percentage (X10) 68-69	Number of Extents / Tape Rewind Option: R/U/N 70	File Conditioners: U1-U8 71-72	Not used 73-74	Program Identification 75-80
2.		01	F	SCRNIN		I	P			F		27								*IMA											
3.		02	F	EMPFILE		I	C			F	100	100	R	8	A	I		1		DISK		S									
4.		03	F	SCRNOUT		O				F		103								*OMA											
		04	F																												
		05	F																												
		06	F																												
		07	F																												
		08	F																												
		09	F																												
		10	F																												

UD1-1166 Rev. 4-78

Figure 3—39A. EMPINQ, Control Card and File Description Specifications Form



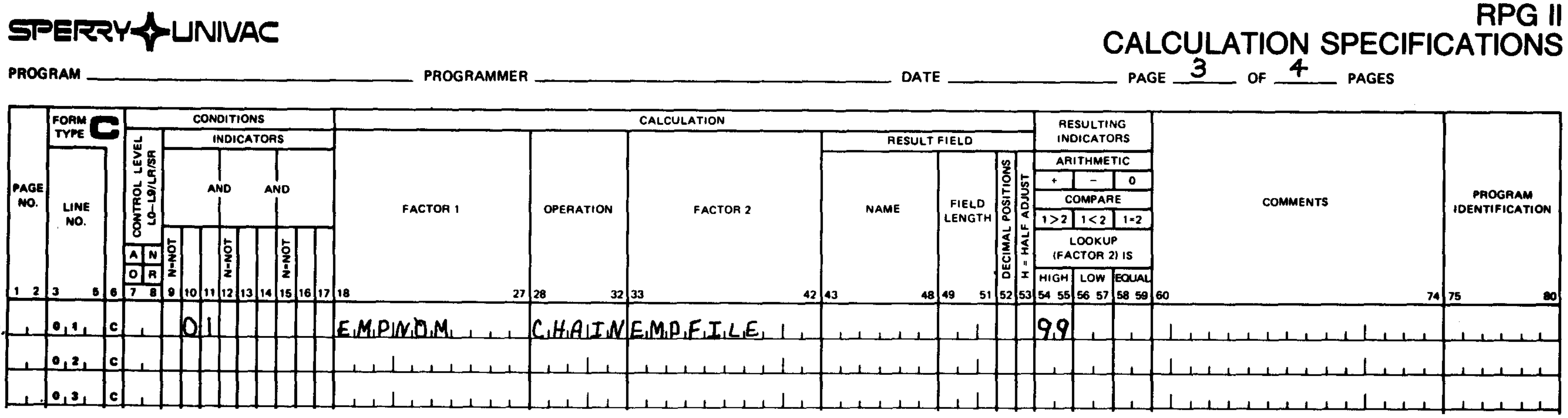
SPERRY UNIVAC

RPG II
INPUT FORMAT SPECIFICATIONS

PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE 2 OF 4 PAGES

	PAGE NO. (1-2)	LINE NO. (3-5)	FORM TYPE I (6)	FILE NAME (7-14)	SEQUENCE 01-99 or AA-ZZ (15-16)	NUMBER 1 or N (17)	O=OPTIONAL (18)	RECORD IDENTIFYING INDICATOR OR ** (19-20)	RECORD IDENTIFICATION CODES 1 (21-27)	RECORD IDENTIFICATION CODES 2 (28-34)	RECORD IDENTIFICATION CODES 3 (35-41)	STACKER SELECT (42)	DATA FORMAT P/B/L/R (43)	FIELD LOCATION FROM (44-47)	FIELD LOCATION TO (48-51)	DECIMAL POSITIONS (52)	FIELD NAME (53-58)	CONTROL LEVEL L1-L9 (59-60)	MATCHING FIELDS OR CHAINING FIELDS (61-62)	FIELD RECORD RELATION (63-64)	FIELD INDICATORS PLUS (65-66)	MINUS (67-68)	ZERO OR BLANK (69-70)	NOT USED (71-74)	PROGRAM IDENTIFICATION (75-80)
5.		01	I	SCRNIN	AA			01																	
6.		02	I											20	27		EMPNO								
7.		03	I	EMPFILE	BB			02						1	8		EMPNUM								
8.		04	I											9	38		NAME								
9.		05	I											39	58		STREET								
10.		06	I											59	78		CTYST								
11.		07	I											79	87		ZIP								
12		08	I																						
		09	I																						

Figure 3—39B. EMPINQ, Input Format Specifications Form



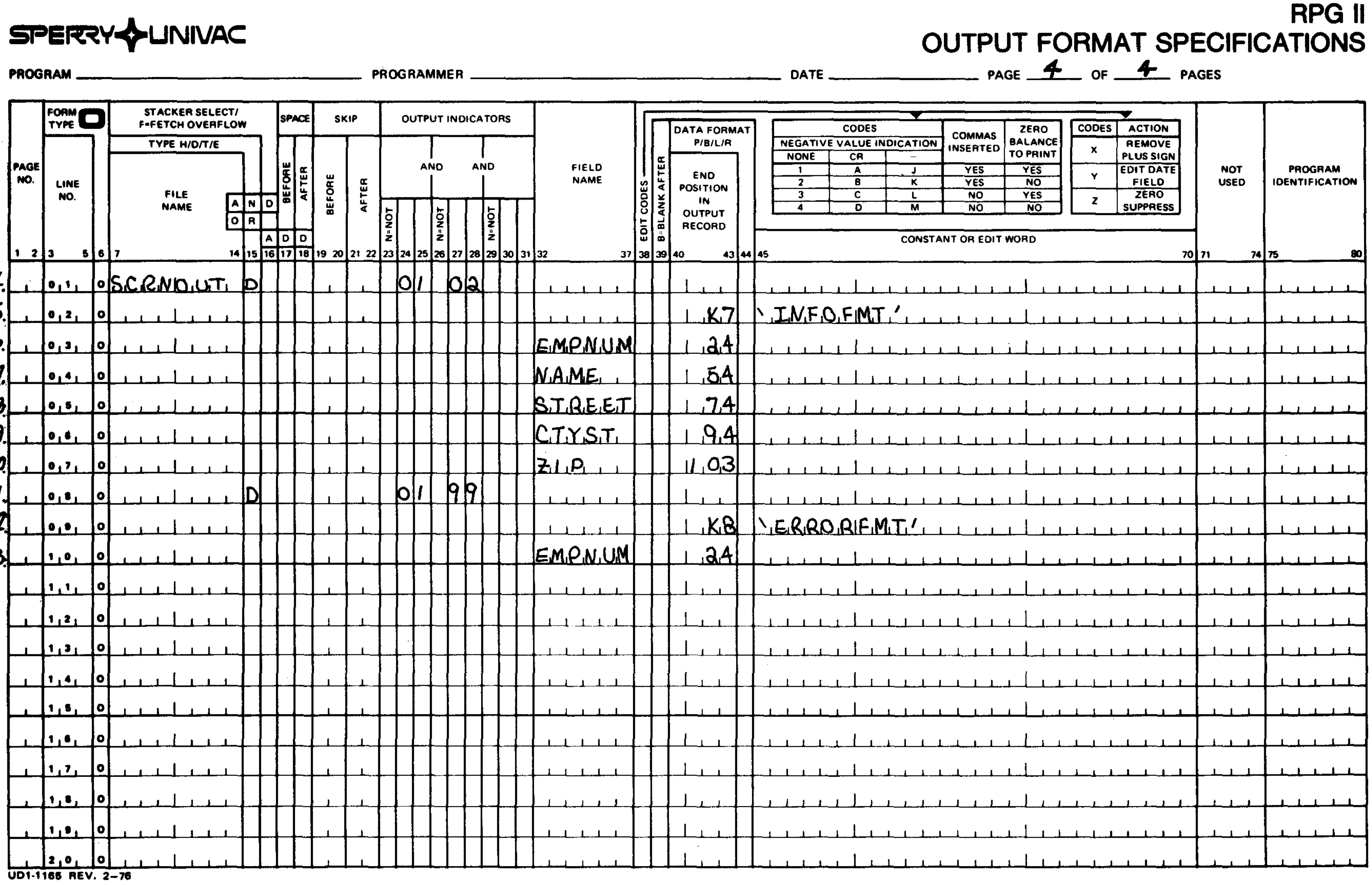
SPERRY UNIVAC

RPG II
CALCULATION SPECIFICATIONS

PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE 3 OF 4 PAGES

	PAGE NO. (1-2)	LINE NO. (3-5)	FORM TYPE C (6)	CONTROL LEVEL L0-L9/LR/SR (7-8)	INDICATORS (9-11)	AND (12-14)	AND (15-17)	FACTOR 1 (18-27)	OPERATION (28-32)	FACTOR 2 (33-42)	RESULT FIELD NAME (43-48)	FIELD LENGTH (49-51)	DECIMAL POSITIONS (52)	H = HALF ADJUST (53)	RESULTING INDICATORS HIGH 1>2 + (54-55)	LOW 1<2 − (56-57)	EQUAL 1=2 0 (58-59)	COMMENTS (60-74)	PROGRAM IDENTIFICATION (75-80)
13.		01	C		01			EMPNOM	CHAIN	EMPFILE					99				
		02	C																
		03	C																

Figure 3—39C. EMPINQ, Calculation Specifications Form



SPERRY UNIVAC

RPG II OUTPUT FORMAT SPECIFICATIONS

PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE 4 OF 4 PAGES

CODES			COMMAS INSERTED	ZERO BALANCE TO PRINT
NEGATIVE VALUE INDICATION				
NONE	CR	–		
1	A	J	YES	YES
2	B	K	YES	NO
3	C	L	NO	YES
4	D	M	NO	NO

CODES	ACTION
X	REMOVE PLUS SIGN
Y	EDIT DATE FIELD
Z	ZERO SUPPRESS

	PAGE NO. 1 2	LINE NO. 3 5	FORM TYPE 6	FILE NAME 7 14	TYPE H/D/T/E 15	STACKER SELECT/ F=FETCH OVERFLOW 16	SPACE BEFORE 17	SPACE AFTER 18	SKIP BEFORE 19 20	SKIP AFTER 21 22	OUTPUT INDICATORS N=NOT 23 24 25	AND N=NOT 26 27 28	AND N=NOT 29 30 31	FIELD NAME 32 37	EDIT CODES 38	B=BLANK AFTER 39	END POSITION IN OUTPUT RECORD 40 43	DATA FORMAT P/B/L/R 44	CONSTANT OR EDIT WORD 45 70	NOT USED 71 74	PROGRAM IDENTIFICATION 75 80
14.		01	O	SCRNOUT	D						01	02									
15.		02	O														K7		'INFOFMT'		
16.		03	O											EMPNUM			24				
17.		04	O											NAME			54				
18.		05	O											STREET			74				
19.		06	O											CTYST			94				
20		07	O											ZIP			103				
21.		08	O		D						01	99									
22		09	O														K8		'ERRORFMT'		
23		10	O											EMPNUM			24				
		11	O																		
		12	O																		
		13	O																		
		14	O																		
		15	O																		
		16	O																		
		17	O																		
		18	O																		
		19	O																		
		20	O																		

UD1-1165 REV. 2-76

Figure 3—39D. EMPINQ, Output Format Specifications Form

Line Explanation

5-6 The IMA file SCRNIN includes an 8-byte employee number entered as an input field from the terminal. Columns 15 and 16 indicate that no retrieval sequence is required for this record type in relation to other record types in the file. There are two record types on input. The first record type is shown by the 01 indicator (columns 19-20). Indicator 01 is set on when the terminal operator enters the employee number.

7-12 These lines define the disk record for the chain file EMPFILE. When the CHAIN operation occurs (line 13) and the employee record is found, the record-identifying indicator 02 (column 19-20) is turned on.

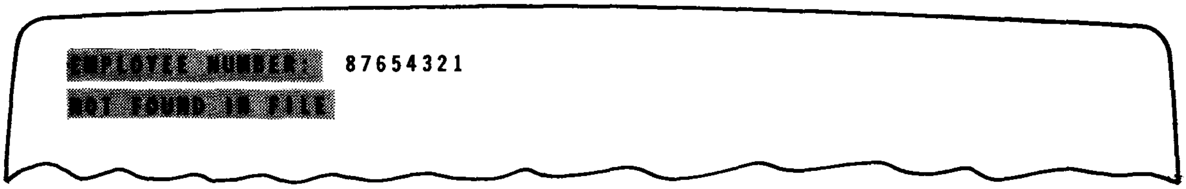
13 The CHAIN operation retrieves the employee record using the employee number as a key. When the record is found in the disk file, indicator 02 is set on. When the record is not found, indicator 99 is set on.

14-20 Line 14 defines the IMS 90 OMA file as a detail type output record (D in column 15). If indicators 01 and 02 are on, IMS 90 sends the screen format INFOFMT and the name, street, city, state, and zip code of the employee (variable fields described on lines 15 through 20) to the terminal. The end position of employee number is 24 because the OMA header uses the first 16 bytes and the employee number is 8 bytes long. The K7 in columns 42 and 43 indicates the number of characters contained in the format name (INFOFMT) that is defined as a constant in columns 46-52.

21-23 Line 21 defines the error screen format as a detail type record. If the employee number is not found in the disk file, RPG II sets indicator 99 on and IMS 90 sends the error screen format ERRORFMT to the terminal. K8 in columns 42 and 43 indicates the number of characters contained in the format name (ERRORFMT) that is defined as a constant in columns 46-53. Line 23 defines the variable data field of the employee number that follows the 16-byte output message header.

Both screen formats INFORFMT and ERRORFMT must be constructed separately at screen format generation time. This process defines the display constants, including the error message NOT FOUND IN FILE, which is itself a display constant. The RPG II action program then must supply the exact variable data needed by the screen formats.

Figure 3-19A shows the results of the complete output screen format (INFOFMT) containing display constants (shaded fields) and the variable data supplied by this action program. Figure 3-39E shows an example of the error screen generated when an employee record cannot be found. Display constants are shaded.



```
EMPLOYEE NUMBER:  87654321
NOT FOUND IN FILE
```

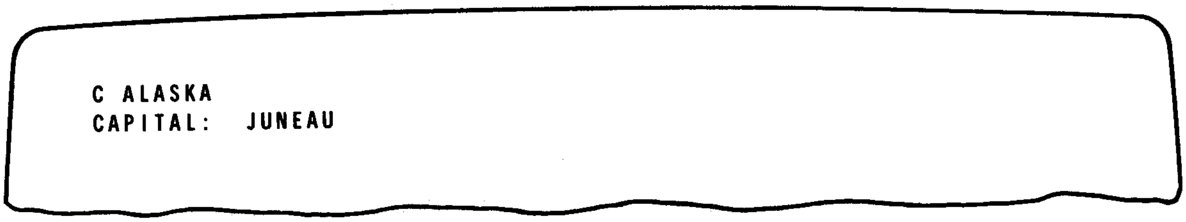
Figure 3—39E. Error Screen Format

3.17. SAMPLE BAL ACTION PROGRAM

The sample user-written action program in Figure 3-40 processes a simple inquiry transaction. A sample input message and the corresponding output message are shown in Figure 3-40.

The file referenced in this transaction is the STATE file of the examples in 3.14. The transaction retrieves and displays the name of the capital city when the state name is given as input.

IMS 90 associates the transaction code C with the action program ACT3 in Figure 3-41. ACT3 uses the name of the state given in the input message as a key to obtain a record from the STATE file. If the record does not exist, an error message is displayed.



```
C ALASKA
CAPITAL:  JUNEAU
```

Figure 3—40. Example of Simple Inquiry Transaction

```
1        10    16
         TITLE 'IMS STATE CAPITAL ACTION PROGRAM'
         PRINT NOGEN
ACT3     START Ø                   ACTION PROGRAM ENTRY POINT
* ALLOCATE REGISTERS TO COVER:
         USING *,R2                THIS ACTION PROGRAM
         USING ZA#DPIB,R3          THE PROGRAM INFORMATION BLOCK
         USING ZA#IMH,R4           THE INPUT MESSAGE AREA
         USING WORK,R5             THE WORK AREA
         USING ZA#OMH,R6           THE OUTPUT MESSAGE AREA
* INITIALIZE
         STM   14,12,12(13)        SAVE ACTION SCHEDULING REGISTERS
*                                  RD$ CONTAINS SAVE AREA ADDRESS WHEN
*                                  CONTROL IS RECEIVED BY
*                                  ACTION PROGRAM.
         LR    R2,R15              ESTABLISH ADDRESSABILITY FOR ACTION
*                                  PROGRAM. RF$ CONTAINS ENTRY POINT
*                                  ADDRESS WHEN CONTROL IS RECEIVED
*                                  BY ACTION PROGRAM.
         LM    R3,R6,Ø(R1)         ESTABLISH ADDRESSABILITY FOR
*                                  ACTIVATION RECORD. R1$ CONTAINS
*                                  PARAMETER LIST ADDRESS WHEN CONTROL
*                                  IS RECEIVED BY
*                                  ACTION PROGRAM.
         LA    R1Ø,SAVEAREA            GET ADDRESS OF ACT3 SAVE AREA
         ST    R1Ø,8(,R13)             SET FORWARD POINTER FROM ACTION
*                                      SCHEDULING SAVE AREA
         ST    R13,4(,R1Ø)             SET BACKWARD POINTER TO ACTION
*                                      SCHEDULING SAVE AREA
         LR    R13,R1Ø                   MAKE ACT3 SAVE AREA THE CURRENT
*                                        SAVE AREA
* GET STATE RECORD FROM FILE USING STATE NAME KEY IN INPUT MESSAGE
         ZG#CALL GET,(STATE,RECORD,SNKEY)    ISSUE CALL TO IMS 9Ø
         CLI   ZA#PSC+1,Ø          TEST STATUS CODE RETURNED IN PROGRAM
*                                  INFORMATION BLOCK BY IMS 9Ø
         BNE   ERROR               NON-ZERO MEANS ERROR
* 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
         B     TERM
* PROCESS ERROR
```

Figure 3—41. Sample BAL Action Program (Part 1 of 2)

```
1       10    16
ERROR    MVC   OUTTEXT(4),NEWLINE              CLEAR TO END OF LINE AND POSITION
*                                              TO BEGINNING OF NEXT LINE
         CLI   ZA#PSC+1,1                      TEST STATUS CODE
         BNE   IOERROR
*                                ONE MEANS INVALID KEY
         MVC   OUTTEXT+4(L'MSGCON2),MSGCON2   PUT'INVALID STATE NAME'
*                                             INTO MESSAGE
         B     TERM
IOERROR  MVC   OUTTEXT+4(L'MSGCON3),MSGCON3   PUT 'IO ERROR' INTO
*                                             MESSAGE
* TERMINATE EXECUTION OF ACTION PROGRAM BY RETURN CALL TO IMS 9Ø.
* THE DEFAULT TERMINATION INDICATOR IN THE PROGRAM
* INFORMATION BLOCK IS 'N' FOR NORMAL TRANSACTION TERMINATION.
* THE DESTINATION TERMINAL ID IN THE OUTPUT MESSAGE AREA HEADER IS
* SET TO THE DEFAULT VALUE OF THE ORIGINATING TERMINAL AND THE TEXT
* LENGTH IS SET TO THE CONFIGURATION STANDARD LENGTH.
TERM     ZG#CALL  RETURN
         EJECT
* CONSTANTS
STATE    DC    CL7'STATE'              ISAM FILE NAME
MSGCON1  DC    C'CAPITAL'
MSGCON2  DC    C'INVALID STATE NAME'
MSGCON3  DC    C'I/O ERROR'
NEWLINE  ZO#POSC,Ø,Ø          ICAM PROCEDURE TO GENERATE DICE SEQUENCE FOR
*                             NEW LINE CONTROL WITH CLEAR
         EJECT
* ACTIVATION RECORD DEFINITION
         ZM#DIMH
TCODE    DS    X          TRANSACTION CODE
         DS    X          SPACE
SNKEY    DS    XL4        STATE NAME KEY
         EJECT
WORK     DSECT    WORK AREA
RECORD   EQU   *
SNAME    DS    XL14      STATE NAME
SPOP     DS    XL8       STATE POPULATION
SCAPITAL DS    XL25      STATE CAPITAL
SAVEAREA DS    18A       REGISTER SAVE AREA
PLIST    DS    4A        PARAMETER LIST REFERENCE BY ZG#CALL MACRO
         EJECT
         ZM#DOMH         OMA CONTROL HEADER
OUTTEXT  DS    XL42      OUTPUT MESSAGE TEXT AREA
         REGEQU          THIS PROC DEFINES MNEMONIC REGISTER NAMES
         END
```

Figure 3—41. Sample BAL Action Program (Part 2 of 2)

4. Generating Edit Tables

4.1. PURPOSE

The validity of input data is a major concern in an online system. Writing validation procedures in an action program, however, can be a complicated and time-consuming process. This process can be greatly simplified by means of an offline IMS 90 utility program, the edit table generator (ZH#EDT), available under single-thread and multithread IMS 90 but not basic IMS 90. The edit table generator offers a convenient means for converting freeform 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.

The output of the edit table generator is written to the named record (NAMEREC) file, from where it is loaded at the appropriate time by applications management. Each edit table is associated with a particular action at configuration time by means of 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.

4.2. INPUT TO EDIT TABLE GENERATOR

Input to the edit table generator is in the form of keyword parameters that define the edit table, each field of the input message to be edited, and the edit criteria for each field.

4.2.1. Coding Rules

The following rules apply to the coding of input to the edit table generator. Note that the statement conventions in Appendix A also apply.

1. Input cards must contain sequence numbers in columns 77 through 80. Input cards must be presented in ascending order. The lowest permissible sequence number is 0001.

2. Parameters can be coded in any columns between 1 and 76. Blanks are ignored and are permitted anywhere in the edit table definition.

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.

4. A new edit table specification must start on a new line. Each field need not begin on a new line.

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 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.

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.

7. Numeric values are positive unless preceded by a minus sign (-). The plus sign (+) is not permitted in numeric values.

4.2.2. Input Parameters

The character specified by the SEP parameter is used as the separator.

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}]
[    {C}]
[    {N}]
[    (P)]
```

where:

SEP=separator-character
: Specifies the field separator character for both the edit table definition and the input message to be edited. It can not 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.

ETAB=tablename

Identifies the edit table and must immediately follow the SEP parameter. This specification associates the edit table with an action at configuration, via the EDIT=tablename option in the ACTION section.

The tablename must be 2 to 7 alphanumeric characters, the first of which must be alphabetic.

KEY

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 another KEY parameter is encountered.

Input message fields can be positional or keyword. Positional fields are defined by a numeric specification in the KEY parameter (e.g., KEY=1). Keyword fields are defined by an alphanumeric specification (e.g., KEY=AMT). Positional fields must precede keyword fields and must be specified in order, beginning with 1. Once a keyword field is defined, all additional fields must be keyword; positional fields are no longer accepted for the current edit table.

KEY=keyword

Specifies a 1- to 3-character alphanumeric identification, the first character of which must be alphabetic, for a keyword field in the input message. Keyword fields must be entered at the terminal in the form *keyword=data.* Once a keyword field is identified in the edit table definition, all subsequent fields must be defined as keyword 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. The first field in an input message is always the transaction code, and this must always be specified as a positional field.

LEN=field-length

Specifies the length of the edited field as required by the action program and is a required parameter. A maximum of 255 characters can be specified for alphanumeric fields and four characters (one full word) for binary fields. Ten characters is the maximum length for numeric fields unless both MIN and MAX parameters are specified for this field. If a numeric field is identified in the action program as packed decimal, up to 16 characters can be specified.

NOTES:

1. *If the field-length is larger than the width of the screen on which data is to be entered, IMS 90 removes the DICE code at the end of each line of terminal input and replaces it with a blank character. These additional blank characters must be provided for in the action program and included in the field-length specified by the LEN parameter.*

2. *The length specified for both binary (TYP=B) and packed (TYP=P) fields must equal the length for each binary or packed item in the input message. For example, if a field is defined as packed with a LEN=3 then a number keyed in at the terminal cannot be 1000 or greater. This will cause an error stating your input field is too large, even though 1000 may be represented in a packed field in 3 bytes.*

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. The space is included in the field-length specified by the LEN parameter. For example, if the transaction code is PAY, the operator must key in PAY△, and the LEN parameter must be specified as LEN=4.*

   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.

POS=starting-position

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. The maximum permissible specification is 32,767.

FIL=fill-character

Optionally specifies the fill character to be inserted in the edited field when the field as input from the terminal is less than the field-length specified by the LEN parameter. The default fill character is 0. If spaces (X'40') are desired as fill characters, this parameter can be coded as either FIL= or FIL=△; i.e., a space can be included or omitted 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.

JUS=L

Specifies left justification of this field in the edited message. Binary and packed fields are always right-justified; therefore, this parameter is ignored if specified for a binary field.

JUS=R

Specifies right justification of this field in the edited message and is the default assumption.

MAN=N

Specifies that this field is not mandatory in the edited message in order for input to be acceptable.

MAN=Y

Specifies that this field is mandatory in the edited message.

MAX=maximum-value

Specifies the maximum value allowed for the field in the input message. This parameter is applicable only to numeric fields. The highest value that can be specified is $2^{31}-1$. The number of characters in this value must not exceed the length specified by the LEN parameter.

MIN=minimum-value

Specifies the minimum value allowed for the field in the input message. This parameter is applicable only to numeric fields. The lowest value that can be specified is $-(2^{31}-1)$. The number of characters in this value must not exceed the length specified by the LEN parameter.

TYP

Specifies 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.

TYP=N

Specifies numeric data.

TYP=P

Specifies packed decimal data.

4.3. EXECUTING EDIT TABLE GENERATOR

4.3.1. Sample Execution Run Stream

Once input parameters describing the edit table format are specified on cards and the NAMEREC file has been initialized, the ZH#EDT edit table generator utility can be executed using the control stream illustrated in Figure 4-1.

```
// JOB ADDEDT,,A000
// DVC 20 // LFD PRNTR
// OPTION DUMP
// DVC 50 // VOL DS9999 // LBL NAMEREC,DS9999 // LFD NAMEREC
// EXEC ZH#EDT
/$
     source cards
        .     .
        .     .
        .     .
     source cards
/*
/&
// FIN
```

Figure 4—1. Sample Execution of Edit Table Generator

Note that the LFD name of the NAMEREC file is NAMEREC, not ISAMNRF as must be specified for the NAMEREC file utility and the data definition processor.

If the input definition is acceptable, the generated edit table is written to the NAMEREC file and the following message is issued:

```
tablename ADDED
```

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 messsage is issued:

```
TABLE ADDED, DUPLICATE DELETED
```

If errors cause rejection of the edit table, the following message is issued:

```
tablename REJECTED
```

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.

4.3.2. Error Processing

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.

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
```

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 4–1.

An example of an input statement error and the resultant error message is as follows:

Input:

```
SEP=,ETAB=EDIT1,KEY=1,LEN=5,POS=0, JUS=X,MAN=Y  0002
```

Error message:

```
0002 39 JUSTIFICATION ILLEGAL
```

Table 4-1 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 4—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*
DIGIT MUST BE NUMERIC	The value specified for the LEN, MAX, or MIN parameter was not a numeric value.
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
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

*These errors set the UPSI byte to 80; all other errors in this table result in an UPSI byte value of 40.

Table 4—1. Edit Table Diagnostic Messages (Part 2 of 2)

Error Message Text	Explanation
LEN OR POS MISSING	Required parameters
LEN ZERO	A zero value was specified for the LEN parameter Value must be greater than zero
MAX OR MIN ABSOLUTE VALUE TOO LARGE	2^{31}-1 is largest absolute value allowed
MAX VALUE LESS THAN MIN	The value specified for the MAX parameter is less than the value specified for the MIN parameter
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.

4.4. ENTERING INPUT MESSAGES FROM TERMINAL

When an input message for which an edit table has been generated is entered from the terminal, it is processed by an IMS 90 component called the expanded input editor. The following considerations apply to the entering of these input messages.

- The transaction code must be the initial field of the message. The first field of the edited message at execution time may be two characters or longer; however, if the first field is longer than five characters, only the first five characters are used as the transaction code to schedule the action programs.

- Positional fields begin with the first nonblank character and extend to the next separator. Positional fields must appear in the input message in the same order as specified in the edit table definition. If a positional field is omitted, a separator character must be entered to indicate its position. Positional field editing is terminated when all positional fields have been edited, when a keyword field is encountered, or when end of data has been reached. A positional field may not contain an equal sign.

- 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.

- Numeric values are positive unless preceded by a minus sign. The plus sign (+) is an invalid character.

- Error messages are displayed on the first line of the display terminal; therefore, it is recommended that input messages be started on the second line so that the input is not erased by an error message.

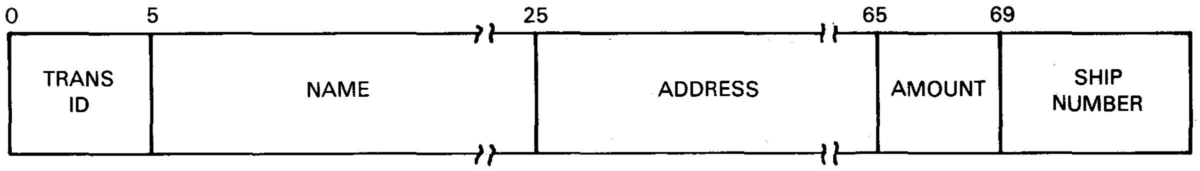
- If fields are continued from one line to another, IMS 90 removes the DICE code at the end of each line and replaces it with a blank character, which is sent to the action program as part of the data. Fields that do not exceed the width of the screen should always be entered on one line. If a field exceeds the screen width and must be continued from one line to another, the terminal operator should avoid splitting a word between lines, and the action program should provide for the additional blank characters.

- If the terminal input ends with a positional parameter (no keyword parameters are specified), the separator character must end the input message; otherwise, the input message could be partially deleted. A correct terminal entry is:

  INFOR,BIOLOGY,CLASS2,MARY J. BLISS,

4.5. SAMPLE APPLICATION

Figure 4-2 and Table 4-2 describe sample input to the edit table generator for an accounts receivable application where the edited input is to be delivered to the action program in the following format:



```
1                                                                  77 80
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 4—2. Sample Input to Edit Table Generator

Table 4—2. Description of Sample Input to Edit Table Generator

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. This field is always the transaction-id.
	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.
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 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.

The following examples represent freeform input from the terminal and the resulting messages sent to the action program in accordance with the specifications in the edit table generated for this application or, in case of error, the output message displayed at the terminal. Note that in these examples, 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:

```
PAYMTJOHN△D.△SMITH△△△△△△△1112△BREEZE△DR.△PHILA.△PA.△19160

△△△△△△△△09C4123456
```

Terminal Input:

```
PAYMT,JOHN D. SMITH,,SN=123456,AMT=2500
```

Edited Message:

```
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. 19160,
AMT=700,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. 19160,SN=123456
```

Output Message:

```
AMT MISSING
```

Explanation:

AMT was specified as mandatory.

5. Terminal Operation

5.1. TERMINAL I/O MESSAGE PROCESSING

IMS 90 terminal operation is concerned with the transmitting and receiving of messages. For every message transmitted by the terminal operator, there is always at least one output response. Input messages are of two types – transaction messages and terminal commands.

Transaction messages are related to action programs, both IMS 90 and user-supplied. Because several actions can be linked together to form a transaction, each transaction can include a series of input and output messages. Each transaction is initiated by a 1- to 5-character transaction code.

When you use the uniform inquiry update element (UNIQUE), file processing transactions are performed by means of UNIQUE commands. (See Section 6.) The OPEN command is the transaction code that initiates UNIQUE processing.

File processing functions also can be performed by user-written action programs via user-defined transaction codes and embedded messages. IMS 90 also provides transaction codes for communicating between terminals (SWTCH), displaying the last valid output message (DLMSG) for recovery purposes, downline loading user programs to a UTS 400 terminal (DLOAD), and displaying statistical information (ZSTAT).

Administrative and control functions are handled by IMS 90 terminal commands. Master terminal commands allow overall monitoring of the system and control of the communications network, while standard terminal commands perform administrative and operational functions for the individual terminal. All terminal commands start with the letters ZZ.

5.1.1. Initiating Online Processing

After the IMS 90 start-up procedure has been performed, as detailed in the IMS 90 system support functions user guide/programmer reference, UP-8364 (current version), the message IMS READY appears on each terminal that is in a ready state, unless you have chosen, by configuration, the option of not receiving the IMS READY message. The terminal operator can then begin transmitting messages. This message appears only once, at start-up time. Terminals coming online at a later time do not receive IMS READY unless they are marked down at the time IMS 90 is initiated.

5.1.2. Transmitting Messages from Display and Hard Copy Devices

When commands and other messages are issued from a display device, the terminal operator must enter the message and press the TRANSMIT key. If input is from a hard copy device (e.g., SPERRY UNIVAC DCT 500 Data Communications Terminal or a teletypewriter), an end-of-text (ETX) character must follow the last character of the message. In both cases, commands and transaction codes must be entered in uppercase if the translate option has not been configured.

When a hard copy device is used, characters or messages erroneously entered can be deleted through the use of control characters. The control characters are:

Function	Control Character
Delete a character	H
Delete a message	X

More than one character can be deleted by entering the H consecutively as many times as there are characters to be deleted.

5.1.2.1. Transmitting DICE Sequences from Hard Copy Devices

On hard copy devices, device-independent control expression (DICE) sequences are transmitted to action programs by means of the carriage return key and the line feed key.

Before keying in the UNIQUE command OK or CANCEL to execute or cancel an ADD, CHANGE, or DELETE transaction on a hard copy device, the operator must press the carriage return key and the line feed key. This is necessary because UNIQUE expects to receive the DICE sequence before an OK or CANCEL command. If the operator does not press these keys before transmitting an OK or CANCEL command, IMS 90 cancels the transaction and returns an INPUT ALTERED message to the terminal. (If this occurs, the operator reenters the UNIQUE update operation.)

On the other hand, whether the operator should or should not press these keys before keying input for a user-written action program depends upon whether editing of input messages has been specified for the action at configuration time and what the action program expects to receive as input. If no editing has been specified and the keys are pressed, IMS 90 passes the 4-byte DICE sequence to the action program that receives control to be processed. If input editing has been specified, the DICE sequence is stripped entirely (except for the (CR) DICE sequence, which is replaced by a blank). The action program must be able to accept such characters in the input, and the terminal operator must be appropriately instructed.

5.1.2.2. Handling Multiline Terminal Messages

When keying in an input message at the terminal, the carriage return (CR) DICE sequence (10040000_{16}) is replaced by one blank character (40_{16}), unless the carriage return DICE sequence precedes valid input text or unless the EDIT=NONE parameter is specified in the ACTION section of the configuration. If the carriage return sequence precedes valid input text, it is deleted completely. If NONE is specified, the input message is transmitted to the action program unedited.

For example, if the word "END" is divided between two lines at the terminal in this manner:

```
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ EN (CR)
D
```

the resulting message in the input message area (IMA) of the action program would be:

```
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ EN△D
```

This example emphasizes that the best approach to multiline message composition is to use the carriage return between words to avoid embedded spaces in the resulting message transmitted to the IMA. Note that UNIQUE rejects fields that are divided across two lines. Consequently, you should avoid splitting fields in messages that use more than one line.

5.1.3. Initiating a Transaction

The first input message of a transaction must contain a 1- to 5-character transaction code beginning with the first character of the message. If the transaction code is less than five characters in length, it must be followed by a blank. If the input message begins with a ZZ, it is interpreted as a terminal command to be processed by IMS 90.

Transaction codes OPEN, SWTCH, DLMSG, DLOAD, and ZSTAT initiate IMS 90 action programs and are reserved. Any other transaction code must be defined to the system at configuration time in a TRANSACT section or it will be rejected as invalid input. In a dialog transaction, all input messages after the first are not examined for a transaction code and, in fact, may contain any characters in the first one to five positions except for ZZ.

Transactions can be initiated only from terminals and not from the system console. A transaction can be initiated by a function key if the function key has been defined at configuration in a TRANSACT section. A transaction can be initiated only when the previous transaction or terminal command has been resolved in its entirety. If a transaction code is entered prior to the resolution of the previous input, it is treated as unrecognizable input and is ignored.

When an output message requires an input response, that response must be transmitted prior to the initiation of another transaction.

5.1.4. Solicited and Unsolicited Output

The initial output received at a terminal in response to an input from the terminal is called solicited output. Solicited output is unconditionally transmitted when it becomes available.

Unsolicited output is any output other than the initial response to a message from the same terminal. It is available only if specified at configuration. There are two types of unsolicited output:

- Additional output to the initiating terminal after the original output (i.e., segmented output). When the volume of output is greater than the screen size of the initiating terminal, it must be transmitted in segments.
- Output received at a terminal as a result of an input from another terminal (i.e., switched output).

To avoid the possibility of unsolicited output conflicting with the terminal operator's efforts (e.g., developing a screen of data for input), IMS 90 warns the terminal operator of pending unsolicited output via an unsolicited output indicator as listed in Table 5-1.

Table 5—1. Unsolicited Output Discipline

	Display Devices	Hard Copy Devices
Unsolicited output indicator	Message waiting light	/CMW*
Acceptance response	Press message waiting light key.	Transmit bell Press CTRL G. Press CTRL C.

*The unsolicited output indicator to a hard copy device is a 4-character message defined by the MSGWAIT keyword parameter of the TERM macro in the CCA definition. The default is /CMW. Refer to the ICAM network definition and operations user guide, UP-8947 (current version).

When output is segmented, the unsolicited output indicator is displayed prior to each segment. The operator must accept all segmented output.

If the unsolicited output indicator is displayed to notify the terminal operator of the availability of a message sent from another terminal, the operator can ignore the signal and transmit an input message, or he can accept the unsolicited switched message by pressing the message waiting key on display devices or by pressing simultaneously the CTRL and G followed by the CTRL and C keys (ETX) on hard copy devices.

If more unsolicited messages are waiting, the unsolicited output indicator is sent again and the operator can again ignore the signal or accept the message.

The switched output indicator will appear at a terminal only if that terminal is not in interactive mode, unless UNSOL=ACTION is specified to the configurator in the TERMINAL section.

When handling segmented or switched output, it is advisable to indicate output disk queueing in the ICAM network definition.

5.1.5. Function Keys

Function keys F1 through F4 on the UNISCOPE display terminal F1 through F22 on the UTS 400 terminal and F1 through F33 on the IBM 3270 terminal provide a fast, efficient method of accessing user-written action programs. A function key can be used in either of two ways - as a transaction code or as a response to an output message. A function key used as a transaction code must be defined to the configurator in the TRANSACT section. A function key used as an input response is defined in the action program.

The same function key can be used for both purposes. If the terminal is in interactive mode, the function key calls in the successor action program identified in the previous action. If the terminal is not in interactive mode, the function key is interpreted as a transaction code.

Function keys can be entered from a hard copy terminal by keying in F#nn, where nn is 01 through 33.

5.1.6. Automatic Status Messages

One of the following automatic status messages is output by IMS 90 (multithread only) to notify a terminal operator of the status of the last input message. The appropriate status message is sent to the terminal periodically until the end of the current action:

Automatic Status Message	Meaning
INPUT IN QUEUE	IMS 90 has received the input message but has not yet delivered it to the action program.
INPUT IN PROCESS	The action program has received the input message and is processing it.
ROLLBACK IN PROCESS	The action program has terminated abnormally, and the updated user data files are being restored to the last rollback point.

At any point after the first status message, the terminal operator can cancel the current transaction via the ZZCNC command (5.2.1.6). Any input other than the ZZCNC command transmitted prior to the response output message is ignored.

5.2. TERMINAL COMMANDS

Terminal commands are used for various administrative, operational, and control functions. Master terminal commands are used to control the overall system and the communications network, while standard terminal commands are used by the individual terminal operator to control the flow of messages at his terminal. All terminal commands begin with the characters ZZ. Table 5-2 summarizes the master and standard terminal commands supported by basic, single-thread, and multithread IMS 90. They are described in 5.2.1 and 5.2.2.

Table 5—2. IMS 90 Terminal Commands

Command	Master/ Standard	Function	IMS 90		
			Basic	Single-Thread	Multithread
ZZALT	Master	Designates an alternate terminal	No	Yes	Yes
ZZBTH	Master	Initiates and controls batch processing of transactions in online mode from the master terminal (Refer to 7.6.1)	No	Yes	Yes
ZZCLS	Master	Closes files from master terminal	Yes	Yes	Yes
ZZCNC	Standard	Cancels the current transaction	Yes	Yes	Yes
ZZDWN	Master	Closes a terminal down from master terminal	Yes	Yes	Yes
ZZHLD	Standard	Suspends output to a terminal temporarily not ready to receive it	Yes	Yes	Yes
ZZHLT	Master	Causes immediate shutdown of activity from master terminal or system console*	Yes	Yes	Yes*
ZZMCH	Standard	Changes master terminal	No	Yes	Yes
ZZNRM	Standard	Reverts to normal mode after terminal has been in test mode (ZZTMD)	Yes	Yes	Yes
ZZOPN	Master	Opens files from the master terminal	Yes	Yes	Yes
ZZPCH	Master	Permits IMS 90 to load recompiled action programs	No	Yes	Yes
ZZRDY	Standard	Reports that terminal is now ready for output suspended by ZZHLD command	Yes	Yes	Yes
ZZRSD	Standard	Causes last initial response message to be retransmitted	No	Yes	Yes
ZZSHD	Master	Causes orderly cessation of IMS 90 from master terminal or system console*	Yes	Yes	Yes*
ZZTCT	Master	Obtains status of terminal control table from master terminal	Yes	Yes	Yes
ZZTMD	Standard	Places a terminal in test mode	Yes	Yes	Yes
ZZTST	Master	Tests whether a terminal on network is able to receive output	No	Yes	Yes
ZZUP	Master	Brings terminal online from master terminal	Yes	Yes	Yes

*If OPCOM=YES has been specified to the configurator in the OPTIONS section, the multithread IMS 90 user has the option of sending this master terminal command from the system console at any time. This command should be keyed in as an unsolicited message to the IMS 90 job.

5.2.1. Standard Terminal Commands

Standard terminal commands can be submitted from any terminal and are used at the operator's discretion to control transmittal, testing, and cancellation of messages. They are discussed in 5.2.1.1 through 5.2.1.6.

5.2.1.1. ZZRSD (Resend)

The ZZRSD command directs IMS 90 to retransmit the last transaction-oriented output message displayed at the calling terminal. Terminal command responses cannot be resent except for ZZCNC, ZZCLS, ZZOPN, and (multihead) ZZTST. Once a new input message (other than the ZZRSD terminal command) is initiated, the preceding output message cannot be retransmitted, and the response NO MSG IN QUEUE is displayed. The ZZRSD command can be initiated only when there is no unresolved input message; it is allowed for only one retransmission of an output message. It is not available if RESEND=NO is specified to the configurator in the OPTIONS section.

5.2.1.2. ZZHLD (Hold)

The ZZHLD command directs IMS 90 to suspend all output to the initiating terminal until a ZZRDY command is submitted. This command can be initiated only when there is no unresolved input. You may not use the ZZHLD command at a local workstation because it locks the workstation. A typical situation for use of the ZZHLD command is if the ribbon broke or the paper jammed on the hard copy device.

On the UNISCOPE 100 and 200 display terminals, the operator must press the WAIT switch to allow subsequent input after transmitting the ZZHLD terminal command; this means that the WAIT switch must be pressed before transmitting the ZZRDY command to resume receiving output. On UTS 400 terminal; the operator must press the KBOARD UNLOCK switch before transmitting the ZZRDY command to resume receiving output.

5.2.1.3. ZZRDY (Ready)

The ZZRDY command directs IMS 90 to release the hold state on the initiating terminal and to transmit output as it becomes available. This command is valid only if initiated while the terminal is in the hold state; if submitted at any other time, it has no effect.

5.2.1.4. ZZTMD (Test Mode)

The ZZTMD terminal command directs IMS 90 to place the initiating terminal in test mode. When a terminal is in the test mode, there is no physical alteration of data files; any command to alter a data file (ADD, DELETE, CHANGE) is simulated. The uses of the ZZTMD command are:

1. to allow new or revised action programs to be tested and debugged without risking the compromise of online files; and

2. to permit training of terminal operators without causing actual update transactions to be made against user files.

When the session of testing or training is completed, the command ZZNRM is issued to revert to normal operating mode. Neither the ZZTMD nor the ZZNRM command can be issued while the initiating terminal is operating in interactive mode; that is, in the midst of a transaction.

5.2.1.5. ZZNRM (Normal Mode)

The ZZNRM command directs IMS 90 to return the initiating terminal to normal mode, thus enabling alteration of data files as authorized. This command is valid only if initiated while the terminal is in the test mode. If submitted at any other time, it has no effect.

5.2.1.6. ZZCNC (Cancel)

The ZZCNC terminal command directs IMS 90 to cancel the transaction currently active. It does not cancel a terminal command. This command is used most often to break deadlock situations under multithread IMS 90 operations. It also clears all output queued to this terminal. It is not a command that the master terminal issues on behalf of another terminal, but must be initiated by the terminal experiencing the deadlock.

Once an input has been sent, the ZZCNC command cannot be transmitted until at least one output message has been sent to the terminal. The output message can be a response message or an automatic status message.

5.2.1.7. ZZMCH (Master Terminal Change)

If the master terminal is down, a new master terminal can be designated by issuing the ZZMCH terminal command. The ZZMCH command can be entered from any terminal. If the old master terminal becomes operable again during the IMS 90 session, it can be reactivated as a network terminal but not as the master terminal (unless the ZZMCH command is used again). The format of the ZZMCH command is:

```
ZZMCH terminal-id
```

where:

`terminal-id`
Is the configured symbolic identification of the new master terminal.

If the command is processed successfully, the following response is transmitted to the terminal that initiated the command:

```
NEW MASTER TERMINAL IS terminal-id
```

If the master terminal is not down, the response is:

```
INVALID MASTER TERMINAL COMMAND
```

If the named terminal is not identified in the network definition, the following response is sent:

TERMINAL terminal-id CANNOT BE FOUND

NOTE:

The ZZMCH command can be entered only from a configured terminal (not the system console).

5.2.2. Master Terminal Commands

Master terminal commands enable privileged control and monitoring of the IMS 90 system. Master terminal commands must be submitted from the master terminal with two exceptions: ZZHLT and ZZSHD.

The ZZHLT and ZZSHD commands can be entered from the system console or master workstation (if present) in a multithread system if OPCOM=YES is configured. Master terminal commands submitted from a regular terminal are rejected as invalid.

An IMS job may have a master workstation associated with it either by means of job control or initiation of that job from a workstation. In either case, enter unsolicited keyins from the system console or the job's master workstation. This applies to multithread systems when OPCOM=YES is configured and to single-thread systems when the console is configured as the master terminal. Responses are always sent to the job's master workstation, if present, and are only sent to the console if the job has no master workstation or if it has been logged off.

The master terminal commands are described in 5.2.2.1 through 5.2.2.12.

5.2.2.1. ZZUP (Terminal Up)

The terminal-up command directs IMS 90 to enable a terminal that has previously been disabled. The typical situation for the use of this command is when a terminal, that has been inoperable and placed in the disabled state by the terminal-down command, becomes operable.

Format:

```
ZZUP terminal-id
```

where:

`terminal-id`
Is the configured symbolic identification of the specified terminal.

5.2.2.2. ZZDWN (Terminal Down)

The terminal-down command directs IMS 90 to disable a specified terminal. A typical situation for the use of this command is to logically disable a terminal that is malfunctioning. A terminal that is physically down is marked logically down by means of the ZZDWN command. Othewise, a degradation in polling can result, causing longer response times for active terminals.

Format:

```
ZZDWN terminal-id
```

where

```
terminal-id
```
Is the configured symbolic identification of the specified terminal.

NOTE:

It is possible to inadvertently disable the master terminal by specifying the terminal-id of the master terminal itself. To remedy this, control of the network can be transferred to a regular terminal by means of the ZZMCH command (5.2.2.8). In a multithread system, an orderly shutdown (ZZSHD, see 5.2.1.7) can be directed from the system console if this option is configured.

5.2.2.3. ZZTST (Test Terminal)

The test terminal command directs IMS 90 to transmit a message to a specified terminal. Upon receipt of the response, the master terminal is advised whether or not the test was successful. If the terminal being tested was disabled earlier using the ZZDWN command, the response message to the master terminal will indicate that the test was unsuccessful. The ZZTST master terminal command is used to determine whether a production terminal is able to receive solicited or unsolicited output. To use this command, you must specify the UNSOL=YES in the OPTIONS section input to the configurator. The ZZTST command is not available in basic IMS 90.

Format:

```
ZZTST terminal-id
```

where:

```
terminal-id
```
Is the configured symbolic identification of the specified terminal.

5.2.2.4. ZZTCT (Terminal Control Table Status)

The terminal control table status command directs IMS 90 to output to the master terminal the current status of the specified terminal. This output includes status of the terminal (up or down), outstanding activity, and volume and type of traffic since startup.

Format:

```
ZZTCT terminal-id
```

where:

```
terminal-id
```

Is the configured symbolic identification of the specified terminal.

In response to the ZZTCT terminal command, the following message is sent to the master terminal:

```
STATUS OF terminal-id(UP  );nnnn MSG;nnnn TRAN;nnnn TRM CMD; ALT=name
                     )DWN (
                     )HLD (
                     (TEST)
```

where:

```
nnnn
```

Is the number of messages, transactions, or terminal commands.

```
name
```

Is the name of the alternate terminal, if specified.

5.2.2.5. ZZALT (Alternate Terminal Designation)

The ZZALT master terminal command directs IMS 90 to assign an alternate terminal for a designated terminal. Once the alternate terminal is assigned, switched output messages destined for the original terminal are rerouted to the alternate terminal whenever the original terminal is physically or logically down. Solicited and nonswitched messages are not affected. When the original terminal becomes operational again, message alternation is discontinued. The ZZALT command also can be used, without the alternate terminal-id, to restore the original terminal. Restrictions on the use of this command are:

- UNSOL=YES must have been specified in the OPTIONS section of configurator input.
- If the alternate terminal is physically or logically down, the switched messages are written on the original terminal's queue.

- Only one level of alternation is permitted; i.e., if B is designated as an alternate terminal to A, and C is designated as an alternate terminal to B, switched messages for A are sent to B, not C. However, the ZZALT command can be entered again to designate C as an alternate terminal to A, overriding the first ZZALT command. The ZZTCT master terminal command can be used to determine what alternate terminal is in effect.

- Test messages generated by the ZZTST command in single thread are not directed to the alternate terminal.

The format of the ZZALT command for naming an alternate terminal is:

```
ZZALT terminal-id,alt-terminal-id
```

where:

`terminal-id`
Is the configured symbolic identification of the original terminal.

`alt-terminal-id`
Is the configured symbolic identification of the alternate terminal to which traffic is to be routed.

If successful, the response is:

terminal-id IS ALTERNATED TO alt-terminal-id

The format of the ZZALT command for restoring an alternated terminal is:

```
ZZALT terminal-id
```

where:

`terminal-id`
Is the configured symbolic identification of the restored terminal.

If successful, the response is:

terminal-id IS RESTORED

If unsolicited output has not been configured, the following response is transmitted:

INVALID MASTER TERMINAL COMMAND

If an identified terminal is not in the network, the response is:

terminal-id CANNOT BE FOUND

5.2.2.6. ZZCLS (Close File)

The close file command directs IMS 90 to close the file specified. Terminal operators are informed of this action only on submission of input that results in an attempt to access the closed file. The ZZCLS command cannot be directed to a defined file. Also, no comments or characters may follow the file name.

Format:

```
ZZCLS filename
```

where:

filename
Is the configuration-specified file name for the file that is to be closed.

When the ZZCLS master terminal command is issued and no currently open transactions have updated the file, IMS 90 issues a CLOSE macroinstruction to data management and sends the following message to the master terminal operator:

filename CLOSED

When currently open transactions are updating the file indicated in the ZZCLS command, IMS 90 sends the following message to the master terminal:

filename - - IN USE BY terminal-name

where:

filename
Identifies the file being used.

terminal-name
Identifies the terminal currently accessing the file specified in the ZZCLS command.

Once the terminal that was accessing the file ends the transaction, the master terminal operator must reissue the ZZCLS command to close the file.

Any new user of this file after the ZZCLS command is accepted receives file close status; i.e., the value 3 in the STATUS-CODE field of the PIB and the value 6 in the DETAILED-STATUS-CODE. Current users of the file, if any, are allowed to access the file.

On the other hand, if the terminal operator keys in a wrong filename in the ZZCLS command, IMS 90 issues the following message to the master terminal:

filename NOT CONFIGURED

When an attempt is made to close a common storage file, IMS 90 issues the following message to the master terminal:

filename INVALID FUNCTION FOR CDA FILE

If an error occurs during the closing of the specified file, IMS 90 issues the following message to the master terminal:

```
filename CLOSE ERROR DMnn
```

In addition, the data management error code is displayed and further access to the file is inhibited.

NOTES:

1. *When ZZCLS is issued to a sequential output file, the EXTEND parameter in the LFD job control statement for this file must be included to resume with the next ascending record number after ZZOPN.*

2. *The ZZCLS command cannot be issued to a common storage area file.*

5.2.2.7. ZZOPN (Open File)

The open file command directs IMS 90 to open a specified file previously closed via the ZZCLS master terminal. No comments or characters may follow the file name.

Format:

```
ZZOPN filename
```

where:

filename
Is the configuration-specified file name for the file.

If the operator keys in an invalid filename on the ZZOPN master terminal command, IMS 90 issues the following message to the master terminal:

```
filename NOT CONFIGURED
```

When a file cannot be opened because of a conflict of access rights, IMS 90 issues the following message at the master terminal:

```
filename CANNOT OPEN WITH ACCESS SPECIFIED
```

If an error occurs during the opening of the specified file, IMS 90 issues the following message to the master terminal:

```
filename OPEN ERROR DMnn
```

where:

nn

Specifies the data management error code. Refer to the OS/3 system messages programmer/operator reference, UP-8076 (current version).

NOTES:

1. *When ZZOPN is issued to a sequential output file, the EXTEND parameter in the LFD job control statement for this file must be included to resume with the next ascending record number after ZZOPN.*

2. *The ZZOPN command cannot be issued to a common storage area file.*

5.2.2.8. ZZBTH (Batch)

The ZZBTH master terminal command initiates and controls batch transaction processing in the online mode. See Section 7 for the format of this command and details of its use.

5.2.2.9. ZZSHD (Shutdown)

The shutdown command causes IMS 90 to terminate gracefully. The initiation of new transactions is inhibited, but transactions previously initiated can continue processing. Any new transactions are not accepted, and a shutdown indicator is sent to terminals attempting to start new transactions. All terminal commands are accepted. Once there are no longer any outstanding transactions or a shutdown time-out has expired, whichever comes first, IMS 90 terminates. Any transactions not completed at that point are rolled back. Shutdown time-out is 180 seconds for single-thread and five times the action time-out value for multithread (the value given in the ACTION parameter of the configurator TIMEOUTS section).

The following message is sent to terminals attempting to enter transactions after the ZZSHD terminal command has been transmitted:

```
SHUTDOWN IN PROCESS
```

5.2.2.10. ZZHLT (Halt)

The halt command terminates IMS 90 immediately and is used only for emergencies. No notification is delivered to terminal operators, and transactions in process are not rolled back. File recovery is usually required after this type of termination, using either the offline recovery utility or the warm restart option at the next IMS 90 execution.

5.2.2.11. ZZPCH (Program Change)

The program change master terminal command is used for debugging action programs. It allows the programmer to execute an action program online, recompile the action program, and then execute the new version.

When the ZZPCH command is issued, IMS 90 locates the disk address of the new version of the designated program and substitutes it for the disk address of the previous version of the program. Then, when the action program is requested, IMS 90 loads the new version. You also can issue the ZZPCH command to allow IMS 90 to find an action program not in the load library at start-up time. In either case, the program must be identified in the PROGRAM section at configuration and must not be configured as resident.

Format:

```
ZZPCH program-name
```

where:

program-name
Specifies the name of the action program changed or updated.

Before issuing the ZZPCH terminal command, the programmer must replace the old version of the action program with the new version in the IMS 90 load library. Then after issuing the ZZPCH command, you can execute the new action program.

The ZZPCH terminal command should not be used for resident subprograms. If you issue a ZZPCH command for a resident subprogram, the first call to the resident subprogram from an action program cancels the transaction and IMS 90 issues the following error message:

```
program-name IS A SUBPROGRAM - CANNOT BE RELOADED
```

In multithread IMS 90, if the named action program is being used by another terminal, the following message is issued:

```
PROGRAM program-name IN USE
```

The operator should wait until the action program is not in use and reissue the ZZPCH command. The new action program can be loaded only when the current action program is not in use.

The size of the new action program can not exceed the size specified in the MAXSIZE parameter of the configurator ACTION section. If the new size does exceed the MAXSIZE value, IMS 90 cancels the transaction and issues the following error message:

```
PROGRAM SIZE EXCEEDS SPECIFIED MAXSIZE
```

The ZZPCH master terminal command can not be issued for basic IMS 90. If you issue a ZZPCH master terminal command under basic IMS 90, IMS 90 issues the following message:

```
ZZPCH NOT CONFIGURED
```

If the action program to be changed is not configured; i.e., was never specified in the PROGRAM section of the configurator, IMS 90 issues the following message:

```
PROGRAM program-name NOT CONFIGURED
```

If, in the ZZPCH command, you fail to enter the name of the action program to be changed, IMS 90 issues the following message:

```
PROGRAM NAME MISSING FOR ZZPCH COMMAND
```

The terminal operator must reenter the ZZPCH terminal command to supply the action program name.

If the operation of loading the new action program is unsuccessful, IMS 90 issues the following message:

```
PROGRAM program-name LOAD ERROR
```

If the ZZPCH command is successful, IMS 90 issues the following message:

```
PROGRAM program-name MARKED RELOADABLE  VERSION  yymmdd  hhmmss
```

where:

program-name
Specifies the name of the action program containing the changes.

yymmdd
Specifies (in the format year, month, and day) the date of the new action program version being executed.

hhmmss
Specifies the clock time (by hour, minutes, and seconds) of the new action program version being executed.

5.3. IMS 90 TRANSACTION CODES

The transaction codes SWTCH, DLMSG, DLOAD, and ZSTAT can be entered by the terminal operator to initiate special-purpose transactions performed by IMS 90 action programs. The SWTCH transaction code initiates terminal-to-terminal communication. DLMSG retrieves the last valid output message from a terminal. The DLOAD transaction code initiates the downline loading of UTS 400 programs from the host system to a UTS 400 terminal system.

5.3.1. SWTCH (Terminal-to-Terminal Communication)

IMS 90 allows communication between terminals; however, excessive use of this feature degrades performance because the system is not designed to be used as a message switcher. Terminal-to-terminal communication is initiated via the SWTCH transaction code in the following format:

```
SWTCH△⎧MAST               ⎫;△△message-text
      ⎨terminal-id n,...  ⎬
      ⎩ALL                ⎭
```

where:

`MAST`
: Transmits the message-text to the master terminal regardless of the configured symbolic identification of the master.

`terminal-id-n`
: Is the configured symbolic identification of the destination terminal.

`ALL`
: Transmits the message to every terminal in the IMS 90 network. This parameter is used only when SWTCH is issued from the master terminal; it cannot be used in a single-thread system if the console is the master terminal.

The SWTCH transaction code must be followed by at least one blank. A comma must separate each terminal identification, which can have no intervening blanks. The semicolon denotes the end of the list of destination terminals and the beginning of the message text. The message text is sent exactly as it is keyed in, starting with the first position after the semicolon.

Each valid destination terminal receives the message (unsolicited output) in the following format:

```
FROM source-terminal-id-n. message text
```

where:

`source-terminal-id-n`
: Is the configured symbolic identification of the source terminal.

The source terminal receives a status message indicating whether the message has been successfully queued to all destination terminals. If success is not complete, errors or exceptions are noted. Each destination terminal specified must be valid; that is, configured as part of the IMS 90 network. The source terminal is also informed if any destination terminals are down. The message is queued even if the destination terminal is physically or logically down.

If the destination terminal is down and alternated (via the ZZALT command), the message is sent automatically to the alternate terminal. The source terminal is not informed that the destination terminal is down. If the alternate terminal is also down, the source terminal is informed that the destination terminal is down.

Three different status messages may be issued:

```
MSG SENT

MSG NOT SENT INVALID DEST.terminal-id-n,...

MSG QUEUED. TERMINAL DOWN.terminal-id-n
```

Example 1:

Input at source terminal (TRM2):

```
SWTCH TRM1,TRM4; THIS IS THE TEXT.
```

Output at source terminal:

```
MSG SENT.
```

Output at destination terminal:

```
FROM TRM2. THIS IS THE TEXT.
```

Example 2:

Input at source terminal (TRM1, the master terminal):

```
SWTCH ALL; THIS IS THE MESSAGE.
```

Output at source terminal (TRM1):

```
MSG SENT.
```

Output at all configured terminals except the source (master) terminal:

```
FROM TRM1. THIS IS THE MESSAGE.
```

Example 3:

Input at source terminal (TRM4):

```
SWTCH TRM1, TRM2, TRM3, TRM5, TRM7; THIS IS THE MESSAGE.
```

Output at source terminal:

```
MSG NOT SENT. INVALID DEST.TRM3

MSG QUEUED. TERMINAL DOWN.TRM2
```

Output at destination terminals (TRM1, TRM5, TRM7):

```
FROM TRM4. THIS IS THE MESSAGE.
```

NOTE:

TRM2 can receive the switched message when the terminal comes up.

Example 4:

TRM2 is down and alternated to TRM3. TRM3 is not down.

Input at source terminal (TRM1):

```
SWTCH TRM2; THIS IS THE MESSAGE.
```

Output at source terminal (TRM1):

```
MSG SENT.
```

Output at terminal (TRM3):

```
FROM TRM1. THIS IS THE MESSAGE.
```

Example 5:

TRM2 is down and alternated to TRM3. TRM3 is also down.

Output at source terminal:

MSG QUEUED TERMINAL DOWN TRM2

NOTE:

The message is on the queue of the terminal TRM3.

Use of the SWTCH transaction code requires specification of the UNSOL=YES parameter in the OPTIONS section of the configurator. Messages switched to or from the master terminal can only be done if the master terminal is configured as a terminal and not as the console. If the unsolicited output indicator is displayed (this is the message-waiting indicator on a UNISCOPE 100 display terminal) to notify the operator of the availability of a message switched from another terminal, the terminal operator can ignore this signal and initiate input or accept the switched message. (See 5.1.4.)

In addition, generate an ICAM network that supports unsolicited ouput. For more details, see the IMS system support functions user guide/programmer reference, UP-8364 (current version).

5.3.2. DLMSG (Displaying the Last Effective Output Message)

When the cold or warm start-up option is specified at IMS 90 initiation, IMS 90 appends the transaction code DLMSG to the IMS READY message. By pressing the transmit key, each terminal operator can retrieve the last valid output message in the terminal output message file (TOMFILE) for the terminal. This determines how far rollback of the files accessed by that terminal has gone so that production can be appropriately resumed.

The same transaction code, DLMSG, can be entered during online operations to display the last "effective" output message at the origining terminal; i.e., all updates performed since the output message was first generated (if any) have since been rolled back or can be rolled back.

The DLMSG transaction code can be transmitted with a configured terminal-id as a following operand. This retrieves the last valid message output at the specified terminal and displays it at the originating terminal.

The format of the DLMSG transaction code is:

```
DLMSG[△terminal-id]
```

where:

```
terminal-id
```
Is the configured symbolic identification of the specified terminal.

If the named terminal cannot be found in the terminal control table (TCT), the following diagnostic is issued to the originating terminal:

INVALID TERMINAL-ID

If no message is found in the TOMFILE for the named terminal, the following message is issued:

SUBFILE IS EMPTY

If an I/O error occurs while the TOMFILE is being read, the following diagnostic message is issued:

**** I/O ERROR ON TOMFILE ****

**** NO MESSAGE RETRIEVED ****

5.3.3. DLOAD (Downline Loading a UTS 400 Program)

The IMS 90-supplied downline load action program, ZUKLOD, enables users to downline load UTS 400 programs from the IMS 90 load library to a UTS 400 terminal. ZUKLOD is activated by the transaction code DLOAD. The format is:

```
DLOAD   program-name[,aux-device-no,prog-id]
```

where:

```
program-name
```
Is the 8-byte name of the load module to be downline loaded.

```
aux-device-no
```
Is the auxiliary device number of the auxiliary storage device (cassette or diskette subsystem). This number must be between 1 and 9 inclusive.

```
prog-id
```
Is the name of the program being downline loaded to either a cassette or a diskette. Only alphanumeric characters (A through Z and 0 through 9) may be used. Prog-id may be a maximum of 4 characters long.

The optional entries are used for downline loading to a peripheral device. If these optional entries are not used, the default loading is directed to the UTS 400 memory.

On initial activation, the ZUKLOD action program issues the function call, SETLOAD. The SETLOAD function prepares a work area defined within the CDA for use by the GETLOAD function. Following the call to the SETLOAD function, ZUKLOD calls the GETLOAD function, which provides a block of code from the UTS 400 program load module. This code is prefixed with proper header information and sent as an output message to the UTS 400 terminal using the continuous output feature with external succession. The same action program (ZUKLOD) is rescheduled to continue issuing GETLOAD functions.

5.3.3.1. Downline Load to Main Storage

After the last block of code is sent to the UTS 400 terminal, the terminal transmits a response message indicating whether or not the downline load was successful. After ZUKLOD terminates normally, a message is displayed indicating the success or error of downline loading. These messages are listed in Table 5-3. All messages are displayed at the UTS 400 terminal requesting the downline load except DELIVERY NOTICE ERROR, which is displayed at the IMS 90 master terminal.

NOTE:

If the UTS 400 program is to be executed directly after a successful downline load, it is responsible for capturing the successful load message at the UTS 400 entry point 'TEXTCM'. If the UTS 400 program does not consider this entry point, the master or primary screen and file control characters will be cleared and the successful load message displayed.

Table 5—3. ZUKLOD Action Program Messages

Message	Meaning
DELIVERY NOTICE ERROR ON UTS 400 TERMINAL: nn	An error occurred when the operator tried to send a block of data to the UTS 400. The letters nn represent a two-character hexadecimal delivery notice error code. (See Tables 3-19 and 3-20 for a description of the error code.)
LOAD ERROR ENCOUNTERED FROM	A host system read error has occurred, reading a block of data from the load library. nn = system loader error code. See OS/3 system messages programmer/operator reference, UP-8076 (current version).
SUCCESSFUL LOAD	The program has loaded successfully.
UTS 400 LOAD ERROR BIT(S) n [&n]	See Table 3-16 in 3.13.1 for a description of load error bits.
INVALID CHARACTER IN OFFLINE	Only alphanumeric characters A through Z and 0 through 9 are allowed for name of program being downline loaded to a cassette/diskette. Reenter the DLOAD transaction code using corrected program name.
BAD AUX NUMBER	Only the numbers 1 through 9 are valid for the one-character auxiliary-device-number.

5.3.3.2. Downline Load to Auxiliary Storage Device

Unlike a downline load to main storage, the UTS 400 terminal does not generate a response message after the last block of code is sent to the auxiliary storage device. Hence, the status of the downline load is not known until the program is read from the auxiliary storage device into main storage.

5.3.4. ZSTAT (Displaying Statistical Information)

The ZSTAT transaction displays the current status of all files, programs, transactions, and terminals in an IMS 90 network. If you use the // PARAM RESTART statement at IMS 90 start-up time (for multithread only) after you have previously shut down normally, you can accumulate these same statistics for multiple sessions.

You can enter the ZSTAT transaction code only from a configured master terminal. You can't use ZSTAT in a single-thread system in which the console is the master terminal.

When using ZSTAT on a UTS 400 device, you must enter VAR on the control page transmit function, XMIT, to indicate transfer of all variable information and field control characters associated with the variable data. Also, you must set the FCC/PROTECT switch to the FCC position.

Issue the ZSTAT transaction code in the following format:

```
ZSTAT⎡⎧ALL [,CONT] [,AUXn] [,TCSassss]⎫⎤
     ⎣⎩HELP                           ⎭⎦
```

where:

n
: Is the auxiliary device identification.

a
: Is the cassette/diskette track address.

ssss
: Is the cassette/diskette starting position.

You can issue the ZSTAT transaction code in three ways:

1. The ZSTAT ALL [,CONT] [,AUXn] [,TCSassss] format allows a choice of displaying statistics for all files, programs, transactions, and terminals in noncontinuous or continuous output modes to the primary output device and the auxiliary output device. When using continuous output and an auxiliary device, you must configure CONTOUT=YES. If ZSTAT ALL is entered without the optional parameters, noncontinuous output mode is assumed and the output is displayed on the primary device only.

2. The ZSTAT HELP format displays the ZSTAT parameters with an explanation of each on the primary device only. You can then either terminate or continue processing ZSTAT.

3. The ZSTAT transaction code entered alone at the terminal displays a menu which allows you to select the statistical information you desire and indicate continuous output and an auxiliary device number ID.

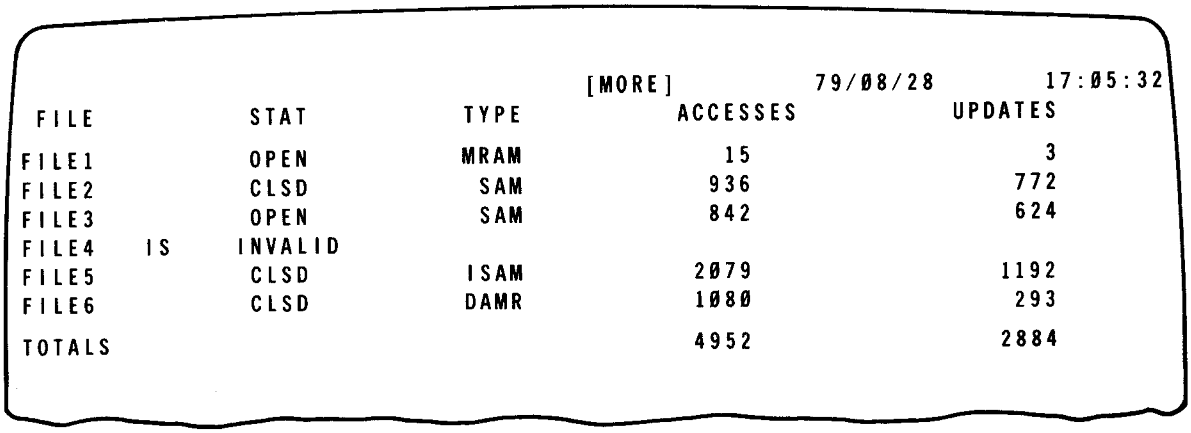
Because ZSTAT produces a menu screen that contains protected fields, only display devices that have a screen protection feature can use the menu form of the ZSTAT transaction code. Hard copy devices or display devices not having the screen protection feature can only use the ZSTAT ALL form of the transaction code.

When you enter ZSTAT ALL, the resulting output is a display of all the current files, programs, transactions, and terminals in the IMS 90 network. Each of these items starts a new page of output. Figures 5-1 through 5-4 show sample noncontinuous mode output displays for each item. The ZSTAT format does not produce menu screen output. Therefore, if you wish to indicate continuous output and an auxiliary device ID, you must enter the CONT and AUXn parameters with the ZSTAT transaction code. If the auxiliary device is a cassette or diskette, the TCS parameter must follow the AUX parameter.

The file status display contains the following information:

- File name (maximum of 13 eight-character names)
- File status
  - OPEN
  - CLSD (CLOSED)
  - IS INVALID

- File type
  - MRAM (MIRAM)
  - ISAM
  - DAMR
  - SAM
  - SAT
  - PRNT (batch printer file)
- Number of file accesses
- Number of file updates
- Total number of accesses and updates for all files.



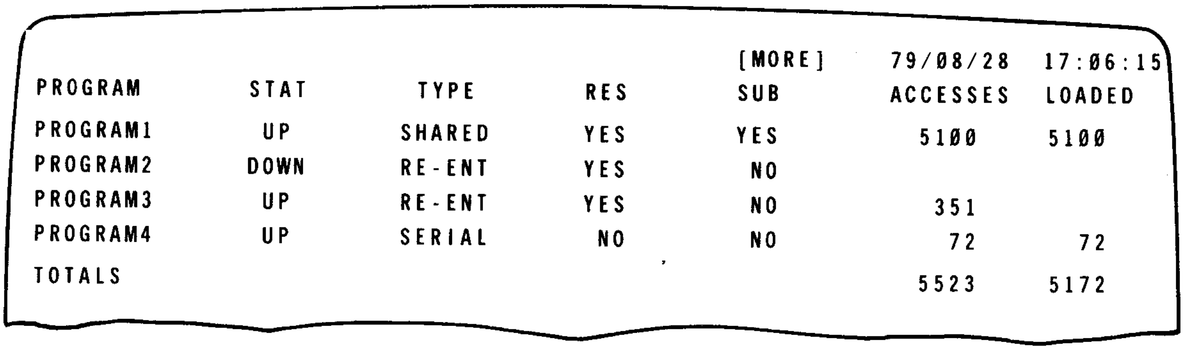
```
                                        [MORE]           79/Ø8/28          17:Ø5:32
 FILE              STAT           TYPE          ACCESSES             UPDATES

FILE1              OPEN           MRAM             15                      3
FILE2              CLSD            SAM            936                    772
FILE3              OPEN            SAM            842                    624
FILE4      IS     INVALID
FILE5              CLSD           ISAM           2Ø79                   1192
FILE6              CLSD           DAMR           1Ø8Ø                    293

TOTALS                                           4952                   2884
```

Figure 5—1. Sample File Status Display

The program status display gives the following information:

- Program name
- Program status
  - UP
  - DOWN
  - IS INVALID

- Program type
  - RE-ENT
  - SERIAL
  - SHARED (multithread only)
- Resident program
  - YES
  - NO
- Subprogram
  - YES
  - NO
- Number of program accesses
- Number of program loads
- Total number of accesses and loads for all programs

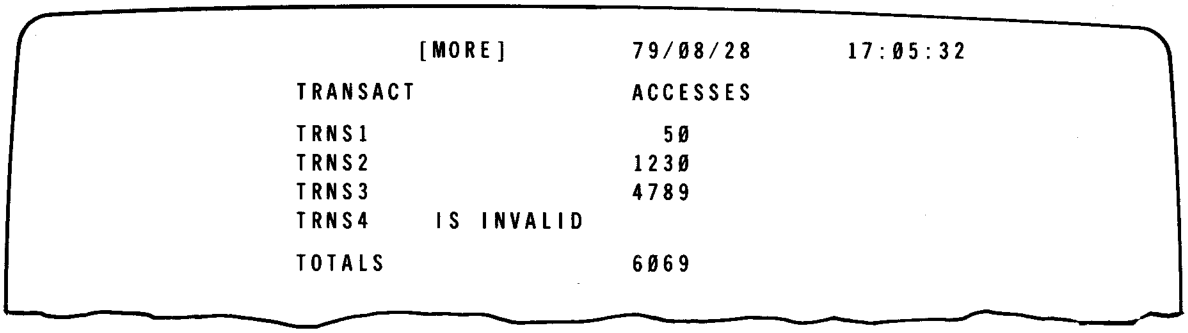


```
                                              [MORE]    79/Ø8/28   17:Ø6:15
PROGRAM      STAT     TYPE      RES           SUB       ACCESSES   LOADED
PROGRAM1     UP       SHARED    YES           YES        51ØØ      51ØØ
PROGRAM2     DOWN     RE-ENT    YES            NO
PROGRAM3     UP       RE-ENT    YES            NO         351
PROGRAM4     UP       SERIAL     NO            NO          72        72
TOTALS                                                   5523      5172
```

Figure 5—2. Sample Program Status Display

The transaction status display information includes:

- Transaction name
- Transaction status (IS INVALID)
- Number of transaction accesses
- Total number of accesses for all transactions

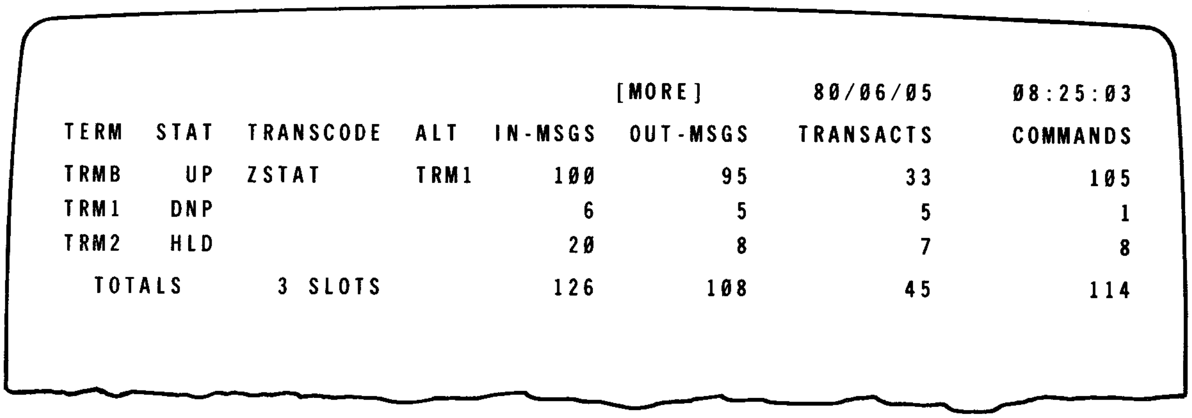


```
            [MORE]         79/Ø8/28       17:Ø5:32
TRANSACT                   ACCESSES
TRNS1                        5Ø
TRNS2                      123Ø
TRNS3                      4789
TRNS4     IS INVALID
TOTALS                     6Ø69
```

Figure 5—3. Sample Transaction Status Display

Terminal information in the terminal status display is:

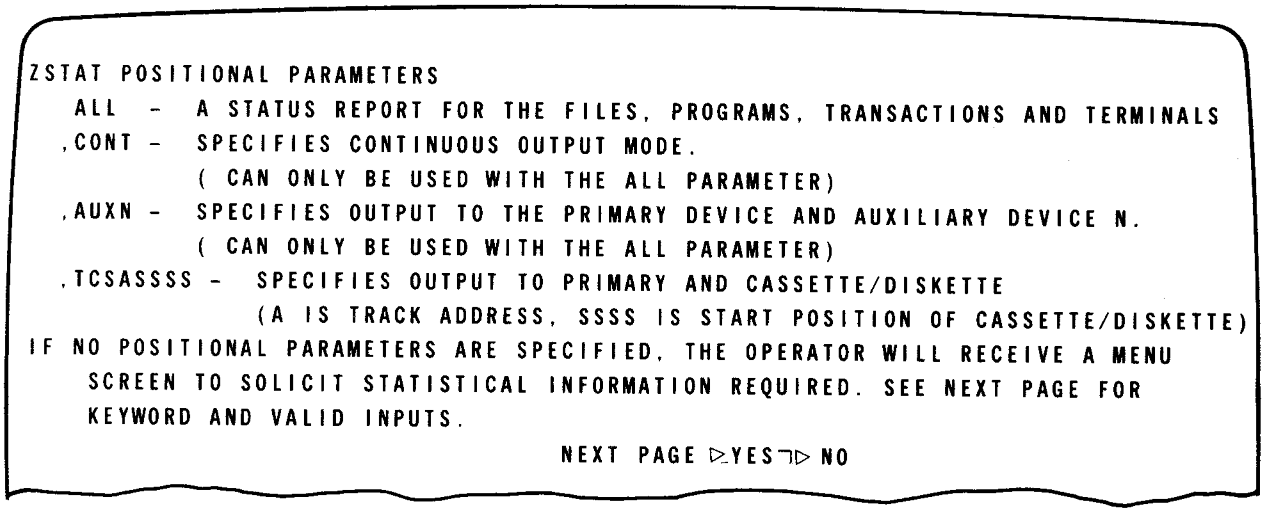
- Terminal ID
- Terminal status
  - DNP(physically down)
  - DNL(logically down)
  - HLD(hold mode)
  - TMD(test mode)
  - UP
- Current transaction code
- Alternate terminal identification (if any)
- Number of input messages processed by this terminal during the current session
- Number of output messages sent to this terminal during the current session
- Number of transactions executed by this terminal during the current session
- Number of terminal commands executed at this terminal during the current session
- Total number of slots available (for global networks only) or number of additional terminals that will be allowed to sign on via $$SON command
- Total number of input messages for all terminals with active sessions
- Total number of output messages for all terminals with active sessions
- Total number of transactions executed for all terminals with active sessions
- Total number of terminal commands executed for all terminals with active sessions.



```
                                         [MORE]       8Ø/Ø6/Ø5      Ø8:25:Ø3
TERM   STAT   TRANSCODE   ALT   IN-MSGS   OUT-MSGS   TRANSACTS     COMMANDS
TRMB     UP   ZSTAT       TRM1      1ØØ         95          33          1Ø5
TRM1    DNP                           6          5           5            1
TRM2    HLD                          2Ø          8           7            8
  TOTALS         3 SLOTS            126        1Ø8          45          114
```

Figure 5—4. Sample Terminal Status Display

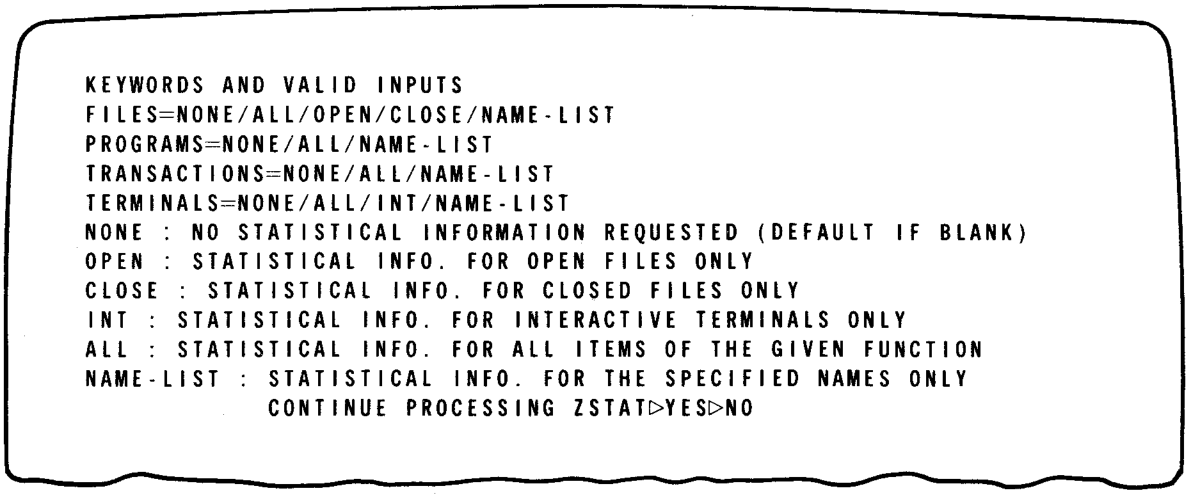
ZSTAT HELP keyed in at the terminal first displays the ZSTAT positional parameters and their meaning. (See Figure 5-5.)



```
ZSTAT POSITIONAL PARAMETERS
   ALL   -   A STATUS REPORT FOR THE FILES, PROGRAMS, TRANSACTIONS AND TERMINALS
  ,CONT  -   SPECIFIES CONTINUOUS OUTPUT MODE.
             ( CAN ONLY BE USED WITH THE ALL PARAMETER)
  ,AUXN  -   SPECIFIES OUTPUT TO THE PRIMARY DEVICE AND AUXILIARY DEVICE N.
             ( CAN ONLY BE USED WITH THE ALL PARAMETER)
  ,TCSASSSS -   SPECIFIES OUTPUT TO PRIMARY AND CASSETTE/DISKETTE
                (A IS TRACK ADDRESS, SSSS IS START POSITION OF CASSETTE/DISKETTE)
IF NO POSITIONAL PARAMETERS ARE SPECIFIED, THE OPERATOR WILL RECEIVE A MENU
    SCREEN TO SOLICIT STATISTICAL INFORMATION REQUIRED. SEE NEXT PAGE FOR
    KEYWORD AND VALID INPUTS.
                                     NEXT PAGE ⊵YES¬▷ NO
```

Figure 5—5. Sample ZSTAT HELP Output Screen Display, Page 1

To respond YES, press TRANSMIT. The second HELP screen then displays the choice of keyword parameter values available for each status display. (See Figure 5-6.) To respond NO, press TAB and then TRANSMIT.

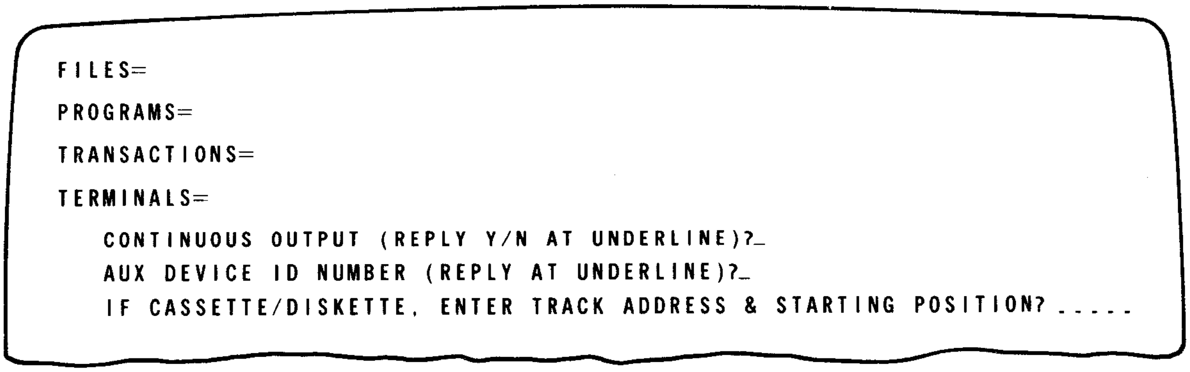


```
KEYWORDS AND VALID INPUTS
FILES=NONE/ALL/OPEN/CLOSE/NAME-LIST
PROGRAMS=NONE/ALL/NAME-LIST
TRANSACTIONS=NONE/ALL/NAME-LIST
TERMINALS=NONE/ALL/INT/NAME-LIST
NONE : NO STATISTICAL INFORMATION REQUESTED (DEFAULT IF BLANK)
OPEN : STATISTICAL INFO. FOR OPEN FILES ONLY
CLOSE : STATISTICAL INFO. FOR CLOSED FILES ONLY
INT : STATISTICAL INFO. FOR INTERACTIVE TERMINALS ONLY
ALL : STATISTICAL INFO. FOR ALL ITEMS OF THE GIVEN FUNCTION
NAME-LIST : STATISTICAL INFO. FOR THE SPECIFIED NAMES ONLY
            CONTINUE PROCESSING ZSTAT▷YES▷NO
```

Figure 5—6. Sample ZSTAT HELP Output Screen Display, Page 2

If you respond YES at the end of the second HELP screen, the menu screen is displayed. (See Figure 5-8.)

If ZSTAT is keyed in with no other parameters, a menu screen is displayed for soliciting further information. (See Figure 5-7.) This menu allows you to specify the parameters you want for files, programs, transactions, and terminals in addition to continuous output and auxiliary device number ID. When ready to enter the keyword parameters, continuous output reply, or auxiliary device ID number on the screen, you can arrive at the space following each equal size or question by pressing the tab key. Do not use the carriage return key (CR) or space bar when skipping from field to field. Instead, use the TAB key.



```
FILES=

PROGRAMS=

TRANSACTIONS=

TERMINALS=

   CONTINUOUS OUTPUT (REPLY Y/N AT UNDERLINE)?_
   AUX DEVICE ID NUMBER (REPLY AT UNDERLINE)?_
   IF CASSETTE/DISKETTE, ENTER TRACK ADDRESS & STARTING POSITION? .....
```

Figure 5—7. ZSTAT Menu Output Screen

You enter a parameter value for each item on the menu including a response to the continuous output and auxiliary device question. The parameter values for each item on the menu follow:

```
FILES=⎧NONE          ⎫
      ⎪OPEN          ⎪
      ⎨CLOSED        ⎬
      ⎪ALL           ⎪
      ⎩file-name-list⎭
PROGRAMS=⎧NONE             ⎫
         ⎨ALL              ⎬
         ⎩program-name-list⎭
TRANSACTIONS=⎧NONE              ⎫
             ⎨ALL               ⎬
             ⎩transact-name-list⎭
TERMINALS=⎧NONE          ⎫
          ⎪INT           ⎪
          ⎨ALL           ⎬
          ⎩term-name-list⎭
```

where:

NONE
Means no statistical information requested.

OPEN
Means information is requested for open files only.

CLOSED
Means information is requested for closed files only.

ALL
Means information for all files, programs, transaction, or terminals is requested.

name-list
Requests information for the files, programs, transactions, or terminals named in the list. A name cannot be split across two lines.

INT
Requests information for interactive terminals only.

Y
Means continuous output is requested.

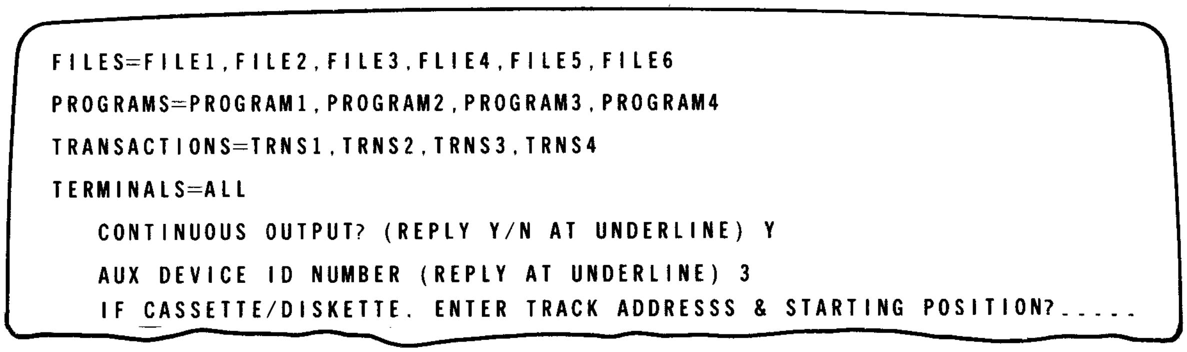
N
Means no continuous output is requested.

–
Means supply a 1-digit auxiliary device ID number.

Figure 5-8 illustrates a sample menu input screen. As a result of entering choices on the menu screen, you receive status displays similar to Figures 5-1 through 5-4. This ZSTAT format allows you to be more selective of status information. Also, note that when you make errors in spelling on the menu (e.g., FLIE4) you will receive an invalid message in your resulting status display. (See Figure 5-1.)

NOTE:

You may enter a maximum of 13 eight-character file names and 13 eight-character program names, 18 five-character transaction codes, and 22 four-character terminal names.



```
FILES=FILE1,FILE2,FILE3,FLIE4,FILE5,FILE6
PROGRAMS=PROGRAM1,PROGRAM2,PROGRAM3,PROGRAM4
TRANSACTIONS=TRNS1,TRNS2,TRNS3,TRNS4
TERMINALS=ALL
    CONTINUOUS OUTPUT? (REPLY Y/N AT UNDERLINE) Y
    AUX DEVICE ID NUMBER (REPLY AT UNDERLINE) 3
    IF CASSETTE/DISKETTE. ENTER TRACK ADDRESSS & STARTING POSITION?.....
```

Figure 5—8. Sample Menu Input Screen

5.3.4.1. Controlling the Terminal

When continuous output is used, the terminal operator usually does not have control of the terminal except in two cases:

1. when ICAM detects an error from the auxiliary device and returns an error delivery notice to ZSTAT and ZSTAT sends an error message to the primary device; or

2. when you press function key 1, 2, 3, or 4 during processing to interrupt continuous output of statistics

Table 5-4 indicates the functions performed when one of the function keys is pressed.

Table 5—4. Responses to Interruptions of ZSTAT

Master Terminal Entry	Function Performed
Transmit Key	Continue processing output messages.
Function Key 1*	Break processing and prompt terminal operator again.
Function Key 2*	Resume processing. ZSTAT reinitializes itself to its state before interruption and retransmits the previous message's output status message.
Function Key 3*	End of processing; ZSTAT terminates.
Function Key 4*	Resume processing at next function.
ZSTAT (with optional parameters, if any)	ZSTAT starts again.

*Key in F#01, F#02, F#03, or F#04 if no function key is available.

When you press function keys, IMS 90 accepts from ICAM either the function key code or the delivery notice (whichever comes first) and passes the first accepted code to ZSTAT via the IMA. Repeated use of the function key eventually produces the results needed by ZSTAT.

Any other replies to an interruption of continous output causes the message INVALID RESPONSE, PLEASE TRY AGAIN to be sent to the primary device.

These function key responses are also valid in noncontinuous output mode. When a user types in ZSTAT ALL with noncontinuous output, the only way the user can stop the ZSTAT transaction from giving all the status information is by using function key 3 to terminate or function key 4 to resume at the next function.

5.3.4.2. ZSTAT Error Messages

ZSTAT generates error messages when ICAM returns an error delivery notice. To determine which error message to issue, ZSTAT tests the specific ICAM delivery notice. If ZSTAT receives an unrecognizable delivery notice, a snap dump is generated in which the OMA contains the message UNRECOGNIZED DELIVERY NOTICE, and a transaction termination message is sent to the source terminal. If this problem occurs, you should contact your local Sperry Univac representative.

Tables 5-5 and 5-6 show ZSTAT recoverable and nonrecoverable error messages and explain their causes and responses.

Table 5—5. ZSTAT Recoverable Error Messages (Part 1 of 2)

ZSTAT Error Message	Explanation
READY GOOD STATUS BUT TERMINAL PRINTER WRITE FUNCTION INOPERATIVE REPLY F2 OR F4 (RESUME) OR F3 (END)	If F2* or F4* is entered, ZSTAT tries to resend the original message. If F3* is entered, ZSTAT terminates.
TERMINATE PRINTER OUT OF PAPER, INOPERATIVE, OR IN TEXT MODE REPLY F2 OR F4 (RESUME) OR F3 (END)	Same
DEVICE NOT RESPONDING - MAY BE DISCONNECTED REPLY F2 OR F4 (RESUME) OR F3 (END)	Same
DISK ERROR - REPLY F2 OR F4 (RESUME) OR F3 (END)	Same
NO ICAM NETWORK BUFFERS AVAILABLE - REPLY F2 OR F4 (RESUME) OR F3 (END)	If F2* or F4* is entered, ZSTAT tries 15 more times to output a terminal status message and then prompts the user again. If F3* is entered, ZSTAT terminates.
01 CONTINUOUS OUTPUT NOT CONFIGURED. DO YOU WISH TO CONTINUE USING NONCONTINUOUS OUTPUT (Y/N)?	The user requested continuous output mode but the CONT parameter was specified without the CONTOUT=YES parameter in the configurator OPTIONS section.

*F2, F3, and F4 are abbreviations for function key 2, 3, or 4; thus, a reply of F2, F3, or F4 means press function key 2, 3, or 4 at the terminal.

Table 5—5. ZSTAT Recoverable Error Messages (Part 2 of 2)

ZSTAT Error Message	Explanation
02 AN AUXILIARY DEVICE WAS REQUESTED AND NOT CONFIGURED. DO YOU WISH TO CONTINUE WITHOUT AN AUXILIARY DEVICE (REPLY Y/N)?	The user requested an auxiliary device without the CONTOUT=YES parameter in the configurator OPTIONS section.
03 AUX-DEVICE ID INVALID, PLEASE INPUT VALID ID ▷ ˥ -	The user keyed in an invalid auxiliary device ID. The correct 2-digit numeric auxiliary device ID must be entered.
04 INVALID TRACK ADDRESS. PLEASE INPUT VALID TRACK ADDRESS ▷ -	The track address for cassette/diskette is invalid. Enter the correct address.
05 INVALID STARTING POSITION. PLEASE INPUT VALID STARTING POSITION ▷ ----	The starting position for cassette/diskette is invalid. Enter a correct starting position.

Table 5—6. ZSTAT Unrecoverable Error Messages

ZSTAT Error Message	Explanation
NOT MASTER TERMINAL	ZSTAT was not issued from the master terminal.
INVALID INPUT PARAMETER	One of the ZSTAT positional parameters is incorrect in the ZSTAT ALL format. Auxiliary device number is invalid or CONTOUT=YES was not configured but was requested.
DATA ERROR ON TCS	An error was found while data was being sent to the cassette/diskette. Retry ZSTAT. If failure persists, check the condition of the cassette/diskette and associated hardware.
CASSETTE/DISKETTE NOT RESPONDING. ZSTAT TERMINATED	The cassette/diskette device was not powered up and in the ready state. The user should check the hardware device.

5.4. GLOBAL NETWORK TERMINAL COMMANDS

IMS 90 can interface ICAM global networks. After you configure IMS 90 and write the global network definition, IMS 90 at start-up issues ICAM macros to attach IMS 90 to the global network you described. Once this interface exists, without redialing, a terminal operator can attach his terminal to IMS 90. This terminal-to-IMS 90 interface is called a dynamic session. The terminal operator initiates a dynamic session by keying in:

```
$$SON terminal-idprogram-name
```

where:

terminal-id
: Is the terminal name specified on the label of the TERM macro in the network definition.

program-name

Is the name of the IMS 90 program specified on the label of the LOCAP macro in the network definition. No space is permitted between terminal-id and program-name.

Example:

```
$$SON TRM1IMS9
```

After issuing the $$SON command, the terminal operator receives the following responses:

```
SPERRY UNIVAC DCA NETWORK, LEVEL n.n NODE ID xxxx

SESSION PATH OPEN      (after session is initiated)

or

SESSION PATH CLOSED    (if the session is rejected)
```

SESSION PATH OPEN means that you have successfully signed onto IMS 90. Note that the SESSION PATH OPEN is used instead of the IMS READY message. SESSION PATH CLOSED means that the sign-on was unsuccessful. This could be because IMS 90 was not loaded into the system or IMS 90 itself did not have enough terminal resources to accept the sign-on command; the terminal tables within IMS 90 could not accommodate another terminal. These terminal resources are controlled by the TERMS keyword in the configurator NETWORK section.

To terminate the dynamic session and detach the terminal from the IMS 90, the terminal operator issues the following command:

```
$$SOFF
```

6. Transaction Processing via UNIQUE

6.1. UNIQUE CONCEPT

The uniform inquiry update element (UNIQUE) is a set of action programs provided by Sperry Univac that perform file retrieval and updating functions through the use of commands from the terminal.

UNIQUE accesses data files through defined record management (DRM). Thus, the user must define the data structure (files, records) and the allowable operations (retrieve, modify, add, delete) by submitting a data definition for each defined file to the data definition processor. (See Section 2.)

IMS 90 provides a password capability for UNIQUE users through both the data definitions processor and the PASSWORD function parameter of the NAMEREC file utility. Access to defined files and subfiles can be restricted to specific terminals by means of the NAMEREC utility which is described in detail in the IMS 90 system support functions user guide/programmer reference, UP-8364 (current version).

6.1.1. UNIQUE Dialog

A UNIQUE dialog consists of a series of commands and responses dealing with a particular defined file. Dialogs with different defined files can be combined to form a UNIQUE transaction.

The response to a UNIQUE command is based on:

- the command itself;
- additional text in the command message;
- the contents of the file being accessed; and
- the previous commands processed during this dialog.

The actual words used in a dialog comprise the UNIQUE lexicon. (See Appendix B.) The UNIQUE syntax is described in the examples supporting the command explanations.

UNIQUE uses the following commands:

- OPEN

  Initiates a dialog with a defined file. The initial OPEN command, or an OPEN command issued after a CLOSE command (or after the ZZCNC terminal command), functions as a transaction code and initiates a UNIQUE transaction. A subsequent OPEN command starts a new dialog with a different defined file but does not initiate a new transaction.

- CLOSE

  Terminates a transaction.

- DISPLAY

  Causes a specified record to be displayed at the terminal.

- LIST

  Displays certain records or parts of records at the terminal; can also display the results of statistical functions.

- DETAIL

  Interrupts the processing of the previous LIST command to obtain a different or more detailed listing.

- SHOW

  Displays the format of the defined file being accessed and the most recent LIST, DETAIL, ADD, DISPLAY, DELETE, and CHANGE commands.

- MORE

  Requests the next screenful of information according to the previous LIST, DETAIL, or SHOW command.

- ADD

  Enables the terminal operator to add a record to the file.

- CHANGE

  Enables the terminal operator to change values in a record.

- DELETE

  Displays a specified record, which can then be deleted by keying in the OK command.

- OK

  Completes the function indicated by the previous command - ADD, DELETE, or CHANGE.

- CANCEL

  Cancels the most recent (current) update command - ADD, DELETE, or CHANGE.

- NEXT

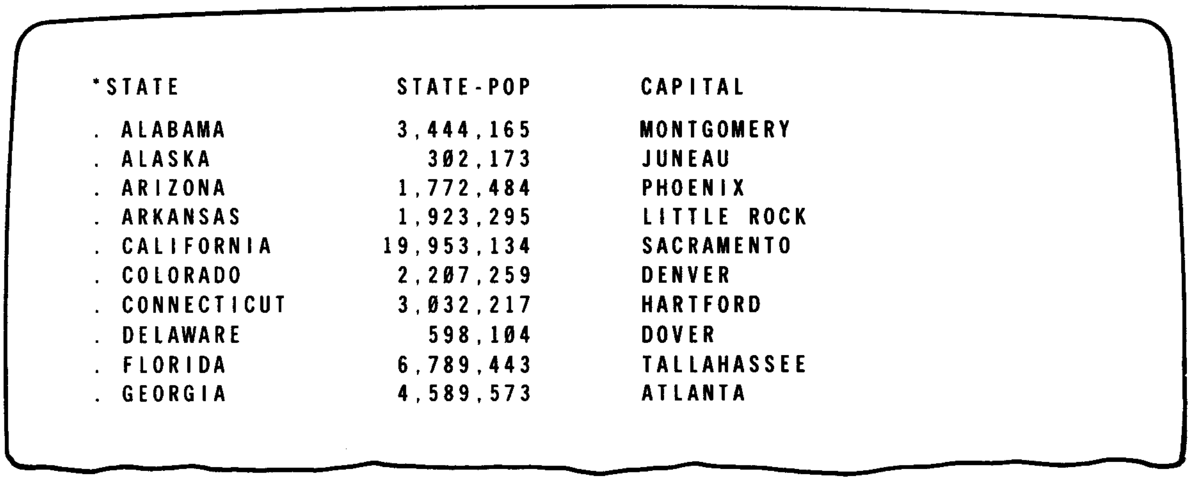
  Selects the next identifier from the most recent DISPLAY, ADD, CHANGE, or DELETE command and performs the same function.

6.1.2. Defined Files Accessed by UNIQUE

The defined files partially shown in Figures 6-1 and 6-2 are referenced in the discussion of UNIQUE commands that follows. The STATES file is a simple defined file; the TOWNS file is a hierarchical defined file.

The file listings presented in both illustrations correspond to the format used by UNIQUE when displaying records in response to the LIST command. The asterisk (*) indicates a header line containing names that serve as column headers. The period (.) indicates a detail line containing values stored in the file.

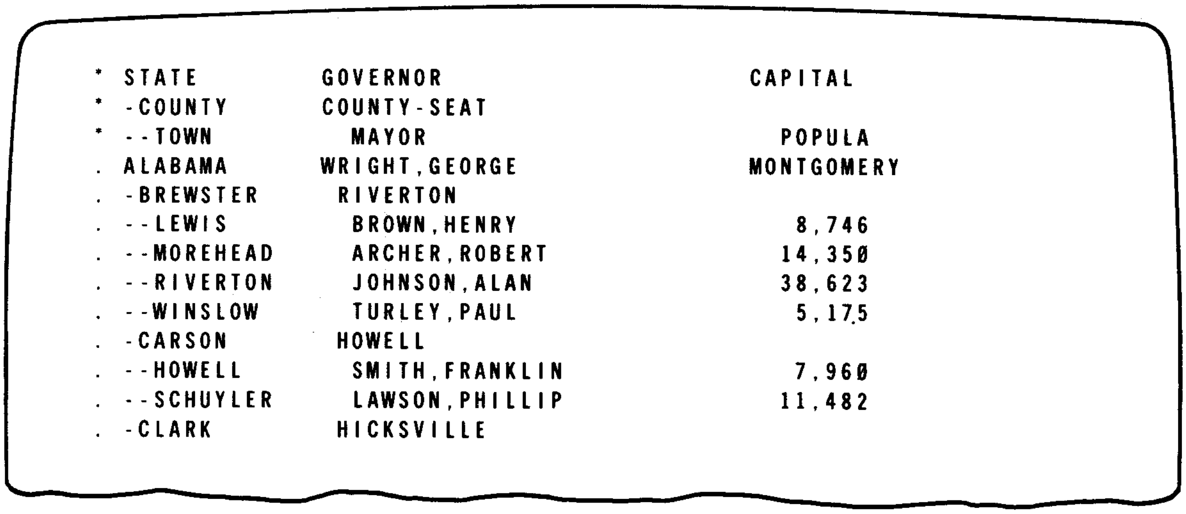
Each record in the STATES file contains the name of the state, the state population, and the capital city. The name of the state serves as the record identifier.



```
*STATE              STATE-POP       CAPITAL
. ALABAMA           3,444,165       MONTGOMERY
. ALASKA              3Ø2,173       JUNEAU
. ARIZONA           1,772,484       PHOENIX
. ARKANSAS          1,923,295       LITTLE ROCK
. CALIFORNIA       19,953,134       SACRAMENTO
. COLORADO          2,2Ø7,259       DENVER
. CONNECTICUT       3,Ø32,217       HARTFORD
. DELAWARE            598,1Ø4       DOVER
. FLORIDA           6,789,443       TALLAHASSEE
. GEORGIA           4,589,573       ATLANTA
```

Figure 6—1. Partial Listing of STATES File

Three types of records comprise the TOWNS file: state, county, and town records. Figure 6-2 represents these records as displayed in response to a LIST command. The state record contains the name of the state, the governor, and the capital city. For each state record, there are one or more county records for each county record, one or more town records. The county record contains the name of the county and the county seat. The town record contains the town name, the mayor's name, and the population.



```
* STATE        GOVERNOR                 CAPITAL
* -COUNTY      COUNTY-SEAT
* --TOWN         MAYOR                    POPULA
. ALABAMA      WRIGHT,GEORGE            MONTGOMERY
. -BREWSTER     RIVERTON
. --LEWIS        BROWN,HENRY                8,746
. --MOREHEAD     ARCHER,ROBERT             14,35Ø
. --RIVERTON     JOHNSON,ALAN              38,623
. --WINSLOW      TURLEY,PAUL                5,175
. -CARSON       HOWELL
. --HOWELL       SMITH,FRANKLIN             7,96Ø
. --SCHUYLER     LAWSON,PHILLIP            11,482
. -CLARK        HICKSVILLE
```

Figure 6—2. Partial Listing of TOWNS File

Each record in a file has an identifier. Records are ordered on these identifiers. The identifier of the first record in the STATES file (Figure 6-1) is ALABAMA, the second is ALASKA, etc. The first record in the TOWNS file (Figure 6-2) is a state record and its identifier is ALABAMA. The identifier of the second record (the first county record) is ALABAMA, BREWSTER; the third, a town record, is ALABAMA, BREWSTER, LEWIS.

Notice in the county records shown in Figure 6-2 (BREWSTER, CARSON, CLARK) that the ALABAMA is replaced by a hyphen (-). This serves to reduce the amount of space occupied on the screen. Similarly, the ALABAMA, BREWSTER part of each town record's identifier is replaced by two hyphens (- -).

→ Note that when UNIQUE adds or changes a defined record, it sets the sign of any new value for a packed decimal item to hexadecimal F instead of C.

6.2. UNIQUE COMMANDS

The data manipulating commands provided by UNIQUE, with examples of their use, are described in 6.2.1 through 6.2.13. The examples intentionally follow in succession as in a stream of commands and are numbered consecutively throughout the discussion. In order to highlight each new input or output message in the examples, previously issued commands and responses remaining on the screen are shaded.

The format for most commands is the same whether the input is on a display device or a hard copy device. The examples given with these commands are for a display device. The formats for the ADD and CHANGE commands, however, vary for display and hard copy devices. Both formats, their parameters, and examples of their use are provided for these commands.

When keying in UNIQUE commands at the terminal, the following general rules apply. Specific rules for individual commands are noted where applicable.

1. Parameters are separated from the UNIQUE command and from each other by spaces.

2. In identifiers, values, and specifications, alphanumeric item-names and literals containing blanks or special characters must be enclosed by apostrophes. These special characters include comma (,), semicolon (;), leading hyphen (-), and apostrophe ('). By convention, the apostrophe is represented by a pair of apostrophes if it is used as a character in an item. For example, the value O'Brien would be entered as 'O''Brien'.

3. Numeric literals must not be enclosed by apostrophes.

4. Decimal points and commas must appear where appropriate in numeric values.

6.2.1. OPEN

The OPEN command denotes the beginning of a UNIQUE dialog and designates the password of the file that the user intends to query or update during this dialog. This command can be issued at any time during a transaction to access a different file. Issuing another OPEN command during a transaction does not initiate a new transaction but starts a new dialog with a different file.

The OPEN command and one other UNIQUE command can be transmitted together to improve throughput. (See example 2.)

Format:

```
OPEN password
```

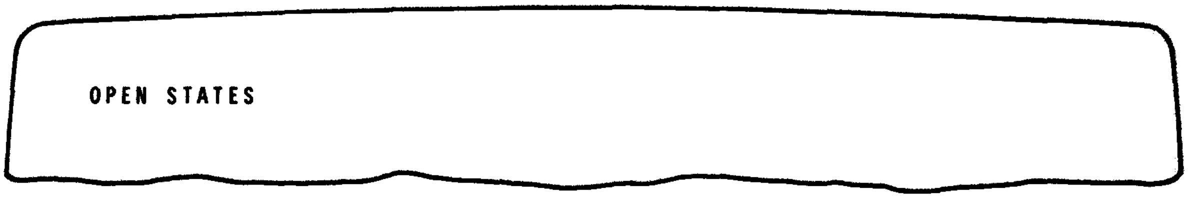
where:

```
password
```

Is an alias for a defined file or is the actual defined file or subfile name defined by the data definition processor.

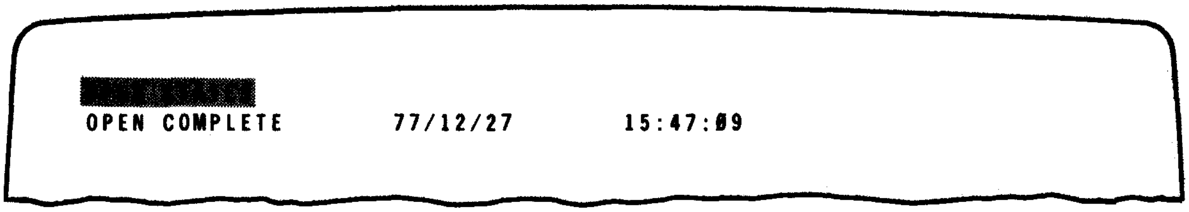
Example 1:

Input



```
OPEN STATES
```

Output

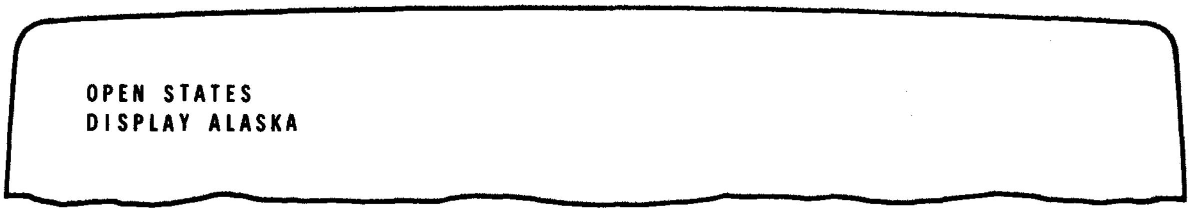


```
[redacted]
OPEN COMPLETE          77/12/27          15:47:Ø9
```

The OPEN command initiates the transaction and makes the STATES file accessible to the terminal operator.

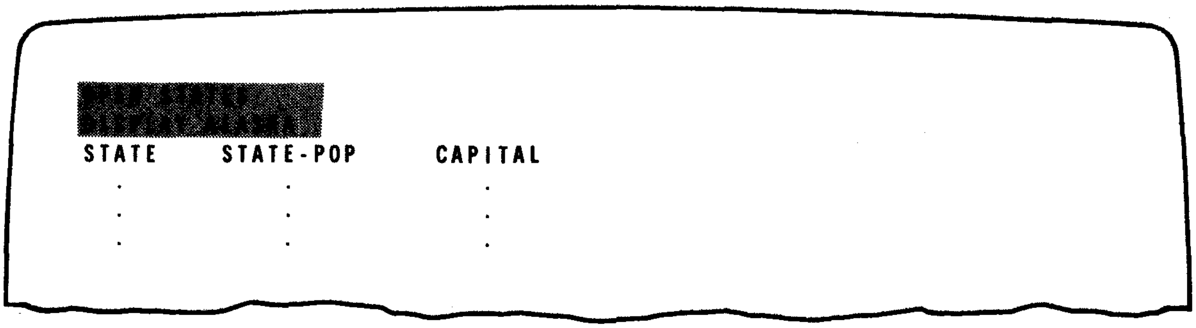
Example 2:

Input



```
OPEN STATES
DISPLAY ALASKA
```

Output



```
[redacted]
STATE     STATE-POP     CAPITAL
  .           .            .
  .           .            .
  .           .            .
```

In this example, the OPEN command transmitted opens the STATES file and displays the first state record (ALASKA). The OPEN COMPLETE message is not sent to the terminal.

6.2.2. CLOSE

The CLOSE command terminates the UNIQUE transaction. No UNIQUE commands are then recognized until another OPEN command is issued.

Format:

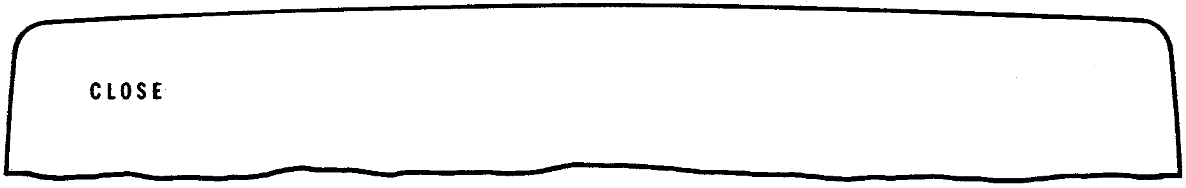
```
CLOSE
```

NOTE:

The CLOSE command is not required to terminate a dialog. Issuing another OPEN command terminates access to the file, then immediately provides access to the new file.

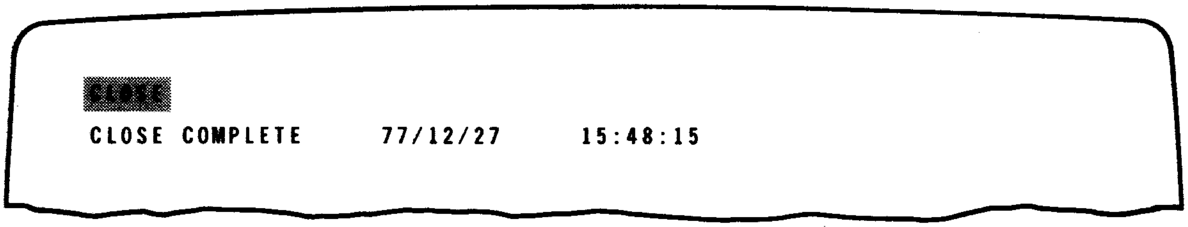
Example:

Input



```
CLOSE
```

Output



```
CLOSE
CLOSE COMPLETE      77/12/27      15:48:15
```

6.2.3. DISPLAY

The DISPLAY command causes the specified record to be displayed at the terminal. Column headings, as well as values, are displayed. The column headings are obtained from the IDENTIFIER and ITEM statements of the data definition.

Format:

```
DISPLAY identifier-1 [;identifier-2]...
```

where:

`identifier-1`

Indicates a specific record. Hyphens (--) are used as ditto characters when portions of an identifier are repeated in a subsequent identifier or a subsequent command. (See example 2.)

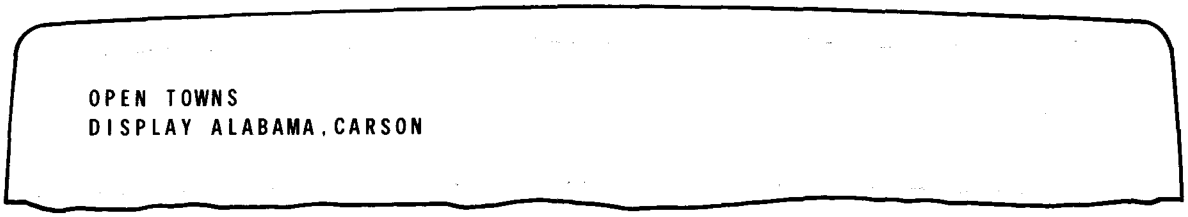
`identifier-2`

Designates another record. This record is not displayed, however, unless the NEXT command follows this DISPLAY command.

In response to a DISPLAY command, UNIQUE displays the record specified by the first identifier. If more identifiers are specified in the DISPLAY command, the records corresponding to the other identifiers are displayed, one at a time, by using the NEXT command. The DISPLAY command and ensuing NEXT commands can be mixed with other, intervening commands that do not alter the effect of the NEXT command. Other commands that can appear prior to the NEXT command are LIST, MORE, DETAIL, or SHOW.

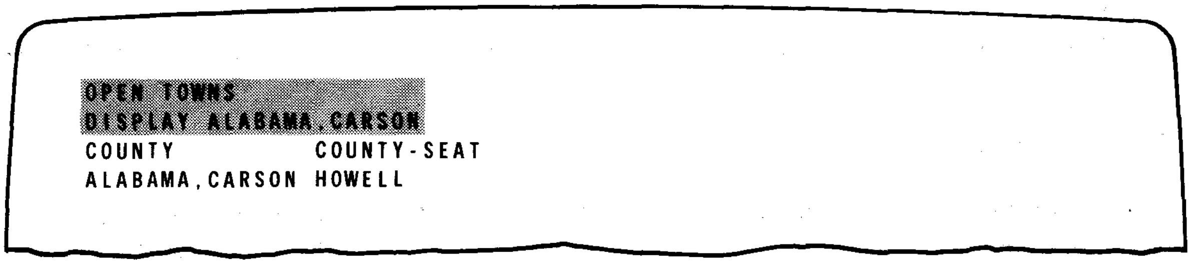
Example 1:

Input



```
OPEN TOWNS
DISPLAY ALABAMA,CARSON
```

Output

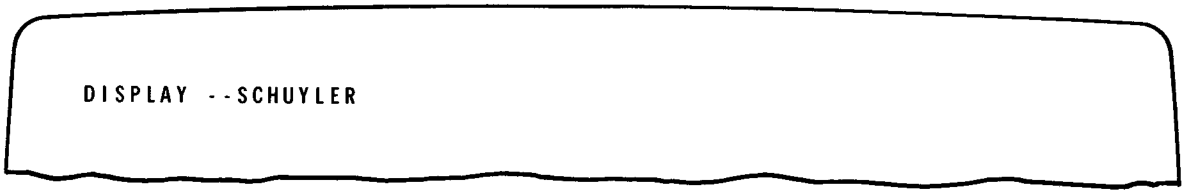


```
OPEN TOWNS
DISPLAY ALABAMA,CARSON
COUNTY          COUNTY-SEAT
ALABAMA,CARSON HOWELL
```

The DISPLAY command presents the data from the record identified by ALABAMA,CARSON in the TOWNS file.

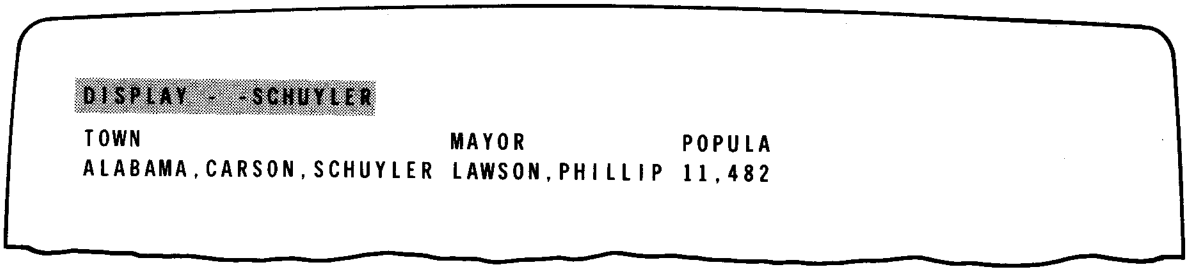
Example 2:

Input



```
DISPLAY --SCHUYLER
```

Output



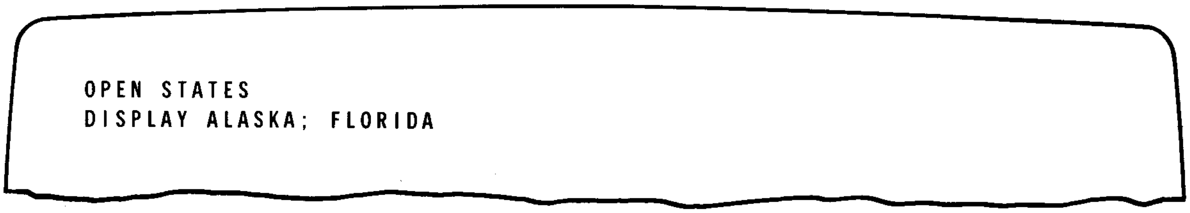
```
DISPLAY - -SCHUYLER
TOWN                     MAYOR           POPULA
ALABAMA,CARSON,SCHUYLER LAWSON,PHILLIP 11,482
```

The terminal operator uses the ditto characters (--) to indicate that ALABAMA,CARSON from the previous DISPLAY should be used for the first part of the town identifier.

It is not necessary to issue another OPEN for the TOWNS file. The OPEN from example 1 is still in effect.

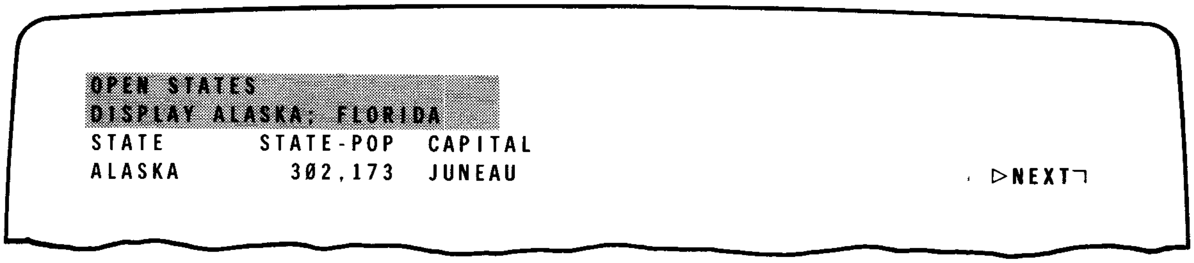
Example 3:

Input



```
OPEN STATES
DISPLAY ALASKA; FLORIDA
```

Output



```
OPEN STATES
DISPLAY ALASKA; FLORIDA
STATE      STATE-POP  CAPITAL
ALASKA       3Ø2,173  JUNEAU                              ▷NEXT⌝
```

The OPEN causes UNIQUE to close the TOWNS file and allow access to the STATES file. The DISPLAY command requests UNIQUE to display the information from the record whose identifier is ALASKA. The embedded NEXT command is the result of multiple identifiers in the DISPLAY command. To display the record associated with the next identifier, FLORIDA, the operator presses the TRANSMIT key. If the embedded NEXT command is not transmitted at this time, it can be specified later, as long as a DELETE, ADD, or CHANGE command (or another DISPLAY) does not intervene.

6.2.4. NEXT

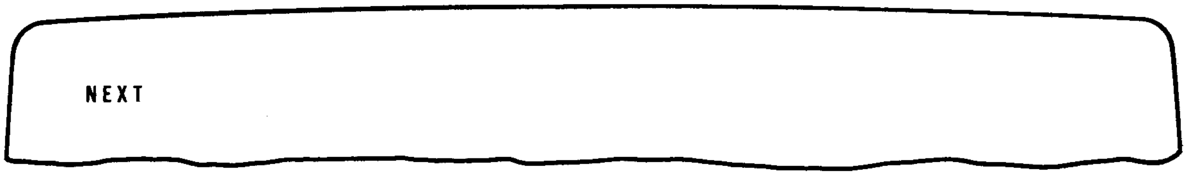
The NEXT command selects the next identifier from the most recent DISPLAY, DELETE, ADD, or CHANGE command. The function performed on the identified record is determined by that previous command.

Format:

```
NEXT
```

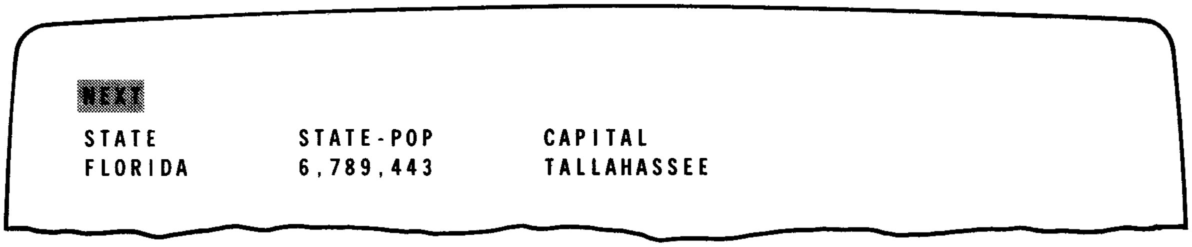
Example 1:

Input



```
NEXT
```

Output

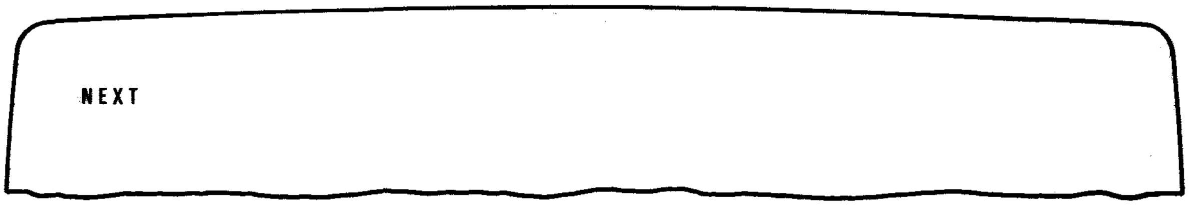


```
NEXT
STATE          STATE-POP        CAPITAL
FLORIDA        6,789,443        TALLAHASSEE
```

The preceding command was DISPLAY ALASKA; FLORIDA. Since the ALASKA record already has been displayed, NEXT requests the display of the FLORIDA record.

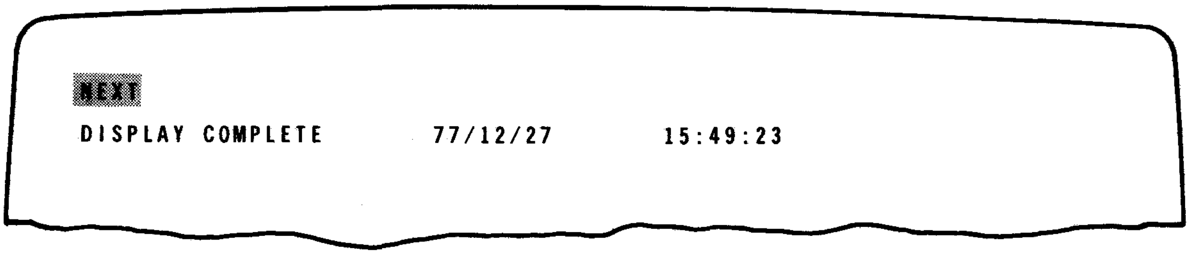
Example 2:

Input



```
NEXT
```

Output



```
NEXT
DISPLAY COMPLETE        77/12/27        15:49:23
```

There are no more identifiers outstanding from the previous DISPLAY command; therefore, the DISPLAY COMPLETE message is sent.

6.2.5. DELETE

The DELETE command gives the terminal operator the ability to delete a record after viewing the information in the record. UNIQUE displays the specified record in response to the DELETE command. To effect the deletion, the terminal operator must key in the OK command. If he decides not to delete the record, he should key in the CANCEL command.

Format:

```
DELETE identifier-1 [;identifier-2]...
```

where:

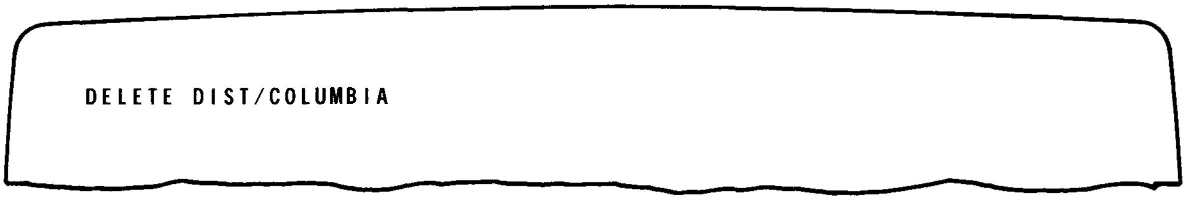
`identifier-1`
Designates a specific record. Ditto characters (--) can replace portions of the identifier.

`identifier-2`
Designates another record in the file. This record is displayed when the NEXT command is specified.

The DELETE command places the terminal in an update state. While the terminal is in this state, no other UNIQUE commands are accepted except OK or CANCEL. The following example illustrates.

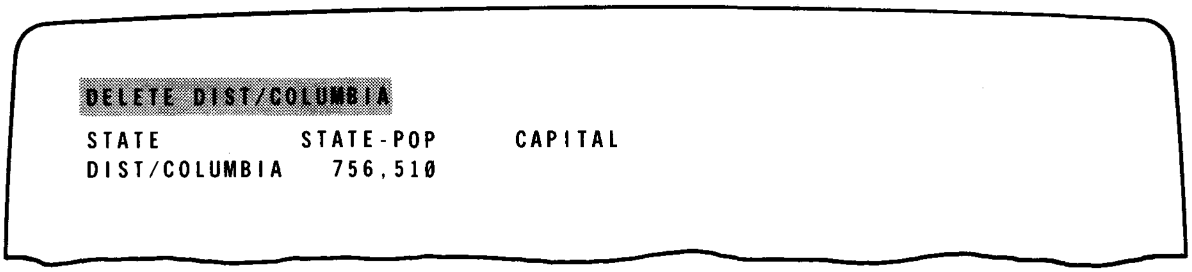
Example:

Input



```
DELETE DIST/COLUMBIA
```

Output



```
DELETE DIST/COLUMBIA
STATE           STATE-POP      CAPITAL
DIST/COLUMBIA     756,510
```

The item CAPITAL for this record has no value.

6.2.6. OK

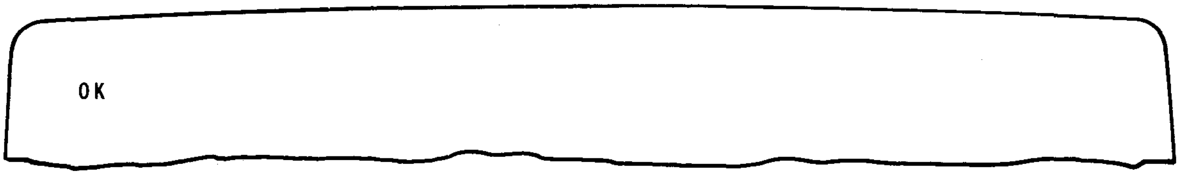
The OK command executes the function associated with the previous command. It is used only with the DELETE, ADD, or CHANGE command. Upon successful completion of that function, UNIQUE establishes a new rollback point for the current transaction.

Format:

```
OK
```

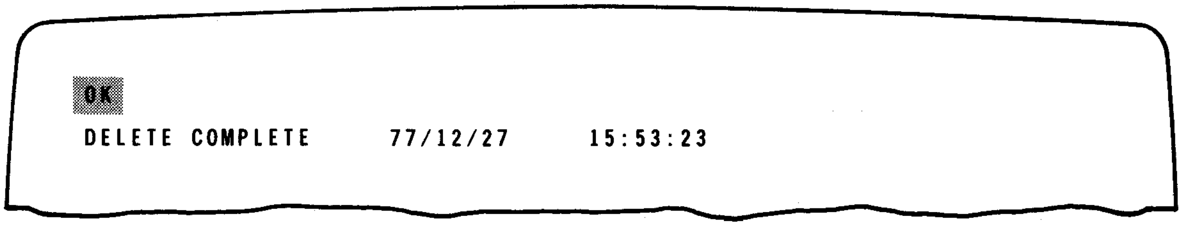
Example:

Input



```
OK
```

Output



```
OK
DELETE COMPLETE      77/12/27      15:53:23
```

Because the previous command was DELETE, the OK command requests UNIQUE to perform the deletion of the DIST/COLUMBIA record.

6.2.7. CANCEL

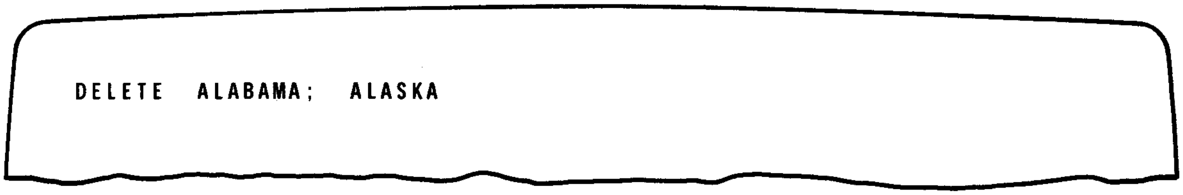
The CANCEL command requests UNIQUE to cancel the previous update command (DELETE, ADD, or CHANGE); thus, the file remains as it was before that update command.

Format:

```
CANCEL
```

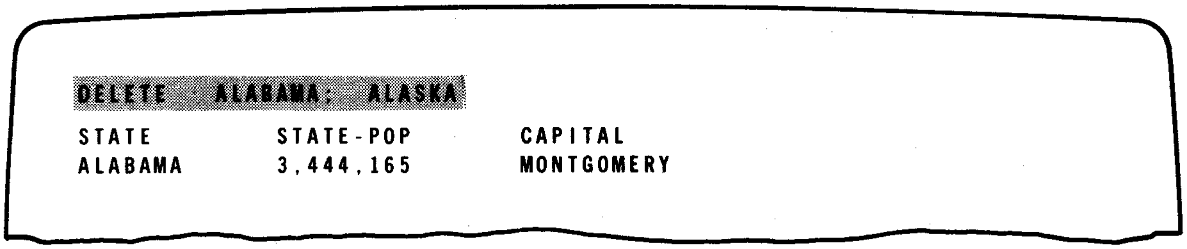
Example:

Input



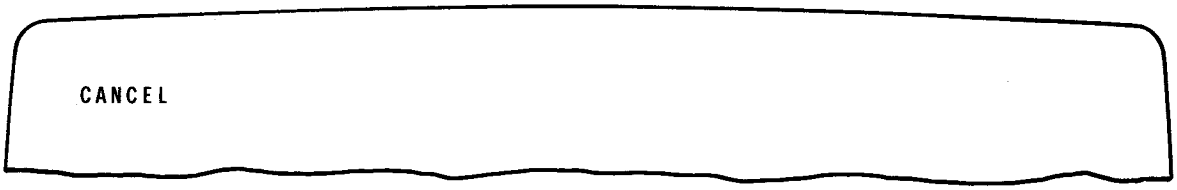
```
DELETE  ALABAMA;  ALASKA
```

Output



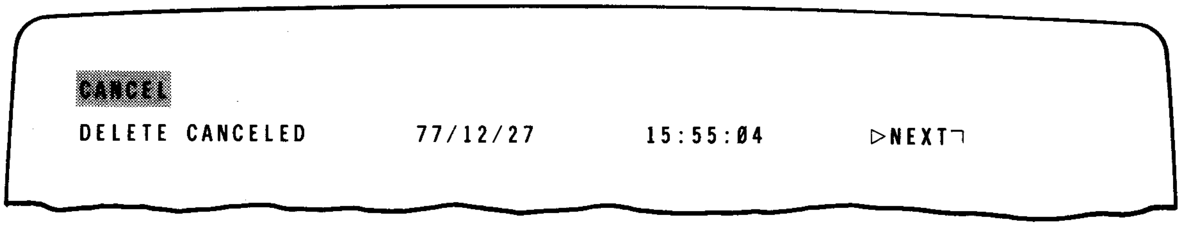
```
DELETE  ALABAMA;  ALASKA
STATE          STATE-POP       CAPITAL
ALABAMA        3,444,165       MONTGOMERY
```

Input



```
CANCEL
```

Output



```
CANCEL
DELETE CANCELED        77/12/27        15:55:Ø4        ▷NEXT⌝
```

This sequence of inputs and outputs results in no change to the file. The terminal operator now can enter NEXT to initiate the delete sequence for the ALASKA record.

6.2.8. ADD

The ADD command initiates a series of inputs and UNIQUE responses by which a record can be added to the file. There are two formats for this command – the display format and the hard copy format.

When using the IBM 3270 display station under UNIQUE, your command input or responses to UNIQUE messages must be in hard copy format rather than display format. The reason for this is the IBM 3270 display station transmits a whole screen of data to the UNIQUE program instead of a data field limited by a start-of-entry (▷) symbol and the cursor.

Upon successful completion of the ADD function, UNIQUE establishes a new rollback point for the current transaction.

The ADD command places the terminal in an update state. While the terminal is in this state, no UNIQUE commands are accepted except OK or CANCEL.

6.2.8.1. Display Format

Format:

```
ADD identifier-1[;identifier-2]...
```

where:

`identifier-1`
Is the record designation. Ditto characters (--) can replace portions of the identifier.

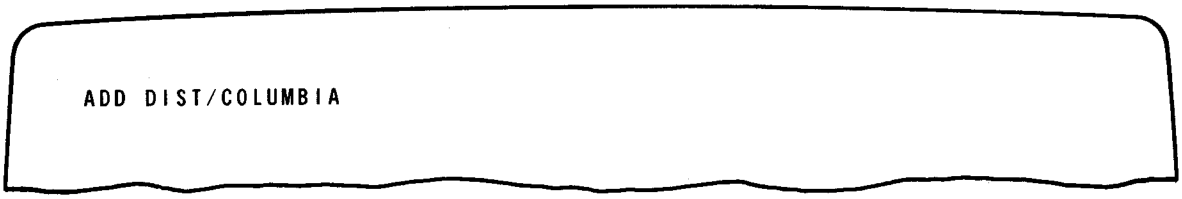
identifier-2
: Is another record designation. The update format for this record is displayed when the NEXT command is entered.

In response to the ADD command, UNIQUE sends to the terminal the item names (column headers) and update formats of the items associated with the specified record. The terminal operator can depress the tab key to advance the cursor to the beginning of each update format. (Note that an extra character position is provided for the sign of numeric items.) The terminal operator overwrites the update formats with values and then transmits the modified screen. If UNIQUE finds no errors, it adds the new record to the file and sends an ADD COMPLETE message to the terminal. If errors are discovered, UNIQUE sends the column headers, update formats, and either valid values or error notations back to the terminal for further modification by the terminal operator.

At any time during this sequence, the terminal operator can key in the CANCEL command instead of overwriting the update formats with values. The ADD command is then canceled.

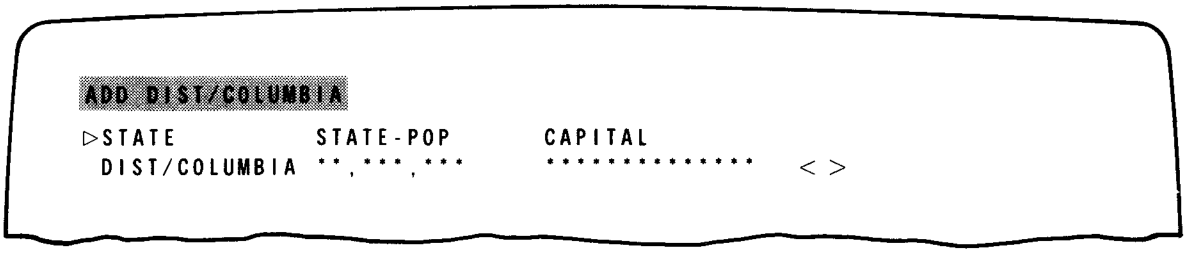
Example 1:

Input



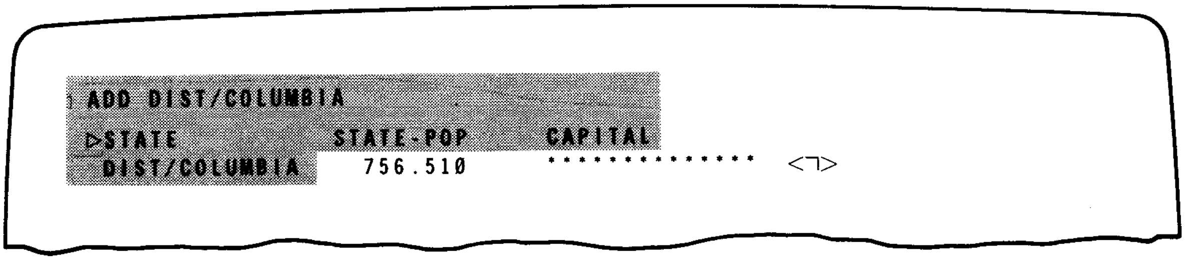
```
ADD DIST/COLUMBIA
```

Output



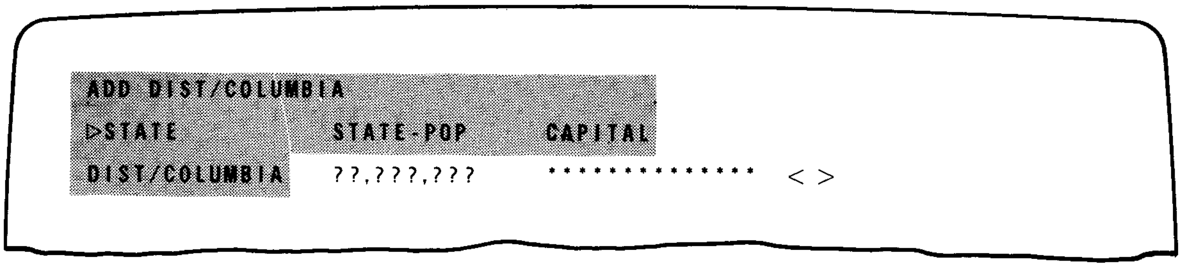
```
ADD DIST/COLUMBIA
▷STATE          STATE-POP      CAPITAL
 DIST/COLUMBIA  **,***,***     **************    < >
```

Input



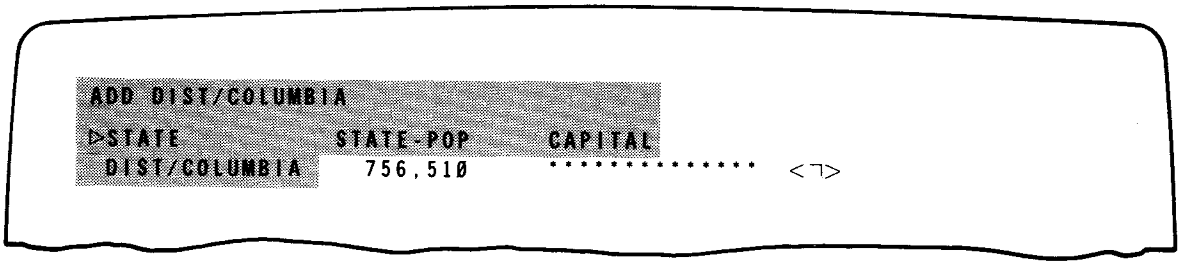
```
ADD DIST/COLUMBIA
▷STATE          STATE-POP      CAPITAL
 DIST/COLUMBIA    756.510      **************   <⌐>
```

Output



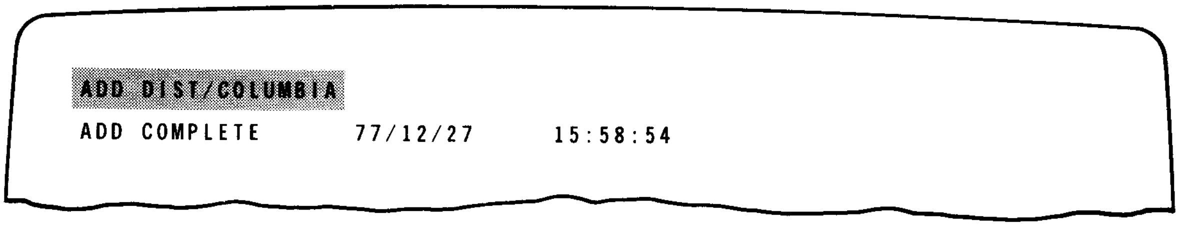
```
ADD DIST/COLUMBIA
▷STATE              STATE-POP      CAPITAL
DIST/COLUMBIA       ??,???,???     **************   < >
```

Input



```
ADD DIST/COLUMBIA
▷STATE              STATE-POP      CAPITAL
 DIST/COLUMBIA        756,51Ø      **************   <⌉>
```

Output



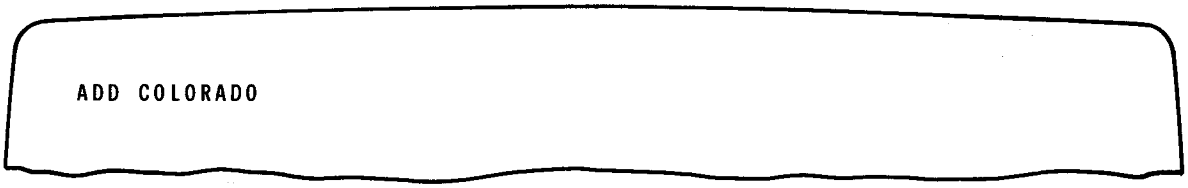
```
ADD DIST/COLUMBIA
ADD COMPLETE        77/12/27       15:58:54
```

The asterisks (*) under STATE-POP and under CAPITAL constitute the update formats. The characters (< >) at the end of the output aid the terminal operator in placing the cursor in subsequent input; that is, the operator should place the cursor between the two special characters (< >) after overwriting the desired items. This ensures that the entire message is transmitted.

The question marks (?) in the second output tell the operator that the value keyed in for STATE-POP is invalid (the period should be a comma). After the operator corrects the item in the third input, UNIQUE adds the record to the file and notifies the operator in the third output message.

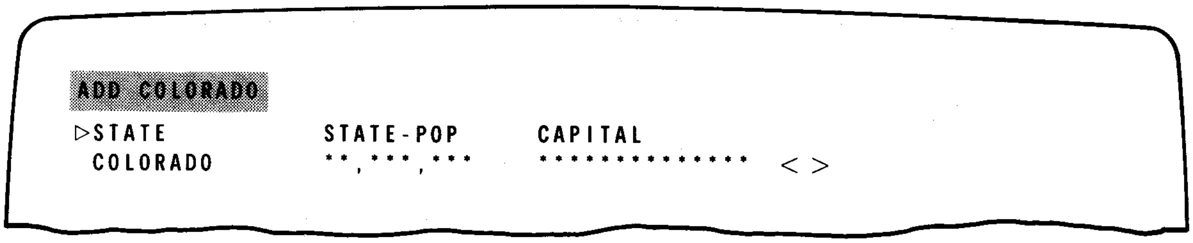
Example 2:

Input



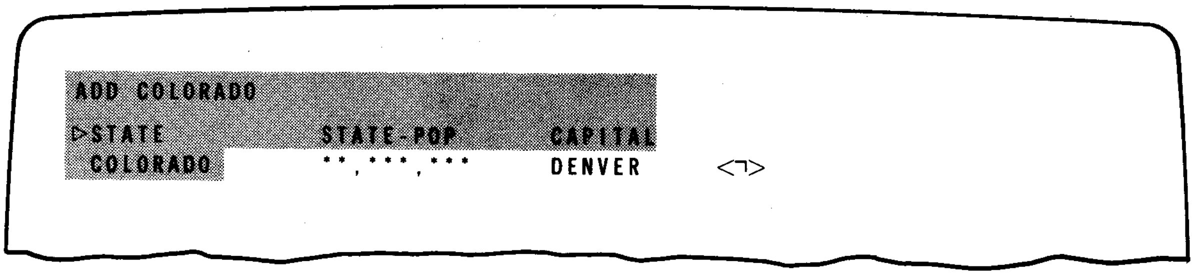
```
ADD COLORADO
```

Output



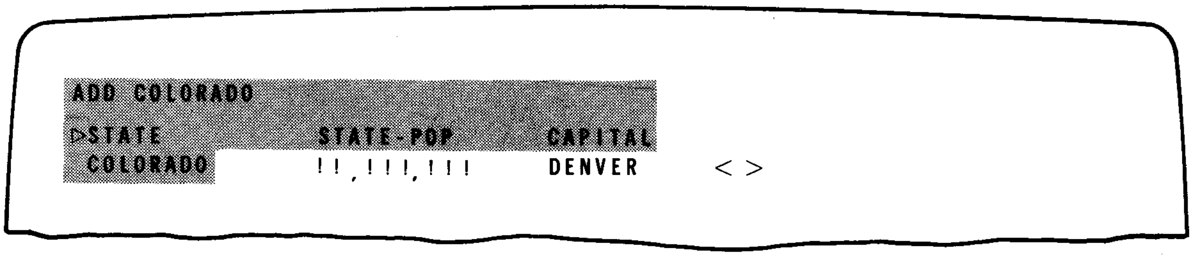
```
ADD COLORADO
▷STATE          STATE-POP      CAPITAL
 COLORADO       **,***,***     **************   < >
```

Input



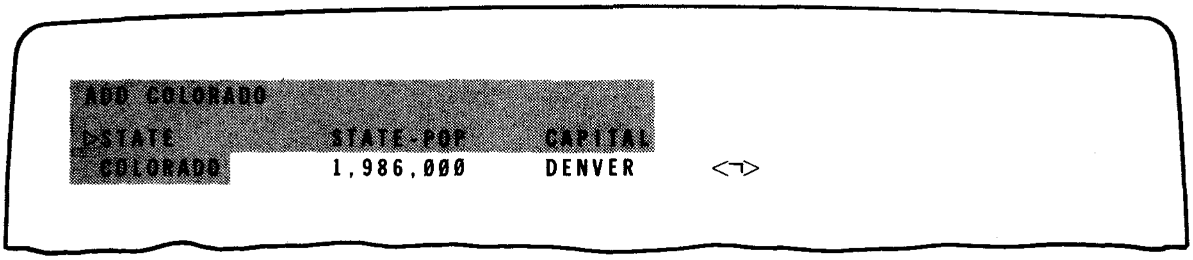
```
ADD COLORADO
▷STATE          STATE-POP        CAPITAL
 COLORADO       **,***,***       DENVER       <¬>
```

Output



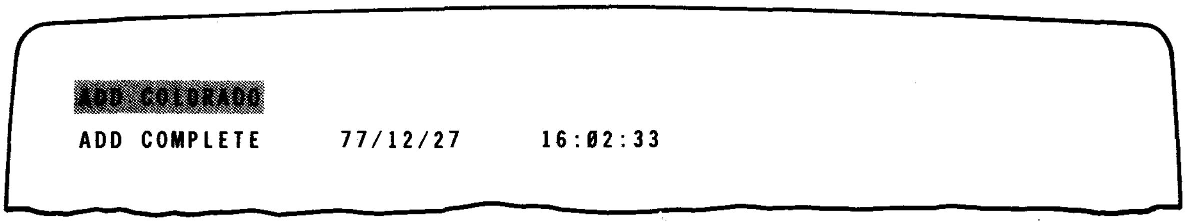
```
ADD COLORADO
▷STATE          STATE-POP        CAPITAL
 COLORADO       !!,!!!,!!!       DENVER       < >
```

Input



```
ADD COLORADO
▷STATE          STATE-POP       CAPITAL
 COLORADO       1,986,ØØØ       DENVER      <¬>
```

Output



```
ADD COLORADO
ADD COMPLETE       77/12/27       16:Ø2:33
```

Example 2 demonstrates an ADD sequence in which the terminal operator attempts to add a state record without giving a value to STATE-POP. The exclamation points in the second output indicate that the STATE-POP item was defined as a MUST ADD item in the data definition for this file. The operator must supply a value for STATE-POP before this record can be added.

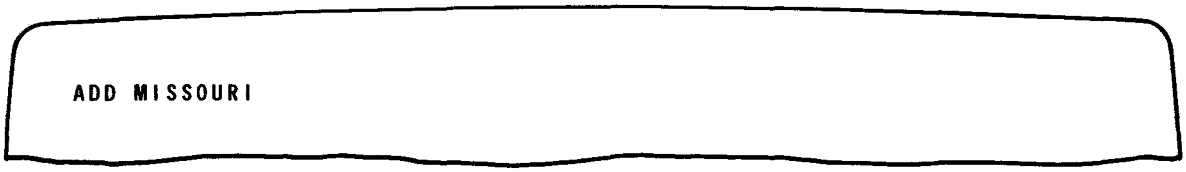
NOTE:

If a numeric item to be added begins with a decimal point, the terminal operator must key in the decimal point, even if the new value is zero.

If the operator enters the OK command during an ADD sequence, UNIQUE adds the designated record with all the valid items. Any invalid items except those identified as MUST ADD in the data definition. The ADD command is canceled if the MUST ADD items are not valid. (See example 3.)

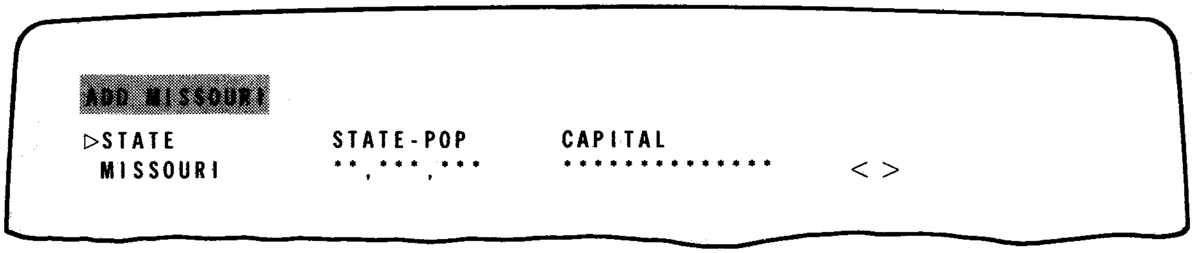
Example 3:

Input



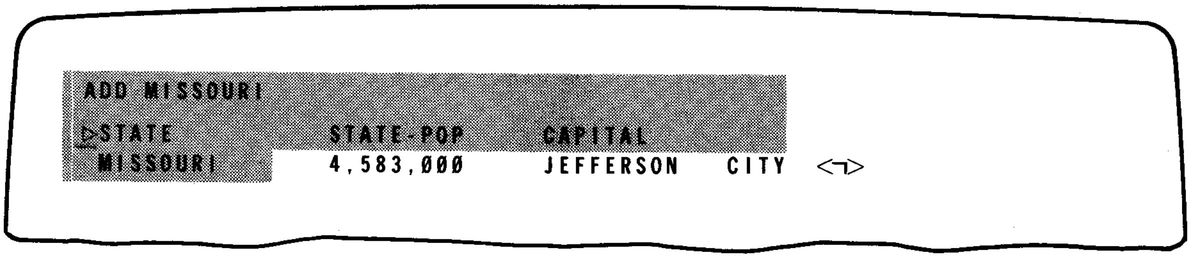
```
ADD MISSOURI
```

Output



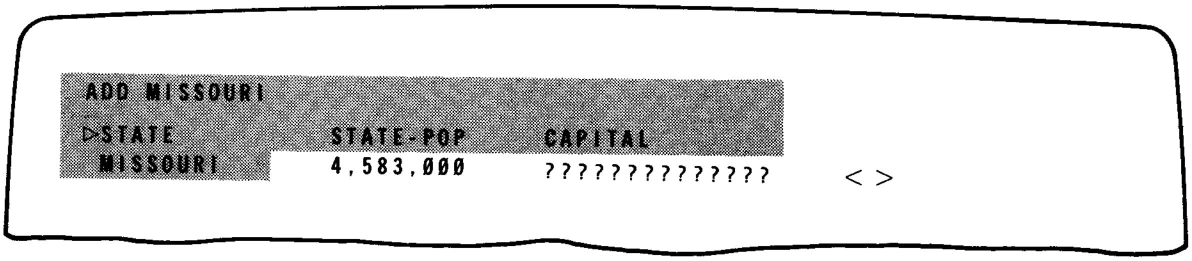
```
ADD MISSOURI
▷STATE          STATE-POP     CAPITAL
 MISSOURI       **,***,***    **************      < >
```

Input



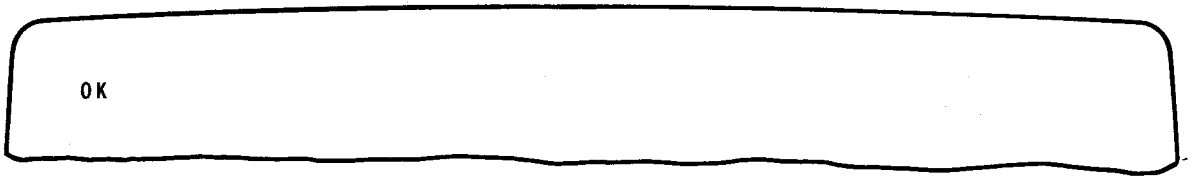
```
ADD MISSOURI
▷STATE            STATE-POP      CAPITAL
  MISSOURI        4,583,ØØØ      JEFFERSON   CITY  <¬>
```

Output



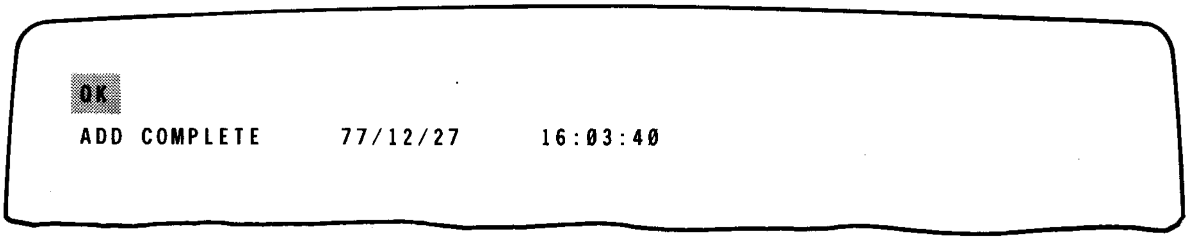
```
ADD MISSOURI
▷STATE            STATE-POP      CAPITAL
  MISSOURI        4,583,ØØØ      ??????????????      < >
```

Input



```
OK
```

Output



```
OK
ADD COMPLETE      77/12/27       16:Ø3:4Ø
```

This example demonstrates the use of the OK command with the display format ADD command. The OK command (input 3) causes UNIQUE to add the MISSOURI record, which supplies values to the valid items. In this case, the STATE-POP item contains the value 4,583,000, and the CAPITAL item contains the null value.

6.2.8.2. Hard Copy Format

Basic Format:

```
ADD identifier item-name=value[;[item-name=]value]...
```

General Format:

```
ADD identifier[;]...[item-name=]value[[;]...,[item-name=]value]...
```

NOTE:

The general format is an expanded form of the basic format.

where:

`identifier`
Is the record designation. (See the DISPLAY command, 6.2.3.)

`item-name`
Names an item. An item name can be any name, other than an identifier name, that appears as a column header in a DISPLAY command. The equal sign must be coded immediately after each item name. The item names need not be coded in the order of their appearance in the display record.

Item names can be omitted, in which case they are inferred from the position of the value in the input message. The positions of values are determined from the last specified item name, forward in a consecutive manner. If no item names are specified, positional locations are assumed from the first item after the identifier. Any consecutive item that is not to be updated can be omitted by coding a semicolon (;) in its positional location.

`value`
Is the value to be stored in item-name. All values, except the last, must be followed immediately by a semicolon (;).

In response to this command, UNIQUE displays the item names and values specified in the command, along with the identifier of the record. If the terminal operator is satisfied with the new record, he keys in OK, thus effecting the addition of this record to the file. If he decides not to add the record, he keys in the CANCEL command.

In response to the hard copy format ADD command, UNIQUE also displays error notations when invalid item names or conditions are discovered. It displays invalid item names showing location in the input message, the invalid name, and the associated values. UNIQUE formats and displays all acceptable input for visual verification. The terminal operator can then decide to cancel the command or to correct the errors.

The operator makes corrections by entering a new input message with either the item name and value (for example, NAME-5=VALUE-5), or just the value (for examples, ;;;;VALUE-5) that is to be respecified. The operator does not enter the ADD command again when he is respecifying. UNIQUE responds by displaying all the original output, with the respecified values replacing the values in error. The terminal operator then keys in OK to effect the update, or CANCEL to cancel the entire ADD command.

NOTE:

The hard copy format may also be used with display devices.

Example 1:

```
Input      ADD DIST/COLUMBIA STATE-POP=756.51Ø
Output   { STATE           STATE-POP
           DIST/COLUMBIA   ??,???,???
Input      STATE-POP=756,51Ø
Output   { STATE             STATE-POP
           DIST/COLUMBIA       756,51Ø
Input      OK
Output     ADD COMPLETE    77/12/27          16:Ø5:Ø5
```

This example illustrates the use of the basic hard copy format with only one value specified. The question marks in the first output message indicate that the value keyed in for STATE-POP is invalid (because it contains a decimal point). The corrected value is entered, and the OK command is issued to add the record to the file.

Example 2:

```
Input    { OPEN TOWNS
           ADD ALABAMA,CLARK,THURMONT   POPULA=16,767
Output   { TOWN                     MAYOR              POPULA
           ALABAMA,CLARK,THURMONT !!!!!!!!!!!!!!         16:767
Input      MAYOR=BAKER,RICHARD
Output   { TOWN                     MAYOR              POPULA
           ALABAMA,CLARK,THURMONT BAKER,RICHARD          16,767
Input      OK
Output     ADD COMPLETE    77/12/27      16:06:13
```

The exclamation points marks under the MAYOR item name indicate that a value for that item must be included to add a town record to the file.

6.2.9. CHANGE

The CHANGE command initiates a series of inputs and UNIQUE responses which change a record in the file. Like the ADD command, the CHANGE command has two formats – the display format and the hard copy format.

When using the IBM 3270 display station under UNIQUE, your command input or responses to UNIQUE messages must be in hard copy format rather than display format. The reason for this is the IBM 3270 display station transmits a whole screen of data to the UNIQUE program instead of a data field limited by a start-of-entry (△) symbol and the cursor.

Upon successful completion of the CHANGE function, UNIQUE establishes a new rollback point for the current transaction.

The CHANGE command places the terminal in an update state. While the terminal is in this state, no UNIQUE commands are accepted except OK or CANCEL.

6.2.9.1. Display Format

Format:

```
CHANGE identifier-1[;identifier-2]...
```

where:

`identifier-1`
Is the record designation. (See DISPLAY command, 6.2.3, for further explanation.)

`identifier-2`
Designates another record in the file.

In response to the CHANGE command, UNIQUE sends to the terminal the item names that serve as column headers, the old values, and the update formats of the items associated with the specified record. The terminal operator can depress the tab key to advance the cursor to the beginning of each update format. (Note that an extra character position is provided for the sign of numeric items.) The terminal operator overwrites the update formats with the associated new values and then transmits the modified screen. If UNIQUE finds no errors, the modified record replaces the old record in the file and the terminal operator is notified.

If errors are discovered, UNIQUE returns the item names, old values, update formats, and either valid new values or error notations to the terminal for further modification by the terminal operator.

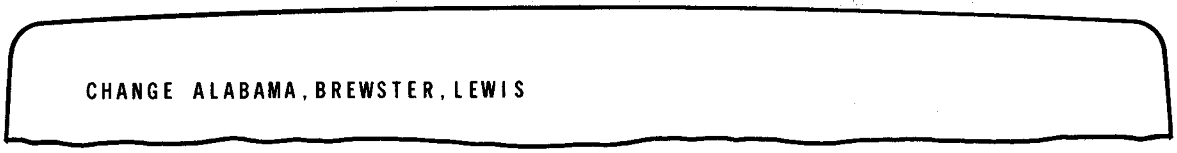
At any time during this sequence, the operator can enter the CANCEL command to cancel the CHANGE command instead of overwriting the update formats with values. The operator can instead key in the OK command to make changes to all valid items. Items for which invalid values have been specified remain unchanged in the file.

NOTE:

If a numeric item to be changed begins with a decimal point, the terminal oprator must key in the decimal point, even if the new value is zero.

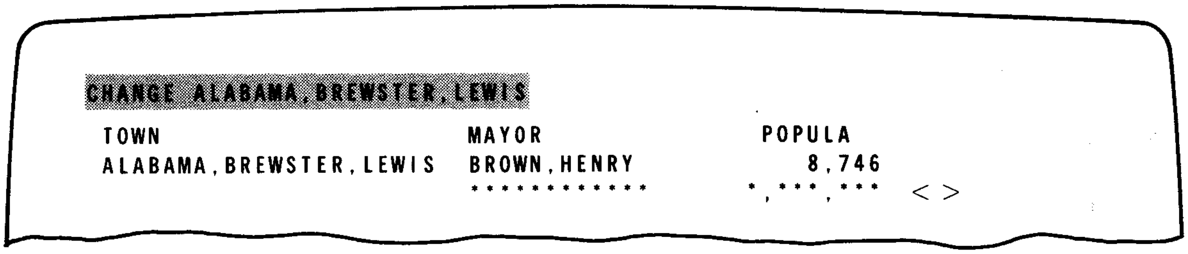
Example:

Input



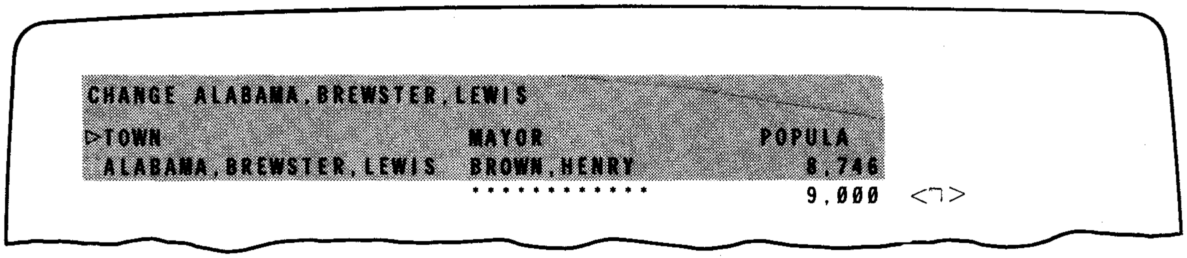
```
CHANGE ALABAMA,BREWSTER,LEWIS
```

Output



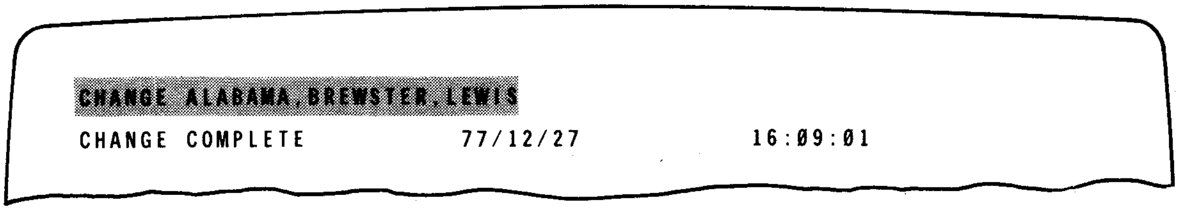
```
CHANGE ALABAMA,BREWSTER,LEWIS
 TOWN                    MAYOR              POPULA
 ALABAMA,BREWSTER,LEWIS  BROWN,HENRY           8,746
                         ************      *,***,***  < >
```

Input



```
CHANGE ALABAMA,BREWSTER,LEWIS
▷TOWN                    MAYOR              POPULA
 ALABAMA,BREWSTER,LEWIS  BROWN,HENRY           8,746
                         ************          9,ØØØ  <¬>
```

Output



```
CHANGE ALABAMA,BREWSTER,LEWIS
CHANGE COMPLETE          77/12/27           16:Ø9:Ø1
```

The absence of asterisks under TOWN in the first output indicates that the operator cannot change that item. Items that cannot be changed contain blanks instead of asterisks in the update format line.

6.2.9.2. Hard Copy Format

Basic Format:

```
CHANGE identifier item-name=value[;[item-name=]value]...
```

General Format:

```
CHANGE identifier[;]...[item-name=]value[[;]...,[item-name=]value]...
```

NOTE:

The general format is an expanded form of the simple format.

where:

```
identifier
```
Is the record designator. (See DISPLAY command, 6.2.3.)

```
item-name
```
Names an item. An item name can be any name, other than an identifier name, that appears as a column header in a DISPLAY command. The equal sign must be coded immediately after each item name. The item names need not be coded in the order of their appearance in the displayed record.

Item names can be omitted, in which case they are inferred from the position of the value in the input message. The positions of values are determined from the last specified item-name forward in a consecutive manner. If no item names are specified, positional locations are assumed from the first item after the identifier. Any consecutive item that is not to be updated is omitted by coding a semicolon (;) in its positional location.

```
value
```
Is the new value to replace the old value in item-name. All values, except the last, must be followed immediately by a semicolon (;).

In response to this command, UNIQUE displays the item names, original values, and values specified in the command, along with the identifier of the record. If the terminal operator is satisfied with the new record, he must key in OK to change this record in the file. If he decides not to change the record, he keys in the CANCEL command.

In response to the hard copy format CHANGE command, UNIQUE also displays error notations when invalid item names or conditions are discovered. UNIQUE displays invalid item names showing location in the input message, the invalid name, and the associated values. UNIQUE formats and displays all acceptable input for visual verification. The terminal operator can then decide to cancel the command or to correct the errors.

Corrections are made by entering a new input message with either the item-name and value (for example, NAME-5=VALUE-5), or just the value (for example, ;;;;VALUE-5) that is to be respecified. The operator does not key in the CHANGE command again when he is respecifying. UNIQUE responds by displaying all the original output, plus the respecified values displayed beneath the item names and old values for items previously in error. The terminal operator must then key in OK to effect the update, or CANCEL to cancel the entire CHANGE command.

NOTE:

The hard copy format also is used with display devices.

Example:

```
Input      CHANGE ALABAMA,BREWSTER,MOREHEAD ;14,725
Output   { TOWN                                        POPULA
         { ALABAMA,BREWSTER,MOREHEAD                   14,35Ø
                                                       14,725
Input      OK
Output     CHANGE COMPLETE        77/12/27            16:Ø9:57
```

In the first input, the item POPULA is inferred from the position of the value, based on the preceding semicolon. An OK command requests UNIQUE to go ahead with the valid change. The resulting new record then changes only the POPULA item.

6.2.10. LIST

The LIST command selects certain records or parts of records for listing at the terminal based on the text of the command. It can also display the results of arithmetic expressions and statistical functions.

LIST parameters are positional; if used, they must be specified in the order shown in the format.

Format:

```
LIST [[display-content-spec][IF selection-criteria];]...
[FOR identifier-1][{AFTER} identifier-2]
                  [{FROM }             ]
[statistical-function [item-name-1 [,item-name-2]...]]...
```

where:

display-content-spec

Indicates the elements of the selected records that are to be displayed. The display content specification can be:

- record-name – the name that represents the entire record;
- ALL – all items of the record (functionally equivalent to record-name)
- subrecord name – a name that represents a subset of items in a record
- item-names – identifying individual items in the selected record

IF selection-criteria

Indicates which records are to be selected on the basis of conditional expressions. A simple conditional expression is a comparison between an item and another item or a literal value. The comparison operators are:

EQ or =	Equal to
NE	Not equal to
GT or >	Greater than
GE	Greater than or equal to
LT or <	Less than
LE	Less than or equal to

If the item is compared to a nonnumeric literal, the literal can have the same number of characters as the item or it can be shorter. If it is shorter, UNIQUE compares the characters in the literal with the same number of characters in the item, starting from the left. The remaining characters in the item are ignored. For example, the command LIST MAYOR='STAN' would select all mayors' names beginning with the letters STAN.

A conditional expression can be negated by preceding it with the word NOT. Conditional expressions also can be combined by Boolean operators (AND,OR) into more complex conditional expressions. Within a conditional expression, NOT is first evaluated, then AND, finally OR. The order of evaluation is changed by the use of parentheses.

FOR identifier-1

Restricts the output of the LIST command to a subset of the file. The record specified by identifier-1 and all subordinate records to that record are then considered for listing.

```
{AFTER} identifier-2
{FROM }
```

Indicates the first record to be considered for listing. The FROM clause starts with the record specified by identifier-2 and lists all selected records, including the one specified by identifier-2. The AFTER clause lists all selected records after the record specified, not including the one specified by identifier-2.

```
statistical-function [item-name-1 [,item-name-2]...]
```

Requests the calculation and display of the results of certain statistical functions. Statistical functions are performed at all levels. The specifications for statistical-function include:

AVG

Calculates and displays the average of the numeric items from the records selected.

COUNT

Displays the number of occurrences of records that meet the selection criteria. No item name is specified.

MAX

Finds and displays the maximum value of the numeric items from the records selected and the identifier of the particular record in which it is found.

MIN

Finds and displays the minimum value of the numeric items from the records selected and the identifier of the particular record in which it is found.

TOTAL

Accumulates and displays the value of the numeric items from the records selected.

```
item-name
```

Names a numeric item. Item-name can be any name, other than an identifier name, that appears as a column header for this record type, so long as its values are defined to be numeric.

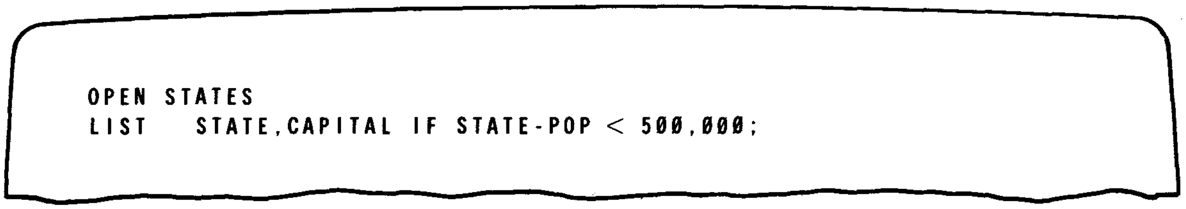
UNIQUE saves the text of the LIST command. This allows the terminal operator to reuse corresponding portions of the last LIST command to construct another LIST command.

When the defined file contains several types of defined records, the user specifies the display-content-spec and/or IF selection-criteria (delimited by the semicolon) separately for each type of defined record.

LIST parameters are positional; if used, they must be specified in the order shown in the command format, and in the same order in which their corresponding defined records were defined. This then is the order in which the different types of defined records appear on the screen in response to the LIST command. UNIQUE uses the position of parameters, indicated by the semicolons, to determine the defined record to which the parameters apply. (See examples 2, 4, and the example for the DETAIL command, 6.2.12.)

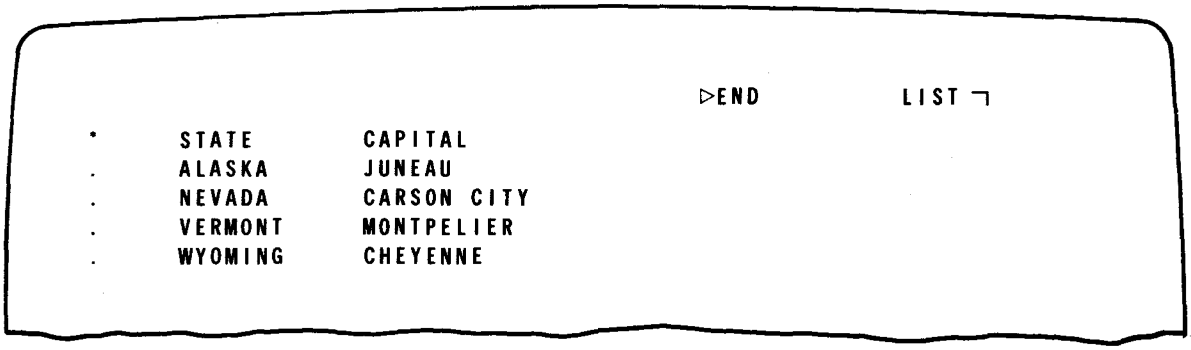
Example 1:

Input



```
OPEN STATES
LIST   STATE,CAPITAL IF STATE-POP < 5ØØ,ØØØ;
```

Output



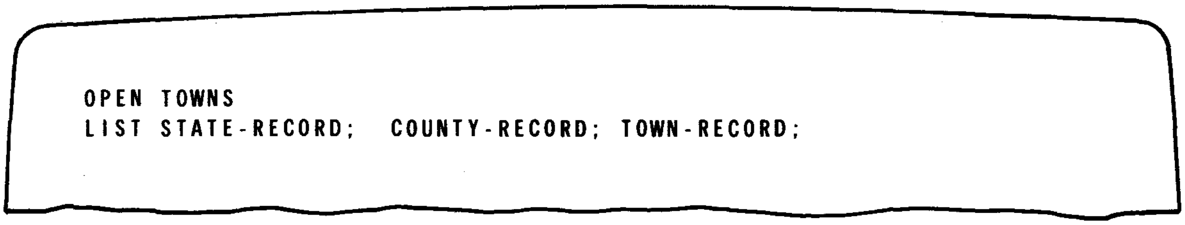
```
                                                  ▷END        LIST ¬
.     STATE      CAPITAL
.     ALASKA     JUNEAU
.     NEVADA     CARSON CITY
.     VERMONT    MONTPELIER
.     WYOMING    CHEYENNE
```

The OPEN command makes the STATES file accessible. The LIST command requests the STATE and CAPITAL item names and values to be listed for those records in which the value of the STATE-POP item is less than 500,000. Note that they required final semicolon (;).

The END LIST status indication is displayed on the final screen of listed information. Another instance when END LIST occurs without data is when a defined file was accessed and the physical file used to create the defined record was closed via the ZZCLS command.

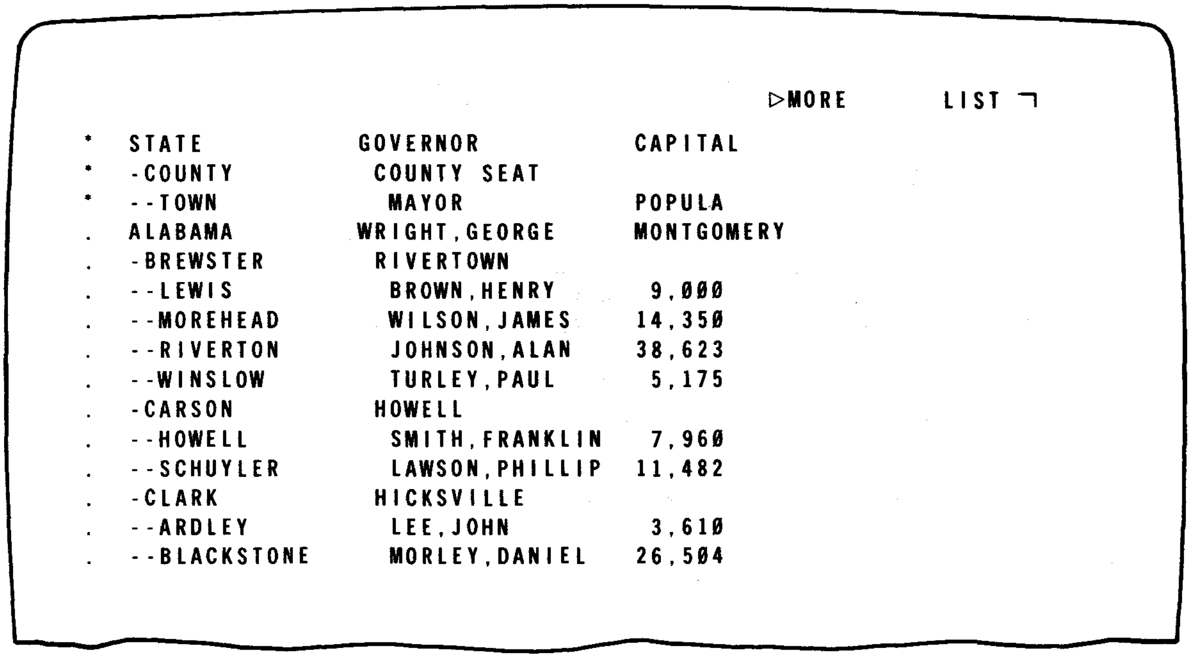
Example 2:

Input



```
OPEN TOWNS
LIST STATE-RECORD;  COUNTY-RECORD; TOWN-RECORD;
```

Output



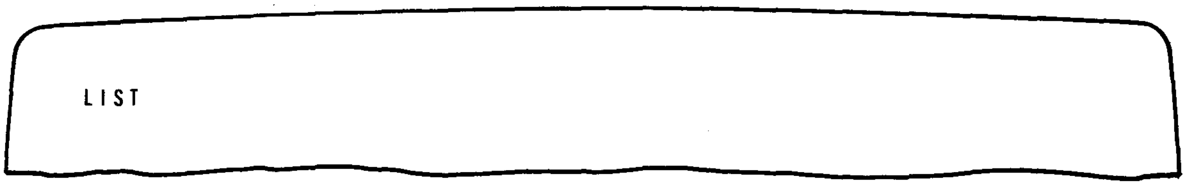
```
                                             ▷MORE       LIST ¬
*  STATE          GOVERNOR         CAPITAL
*  -COUNTY         COUNTY SEAT
*  --TOWN           MAYOR            POPULA
.  ALABAMA        WRIGHT,GEORGE    MONTGOMERY
.  -BREWSTER       RIVERTOWN
.  --LEWIS          BROWN,HENRY       9,000
.  --MOREHEAD       WILSON,JAMES     14,350
.  --RIVERTON       JOHNSON,ALAN     38,623
.  --WINSLOW        TURLEY,PAUL       5,175
.  -CARSON         HOWELL
.  --HOWELL         SMITH,FRANKLIN    7,960
.  --SCHUYLER       LAWSON,PHILLIP   11,482
.  -CLARK          HICKSVILLE
.  --ARDLEY         LEE,JOHN          3,610
.  --BLACKSTONE     MORLEY,DANIEL    26,504
```

This LIST command applies to the TOWNS file since that is the presently accessible file. The display-content-spec is a list of record names. The order of the record names is the order in which they are listed and must coincide with the order in which they are defined in the data definition.

This request is not satisfied in a single screen; that is, more state, county, and town records exist in the file. Therefore, a MORE LIST status indication is sent to the terminal operator. He requests the next screenful by keying in the MORE command.

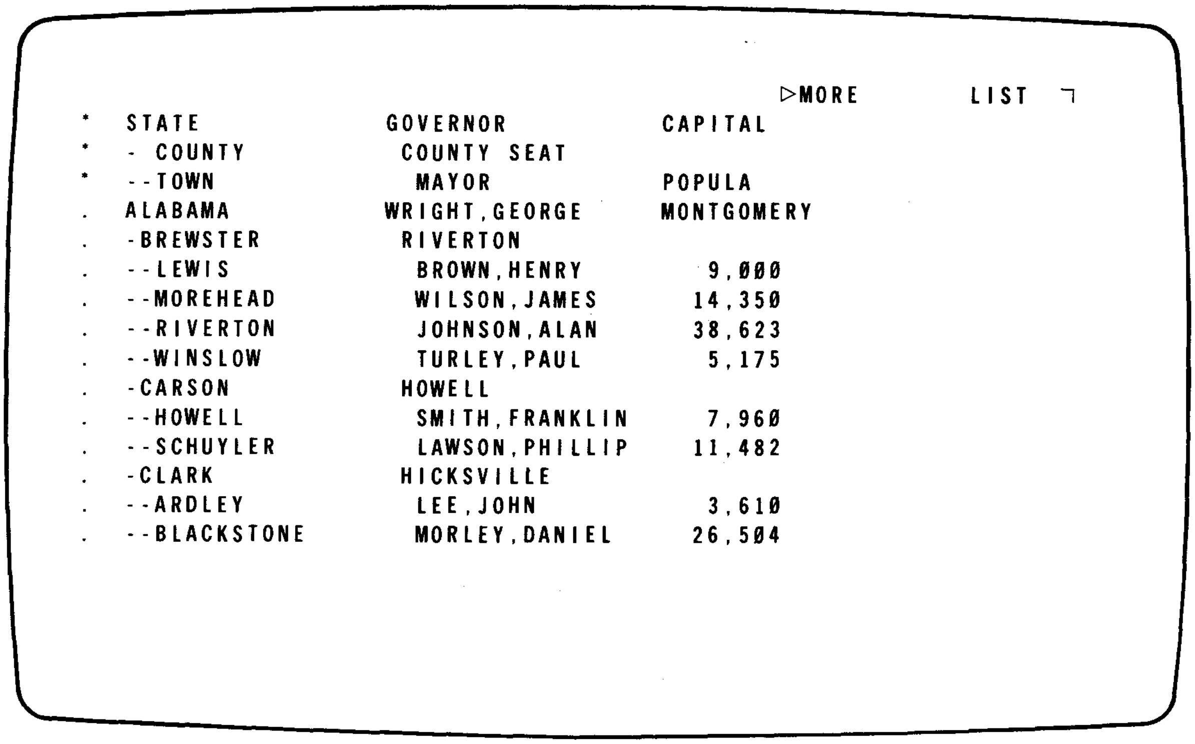
Example 3:

Input



```
LIST
```

Output

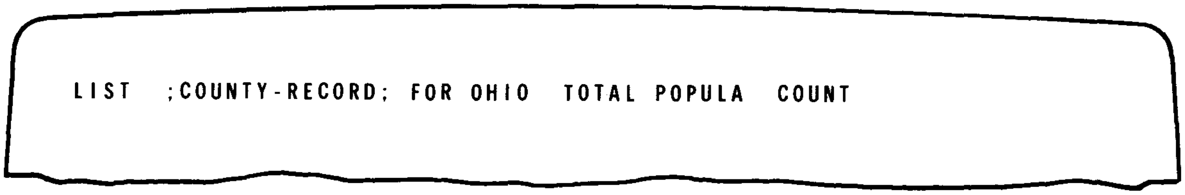


```
                                                   ▷MORE        LIST  ¬
*  STATE             GOVERNOR          CAPITAL
*  - COUNTY           COUNTY SEAT
*  --TOWN              MAYOR           POPULA
.  ALABAMA           WRIGHT,GEORGE     MONTGOMERY
.  -BREWSTER          RIVERTON
.  --LEWIS             BROWN,HENRY       9,ØØØ
.  --MOREHEAD          WILSON,JAMES     14,35Ø
.  --RIVERTON          JOHNSON,ALAN     38,623
.  --WINSLOW           TURLEY,PAUL       5,175
.  -CARSON             HOWELL
.  --HOWELL            SMITH,FRANKLIN    7,96Ø
.  --SCHUYLER          LAWSON,PHILLIP   11,482
.  -CLARK              HICKSVILLE
.  --ARDLEY            LEE,JOHN          3,61Ø
.  --BLACKSTONE        MORLEY,DANIEL    26,5Ø4
```

Note that LIST with no further text requests the entire file to be listed. In this case, the LIST command keyed in produces the same output as example 2.

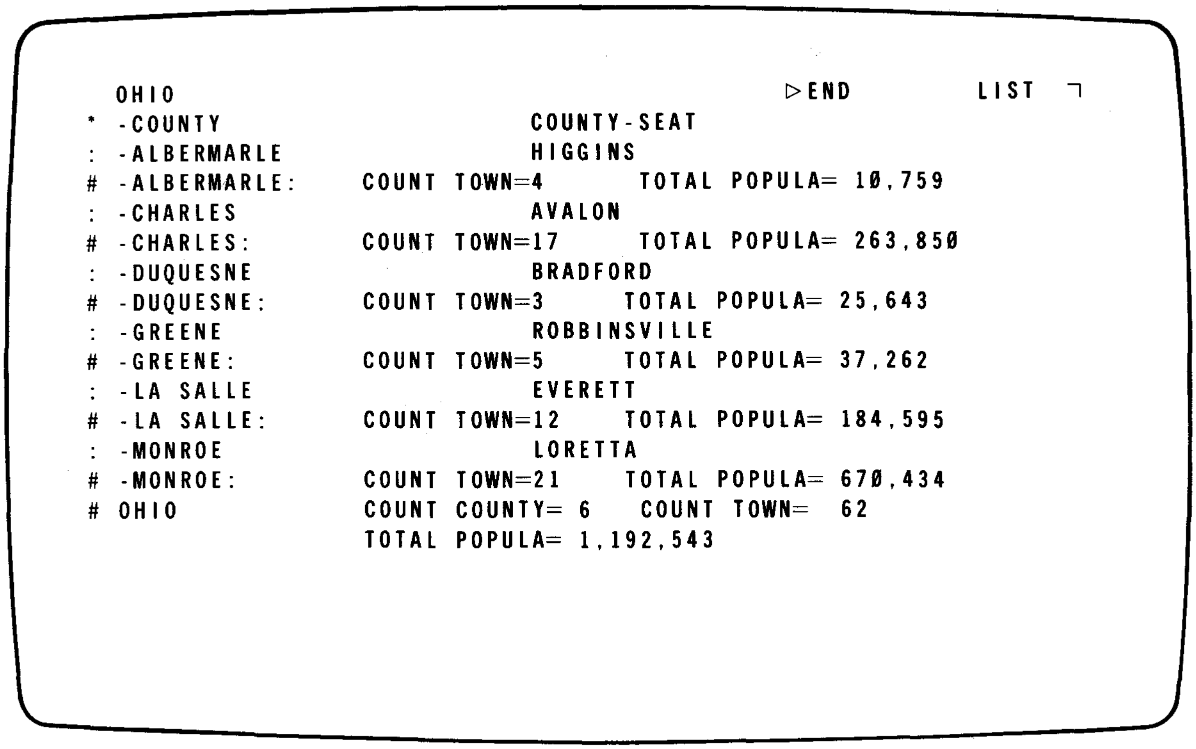
Example 4:

Input



```
LIST  ;COUNTY-RECORD; FOR OHIO  TOTAL POPULA  COUNT
```

Output



```
     OHIO                                               ▷END        LIST ⌝
   * -COUNTY                   COUNTY-SEAT
   : -ALBERMARLE               HIGGINS
   # -ALBERMARLE:    COUNT TOWN=4      TOTAL POPULA= 1Ø,759
   : -CHARLES                  AVALON
   # -CHARLES:       COUNT TOWN=17     TOTAL POPULA= 263,85Ø
   : -DUQUESNE                 BRADFORD
   # -DUQUESNE:      COUNT TOWN=3     TOTAL POPULA= 25,643
   : -GREENE                   ROBBINSVILLE
   # -GREENE:        COUNT TOWN=5     TOTAL POPULA= 37,262
   : -LA SALLE                 EVERETT
   # -LA SALLE:      COUNT TOWN=12    TOTAL POPULA= 184,595
   : -MONROE                   LORETTA
   # -MONROE:        COUNT TOWN=21    TOTAL POPULA= 67Ø,434
   # OHIO            COUNT COUNTY= 6   COUNT TOWN=  62
                     TOTAL POPULA= 1,192,543
```

The display-content-spec for this LIST command requests the listing of only the county records from the TOWNS file. The first semicolon (;) indicates that no detail from the STATE record is desired. This semicolon is used to indicate omission; record names must appear in the order in which they have been described in the data definition. Also, the listing is restricted to those county records whose identifiers begin with OHIO (that is, those counties that are in the state of OHIO), as requested in the FOR clause.

Total lines also are produced because the terminal operator requests the total of the item POPULA and the count of town records and county records. Total lines are indicated by the number sign (#). In addition to the totals of POPULA for each county, a grand total for the state is automatically calculated. Similarly, the number of town records for the entire state is calculated, along with the number of town records for each county and the number of counties in Ohio.

6.2.11. MORE

The MORE command requests the next screenful of information according to the previous LIST or DETAIL command.

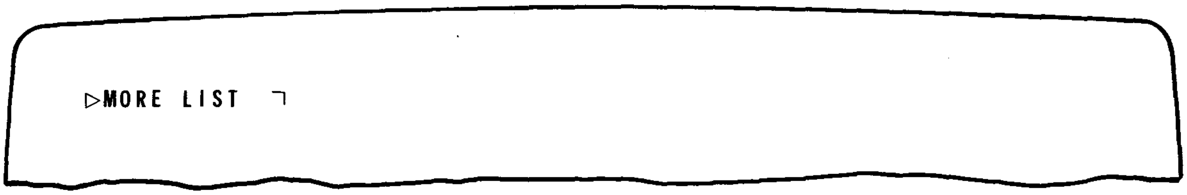
Format:

```
MORE ⎡⎧DETAIL⎫⎤
     ⎣⎩LIST  ⎭⎦
```

The DETAIL and LIST options indicate which of the previous commands (LIST or DETAIL) that UNIQUE is to continue processing to provide the next screen of information. If only MORE is keyed in, UNIQUE outputs a screen in response to the last LIST or DETAIL command entered by the terminal operator.

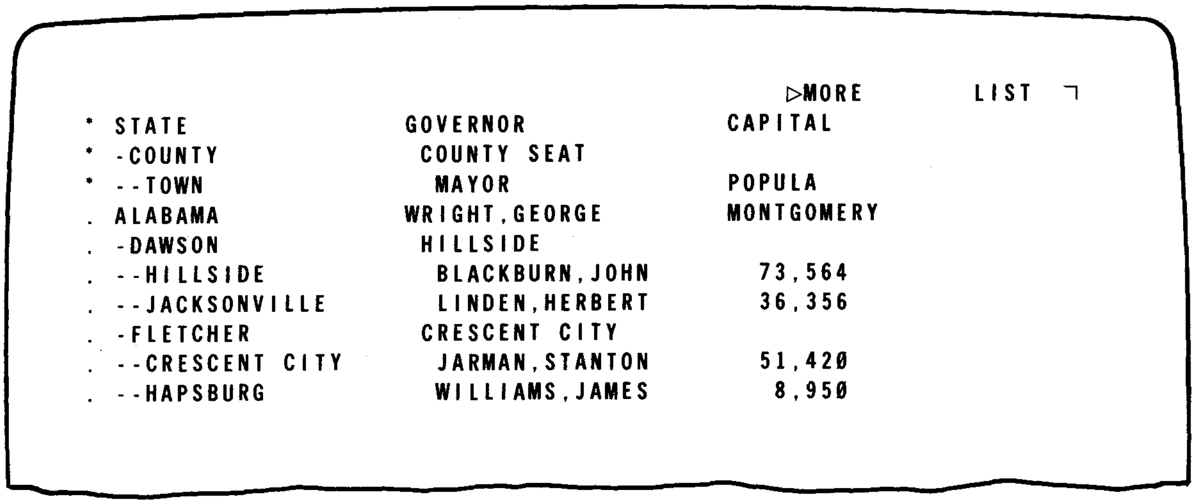
Example:

Input



```
▷MORE LIST ¬
```

Output



```
                                              ▷MORE        LIST ¬
• STATE             GOVERNOR             CAPITAL
• -COUNTY            COUNTY SEAT
• --TOWN              MAYOR              POPULA
. ALABAMA           WRIGHT,GEORGE        MONTGOMERY
. -DAWSON            HILLSIDE
. --HILLSIDE          BLACKBURN,JOHN       73,564
. --JACKSONVILLE      LINDEN,HERBERT       36,356
. -FLETCHER          CRESCENT CITY
. --CRESCENT CITY     JARMAN,STANTON       51,42Ø
. --HAPSBURG          WILLIAMS,JAMES        8,95Ø
```

UNIQUE anticipates the request by the operator to send more LIST information by supplying the MORE LIST phrase as part of its output (example 3 of LIST, 6.2.10). Thus, the terminal operator need only press the transmit button on the display device to input the MORE command. The data that is found between the start-of-message character (▷) and the cursor (¬) is transmitted to UNIQUE. Thus, UNIQUE receives the message MORE LIST and supplies the next screenful of information.

6.2.12. DETAIL

The DETAIL command interrupts the processing of the previous LIST command to request the processing of a higher priority list. This command is especially useful for nesting list requirements.

Format:

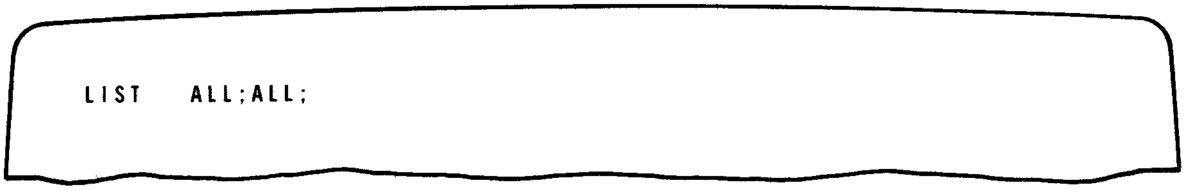
```
DETAIL [[display-content-spec][IF selection-criteria];]...
       [FOR identifier-1]
       [{AFTER} identifier-2]
       [{FROM }             ]
       [statistical-function [item-name-1 [,item-name-2]...]]...
```

See LIST command, 6.2.10, for explanation of the parameters.

The DETAIL command is identical to the LIST command, except that its text is retained separately from that of the LIST command and may be used repeatedly. This means that a different listing operation can be accomplished with a DETAIL command without destroying the content specification, selection criteria, or position within the file of the previous LIST command.

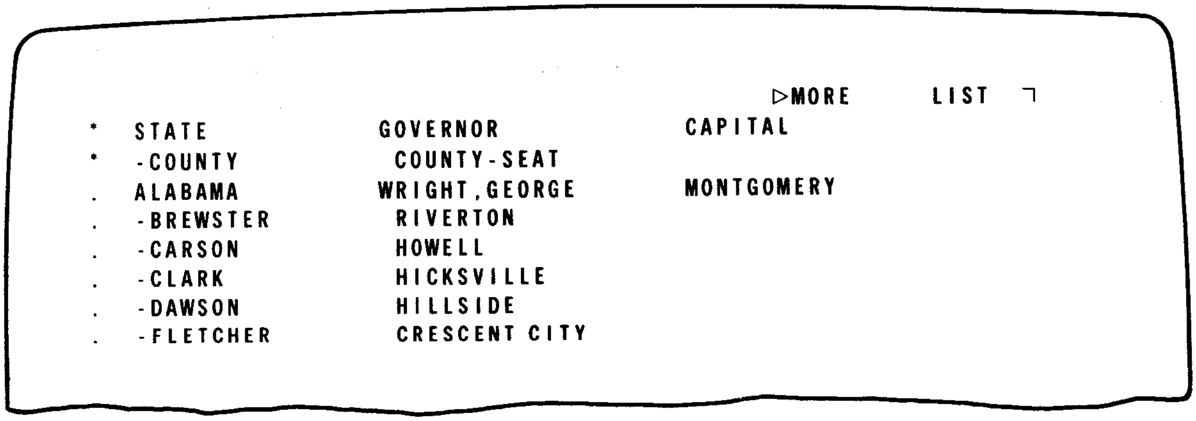
Example:

Input



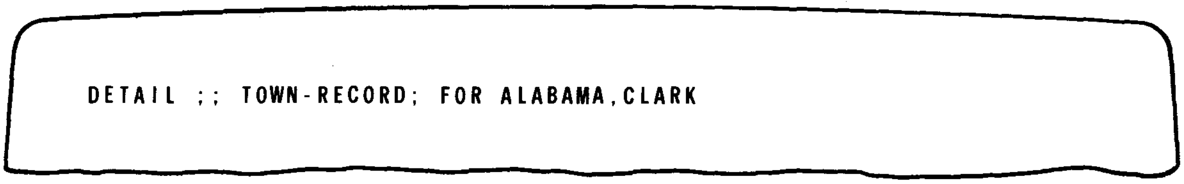
```
LIST   ALL;ALL;
```

Output



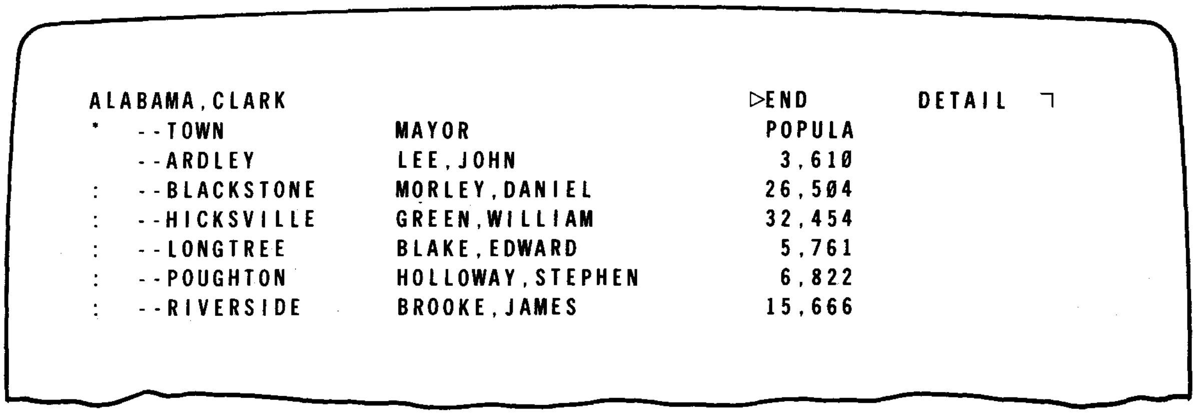
```
                                                  ▷MORE     LIST  ┐
*  STATE          GOVERNOR          CAPITAL
*  -COUNTY         COUNTY-SEAT
.  ALABAMA        WRIGHT,GEORGE     MONTGOMERY
.  -BREWSTER       RIVERTON
.  -CARSON         HOWELL
.  -CLARK          HICKSVILLE
.  -DAWSON         HILLSIDE
.  -FLETCHER       CRESCENT CITY
```

Input



```
DETAIL ;; TOWN-RECORD; FOR ALABAMA,CLARK
```

Output



```
ALABAMA,CLARK                                ▷END       DETAIL  ¬
*   --TOWN            MAYOR                  POPULA
    --ARDLEY          LEE,JOHN                3,61Ø
:   --BLACKSTONE      MORLEY,DANIEL          26,5Ø4
:   --HICKSVILLE      GREEN,WILLIAM          32,454
:   --LONGTREE        BLAKE,EDWARD            5,761
:   --POUGHTON        HOLLOWAY,STEPHEN        6,822
:   --RIVERSIDE       BROOKE,JAMES           15,666
```

In this example, the terminal operator uses the DETAIL command to interrupt the listing of states and counties to obtain a more detailed listing of the towns in a particular county. The MORE command at this point requests the next screen from the LIST command, since the DETAIL command has completed its processing.

6.2.13. SHOW

The SHOW command provides the terminal operator with a capsule data description of the defined file that is being accessed. This includes item names and subrecord names and their attributes. The command also informs the user about the most recent LIST and DETAIL operations. In addition, the SHOW command lists any DISPLAY command or any update command (ADD display format, CHANGE display format, or DELETE) having outstanding, unprocessed identifiers awaiting the entry of a NEXT command. These commands are listed, each followed by its unprocessed identifiers in their order of entry.

Format:

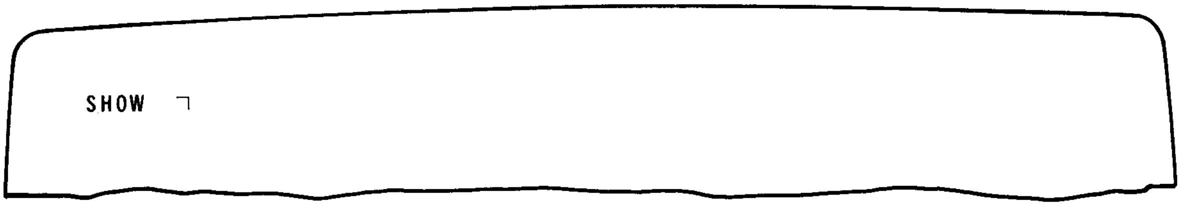
```
SHOW
```

UNIQUE uses the following symbols to format the output display in response to the SHOW command:

Symbol	Meaning
I	An identifier
A	A value must be supplied for an ADD command.
*	A new value can be supplied for a CHANGE command.
!	A value must be supplied for an ADD command, and a new value can be supplied for a CHANGE command.
D	A new value cannot be supplied for a CHANGE command.

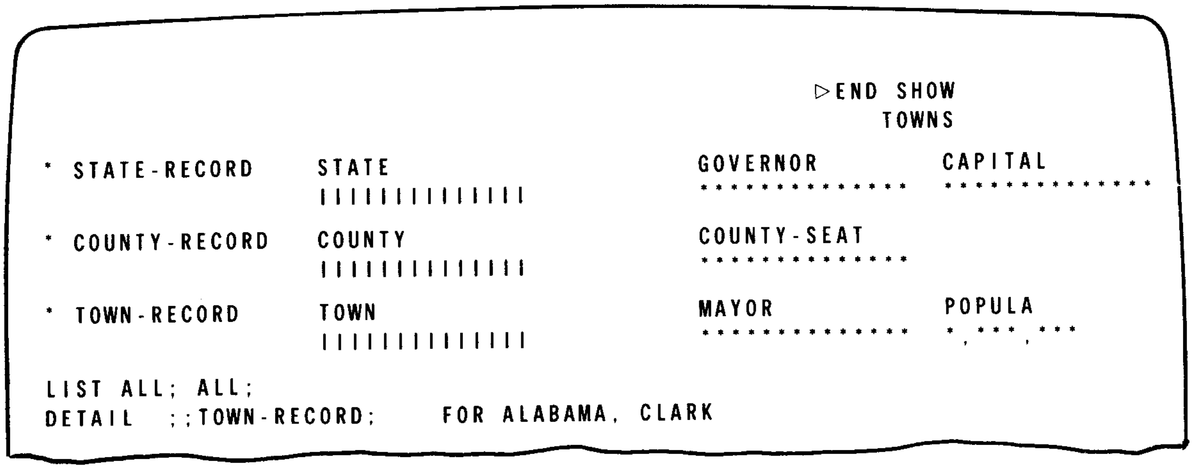
Example:

Input



```
SHOW ⌐
```

Output



```
                                                        ▷END SHOW
                                                            TOWNS

* STATE-RECORD     STATE                           GOVERNOR        CAPITAL
                   IIIIIIIIIIIIII                  **************  **************

* COUNTY-RECORD    COUNTY                          COUNTY-SEAT
                   IIIIIIIIIIIIII                  **************

* TOWN-RECORD      TOWN                            MAYOR           POPULA
                   IIIIIIIIIIIIII                  **************  *,***,***

LIST ALL; ALL;
DETAIL  ;;TOWN-RECORD;     FOR ALABAMA, CLARK
```

The SHOW command displays the password TOWNS for the presently accessible file and lists the records STATE, COUNTY, and TOWN with the items they contain, and appropriate update formats for those items. The I's indicate an identifier for STATE, COUNTY, or TOWN. The asterisks indicate that new values can be supplied for GOVERNOR, CAPITAL, COUNTY-SEAT, MAYOR, and POPULA on a CHANGE command. In addition, the SHOW command displays the most recent LIST and DETAIL commands and their current parameters.

7. Batch Processing of Transactions

7.1. PURPOSE AND USES OF BATCH TRANSACTION PROCESSOR

The batch processor is an optional component of IMS 90 that can be added to any single-thread or multithread configuration having adequate main storage and other resources. It is not available to the user of basic IMS 90. You include batch transaction processing by specifying the BATCH parameter in the NETWORK section of the configurator. The batch processor enables you to input transactions in card format, instead of through a display terminal; its output is directed to the printer or a printer file. Batched transactions can be processed online (that is, concurrently with routine production operations involving the normal terminal communications network) or offline, when no terminal network need be configured or active.

You can submit transactions to the batch processor either from card images filed in disk source modules or from card images included as embedded data in the job control stream at IMS 90 start-up. In neither case is it necessary to allocate a card reader to the IMS 90 job.

A primary use of the batch transaction processor is to display a file in print at the end of a day's production, as a data base administrator, for example, might need to do to review in detail the state of a file after massive update activity. It also is useful for obtaining a hard-copy listing of the data contained in a defined file or subfile – an activity that is not practical in normal operations from a display terminal.

Another important use of the batch transaction processor is to test new UNIQUE-based file query and update dialogs, designed for routine use at production terminals, as well as to test new user-written action programs that your operators initiate as transactions during normal production. Printed output listed by the batch processor reproduces all input and output messages, as well as unsolicited output, and thus provides you with a permanent, hard-copy record of each transaction.

7.2. PROCESSING AND OUTPUT

Batched transactions are processed as if they originate from one or more actual terminals; the batch processor responds to one or a number of *pseudoterminals* created by the IMS 90 configurator and lists output on a print file that is assigned to each pseudoterminal. This output is a step-by-step record of each transaction initiated.

Immediately after each input message, the batch processor prints the output message issued by the action program – with the exception of immediate or delayed internal succession. All input messages and output messages generated from one batch input source module (that is, from one batch pseudoterminal) are listed in the same print file and are not mixed with messages from any other source module.

For UNIQUE dialogs initiated as batch transactions, the batch transaction processor lists what is normally seen as output by the terminal operator, formatted as it would appear on the remote terminal. In this case, each input message is printed above the output message response and starts in column 1. The SOE (start of entry) character and the cursor are not represented. Terminal diagnostic and error messages are included in the output listing. There is no identification of any part of the output listing with a specific batch pseudoterminal.

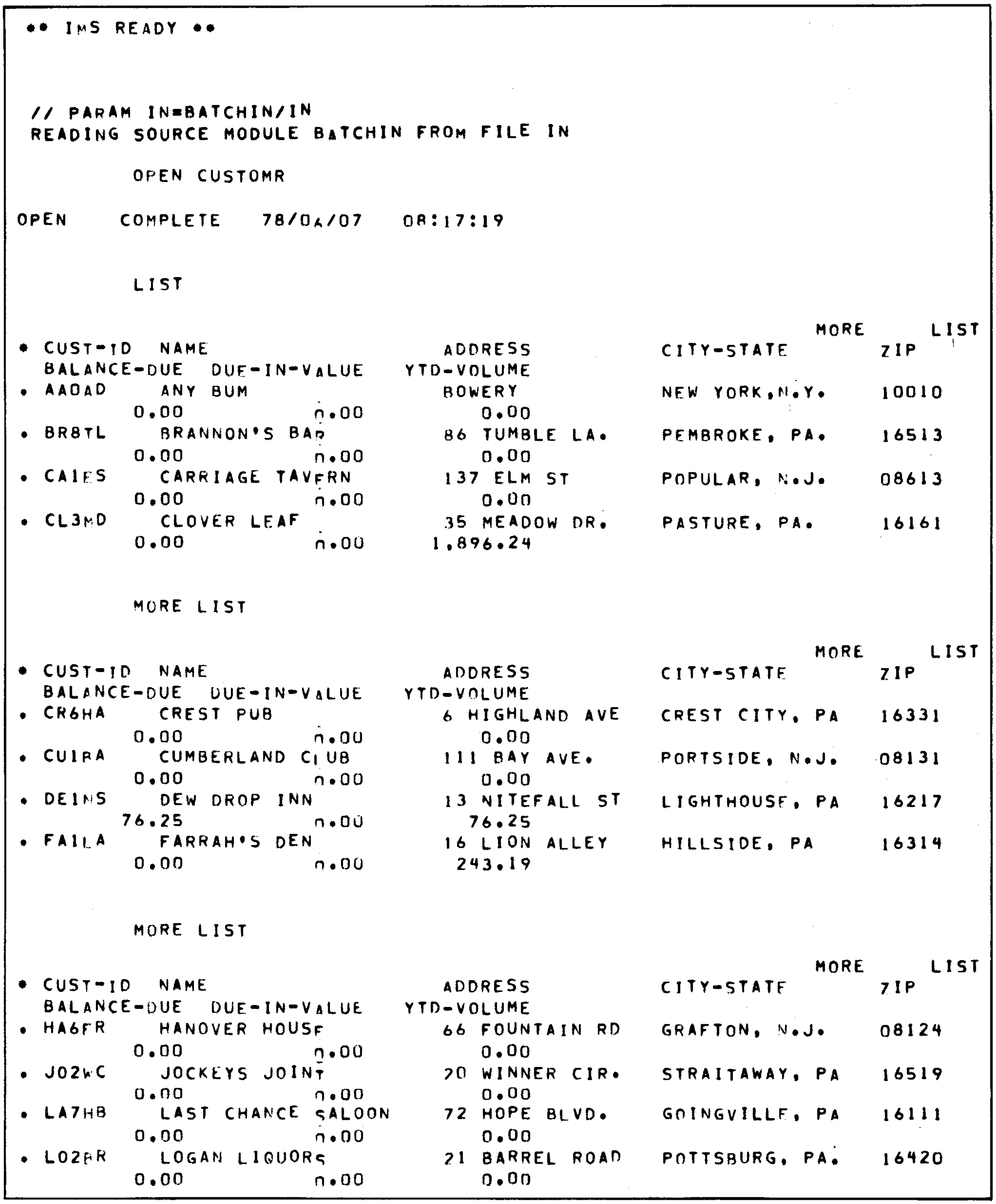
Figure 7–1 illustrates an output listing created by the batch transaction processor for a UNIQUE dialog that opens a defined file to list its contents. The input messages include an OPEN, a LIST, six MORE LIST commands, and a CLOSE.

The output message that follows the **IMS READY** output message contains the name of the source input module, BATCHIN, and the name of the file, IN, on which it resides. These are specified in the PARAM IN statement that immediately precedes it. (Automatic status messages – INPUT IN PROCESS, INPUT IN QUEUE, and ROLLBACK IN PROCESS – are never sent to a batch pseudoterminal.)

The first input message, OPEN CUSTOMR, initiates the UNIQUE transaction; its response is the OPEN COMPLETE output message normally returned to the originating terminal. The output message in response to the LIST command is the first screenful of data records from the defined file, CUSTOMR; the entire file is listed by the response to the fifth MORE LIST command, as indicated by the END LIST response above the seventh output message. The next MORE LIST command, therefore, produces the normal error return displayed next, and the CLOSE command is followed by the routine CLOSE COMPLETE output message.

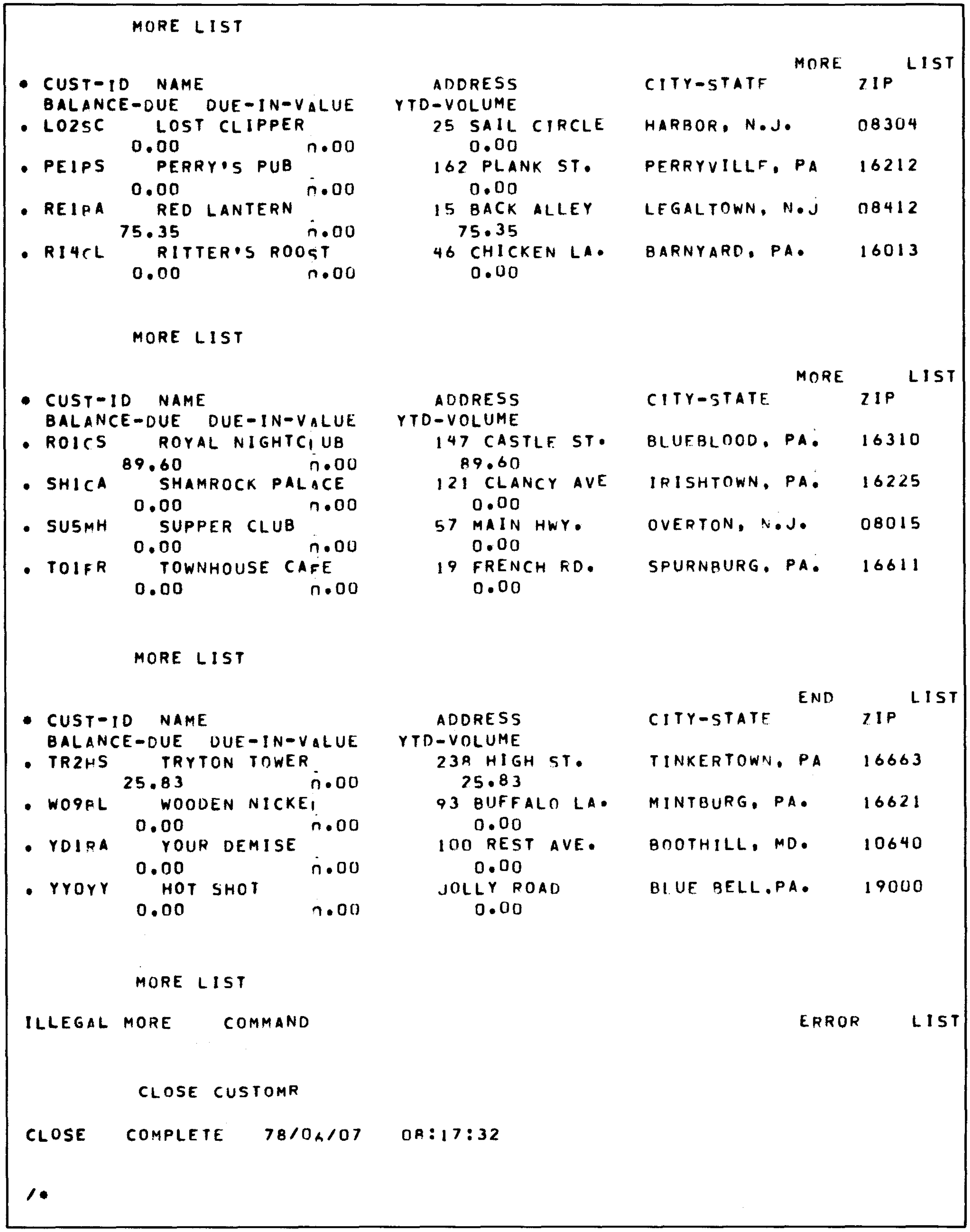
For readability of the output listing, the input messages shown in Figure 7–1 are punched beginning in column 10. This is feasible because all of them are UNIQUE commands, and UNIQUE allows this measure of free-form input. Normally, input messages begin in column 1.

In normal nonbatch operations, when a user-written action program terminates in delayed internal succession, no message is sent to the terminal operator; what otherwise would be the output message is queued by IMS 90 as input to the succeeding action and not displayed. For immediate internal succession, no message is sent to the operator or queued. Instead, IMS 90 makes the input and output message areas in main storage available to the successor action program. In batch operations, the message is not listed by the batch processor as either an output or an input message. With immediate or delayed internal succession, the next message the batch transaction processor lists is the output message from the succeeding action.



```
 •• IMS READY ••


 // PARAM IN=BATCHIN/IN
 READING SOURCE MODULE BATCHIN FROM FILE IN

           OPEN CUSTOMR

OPEN     COMPLETE    78/04/07    08:17:19


           LIST

                                                                       MORE     LIST
• CUST-ID  NAME                        ADDRESS             CITY-STATE           ZIP
  BALANCE-DUE  DUE-IN-VALUE        YTD-VOLUME
• AA0AD     ANY BUM                    BOWERY              NEW YORK,N.Y.        10010
           0.00              0.00          0.00
• BR8TL     BRANNON'S BAR              86 TUMBLE LA.       PEMBROKE, PA.        16513
           0.00              0.00          0.00
• CA1ES     CARRIAGE TAVERN            137 ELM ST          POPULAR, N.J.        08613
           0.00              0.00          0.00
• CL3MD     CLOVER LEAF                35 MEADOW DR.       PASTURE, PA.         16161
           0.00              0.00       1,896.24


           MORE LIST

                                                                       MORE     LIST
• CUST-ID  NAME                        ADDRESS             CITY-STATE           ZIP
  BALANCE-DUE  DUE-IN-VALUE        YTD-VOLUME
• CR6HA     CREST PUB                  6 HIGHLAND AVE      CREST CITY, PA       16331
           0.00              0.00          0.00
• CU1BA     CUMBERLAND CLUB            111 BAY AVE.        PORTSIDE, N.J.       08131
           0.00              0.00          0.00
• DE1NS     DEW DROP INN               13 NITEFALL ST      LIGHTHOUSE, PA       16217
         76.25               0.00         76.25
• FA1LA     FARRAH'S DEN               16 LION ALLEY       HILLSIDE, PA         16314
           0.00              0.00        243.19


           MORE LIST

                                                                       MORE     LIST
• CUST-ID  NAME                        ADDRESS             CITY-STATE           ZIP
  BALANCE-DUE  DUE-IN-VALUE        YTD-VOLUME
• HA6FR     HANOVER HOUSE              66 FOUNTAIN RD      GRAFTON, N.J.        08124
           0.00              0.00          0.00
• JO2WC     JOCKEYS JOINT              20 WINNER CIR.      STRAITAWAY, PA       16519
           0.00              0.00          0.00
• LA7HB     LAST CHANCE SALOON         72 HOPE BLVD.       GOINGVILLE, PA       16111
           0.00              0.00          0.00
• LO2BR     LOGAN LIQUORS              21 BARREL ROAD      POTTSBURG, PA.       16420
           0.00              0.00          0.00
```

Figure 7—1. Example of Output Listed by Batch Transaction Processor (Part 1 of 2)



```
        MORE LIST

                                                              MORE     LIST
• CUST-ID  NAME                    ADDRESS          CITY-STATE        ZIP
  BALANCE-DUE  DUE-IN-VALUE     YTD-VOLUME
• L02SC     LOST CLIPPER          25 SAIL CIRCLE    HARBOR, N.J.      08304
        0.00           0.00         0.00
• PE1PS     PERRY'S PUB           162 PLANK ST.     PERRYVILLE, PA    16212
        0.00           0.00         0.00
• RE1PA     RED LANTERN           15 BACK ALLEY     LEGALTOWN, N.J    08412
       75.35           0.00        75.35
• RI4CL     RITTER'S ROOST        46 CHICKEN LA.    BARNYARD, PA.     16013
        0.00           0.00         0.00


        MORE LIST

                                                              MORE     LIST
• CUST-ID  NAME                    ADDRESS          CITY-STATE        ZIP
  BALANCE-DUE  DUE-IN-VALUE     YTD-VOLUME
• R01CS     ROYAL NIGHTCLUB       147 CASTLE ST.    BLUEBLOOD, PA.    16310
       89.60           0.00        89.60
• SH1CA     SHAMROCK PALACE       121 CLANCY AVE    IRISHTOWN, PA.    16225
        0.00           0.00         0.00
• SU5MH     SUPPER CLUB           57 MAIN HWY.      OVERTON, N.J.     08015
        0.00           0.00         0.00
• T01FR     TOWNHOUSE CAFE        19 FRENCH RD.     SPURNBURG, PA.    16611
        0.00           0.00         0.00


        MORE LIST

                                                              END      LIST
• CUST-ID  NAME                    ADDRESS          CITY-STATE        ZIP
  BALANCE-DUE  DUE-IN-VALUE     YTD-VOLUME
• TR2HS     TRYTON TOWER          238 HIGH ST.      TINKERTOWN, PA    16663
       25.83           0.00        25.83
• W09BL     WOODEN NICKEL         93 BUFFALO LA.    MINTBURG, PA.     16621
        0.00           0.00         0.00
• YD1RA     YOUR DEMISE           100 REST AVE.     BOOTHILL, MD.     10640
        0.00           0.00         0.00
• YY0YY     HOT SHOT              JOLLY ROAD        BLUE BELL,PA.     19000
        0.00           0.00         0.00


        MORE LIST

ILLEGAL MORE    COMMAND                                       ERROR    LIST


         CLOSE CUSTOMR

CLOSE    COMPLETE    78/04/07    08:17:32


/*
```

Figure 7—1. Example of Output Listed by Batch Transaction Processor (Part 2 of 2)

Unsolicited, output can be generated in any online batch-initiated transaction. This is the result of issuing the SEND function call in a user-written action program or including a SWTCH transaction in your input. Unsolicited output in offline mode is not supported. In the online batch mode, if its destination is one or more online active terminals, unsolicited output is routed as for any other unsolicited output.

Unsolicited output destined for another batch pseudoterminal is not supported. Unsolicited output addressed to the originating pseudoterminal is listed on its own print file. An online terminal must not send unsolicited output to a batch pseudoterminal.

Transaction types processed in batch mode, online or offline, include the following:

- UNIQUE dialogs - Initiated by the OPEN command and comprising any of the UNIQUE command repertoire
- Other transactions - Initiated by a 5-character *transaction code* (typically, these activate user-written action programs)
- Two of the standard terminal commands (ZZTMD and ZZNRM). The ZZHLD, ZZRDY, ZZCNC, and ZZRSD terminal commands have no useful role in a batch transaction environment
- SWTCH transaction - For terminal-to-terminal communication.

No batch input message should include any of the master terminal commands; these are not allowed because the batch pseudoterminals may not be master terminals.

7.3. CONTROLLING BATCH TRANSACTION PROCESSING

Control of batch transaction processing begins with the specification of batch processing in the IMS 90 configuration. Certain modifications to the IMS 90 execution run stream also are needed.

Batch transactions can be processed in offline or online modes. Control of offline processing is essentially a matter of the order in which source input is presented in the control stream (7.5); whereas the ZZBTH master terminal command gives the online user considerable flexibility in determining when batch processing is to begin and end, and in specifying which input is to be processed and in what order (7.6).

7.3.1. Effect of IMS 90 Configuration Options

When considering how to control batch processing, you should first review the description of the BATCH keyword parameter in the NETWORK section of your input to the IMS 90 configurator. This is the parameter that adds the batch processing modules to your IMS 90 system.

When you have specified that the batch processor is to be configured using this keyword, the IMS 90 configurator generates one or more batch pseudoterminals, depending on the specification of the BATCH keyword parameter. If BATCH=YES is specified, one pseudoterminal is generated and assigned the *terminal id* BTH1. When the BATCH=n specification is used, the configurator generates the number of pseudoterminals designated by *n;* the *terminal ids* are BTH1,...,BTH4.

In a single-thread IMS 90 system, you process only one batch input source module or embedded data set at a time; the specification BATCH=YES and BATCH=1 are equally valid. In a multithread IMS 90 configuration, on the other hand, although you still process only one embedded data set at a time, you can process simultaneously up to the maximum number *(n)* of batch input source modules. This means that, if you have specified BATCH=3 to the IMS 90 configurator, you can process up to three source modules at one time, or one embedded data set and up to two source modules. In online mode, which you control by issuing the ZZBTH master terminal command, the number of ZZBTH commands that can be active simultaneously is likewise limited by the maximum number specified by the BATCH configurator keyword parameter. (The ZZBTH command is described in 7.6.1.)

7.3.2. IMS 90 Control Streams for Batch Processing

You have four areas of concern in coding the IMS 90 execution run stream when you are running batch transactions:

1. assigning source module input files;

2. assigning printer files;

3. inserting PARAM statements to control the batch processor (these must always follow any other PARAM statements present); and

4. embedding sets of input source data in the control stream.

7.3.2.1. Assigning Source Module Input Files

Unless all batched transactions are to be represented by data sets embedded in the control stream, you must name and create one or more source modules, containing the card images of your input messages, and place these modules in a disk library file. Each input disk file contains several modules. In the control steam, you must assign these source module input files to the IMS 90 execution job, using OS/3 job control conventions. The LFD-names of these input files are used in the PARAM IN statements described in 7.3.2.3.

7.3.2.2. Assigning Print Files to Batch Pseudoterminals

A print file must be assigned to each batch pseudoterminal that the IMS 90 configurator has been directed to generate by your specification of the BATCH configurator keyword parameter. The LFD-names expected by the batch processor for these print files are:

terminal-id	print file LFD-name
BTH1	PRNTR1
BTH2	PRNTR2
BTH3	PRNTR3
BTH4	PRNTR4

Again, the assignment of these files follows OS/3 job control conventions.

7.3.2.3. Invoking and Controlling Batch Processor (PARAM Statements)

Batch processor parameter statements always follow any other PARAM statements in an IMS 90 execution run stream. If there are no other PARAM statements in a single-thread IMS 90 run stream, batch processor parameter statements immediately follow the EXEC statement. In the run stream for a multithread IMS 90 load module, the PARAM BATCH statement immediately follows the PARAM START or PARAM RESTART statement, if there are no other PARAM statements to be placed there. Optional PARAM IN statements indicating source module input files to be processed (if any), immediately follow the PARAM BATCH or PARAM BA statement.

PARAM BATCH format:

```
// PARAM {BA   }={NO     }
         {BATCH} {OFFLINE}
                 {ONLINE }
```

where:

BA

Is the alternate, short form of the BATCH keyword parameter.

NO

Specifies that the configured batch processor is not to be invoked during this execution of IMS 90 and causes the main storage allotted to batch subroutines to be released for other uses by IMS 90.

OFFLINE

Specifies that batched transactions are to be processed in offline mode; that is, without an active terminal network.

ONLINE

Specifies that batched transactions are to be processed in online mode. This mode requires a communications network, including at least the master terminal, to be configured and active. Batch processing is controlled by the ZZBTH master terminal command.

If you have created and filed batch input source modules, you insert into the control stream PARAM IN statements for those that are to be processed, following the PARAM BATCH statement. There may be any number of PARAM IN statements (or none), and the order in which they appear in the control stream is your option. If sets of batch input card images are embedded in the control stream, these may be interspersed freely among the PARAM IN statements in whatever order you want.

PARAM IN format:

```
// PARAM IN=module-name/file-name
```

where:

module-name

Is a 1- to 8-character name of a source module containing card images of input messages to be processed by the batch transaction processor. This module resides in the disk file named by the *file-name* parameter.

file-name

Is the 1- to 8-character name of the disk file on which the source module resides. The *file-name* parameter must match the LFD-name specified in a job control statement assigning the file to the current execution of IMS 90.

Is a separator between the *module-name* and *file-name* parameters and must be coded as shown. It cannot be preceded by a blank, nor be followed by a blank.

7.3.2.4. Embedding Source Data in Control Stream

At your option, you embed sets of card images of input messages in the batch processor control stream, following OS/3 job control conventions. These can be interspersed freely among the PARAM IN statements (Figure 7-2). However, for better performance in multithread batch processing, you should group all embedded data sets last, behind (after) all your PARAM IN statements.

7.3.2.5. Sample Control Stream

Figure 7-2 illustrates a control stream for executing a multithread IMS 90 system for online batch transaction processing. Note that the arbitrary LFD-name of the input source file (SRFILE), assigned in a DVC-LFD job control sequence, is repeated in the two PARAM IN statements later in the control steam; one PARAM IN statement calls for processing of transactions in a module of SRFILE named TEST. The other statement refers to a module in this file named PROD. This example assumes that BATCH=4 is specified in the NETWORK section input to the IMS 90 configurator (the number of printer files assigned to the job indicates this).

```
// JOB IMSBATCH,,36EE8
// DVC 50 // VOL 123456 // LBL NAMEREC // LFD NAMEREC
// DVC 50 // VOL 123456 // LBL AUDFILE // LFD AUDFILE }
// DVC 50 // VOL 123456 // LBL CONDATA // LFD CONDATA } ①
// DVC 50 // VOL 123456 // LBL TCIDTF  // LFD TCIDTF,,INIT   }
// DVC 50 // VOL 123456 // LBL DQFILE1 // LFD DQFILE1,,INIT  }
// DVC 50 // VOL 123456 // LBL DQFILE2 // LFD DQFILE2,,INIT  } ②
// DVC 50 // VOL 123456 // LBL DQFILE3 // LFD DQFILE3,,INIT  }
      .                       .                 .
      .                       .                 . ③
      .                       .                 .
// DVC 50 // VOL 123456 // LBL DQFILE4 // LFD SRFILE
      .                       .                 .
      .                       .                 . ④
      .                       .                 .
// DVC 20 // LFD PRNTR1 }
// DVC 20 // LFD PRNTR2 }
// DVC 20 // LFD PRNTR3 } ⑤
// DVC 20 // LFD PRNTR4 }
/&/ OPTION DUMP
// EXEC ZQ#IMS,LOAD,1 ⑥
// PARAM START ⑦
// PARAM BA=ONLINE
// PARAM IN=TEST/SRFILE  }
/$                       }
     .                   }
     . ⑧                 }
     .                   }
/*                       } ⑨
// PARAM IN=PROD/SRFILE  }
/$                       }
     .                   }
     . ⑧                 }
     .                   }
/*
/&
// FIN
```

NOTES:

① A single-thread IMS 90 system uses the AUDCONF file in lieu of these.

② Not required for offline batch mode, which does not use a terminal network.

③ Assign user data files

④ Assign source module input files referenced in ZZBTH master terminal commands.

⑤ Assign printer files; one for each batch pseudoterminal created by the configurator. Equal in number to the number specified in the BATCH configurator keyword.

⑥ The program name on the EXEC statement must match the LOADM parameter specification on the IMSCONF jproc at configuration. The default name for multithread IMS 90 is ZQ#IMS; the default name for single-thread IMS 90 is ZS#IMS.

⑦ For offline mode, code this as // PARAM BA=OFFLINE.

⑧ Batch input cards

⑨ Param IN statements, or embedded data sets, or both, are required for offine batch mode. Data sets may be freely interspersed among PARAM IN statements. These are optional in online batch mode and when present are processed through one or more ZZBTH*(,ALL) master terminal commands. If absent in online batch mode, the *module-name,filename* form of the ZZBTH command is used to specify all input source modules to be processed.

Figure 7—2. Sample IMS 90 Execution Run Stream for Online Batch Processing in a Multithr[illegible]tem

7.4. PREPARING TRANSACTION INPUT FOR BATCH PROCESSOR

Input for the batch processor is in the form of card images, either filed on disk or included as embedded data sets in the IMS 90 execution run stream. This capability provides the flexibility needed for batch testing as well as for standard batch production runs. Note that action programs using screen formatting services may not run in a batch (online or offline) environment.

7.4.1. Input Message Coding

Each input message is punched into a card, or as many as 18 cards; messages are grouped in the sequence in which the operator enters them at the terminal. The batch processor takes input message text from card image columns 1 through 71. (Individual messages longer than 71 characters are continued onto the next card by coding a nonblank character in column 72; coding on the continuation card begins in column 1.) Up to 17 continuation cards are allowed, giving a maximum length of 1346 bytes for any one message. Columns 73 through 80 are reserved for sequence numbers which are not required or processed. Both input message text and device independent control expressions (DICE) are in EBCDIC code.

When coding UNIQUE dialogs for batch transaction processing, you use the hard copy format of the ADD and CHANGE commands to avoid the laborious preparation of DICE sequences necessary if you use the display format.

When coding an input message for a user action program that does not allow free-form placement of data characters, you code data in specific card columns. The SWTCH transaction, on the other hand, must always be coded beginning in input card column 1 because this is the position expected by the SWTCH action program.

Figure 7-3 illustrates a sample UNIQUE dialog transaction prepared as input for batch processing, comprising 10 input messages and followed by two SWTCH transactions for sending unsolicited output to active online terminals. These card images constitute an embedded data set for inclusion in the control stream. Notice that no delimiter is required to separate the UNIQUE dialog from the SWTCH transaction code that follows it (the CLOSE command serves this purpose) and that nothing is used to mark the beginning or end of the source module. Notice also the handling of continuation for the input message that contains more than 71 characters.

```
OPEN CUSTOMR
LIST
MORE LIST
MORE LIST
MORE LIST
MORE LIST
DISPLAY LA7HB
DETAIL CUST-ID IF NAME GT .LOGAN LIQUORS      .AFTER .LAST CHANCE SAL*
OON .
MORE DETAIL
CLOSE
SWTCH T1,T2 THIS IS A MESSAGE
SWTCH T3,TØ THIS IS A MESSAGE ...............................
```

Figure 7—3. Sample UNIQUE Dialog Transaction Prepared as Input to Batch Transaction Processor

7.4.2. Handling DICE Characters

The characters used in DICE sequences that certain user-written action programs employ to format output messages are written as hexadecimal symbol pairs (using EBCDIC characters) in input messages prepared as batched transactions. To eliminate the requirement that the hexadecimal codes be multipunched, the batch processor employs a shift character (the ¬ character, multipunch 11-7-8) to designate that all EBCDIC characters following it are to be treated as hexadecimal digits (0-9,A-F) until another ¬ character is scanned. Each pair of hexadecimal digits is converted to a single byte before being passed to the batch processor. Two ¬ symbols in succession are converted to a single ¬. For example, the following EBCDIC sequence in an input message:

¬10040000¬YES¬0571¬

is transmitted to the processor as the following:

10	04	00	00	E8	C5	E2	05	71

NOTE:

The first four bytes are the DICE sequence for position control to new line on a CRT device.

When the batch processor encounters DICE control sequences or the ESC (escape) character (27_{16}) in an output message area, it strips them before printing the message. This corresponds to the online handling of DICE codes and hardware characters which are not displayed on the terminal. An OMA could look as follows:

10	01	01	00	C1	C2	C3	C4	C5	C6	10	04	00	00	C7	C8	27	05	C9
DICE Control Sequence				Message Text						Dice Control Sequence				Msg. Text		ESC	HT	Msg. Text

The print lines of this message after batch processing looks as follows:

First line: ABCDEF

Second line: GHI

Note the deletion of DICE control sequences and the ESC character (27_{16}) and HT (horizontal tab, 05_{16}).

7.5. CONTROLLING BATCH PROCESSING IN OFFLINE MODE

Because IMS 90 executes without a communications network in offline batch mode, there is no possibility of controlling batch processing via the master terminal. Batch input source files are assigned with statements in the IMS 90 execution run stream.

The BATCH=OFFLINE parameter statement is followed by IN parameter cards and/or embedded data sets of input card images; data sets are freely interspersed. The order in which these appear in the streams is the order in which the batched transactions they represent are processed. Listed output is ordered by the batch processor automatically. You can only indirectly control its order by the sequence in which you submit your input and, in a multithread IMS 90 system, by the number of batch pseudoterminals you have generated via the BATCH configurator parameter.

7.6. CONTROLLING BATCH PROCESSING IN ONLINE MODE

IMS 90 requires an active communications network in the online batch processing mode. This requirement allows close control of batch processing from the master terminal, using the ZZBTH master terminal command to specify which source modules are to be processed and when.

Batch processing that is conducted in online mode during normal production hours, when the regular operating terminals are busy, should be expected to downgrade performance. For this reason, online batch processing should be invoked during slow periods or prior to shutdown. An alternative is to invoke online batch mode during nonproductive time, perhaps configuring a special network consisting of the master terminal alone.

Another factor affecting online batch performance is the number of batch modules being processed concurrently in a multithread IMS 90 system. The fewer the modules, the better the performance. The maximum number that can be processed concurrently is determined by the number specified with the BATCH configurator keyword, but the master terminal operator controls the number of modules that *are* submitted simultaneously to the batch processor (within the maximum limit) by his timing of the ZZBTH command.

You assign batch input source files and printer files with job control statements in the IMS 90 execution run stream. Following the PARAM START statement, you submit PARAM BATCH=ONLINE, optionally followed by PARAM IN statements and embedded data sets. No batch processing is performed until the first ZZBTH master terminal command is entered. Note that PARAM IN statements or embedded data are appropriate only if you use the ZZBTH *[,ALL] form of the ZZBTH master terminal command to control batch processing in online mode. The *module-name,file-name* form of the ZZBTH command does not take source input from the control stream.

If you want to list the output printed by the batch processor before the termination of the IMS 90, you must instruct the system console operator to invoke the output writer by entering PR BU; if any print files are ready, printing will begin immediately. The purpose of entering the BU (burst mode) parameter is to avoid the normal printing of the IMS 90 job's log file first. If only one printer is available, a spooling supervisor should be configured at system installation time. (This is necessary because a nonspooling supervisor prints the output as soon as it is generated by the batch processor.) Another consideration in a multithread system is that the system console operator cannot be reached via the IMS 90 master terminal, although it is at this terminal that the operator controlling the batch processing online ascertains that processing has been completed for one or more modules and that the results are ready to be printed, if desired.

The function, format, and parameters of the ZZBTH master terminal command and how to use it in controlling batch processing in online mode are described in 7.6.1 through 7.6.5.

7.6.1. ZZBTH Master Terminal Command

The ZZBTH terminal command is entered at the master terminal to initiate and control the batch transaction processor in online mode. Parameters are available to specify the source of input messages for each execution of the command, to control sequential progress through control stream input, and to terminate, suspend, or resume the batch transaction processing currently in progress.

Format:

```
ZZBTH⎧module-name,file-name⎫
     ⎪*[,ALL]              ⎪
     ⎨CANCEL               ⎬
     ⎪PAUSE                ⎪
     ⎩RESUME               ⎭
```

where:

`module-name`

Is a 1- to 8-character name of the source module that contains the card images of input messages to be processed by the batch transaction processor in online mode. When this *module-name,file-name* form of the ZZBTH command is used, the batch processor does not seek the input source module in the job control stream, but locates it in the file named by the second parameter, *file-name*.

`file-name`

Is a 1- to 8-character name of the file on which the foregoing source module resides. The *file-name* parameter must match the LFD-name specified in a job control statement assigning the file to the current execution of IMS 90.

`*`

Specifies that the next PARAM IN card or the next set of embedded data encountered in the IMS 90 command stream represents the source of input messages for this execution of the ZZBTH command. When this execution of the ZBTH command is the first for the current execution of IMS 90, the PARAM IN card or embedded data set used by the batch processor is the first one encountered in the run stream. If the current execution of the ZZBTH * command is not the first for the job, the card or embedded data set selected from the run stream is one occurring after the one used as the source of input messages processed by the immediately preceding ZZBTH * command.

ALL

Specifies that the entire run stream will be processed, starting with the next source of input messages (represented by the * parameter specified with this execution of the ZZBTH * command). If the optional ALL parameter is omitted, processing of the run stream input stops when the next PARAM IN card or set of embedded data is processed. If a subsequent source module is represented in the run stream, therefore, you must enter another ZZBTH * [,ALL] command to process it.

CANCEL

Specifies that all batch transaction processing currently in progress is to terminate. Processing of each transaction ceases with the issue of the current output message to the print file, and the print file is closed. File modifications made since the beginning of the current transaction are not rolled back. The batch processor substitutes a ZZCNC terminal input command for the next expected input message.

PAUSE

Specifies that all batch transaction processing currently in progress is to be suspended. Processing of each transaction ceases with the issue of the current output message to the print file, but the print file is not closed.

RESUME

Specifies that all batch processing temporarily suspended by the preceding ZZBTH PAUSE command is to resume from the suspended state.

7.6.2. Initiating Online Batch Processing

When the ZZBTH command is entered at the master terminal and the batch processor recognizes batch input, the message BATCH INITIATED is output to the master terminal, and processing begins.

With single-thread IMS 90, only one ZZBTH command can be entered at a time; transaction processing in response to it must be completed before another ZZBTH command can be entered. In a multithread IMS 90 system, on the other hand, several ZZBTH commands can be processed simultaneously, up to the maximum number specified by the BATCH configurator keyword.

If the master terminal operator attempts to enter a ZZBTH command in excess of the maximum number of batch modules that can be processed concurrently (or a single-thread operator enters a second command before the processing of the current command is done), IMS 90 responds with the following diagnostic message:

```
TOO MANY ZZBTH COMMANDS ENTERED - COMMAND IGNORED
```

7.6.3. Tracking Progress of Batch Processing

To determine when a batch run is complete, or to keep track of the processing of batch modules, the master terminal operator at any time can enter the ZZTCT master terminal command to access the terminal control table generated for any batch pseudoterminal, and specify the batch *terminal id* desired (BTH1,...,BTH4). The ZZTCT command provides the master terminal with a report of the activity outstanding at the specified terminal. (See 5.2.2.4.)

7.6.4. Resuming Batch Processing Once Terminated

Issuing the ZZBTH CANCEL command terminates only the ongoing batch processing; the transaction processing that has been accomplished is listed. There is nothing to prevent reprocessing the same modules or to prevent batch processing from being performed on other source modules. You can continue by issuing the appropriate ZZBTH command after having received the response to the ZZBTH CANCEL command in effect. The response to the ZZBTH CANCEL command is the message:

```
BATCH PROCESSING CANCELLED
```

The effect of issuing a ZZBTH *[,ALL] command after you have issued a ZZBTH CANCEL command is to read the control stream at the next data set present. If you issue the ZZBTH *module-name, file-name* form of the command, the specified module is processed from its beginning, not from where processing ceased if this was the module being processed when you issued ZZBTH CANCEL.

7.6.5. Repetitive Use of Batch Mode

There is no feature in the batch transaction processor that checks whether your files have been processed by the same input – in fact, batch mode facilitates reprocessing with repetitive use of the same input. This enables you to use the same program at the end of each day to list the state of your files and to review the day's activity.

7.7. CONTINUOUS OUTPUT CONSIDERATIONS

Output delivery notification to a batch pseudoterminal is always generated as successful by the batch transaction processor. A program using output-for-input queueing can be testing in batch mode, but it is not reasonable to perform continuous output by this means in a batch production run.

7.8. BATCH PROCESSOR DIAGNOSTIC MESSAGES

In addition to the diagnostic and error messages normally output by IMS 90 to the operator at a production terminal (which the batch processor includes in the output listing for each batch pseudoterminal), the batch processor generates a set of its own messages. These it sends to the master terminal or includes in the output listing for the batch pseudoterminal, as appropriate. Table 7-1 lists the texts of these messages, describes their causes, and indicates the corrective action that can be taken.

7.9. RECOVERY CONSIDERATIONS

When recovery is invoked for batch processing, either cold or warm restart, the user enters a DLMSG input transaction via cards to the batch processor. One card is supplied for each batch terminal. For single-thread IMS 90, only one card is required. Using the printed output from this transaction (DLMSG), the user can decide what portion of his input batch module was processed before the system aborted. From this information the user can reconstruct his batch input card deck or use the librarian to edit a source library input message module.

An example of a warm restart batch input for a single-thread batch system is:

```
/$
DLMSG BTH1
/*
```

For warm restart, the before images for the files are restored at the start of the last transaction completed.

Table 7—1. Batch Transaction Processor (BTP) Diagnostic Messages (Part 1 of 2)

Message Text	Description or Cause	Corrective Action
BATCH INPUT MODULE READ ERROR	BTP attempts to access a non-existent input module; or an I/O error has occurred; or the module contains no input messages.	Check that module name is correct and that input messages are indeed contained. Otherwise, action as for hardware error.
INVALID BATCH PARAM CARD	The control stream contains an incorrect PARAM card.	Correct the PARAM card. Refer to 7.3.2.3.
PRINTER FILE ERROR	This message is sent to the master terminal when a printer file cannot be opened or an unrecoverable hardware error occurs.	Check that the file exists and that the LFD-name is correct. Otherwise, action as for hardware error.
READING SOURCE MODULE module-name FROM FILE file-name	This message is printed upon successfully opening the source input module file.	None
SOURCE MODULE module-name IN FILE file-name NOT FOUND	This message is printed if the batch input module is not in the library file.	Ensure that the source module exists in the file.
READ ERROR ON MODULE module-name FILE file-name	An I/O error occurs while the BTP is reading the batch input module.	As for hardware error
TOO MANY CONTINUATION CARDS	The BTP has encountered more than 17 continuation cards.	Restrict each input message to the maximum number of cards: 18. Refer to 7.4.1.
HEX CHARACTER ERROR	The BTP prints this message to the right of the input card image when an EBCDIC character other than 0–9 or A–F is coded between two input shift characters.	Revise coding between shift characters to represent hexadecimal codes; see 7.4.2.
HEX CHARACTER PAIRS INCOMPLETE	The BTP prints this message to the right of the input card image when the EBCDIC characters coded between two ¬ shift characters are within the set 0–9 or A–F but are odd in number.	Revise coding between ¬ shift characters to represent hexadecimal pairs; see 7.4.2.

Table 7—1. Batch Transaction Processor (BTP) Diagnostic Messages (Part 2 of 2)

Message Text	Description or Cause	Corrective Action
BATCH NOT CONFIGURED – COMMAND IGNORED	Output to master terminal in response to a ZZBTH command entered when BATCH=NO has been specified or the BATCH keyword is omitted from the NETWORK section of input to IMS 90 configurator.	If batch processing of transactions is intended, specify the BATCH keyword to the IMS 90 configurator according to 7.3.1.1.
BATCH=NO SPECIFIED IN PARAM CARD –COMMAND IGNORED	Output to master terminal in response to a ZZBTH command entered when batch processing has been specified to the IMS 90 configurator but the PARAM BA statement has been omitted from the IMS 90 run stream or PARAM BA=NO is specified.	Specify the intended PARAM BA statement in the IMS 90 run stream according to 7.3.2.3.
INVALID ZZBTH PARAMETERS – COMMAND IGNORED	Output to master terminal in response to a ZZBTH command entered with incorrect parameters (syntax error).	Reenter ZZBTH command with parameters conforming to those documented in 7.6.1.
TOO MANY ZZBTH COMMANDS ENTERED – COMMAND IGNORED	Output to master terminal in response to a ZZBTH command entered in excess of the number which may be processed at one time is this configuration. In a single-thread IMS 90 system, only one ZZBTH command may be active at one time; entry of a second must be deferreduuntil conpletion of processing for the current command. In a multithread system, the number of commands that may be processed simultaneously is linited by the maximum number specified withtthe BATCH configurator keyword.	Defer reentry of the ZZBTH command until response to the ZZTCT command indicates that processing is again possible (7.6.3).
BATCH PROCESSING CANCELLED	Normal response to successful ZZBTH CANCEL command; only the batch processing currently in progress terminates.	None required. Batch processing may be reinitiated during this execution of IMS 90 by entering an appropriate ZZBTH command (7.6.4).
ALREADY READING BATCH CONTROL STREAM DATA SET – COMMAND IGNORED	Output to master terminal in response to a ZZBTH command entered before the processing of a data set in the control stream has been completed by the current ZZBTH command. Only one data set may be processed at a time; while one is being processed, no further PARAM IN statements may be processed.	Defer reentry of the ZZBTH command until response to a ZZTCT command indicates that the current processing of a control stream data set is complete (7.6.3).
BATCH RESUMED	Normal response to successful ZZBTH RESUME command.	None required.
BATCH INITIATED	Normal response to successful start of batch processing with the ZZBTH command.	None required.
ZZBTH PARAMETER PROCESSING ERROR	Output to master terminal in response to a ZZBTH command when an input module is missing, the end of control stream input is accessed, and so forth.	Check for correct parameter information and reissue command.
BATCH PROCESSOR NOT IN PAUSE – COMMAND IGNORED	Output to master terminal in response to a ZZBTH RESUME command entered when the BTP is not in a pause state.	Do not reenter ZZBTH RESUME command until BTP is in a pause state: that is, until after receipt of normal response to ZZBTH PAUSE command.
BATCH PROCESSOR NOT RUNNING – COMMAND IGNORED	Output to master terminal in response to a ZZBTH PAUSE or ZZBTH CANCEL command when the BTP is not running or has completed processing.	None required.
BATCH PROCESSOR IN PAUSE STATE	Normal response to a successful entry of the ZZBTH PAUSE command.	None required. The only valid commands following the ZZBTH PAUSE commands are the ZZBTH RESUME or the ZZBTH CANCEL command.

Appendix A. Statement Conventions

Throughout this document, certain conventions are observed for formats for user-supplied statements and commands. General rules with examples pertaining to these conventions follow.

- Capital letters and punctuation marks (except braces, brackets, and ellipses) must be coded exactly as shown. For example:

  ```
  CALL 'GET' USING file-name record-area record-number.
  ```

  is coded:

  ```
  CALL 'GET' USING CUSTFIL CUS-REC REC-KEY.
  ```

- Lowercase words that are not underlined represent information to be supplied by the user. For example:

  ```
  ZZTCT terminal-id
  ```

  is entered on the terminal as:

  ```
  ZZTCT TRM1
  ```

- 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:

  ```
  ZG#CALL GET.(STATE,RECORD,SNKEY)
  ```

- Information within brackets [ ], including commas and semicolons, represents optional entries that are included or omitted, depending on program requirements. Braces within brackets indicate that one of the entries must be chosen if that operand is included. For example:

  $$\left[\text{JUS=}\begin{Bmatrix}\text{L}\\ \text{M}\end{Bmatrix}\right]$$

  is coded:

  ```
  JUS=L
  ```

- 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 for the designated file.

  $$\left[\text{TYP=}\begin{Bmatrix}\text{A}\\ \text{B}\\ \text{M}\\ \text{N}\\ \text{P}\end{Bmatrix}\right]$$

- An ellipsis (...) indicates the presence of a variable number of entries. For example:

  ```
  SWTCH△ terminal-id-n,... ;△△message-text
  ```

  is entered at the terminal as:

  ```
  SWTCH TRM1,TRM2,TRM4,TRM6;  THIS IS THE UPDATE FILE:
  ```

- A delta (△) indicates the presence of a space as shown in the example.

Statement conventions and coding rules specific to individual functions are described where applicable throughout this document.

Appendix B. UNIQUE Language Elements

The standard lexicon for UNIQUE is automatically defined at configuration time. This appendix lists the elements of this language and how they are used by UNIQUE.

Language Element	Description
Commands	
OPEN	Begins a dialog
CLOSE	Terminates a dialog
DISPLAY	Displays a record in a file
DELETE	Deletes a record in a file
OK	Finalizes update operation
CANCEL	Cancels current update operation
ADD	Inserts a record into a file
CHANGE	Changes a record in a file
NEXT	Processes the next record, as mentioned in preceding command
LIST	Lists records from a file
MORE	Continues processing previous LIST or DETAIL command output
DETAIL	Performs secondary listing operation
SHOW	Displays information about current dialog

Language Element	Description
Punctuation	
;	Semicolon. Used to delimit identifiers and LIST requests
'	Single quote or apostrophe. Used to indicate special character in identifiers
-	Ditto character. Used to copy parts of previous commands
,	Comma. Used to separate segments of identifiers and also used in the editing of numbers
=	Set symbol. Used to set data values, in update commands (hard copy format)
.	Decimal point. Used to indicate decimal places
()	Parentheses. Used in arithmetic expressions in LIST command
$	Dollar sign. Used to designate dollar values
CR	Credit symbol. Used to indicate a negative money amount
Logical Operators*	
AND	Intersection operator
OR	Union operator
NOT	Negation operator
EQ	Equals operator
NE	Not equals operator
GT	Greater-than operator
GE	Greater-than-or-equal-to operator
LT	Less-than operator
LE	Less-than-or-equal-to operator
Statistical Functions*	
TOTAL	Total operation
COUNT	Count operation
MAX	Maximum operation
MIN	Minimum operation
AVG	Average operation

*Used in LIST command

Language Element	Description
LIST Keywords	
FOR	Restricts output from LIST command
FROM	Specifies records to be listed
AFTER	Specifies records to be listed
IF	Initiates selection criteria in LIST command
ALL	Specifies records to be listed
Status Indications	
COMPLETE	Reports successful completion of a request
END	Indicates the end of the LIST or DETAIL output
NO	Used in error messages (e.g., NO RECORD)
PASSWORD	Used in error messages in response to OPEN command
INVALID	Used in error messages (e.g., INVALID COMMAND)
ILLEGAL	Used in error messages (e.g., ILLEGAL IDENT.)
COMMAND	Used in error messages
MESSAGE	Used in error messages
RECORD	Used in error messages
TOO BIG	Used in error messages (e.g., RECORD TOO BIG)
IDENT.	Used in error messages to convey information concerning identifiers
EOM	Used in error messages to indicate the end of some input was unexpected (e.g., INVALID EOM)
DATATYPE	Used in error messages
I/O	Used in error messages (e.g., I/O ERROR)
ERROR	Used in error messages
BAD	Used in error messages
DATA	Used in error messages to indicate an inconsistency between the value of an item and its description (e.g., BAD DATA)
EXISTS	Used in error messages

Appendix C. Sample IMS 90 Application

C.1. PREPARING AN RPG II APPLICATION

This appendix is the step-by-step procedure for preparing an IMS 90 application using an RPG II action program. The example action program (LSTLIM) accesses an ISAM file (STOCKS) and is activated by entering the transaction code STK at the terminal.

The processing steps used to prepare the system for online RPG II action program execution are as follows:

1. ICAM network generation
2. IMS 90 configuration
3. File space allocation
4. User logical file creation
5. RPG II action program compilation
6. RPG II action program linkage
7. IMS 90 execution

ICAM network generation must precede IMS 90 configuration. Action program compilation must precede linkage. All steps must be completed before IMS 90 execution, but otherwise, except as noted, may be done in any order.

In this sample application, steps 3 and 4 are assumed complete; i.e., the disk space has already been allocated and the user logical file has been created.

C.2. ICAM NETWORK GENERATION

The first step is to generate the ICAM network for the action program, LSTLIM. Figure C-1 shows the ICAM specifications for a resident transaction control interface (TCI) network.

```
1        10    16
COMMCT
TCI1     CCA   TYPE=(TCI),PASSWORD=RPGIMS9Ø,FEATURES=(TRACEMAX)
         BUFFERS 1Ø,128,1,ARP=35
LNE1     LINE  DEVICE=(UNISCOPE),TYPE=(24ØØ,SYNC,SWCH),ID=Ø8
TRM1     TERM  ADDR=(28,51),FEATURES=(U2ØØ),LOW=DQFILE1,MEDIUM=MAIN,HIGH=MAIN
TRM2     TERM  ADDR=(28,52),FEATURES=(U2ØØ),LOW=DQFILE1,MEDIUM=MAIN,HIGH=MAIN
TRM3     TERM  ADDR=(29,53),FEATURES=(U2ØØ),LOW=DQFILE1,MEDIUM=MAIN,HIGH=MAIN
TRM4     TERM  ADDR=(29,54),FEATURES=(U2ØØ),LOW=DQFILE1,MEDIUM=MAIN,HIGH=MAIN
PØØ1     PRCS  LOW=DQFILE1
DQFILE1  DISCFILE FILEDIV=8
TCIDTF   DISCFILE MSGSIZE=2Ø48
         ENDCCA
         MCP   MCPNAME=C5
               CACH=(Ø8,TCI1,Ø1)
END
// FIN
```

Figure C—1. ICAM Network Generation for RPG II Action Program, LSTLIM

The ICAM network name (CCA label) is TCI1. This name must agree with the CCA name specified in the NETWORK section of the IMS 90 configuration (Figure C-2). In addition, the password RPGIMS90 must also agree with the password indicated in the NETWORK section of the IMS 90 configuration.

The ICAM network (Figure C-1) describes one line with four UNISCOPE 200 terminals, each having three queues. The output queueing file is DQFILE1, and TCIDTF is the input buffer file. The name of the ICAM symbiont (MCPNAME) is C5, and the communications adapter port number of 08 is associated with this network in the CACH parameter.

The system console operator places the card deck shown in Figure C-2 in the card reader and types in the following commands one at a time:

```
RU SG$PARAM
RU SG$COMMK
```

After the SG$PARAM module is executed, the operator types in the second command to execute the SG$COMMK module. This produces the ICAM symbiont and places it in the $Y$LOD library.

C.3. IMS 90 CONFIGURATION

The second step is IMS 90 configuration. Figure C-2 shows the IMS 90 configuration used for the LSTLIM action program.

```
// JOB IMSCONF,,14ØØØ,14ØØØ
// IMSCONF ZCNF=MT,CCA=TCI1,LOADM=IS1,INIT=(I)
/&
// FIN
NETWORK  CONFID=4 NAME=TCI1 PASSWORD=RPGIMS90
GENERAL
OPTIONS  UNSOL=YES
FILE     STOCKS FILETYPE=ISAM
         BLKSIZE=87
         RECSIZE=8Ø
         IOROUT=ADDRTR
         KEYLEN=3
         KEYLOC=Ø
         IOREG=(8)
         KEYARG=STOCKS
         IOAREA1=STOCKS
         TYPEFLE=RANSEQ
TERMINAL TRM1  MASTER=YES
TERMINAL TRM2
TERMINAL TRM3
TERMINAL TRM4
ACTION   LSTLIM EDIT=# FILES=STOCKS INSIZE=27 OUTSIZE=96Ø
TRANSACT STK ACTION=LSTLIM
PROGRAM  LSTLIM ERET=YES
// FIN
```

Figure C—2. IMS 90 Configuration for RPG II Action Program, LSTLIM

The IMSCONF jproc calls five subordinate procedures that generate the control streams necessary to perform the IMS 90 configuration. The IMSCONF jproc specifies a multithread IMS 90 with a CCA named TCI1, an online IMS 90 load module named IS1, and file space initialized for NAMEREC, AUDFILE, and CONDATA IMS 90 files (INT=(I)).

The NETWORK section of this configuration identifies the IMS 90 system records for the NAMEREC file, and names the CCA network and password in agreement with the ICAM generation specifications (TCI1 and RPGIMS90).

The GENERAL section is optional.

The OPTIONS section specifies unsolicited output, i.e., the SEND function and SWTCH command can be used. RPG II uses the SEND function in the action program, LSTLIM, since multiple screens may be transmitted.

The FILE section describes the user logical file used by the RPG II action program, LSTLIM. The user logical file is STOCKS. This name must agree with the LFD card on the device assignment for the user logical file (see Figure C-9).

The TERMINAL section names four terminals with TRM1 as the master terminal. These names must agree with the terminal names on the ICAM generation (Figure C-1).

The action section names the RPG II action program. This name must agree with the linked load module name (Figure C-3). If you anticipate the transmission of multiple blanks in a screen of data, you should use the EDIT=c parameter in the ACTION section where c indicates some infrequently used special character. Otherwise, multiple blanks are removed by the IMS 90 editing process. Do not use EDIT=NONE. RPG II cannot process the incoming DICE codes. The user logical file, STOCKS, is to be referenced and used by the LSTLIM action program. The INSIZE parameter must be used for all RPG,II action programs and the value must be equal to the record size specified on the file description specifications form for the IMA.

The TRANSACT section specifies the transaction code of STK, which must agree with the transaction code keyed in by the terminal operator at RPG II action program execution time.

Finally, the PROGRAM section names the action program and specifies that control is returned to LSTLIM if an error occurs.

C.4. COMPILING AND LINKING THE RPG II ACTION PROGRAM

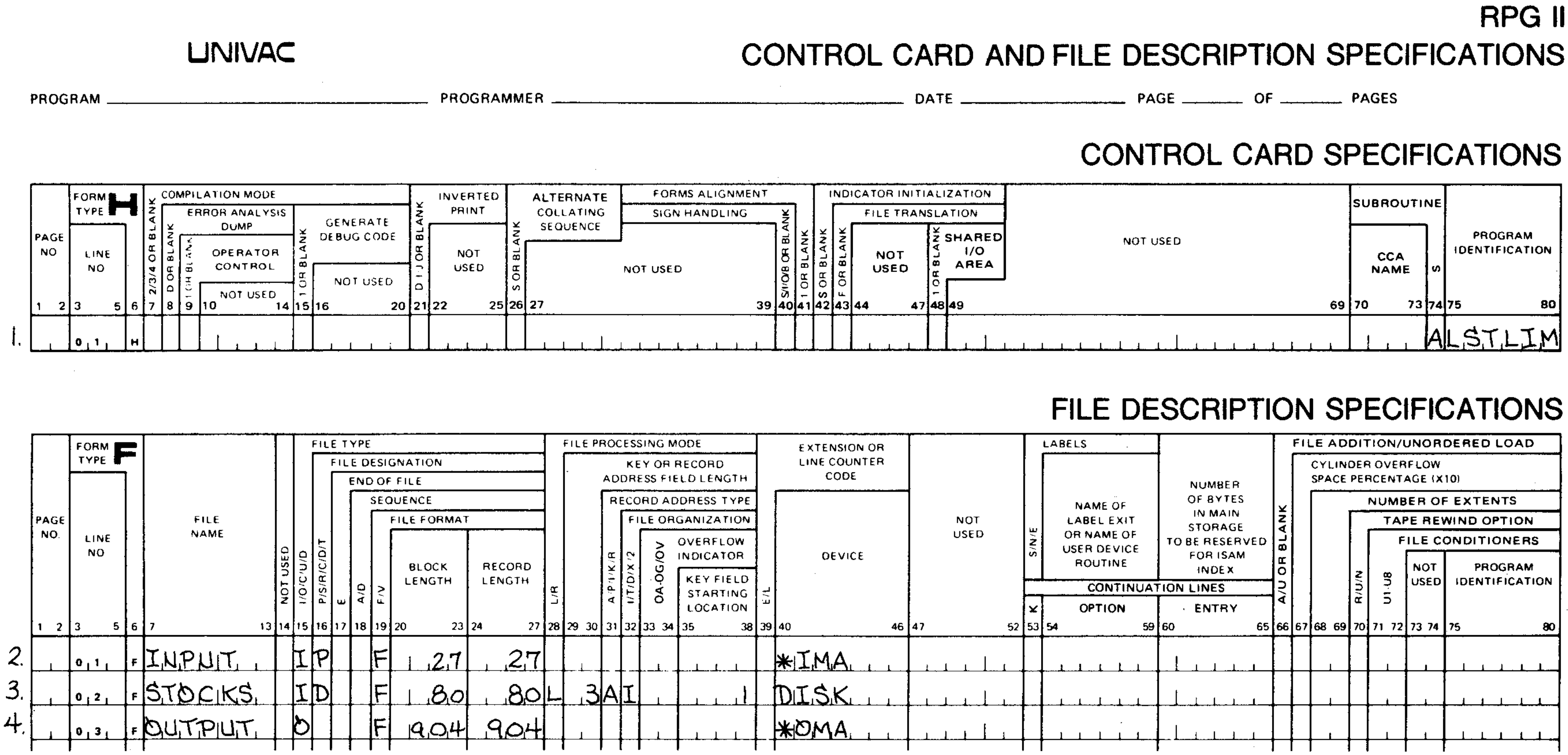
To compile and link the RPG II action program, use the simple RPG and LINK jprocs shown in Figure C-3.

```
// JOB RPGACT
// RPG
/$
 .
 .  (action program)
 .
/*
// LINK LSTLIM,OUT=(RES,$Y$LOD)
/&
// FIN
```

Figure C—3. Compile and Link for RPG II Action Program, LSTLIM

Note that in linking the RPG II action program object module, the name of the load library where the action program load module is stored ($Y$LOD) must agree with the library name ($Y$LOD by default) on the LBL parameter of the IMSCONF jproc.

Figure C-4 through C-8 show the specifications forms required to code action program LSTLIM, which accesses the STOCKS file for records within the lower and upper limits keyed in by the terminal operator. A line-by-line description also is provided.



UNIVAC

RPG II
CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE ____ OF ____ PAGES

CONTROL CARD SPECIFICATIONS

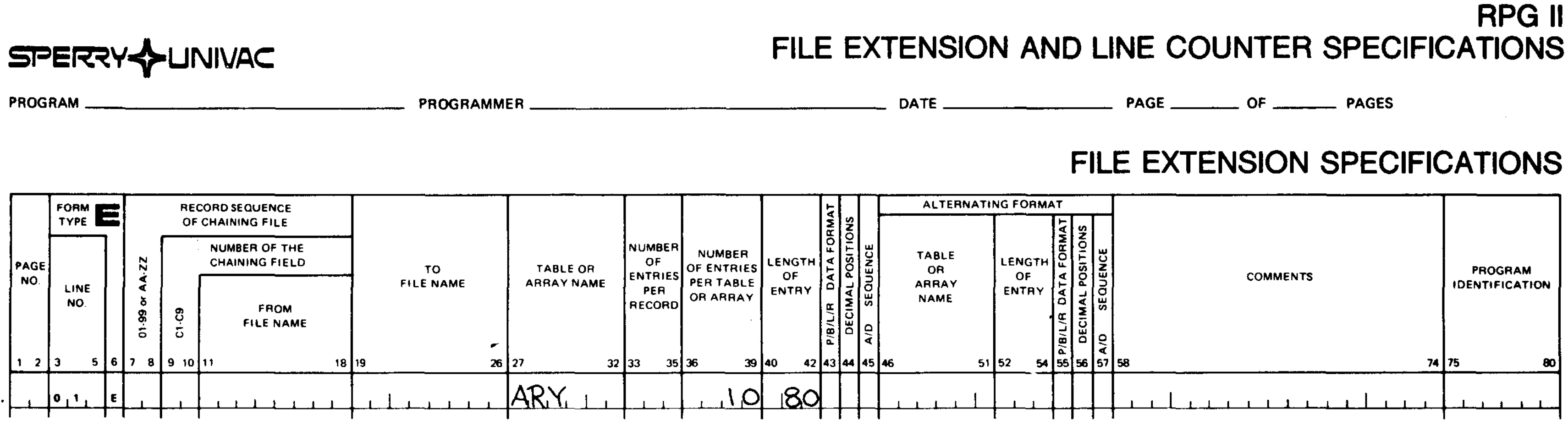
	PAGE NO 1-2	LINE NO 3-5	FORM TYPE H 6	2/3/4 OR BLANK 7	D OR BLANK 8	9	OPERATOR CONTROL NOT USED 10-14	1 OR BLANK 15	GENERATE DEBUG CODE NOT USED 16-20	D I J OR BLANK 21	INVERTED PRINT NOT USED 22-25	S OR BLANK 26	ALTERNATE COLLATING SEQUENCE / FORMS ALIGNMENT / SIGN HANDLING NOT USED 27-39	40	1 OR BLANK 41	S OR BLANK 42	F OR BLANK 43	FILE TRANSLATION NOT USED 44-47	1 OR BLANK 48	SHARED I/O AREA / NOT USED 49-69	SUBROUTINE CCA NAME 70-73	S 74	PROGRAM IDENTIFICATION 75-80
1.		01	H																			A	LSTLIM

FILE DESCRIPTION SPECIFICATIONS

	PAGE NO. 1-2	LINE NO 3-5	FORM TYPE F 6	FILE NAME 7-13	NOT USED 14	FILE TYPE I/O/C/U/D 15	FILE DESIGNATION P/S/R/C/D/T 16	END OF FILE E 17	SEQUENCE A/D 18	FILE FORMAT F/V 19	BLOCK LENGTH 20-23	RECORD LENGTH 24-27	FILE PROCESSING MODE L/R 28	KEY OR RECORD ADDRESS FIELD LENGTH 29-30	RECORD ADDRESS TYPE A/P/K/R 31	FILE ORGANIZATION I/T/D/X/2 32	OVERFLOW INDICATOR OA-OG/OV 33-34	KEY FIELD STARTING LOCATION 35-38	EXTENSION OR LINE COUNTER CODE E/L 39	DEVICE 40-46	NOT USED 47-52	LABELS S/N/E 53	NAME OF LABEL EXIT OR NAME OF USER DEVICE ROUTINE 54-59	NUMBER OF BYTES IN MAIN STORAGE TO BE RESERVED FOR ISAM INDEX 60-65	A/U OR BLANK 66	67	CYLINDER OVERFLOW SPACE PERCENTAGE (X10) 68-69	TAPE REWIND OPTION R/U/N 70	U1-U8 71-72	NOT USED 73-74	PROGRAM IDENTIFICATION 75-80
2.		01	F	INPUT		I	P			F	27	27								*IMA											
3.		02	F	STOCKS		I	D			F	80	80	L	3	A	I		1		DISK											
4.		03	F	OUTPUT		O				F	904	904								*OMA											

CONTINUATION LINES: K 53 | OPTION 54-59 | ENTRY 60-65

Figure C—4. LSTLIM Control Card and File Description Specifications Forms



SPERRY✦UNIVAC

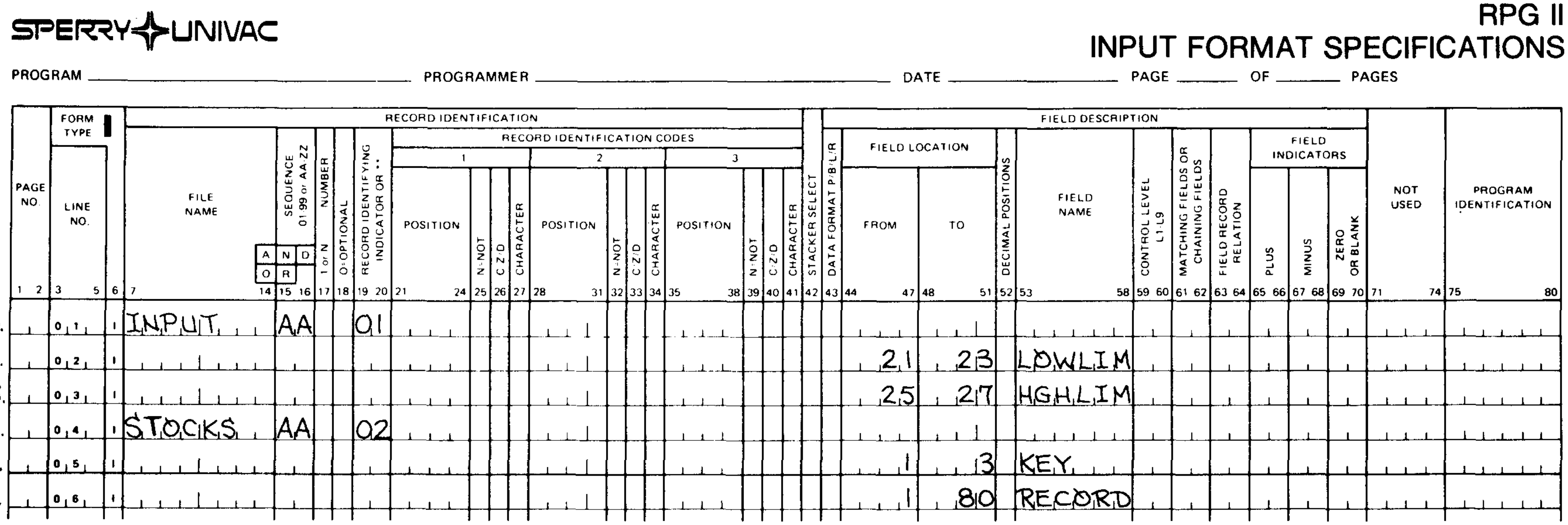
RPG II
FILE EXTENSION AND LINE COUNTER SPECIFICATIONS

PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE ____ OF ____ PAGES

FILE EXTENSION SPECIFICATIONS

	PAGE NO. 1-2	FORM TYPE E LINE NO. 3-5	6	RECORD SEQUENCE OF CHAINING FILE 01-99 or AA-ZZ 7-8	NUMBER OF THE CHAINING FIELD C1-C9 9-10	FROM FILE NAME 11-18	TO FILE NAME 19-26	TABLE OR ARRAY NAME 27-32	NUMBER OF ENTRIES PER RECORD 33-35	NUMBER OF ENTRIES PER TABLE OR ARRAY 36-39	LENGTH OF ENTRY 40-42	P/B/L/R DATA FORMAT 43	DECIMAL POSITIONS 44	A/D SEQUENCE 45	ALTERNATING FORMAT TABLE OR ARRAY NAME 46-51	ALTERNATING FORMAT LENGTH OF ENTRY 52-54	ALTERNATING FORMAT P/B/L/R DATA FORMAT 55	ALTERNATING FORMAT DECIMAL POSITIONS 56	ALTERNATING FORMAT A/D SEQUENCE 57	COMMENTS 58-74	PROGRAM IDENTIFICATION 75-80
5.		01	E					ARY		10	80										

Figure C—5. LSTLIM File Extension Specifications Form



SPERRY✦UNIVAC

RPG II
INPUT FORMAT SPECIFICATIONS

PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE ____ OF ____ PAGES

	PAGE NO 1-2	FORM TYPE I LINE NO. 3-5	6	FILE NAME 7-14	SEQUENCE 01-99 or AA-ZZ 15-16 (AND / OR)	NUMBER 1 or N 17	O: OPTIONAL 18	RECORD IDENTIFYING INDICATOR OR ** 19-20	RECORD IDENTIFICATION CODES 1 POSITION 21-24	N: NOT 25	C/Z/D 26	CHARACTER 27	2 POSITION 28-31	N: NOT 32	C/Z/D 33	CHARACTER 34	3 POSITION 35-38	N: NOT 39	C/Z/D 40	CHARACTER 41	STACKER SELECT 42	DATA FORMAT P/B/L/R 43	FIELD LOCATION FROM 44-47	FIELD LOCATION TO 48-51	DECIMAL POSITIONS 52	FIELD NAME 53-58	CONTROL LEVEL L1-L9 59-60	MATCHING FIELDS OR CHAINING FIELDS 61-62	FIELD RECORD RELATION 63-64	FIELD INDICATORS PLUS 65-66	FIELD INDICATORS MINUS 67-68	FIELD INDICATORS ZERO OR BLANK 69-70	NOT USED 71-74	PROGRAM IDENTIFICATION 75-80
6.		01	I	INPUT	AA			01																										
7.		02	I																				21	23		LOWLIM								
8.		03	I																				25	27		HGHLIM								
9.		04	I	STOCKS	AA			02																										
10.		05	I																				1	3		KEY								
11.		06	I																				1	80		RECORD								

Figure C—6. LSTLIM Input Format Specifications Form

Figure C-4:

LIne Explanation

1 The program is an action program named LSTLIM.

2 The primary input file is INPUT and is the IMS 90 IMA consisting of one record 27 bytes long.

3 STOCKS is an indexed disk logical file processed by limits using the SETLL operation (see Figure C-7, Line 12).

4 Output will go to the IMS 90 OMA, 904 bytes long, including ten 80-byte records and 104 bytes for headers and control characters.

Figure C-5:

Line Explanation

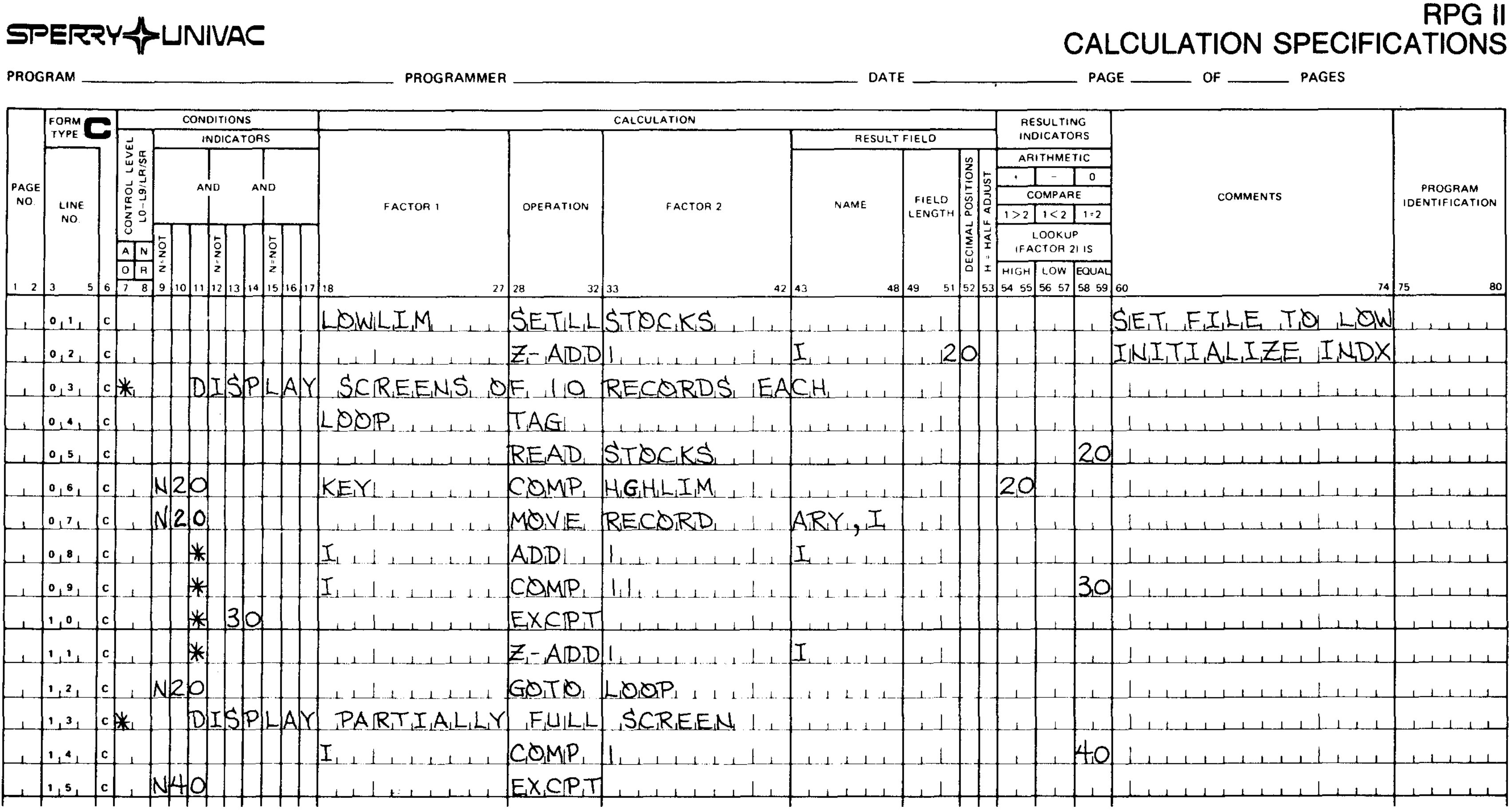
5 ARY is defined as an array having 10 entries of 80 characters each.

Figure C-6:

Line Explanation

6-8 The input record format is defined. The first 16 bytes contain the IMA header. Bytes 17 through 20 contain the 3-character transaction code (STK) followed by a blank. This transaction code must also be specified in the TRANSACT section of the IMS 90 configuration (Figure C-2). The header information and transaction code are not referenced in this action program. Bytes 21, 22, and 23 contain the starting stock symbol (lower limit of the search). Bytes 25, 26, and 27 contain the ending stock symbol (upper limit of the search).

9-11 The logical file disk records are each 80 bytes long with a key (the stock symbol) in the first three bytes.



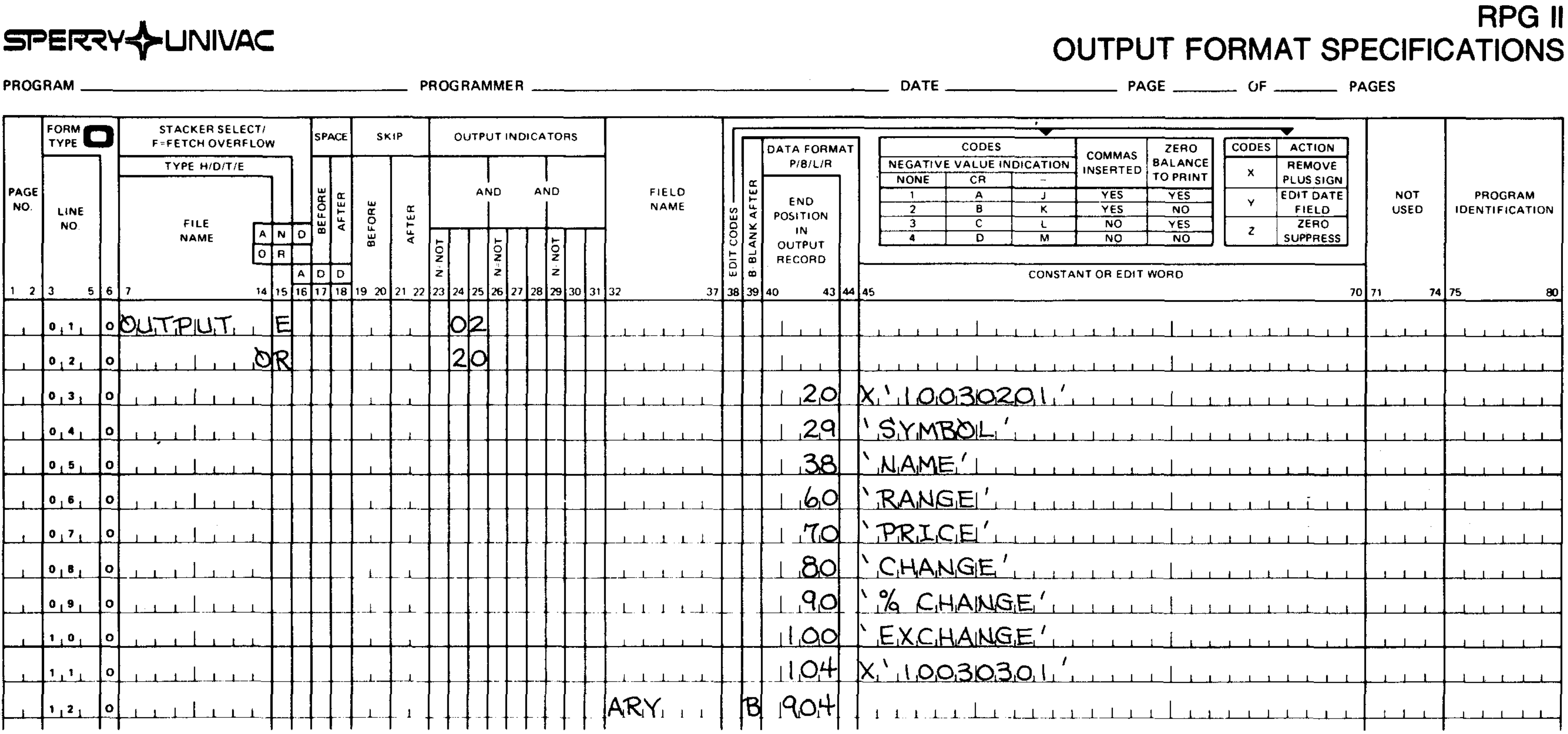
SPERRY UNIVAC

RPG II
CALCULATION SPECIFICATIONS

PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE ____ OF ____ PAGES

	PAGE NO 1–2	LINE NO 3–5	FORM TYPE C 6	CONTROL LEVEL (L0–L9/LR/SR) 7–8	INDICATORS 9–11	AND 12–14	AND 15–17	FACTOR 1 18–27	OPERATION 28–32	FACTOR 2 33–42	RESULT FIELD NAME 43–48	FIELD LENGTH 49–51	DECIMAL POSITIONS 52	H = HALF ADJUST 53	HIGH (1>2) 54–55	LOW (1<2) 56–57	EQUAL (1=2) 58–59	COMMENTS 60–74	PROGRAM IDENTIFICATION 75–80
12.		01	C					LOWLIM	SETLL	STOCKS								SET FILE TO LOW	
13.		02	C						Z-ADD	1	I	2	0					INITIALIZE INDX	
14.		03	C*		DISPLAY SCREENS OF 10 RECORDS EACH														
15.		04	C					LOOP	TAG										
16.		05	C						READ	STOCKS							20		
17.		06	C		N20			KEY	COMP	HGHLIM					20				
18.		07	C		N20				MOVE	RECORD	ARY,I								
19.		08	C		*			I	ADD	1	I								
20.		09	C		*			I	COMP	11							30		
21.		10	C		*	30			EXCPT										
22.		11	C		*				Z-ADD	1	I								
23.		12	C		N20				GOTO	LOOP									
24.		13	C*		DISPLAY PARTIALLY FULL SCREEN														
25.		14	C					I	COMP	1							40		
26.		15	C		N40				EXCPT										

Figure C—7. LSTLIM Calculation Specifications Form



SPERRY UNIVAC

RPG II
OUTPUT FORMAT SPECIFICATIONS

PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE ____ OF ____ PAGES

CODES	NEGATIVE VALUE INDICATION: NONE	CR	–	COMMAS INSERTED	ZERO BALANCE TO PRINT
	1	A	J	YES	YES
	2	B	K	YES	NO
	3	C	L	NO	YES
	4	D	M	NO	NO

CODES	ACTION
X	REMOVE PLUS SIGN
Y	EDIT DATE FIELD
Z	ZERO SUPPRESS

	PAGE NO. (1-2)	LINE NO. (3-5)	FORM TYPE (6)	FILE NAME (7-14)	TYPE H/D/T/E (15)	AND/OR (14-16)	SPACE BEFORE (17)	SPACE AFTER (18)	SKIP BEFORE (19-20)	SKIP AFTER (21-22)	OUTPUT INDICATORS (23-31)	FIELD NAME (32-37)	EDIT CODES (38)	B BLANK AFTER (39)	END POSITION IN OUTPUT RECORD (40-43)	DATA FORMAT P/B/L/R (44)	CONSTANT OR EDIT WORD (45-70)	NOT USED (71-74)	PROGRAM IDENTIFICATION (75-80)
27.		01	O	OUTPUT	E						02								
28.		02	O			OR					20								
29.		03	O												20		X'10030201'		
30.		04	O												29		'SYMBOL'		
31.		05	O												38		'NAME'		
32.		06	O												60		'RANGE'		
33.		07	O												70		'PRICE'		
34.		08	O												80		'CHANGE'		
35.		09	O												90		'% CHANGE'		
36.		10	O												100		'EXCHANGE'		
37.		11	O												104		X'10030301'		
38.		12	O									ARY		B	904				

Figure C—8. LSTLIM Output Format Specifications Form

Figure C-7:

Line Explanation

12 The starting point is set to the lower limit entered by the terminal operator.

13 The array index is set to 1.

16 A record is read from the disk file. If end-of-file has been reached, indicator 20 is set on.

17 If the key is beyond the upper limit entered by the terminal operator, indicator 20 is set on.

18 A record is moved to the array.

19 The array index is incremented.

20 If the array is full, indicator 30 is set on.

21, 22 If the array is full, it is displayed using the EXCPT operation and the index is reset to 1.

23 If there are more records to process, the program returns to LOOP.

25, 26 If there are no more records to process, a test is made to determine whether the array is partially full. If it contains any records, the last screen is displayed.

Figure C-8:

Line Explanation

27, 28 The output record is defined. Exception output is used where the array is full or when a last partial screen is reached.

29 The first 16 positions contain the OMA header, which is unreferenced in this program. Positions 17–20 contain a DICE output control functions that positions the cursor at line 2, position 1. (See the current version of the ICAM user guide, UP-8194).

30–36 Heading information is displayed.

37 The cursor is positioned at line 3, position 1.

38 The 800-character array is displayed using "blank after".

C.5. IMS 90 EXECUTION

Finally, the IMS 90 load module IS1 is executed. Figure C-9 shows the job stream required to execute IMS 90 (IMS 90 start-up). Do not confuse this execution of the configured IMS 90 load module with execution of the RPG II action program. Remember, the action program LSTLIM is executed whenever the transaction code STK and new lower and upper limits are keyed in at the terminal for a search of the STOCKS file.

NOTE:

The operator must load ICAM by typing in the MCPNAME (C5 in this case) on the system console before executing the IMS 90 load module.

The first device assignments are for the printer, IMS 90 files, and user logical file. Note that after the first run the EXT cards should be removed. The next device assignments are for allocating space for disk queue files. Ultimately, the EXEC statement executes the configured IMS 90 load module IS1. Note also that the name on the EXEC statement must agree with the name on the LOADM parameter of the IMSCONF jproc.

```
     // JOB IMS,,14ØØØ,14ØØØ,4
     // DVC 2Ø // LFD PRNTR
     // DVC 64 // VOL RELØ5Ø // LBL NAMEREC // LFD NAMEREC
     // DVC 64 // VOL RELØ5Ø // LBL AUDFILE // LFD AUDFILE
     // DVC 64 // VOL RELØ5Ø // LBL CONDATA // LFD CONDATA
     // DVC 64 // VOL RELØ5Ø // LBL DISC1 // LFD STOCKS
     // DVC 64 // VOL RELØ5Ø
*    // EXT ST,C,,CYL,5
     // LBL DQFILE1 // LFD DQFILE1,,INIT
     // DVC 64 // VOL RELØ5Ø
*    // EXT ST,C,O,BLK,(256,6ØØ)
     // LBL TCIDTF // LFD TCIDTF,2,INIT
     // OPTION DUMP
     // EXEC IS1
     /&
     // FIN
```

*Remove this card after first run.

Figure C—9. Execution of Configured IMS 90

C.6. EXECUTING THE RPG II ACTION PROGRAM

By following this step-by-step process from ICAM network generation to IMS 90 execution, the user prepares the system for the online execution of RPG II action program.

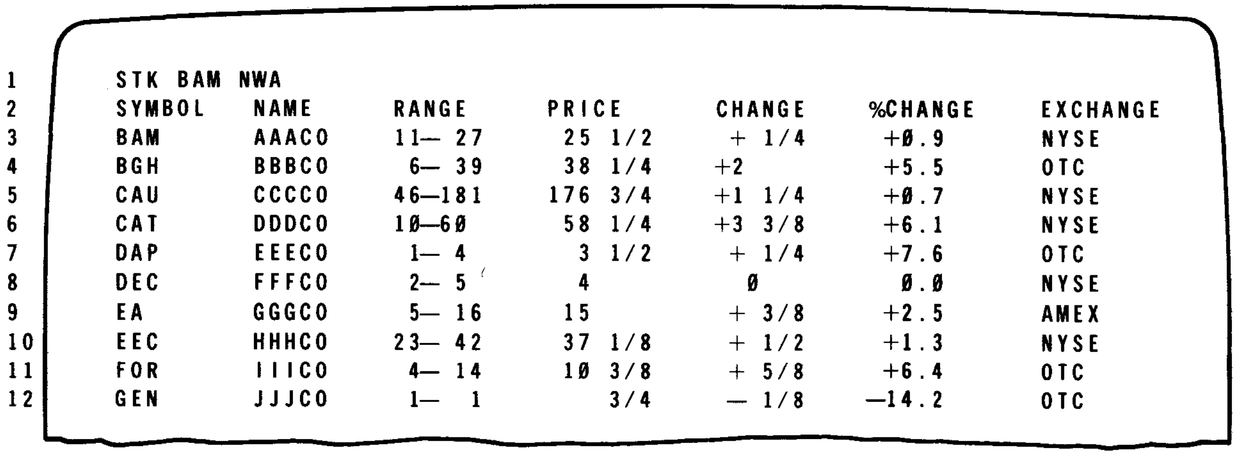
The RPG II action program LSTLIM displays the records of an indexed user logical file between the limits specified by the terminal operator. Each UNISCOPE 200 screen contains 10 records except the last screen, which may contain fewer records.

LSTLIM references two IMS 90 interface areas, the IMA and OMA, as well as one user logical disk file, STOCKS, configured as an ISAM sequential file. The STOCKS file contains stock market information which is accessed by a 3-character stock symbol (alphanumeric key).

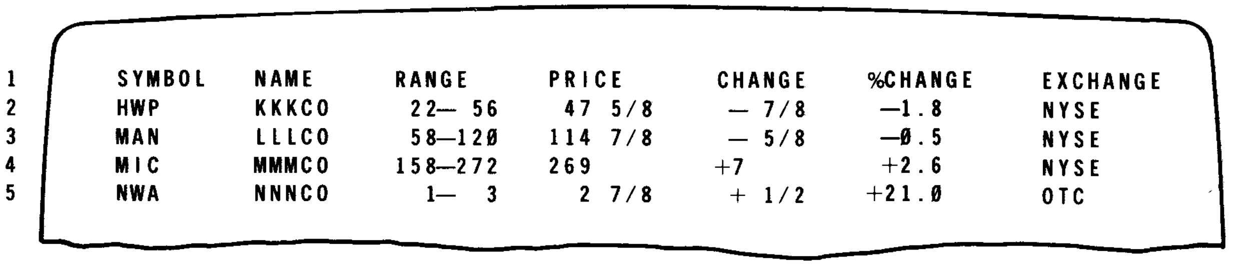
The terminal operator executes the LSTLIM action program when he enters the configured transaction code (STK) followed by a 3-character key for the lower limit and a 3-character key for the upper limit. Records from the STOCKS file within the requested limits are displayed in sequence by key, 10 at a time, until the upper limit is reached.

Figure C-10 shows the user's transaction code, STK, the lower and upper key limits requested from the STOCKS logical file (line 1), and the 10-record screen display from a full array (lines 3-12, screen 1) and from a partially filled array (lines 2-5, screen 2).

When more records exist for the specified key range, the MESSAGE WAIT indicator lights at the terminal, and the user must press the MESSAGE WAITING key to receive the additional records within the specified limits.



```
1   STK BAM NWA
2   SYMBOL  NAME   RANGE    PRICE     CHANGE   %CHANGE  EXCHANGE
3   BAM     AAACO  11— 27    25 1/2    + 1/4    +Ø.9    NYSE
4   BGH     BBBCO   6— 39    38 1/4   +2        +5.5    OTC
5   CAU     CCCCO  46—181   176 3/4   +1 1/4    +Ø.7    NYSE
6   CAT     DDDCO  1Ø—6Ø     58 1/4   +3 3/8    +6.1    NYSE
7   DAP     EEECO   1— 4      3 1/2    + 1/4    +7.6    OTC
8   DEC     FFFCO   2— 5      4          Ø       Ø.Ø    NYSE
9   EA      GGGCO   5— 16    15        + 3/8    +2.5    AMEX
10  EEC     HHHCO  23— 42    37 1/8    + 1/2    +1.3    NYSE
11  FOR     IIICO   4— 14    1Ø 3/8    + 5/8    +6.4    OTC
12  GEN     JJJCO   1— 1        3/4    — 1/8   —14.2    OTC
```



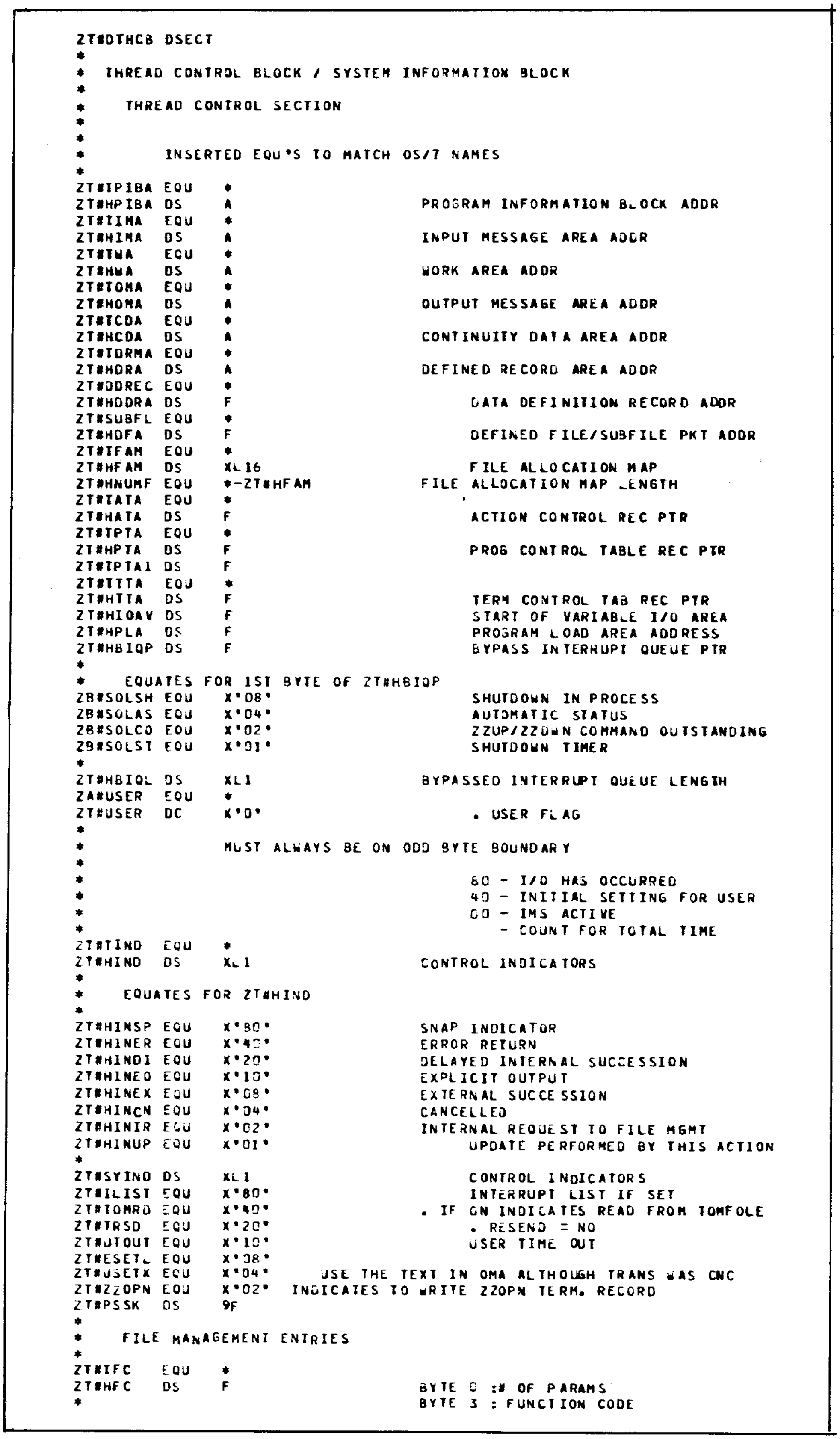
```
1   SYMBOL  NAME   RANGE    PRICE     CHANGE   %CHANGE  EXCHANGE
2   HWP     KKKCO   22— 56   47 5/8    — 7/8    —1.8    NYSE
3   MAN     LLLCO   58—12Ø  114 7/8    — 5/8    —Ø.5    NYSE
4   MIC     MMMCO  158—272  269       +7        +2.6    NYSE
5   NWA     NNNCO    1— 3     2 7/8    + 1/2   +21.Ø    OTC
```

Figure C—10. Sample Screen Displays of Simple Transaction Requesting Records From STOCKS File

Appendix D. IMS 90 Internal Tables

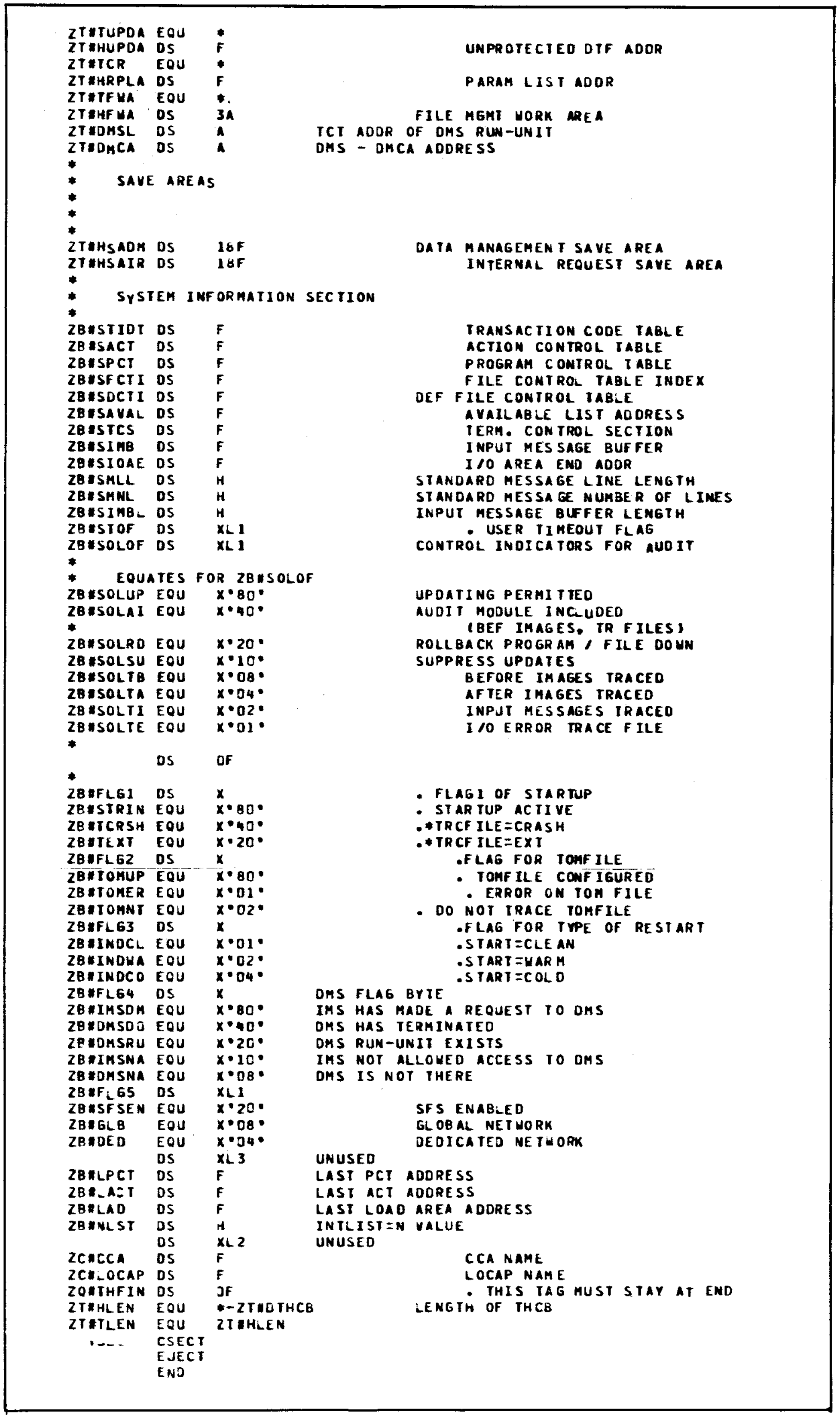
The following single-thread and multithread IMS 90 internal thread control blocks (THCB) and terminal control tables (TCT), Figures D–1 through D–3, may be used with an edited snap dump to further assist the user in determining and solving software problems.

The snap dump lists areas such as the PIB, IMA, WA, OMA, CDA, THCB, and TCT. By examining the following figures, the user can associate the single-thread or multithread DSECT for thread control blocks or terminal control tables with those areas shown in the snap dump.



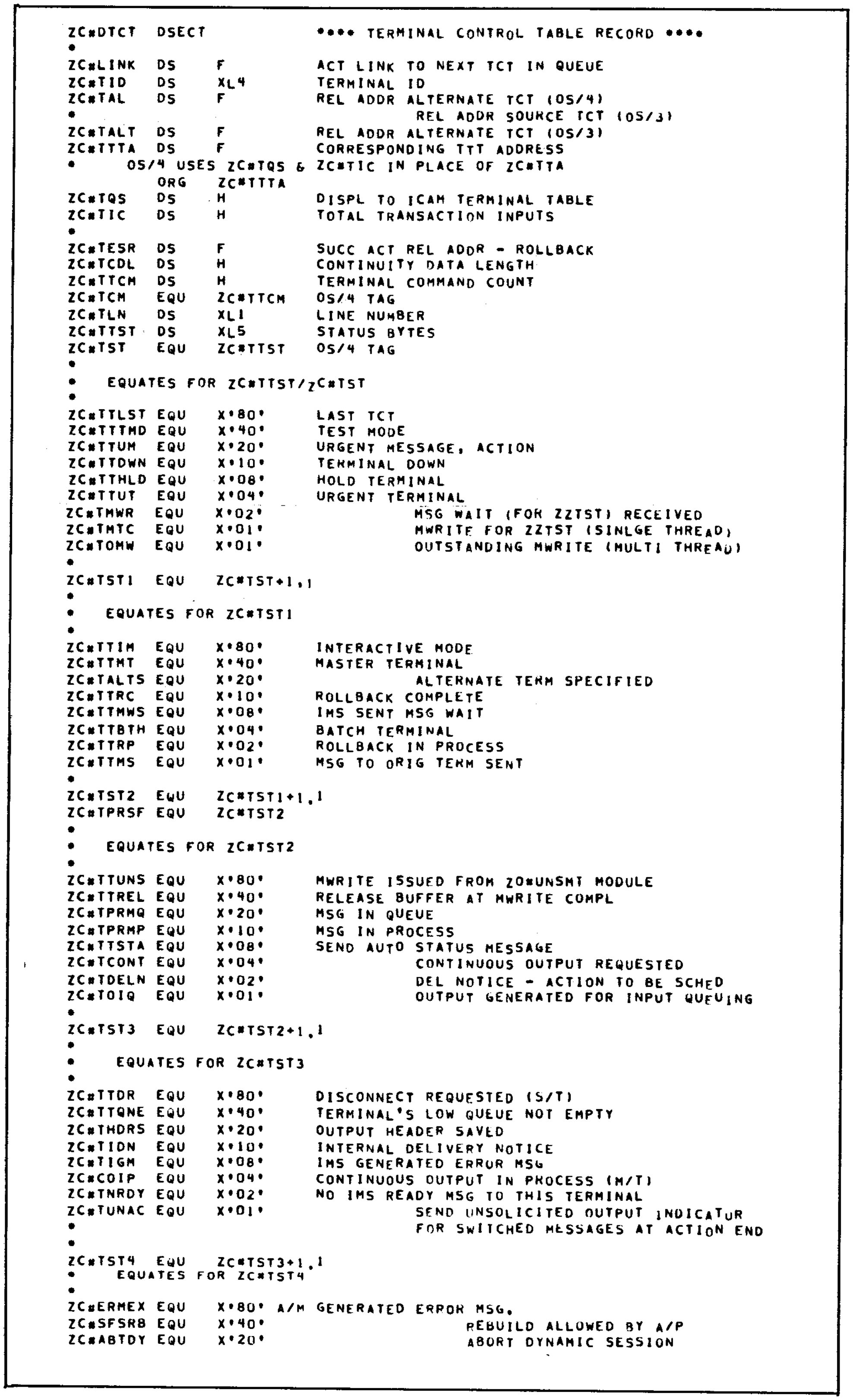
```
ZT#DTHCB DSECT
*
*  THREAD CONTROL BLOCK / SYSTEM INFORMATION BLOCK
*
*    THREAD CONTROL SECTION
*
*
*        INSERTED EQU'S TO MATCH OS/7 NAMES
*
ZT#IPIBA EQU   *
ZT#HPIBA DS    A                  PROGRAM INFORMATION BLOCK ADDR
ZT#TIMA  EQU   *
ZT#HIMA  DS    A                  INPUT MESSAGE AREA ADDR
ZT#TWA   EQU   *
ZT#HWA   DS    A                  WORK AREA ADDR
ZT#TOMA  EQU   *
ZT#HOMA  DS    A                  OUTPUT MESSAGE  AREA ADDR
ZT#TCDA  EQU   *
ZT#HCDA  DS    A                  CONTINUITY DATA AREA ADDR
ZT#TDRMA EQU   *
ZT#HDRA  DS    A                  DEFINED RECORD AREA ADDR
ZT#DDREC EQU   *
ZT#HDDRA DS    F                       DATA DEFINITION RECORD ADDR
ZT#SUBFL EQU   *
ZT#HDFA  DS    F                       DEFINED FILE/SUBFILE PKT ADDR
ZT#TFAM  EQU   *
ZT#HFAM  DS    XL16                    FILE ALLOCATION MAP
ZT#HNUMF EQU   *-ZT#HFAM          FILE ALLOCATION MAP LENGTH
ZT#TATA  EQU   *
ZT#HATA  DS    F                       ACTION CONTROL REC PTR
ZT#TPTA  EQU   *
ZT#HPTA  DS    F                       PROG CONTROL TABLE REC PTR
ZT#TPTA1 DS    F
ZT#TTTA  EQU   *
ZT#HTTA  DS    F                       TERM CONTROL TAB REC PTR
ZT#HIOAV DS    F                       START OF VARIABLE I/O AREA
ZT#HPLA  DS    F                       PROGRAM LOAD AREA ADDRESS
ZT#HBIQP DS    F                       BYPASS INTERRUPT QUEUE PTR
*
*    EQUATES FOR 1ST BYTE OF ZT#HBIQP
ZB#SOLSH EQU   X'08'                   SHUTDOWN IN PROCESS
ZB#SOLAS EQU   X'04'                   AUTOMATIC STATUS
ZB#SOLCO EQU   X'02'                   ZZUP/ZZDWN COMMAND OUTSTANDING
ZB#SOLST EQU   X'01'                   SHUTDOWN TIMER
*
ZT#HBIQL DS    XL1                BYPASSED INTERRUPT QUEUE LENGTH
ZA#USER  EQU   *
ZT#USER  DC    X'0'                    . USER FLAG
*
*              MUST ALWAYS BE ON ODD BYTE BOUNDARY
*
*                                      80 - I/O HAS OCCURRED
*                                      40 - INITIAL SETTING FOR USER
*                                      00 - IMS ACTIVE
*                                         - COUNT FOR TOTAL TIME
ZT#TIND  EQU   *
ZT#HIND  DS    XL1                CONTROL INDICATORS
*
*    EQUATES FOR ZT#HIND
*
ZT#HINSP EQU   X'80'              SNAP INDICATOR
ZT#HINER EQU   X'40'              ERROR RETURN
ZT#HINDI EQU   X'20'              DELAYED INTERNAL SUCCESSION
ZT#HINEO EQU   X'10'              EXPLICIT OUTPUT
ZT#HINEX EQU   X'08'              EXTERNAL SUCCESSION
ZT#HINCN EQU   X'04'              CANCELLED
ZT#HINIR EQU   X'02'              INTERNAL REQUEST TO FILE MGMT
ZT#HINUP EQU   X'01'                   UPDATE PERFORMED BY THIS ACTION
*
ZT#SYIND DS    XL1                     CONTROL INDICATORS
ZT#ILIST EQU   X'80'                   INTERRUPT LIST IF SET
ZT#TOMRD EQU   X'40'              . IF ON INDICATES READ FROM TOMFOLE
ZT#TRSD  EQU   X'20'                   . RESEND = NO
ZT#UTOUT EQU   X'10'                   USER TIME OUT
ZT#ESETL EQU   X'08'
ZT#USETX EQU   X'04'      USE THE TEXT IN OMA ALTHOUGH TRANS WAS CNC
ZT#ZZOPN EQU   X'02'  INDICATES TO WRITE ZZOPN TERM. RECORD
ZT#PSSK  DS    9F
*
*    FILE MANAGEMENT ENTRIES
*
ZT#IFC   EQU   *
ZT#HFC   DS    F                  BYTE 0 :# OF PARAMS
*                                 BYTE 3 : FUNCTION CODE
```

Figure D—1. Single-thread Thread Control Block (THCB) (Part 1 of 2)



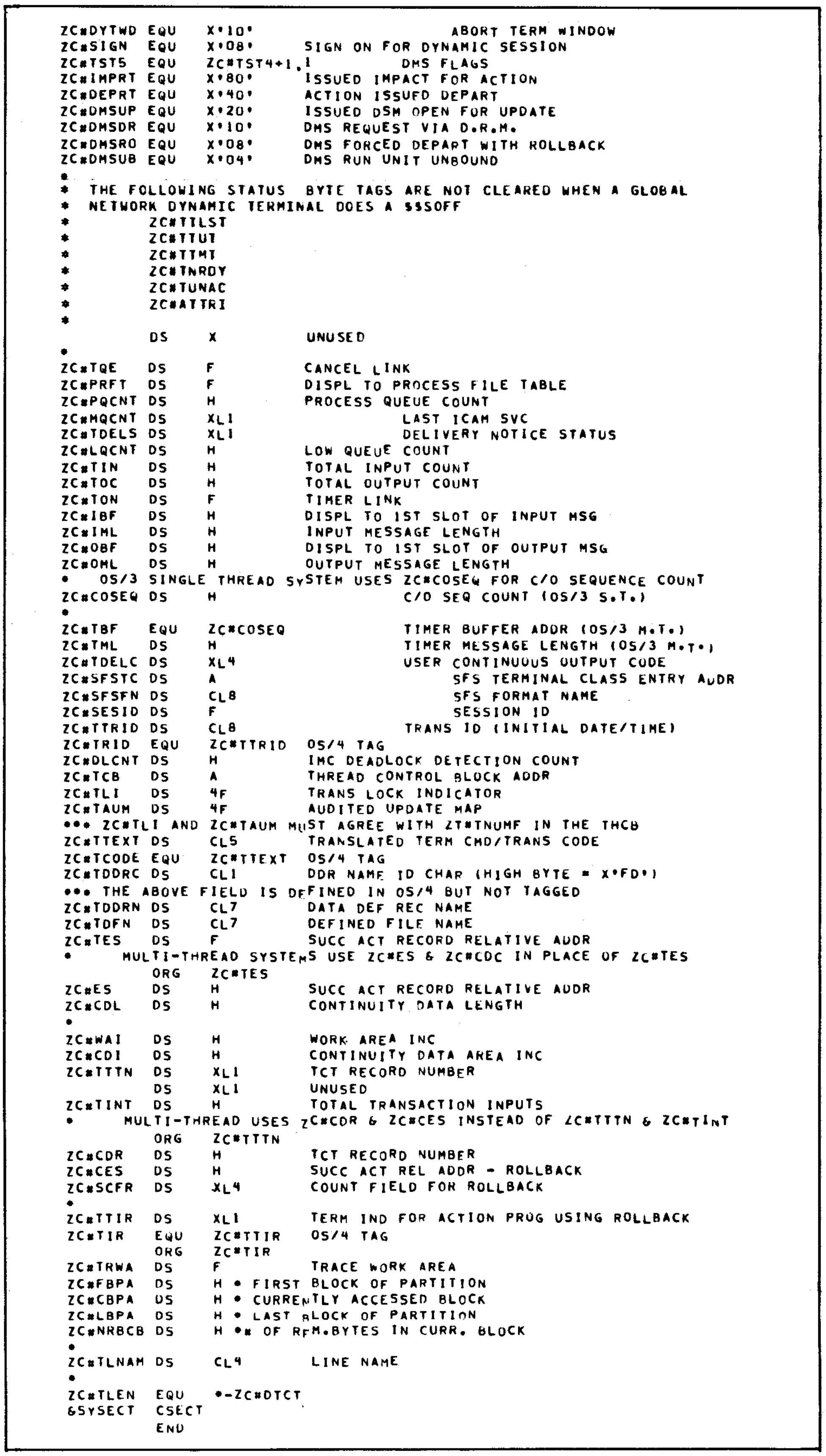
```
ZT#TUPDA EQU   *
ZT#HUPDA DS    F                        UNPROTECTED DTF ADDR
ZT#TCR   EQU   *
ZT#HRPLA DS    F                        PARAM LIST ADDR
ZT#TFWA  EQU   *.
ZT#HFWA  DS    3A                  FILE MGMT WORK AREA
ZT#DMSL  DS    A         TCT ADDR OF DMS RUN-UNIT
ZT#DMCA  DS    A         DMS - DMCA ADDRESS
*
*    SAVE AREAS
*
*
*
ZT#HSADM DS    18F                 DATA MANAGEMENT SAVE AREA
ZT#HSAIR DS    18F                      INTERNAL REQUEST SAVE AREA
*
*    SYSTEM INFORMATION SECTION
*
ZB#STIDT DS    F                        TRANSACTION CODE TABLE
ZB#SACT  DS    F                        ACTION CONTROL TABLE
ZB#SPCT  DS    F                        PROGRAM CONTROL TABLE
ZB#SFCTI DS    F                        FILE CONTROL TABLE INDEX
ZB#SDCTI DS    F                   DEF FILE CONTROL TABLE
ZB#SAVAL DS    F                        AVAILABLE LIST ADDRESS
ZB#STCS  DS    F                        TERM. CONTROL SECTION
ZB#SIMB  DS    F                        INPUT MESSAGE BUFFER
ZB#SIOAE DS    F                        I/O AREA END ADDR
ZB#SMLL  DS    H                   STANDARD MESSAGE LINE LENGTH
ZB#SMNL  DS    H                   STANDARD MESSAGE NUMBER OF LINES
ZB#SIMBL DS    H                   INPUT MESSAGE BUFFER LENGTH
ZB#STOF  DS    XL1                      . USER TIMEOUT FLAG
ZB#SOLOF DS    XL1                 CONTROL INDICATORS FOR AUDIT
*
*    EQUATES FOR ZB#SOLOF
ZB#SOLUP EQU   X'80'               UPDATING PERMITTED
ZB#SOLAI EQU   X'40'               AUDIT MODULE INCLUDED
*                                       (BEF IMAGES, TR FILES)
ZB#SOLRD EQU   X'20'               ROLLBACK PROGRAM / FILE DOWN
ZB#SOLSU EQU   X'10'               SUPPRESS UPDATES
ZB#SOLTB EQU   X'08'                    BEFORE IMAGES TRACED
ZB#SOLTA EQU   X'04'                    AFTER IMAGES TRACED
ZB#SOLTI EQU   X'02'                    INPJT MESSAGES TRACED
ZB#SOLTE EQU   X'01'                    I/O ERROR TRACE FILE
*
         DS    0F
*
ZB#FLG1  DS    X                   . FLAG1 OF STARTUP
ZB#STRIN EQU   X'80'               . STARTUP ACTIVE
ZB#TCRSH EQU   X'40'               .*TRCFILE=CRASH
ZB#TEXT  EQU   X'20'               .*TRCFILE=EXT
ZB#FLG2  DS    X                       .FLAG FOR TOMFILE
ZB#TOMUP EQU   X'80'                   . TOMFILE CONFIGURED
ZB#TOMER EQU   X'01'                    . ERROR ON TOM FILE
ZB#TOMNT EQU   X'02'               . DO NOT TRACE TOMFILE
ZB#FLG3  DS    X                       .FLAG FOR TYPE OF RESTART
ZB#INDCL EQU   X'01'                   .START=CLEAN
ZB#INDWA EQU   X'02'                   .START=WARM
ZB#INDCO EQU   X'04'                   .START=COLD
ZB#FLG4  DS    X         DMS FLAG BYTE
ZB#IMSDM EQU   X'80'     IMS HAS MADE A REQUEST TO DMS
ZB#DMSDO EQU   X'40'     DMS HAS TERMINATED
ZP#DMSRU EQU   X'20'     DMS RUN-UNIT EXISTS
ZB#IMSNA EQU   X'10'     IMS NOT ALLOWED ACCESS TO DMS
ZB#DMSNA EQU   X'08'     DMS IS NOT THERE
ZB#FL65  DS    XL1
ZB#SFSEN EQU   X'20'               SFS ENABLED
ZB#GLB   EQU   X'08'               GLOBAL NETWORK
ZB#DED   EQU   X'04'               DEDICATED NETWORK
         DS    XL3       UNUSED
ZB#LPCT  DS    F         LAST PCT ADDRESS
ZB#LACT  DS    F         LAST ACT ADDRESS
ZB#LAD   DS    F         LAST LOAD AREA ADDRESS
ZB#NLST  DS    H         INTLIST=N VALUE
         DS    XL2       UNUSED
ZC#CCA   DS    F                        CCA NAME
ZC#LOCAP DS    F                        LOCAP NAME
ZQ#THFIN DS    0F                       . THIS TAG MUST STAY AT END
ZT#HLEN  EQU   *-ZT#DTHCB          LENGTH OF THCB
ZT#TLEN  EQU   ZT#HLEN
         CSECT
         EJECT
         END
```

Figure D—1. Single-thread Thread Control Block (THCB) (Part 2 of 2)



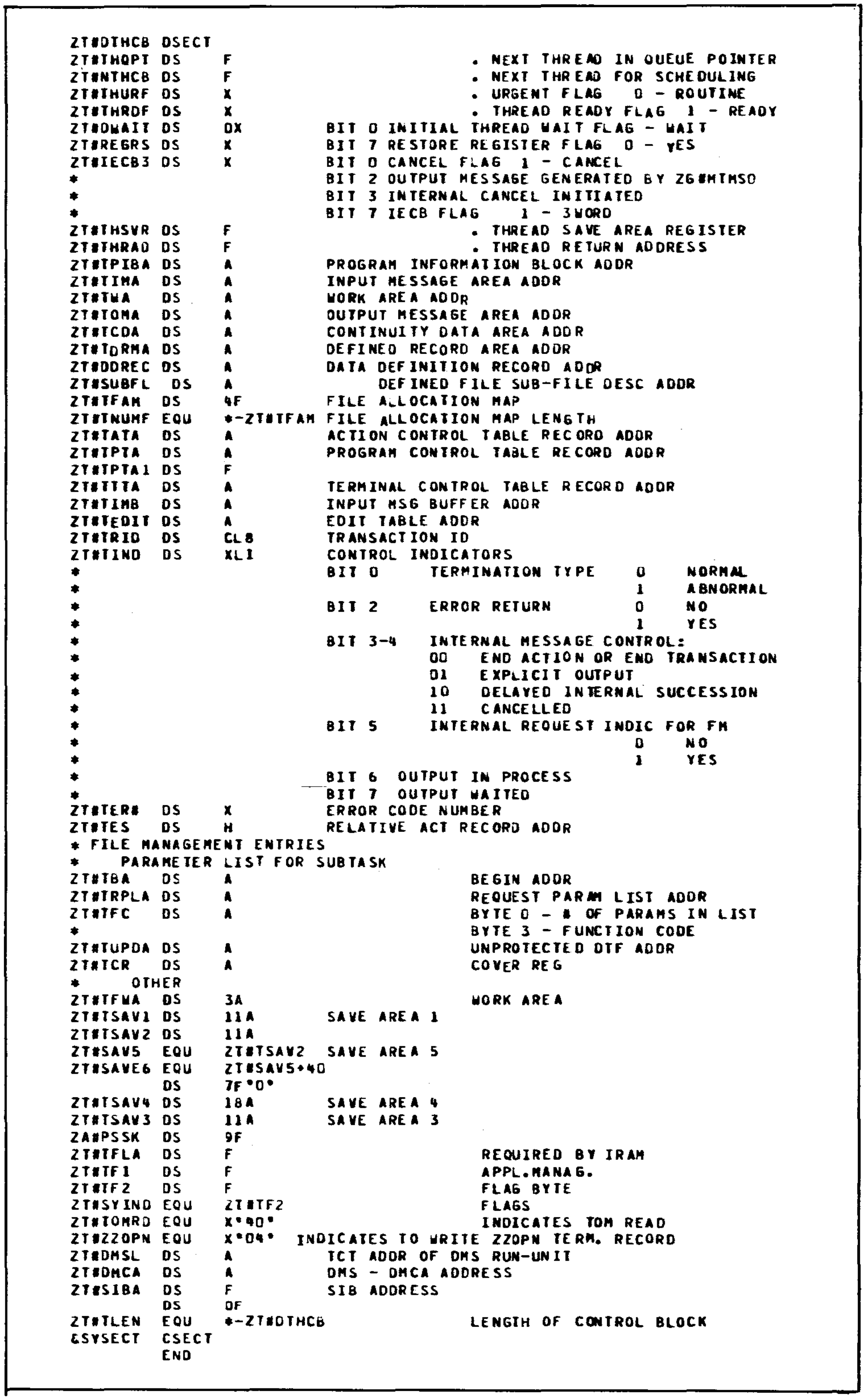
```
ZC#DTCT  DSECT              •••• TERMINAL CONTROL TABLE RECORD ••••
•
ZC#LINK  DS     F           ACT LINK TO NEXT TCT IN QUEUE
ZC#TID   DS     XL4         TERMINAL ID
ZC#TAL   DS     F           REL ADDR ALTERNATE TCT (OS/4)
•                                     REL ADDR SOURCE TCT (OS/3)
ZC#TALT  DS     F           REL ADDR ALTERNATE TCT (OS/3)
ZC#TTTA  DS     F           CORRESPONDING TTT ADDRESS
•     OS/4 USES ZC#TQS & ZC#TIC IN PLACE OF ZC#TTA
         ORG    ZC*TTTA
ZC#TQS   DS     H           DISPL TO ICAM TERMINAL TABLE
ZC#TIC   DS     H           TOTAL TRANSACTION INPUTS
•
ZC#TESR  DS     F           SUCC ACT REL ADDR - ROLLBACK
ZC#TCDL  DS     H           CONTINUITY DATA LENGTH
ZC#TTCM  DS     H           TERMINAL COMMAND COUNT
ZC#TCM   EQU    ZC*TTCM     OS/4 TAG
ZC#TLN   DS     XL1         LINE NUMBER
ZC#TTST  DS     XL5         STATUS BYTES
ZC#TST   EQU    ZC*TTST     OS/4 TAG
•
•   EQUATES FOR ZC#TTST/ZC#TST
•
ZC#TTLST EQU    X'80'       LAST TCT
ZC#TTTMD EQU    X'40'       TEST MODE
ZC#TTUM  EQU    X'20'       URGENT MESSAGE, ACTION
ZC#TTDWN EQU    X'10'       TERMINAL DOWN
ZC#TTHLD EQU    X'08'       HOLD TERMINAL
ZC#TTUT  EQU    X'04'       URGENT TERMINAL
ZC#TMWR  EQU    X'02'                 MSG WAIT (FOR ZZTST) RECEIVED
ZC#TMTC  EQU    X'01'                 MWRITE FOR ZZTST (SINLGE THREAD)
ZC#TOMW  EQU    X'01'                 OUTSTANDING MWRITE (MULTI THREAD)
•
ZC#TST1  EQU    ZC*TST+1,1
•
•   EQUATES FOR ZC#TST1
•
ZC#TTIM  EQU    X'80'       INTERACTIVE MODE
ZC#TTMT  EQU    X'40'       MASTER TERMINAL
ZC#TALTS EQU    X'20'                 ALTERNATE TERM SPECIFIED
ZC#TTRC  EQU    X'10'       ROLLBACK COMPLETE
ZC#TTMWS EQU    X'08'       IMS SENT MSG WAIT
ZC#TTBTH EQU    X'04'       BATCH TERMINAL
ZC#TTRP  EQU    X'02'       ROLLBACK IN PROCESS
ZC#TTMS  EQU    X'01'       MSG TO ORIG TERM SENT
•
ZC#TST2  EQU    ZC*TST1+1,1
ZC#TPRSF EQU    ZC*TST2
•
•   EQUATES FOR ZC#TST2
•
ZC#TTUNS EQU    X'80'       MWRITE ISSUED FROM ZO#UNSMT MODULE
ZC#TTREL EQU    X'40'       RELEASE BUFFER AT MWRITE COMPL
ZC#TPRMQ EQU    X'20'       MSG IN QUEUE
ZC#TPRMP EQU    X'10'       MSG IN PROCESS
ZC#TTSTA EQU    X'08'       SEND AUTO STATUS MESSAGE
ZC#TCONT EQU    X'04'                 CONTINUOUS OUTPUT REQUESTED
ZC#TDELN EQU    X'02'                 DEL NOTICE - ACTION TO BE SCHED
ZC#TOIQ  EQU    X'01'                 OUTPUT GENERATED FOR INPUT QUEUING
•
ZC#TST3  EQU    ZC*TST2+1,1
•
•    EQUATES FOR ZC#TST3
•
ZC#TTDR  EQU    X'80'       DISCONNECT REQUESTED (S/T)
ZC#TTQNE EQU    X'40'       TERMINAL'S LOW QUEUE NOT EMPTY
ZC#THDRS EQU    X'20'       OUTPUT HEADER SAVED
ZC#TIDN  EQU    X'10'       INTERNAL DELIVERY NOTICE
ZC#TIGM  EQU    X'08'       IMS GENERATED ERROR MSG
ZC#COIP  EQU    X'04'       CONTINUOUS OUTPUT IN PROCESS (M/T)
ZC#TNRDY EQU    X'02'       NO IMS READY MSG TO THIS TERMINAL
ZC#TUNAC EQU    X'01'                 SEND UNSOLICITED OUTPUT INDICATOR
•                                     FOR SWITCHED MESSAGES AT ACTION END
•
ZC#TST4  EQU    ZC*TST3+1,1
•    EQUATES FOR ZC#TST4
•
ZC#ERMEX EQU    X'80' A/M GENERATED ERROR MSG.
ZC#SFSRB EQU    X'40'                      REBUILD ALLOWED BY A/P
ZC#ABTDY EQU    X'20'                      ABORT DYNAMIC SESSION
```

Figure D—2. Single-thread and Multithread Terminal Control Table (TCT) (Part 1 of 2)



```
ZC#DYTWD EQU    X'10'                        ABORT TERM WINDOW
ZC#SIGN  EQU    X'08'       SIGN ON FOR DYNAMIC SESSION
ZC#TST5  EQU    ZC#TST4+1,1           DMS FLAGS
ZC#IMPRT EQU    X'80'       ISSUED IMPACT FOR ACTION
ZC#DEPRT EQU    X'40'       ACTION ISSUFD DEPART
ZC#DMSUP EQU    X'20'       ISSUED DSM OPEN FOR UPDATE
ZC#DMSDR EQU    X'10'       DMS REQUEST VIA D.R.M.
ZC#DMSRO EQU    X'08'       DMS FORCED DEPART WITH ROLLBACK
ZC#DMSUB EQU    X'04'       DMS RUN UNIT UNBOUND
*
*  THE FOLLOWING STATUS  BYTE TAGS ARE NOT CLEARED WHEN A GLOBAL
*  NETWORK DYNAMIC TERMINAL DOES A $$SOFF
*        ZC#TTLST
*        ZC#TTUT
*        ZC#TTMT
*        ZC#TNRDY
*        ZC#TUNAC
*        ZC#ATTRI
*
         DS     X           UNUSED
*
ZC#TQE   DS     F           CANCEL LINK
ZC#PRFT  DS     F           DISPL TO PROCESS FILE TABLE
ZC#PQCNT DS     H           PROCESS QUEUE COUNT
ZC#MQCNT DS     XL1                     LAST ICAM SVC
ZC#TDELS DS     XL1                     DELIVERY NOTICE STATUS
ZC#LQCNT DS     H           LOW QUEUE COUNT
ZC#TIN   DS     H           TOTAL INPUT COUNT
ZC#TOC   DS     H           TOTAL OUTPUT COUNT
ZC#TON   DS     F           TIMER LINK
ZC#IBF   DS     H           DISPL TO 1ST SLOT OF INPUT MSG
ZC#IML   DS     H           INPUT MESSAGE LENGTH
ZC#OBF   DS     H           DISPL TO 1ST SLOT OF OUTPUT MSG
ZC#OML   DS     H           OUTPUT MESSAGE LENGTH
*   OS/3 SINGLE THREAD SYSTEM USES ZC#COSEQ FOR C/O SEQUENCE COUNT
ZC#COSEQ DS     H                       C/O SEQ COUNT (OS/3 S.T.)
*
ZC#TBF   EQU    ZC#COSEQ                TIMER BUFFER ADDR (OS/3 M.T.)
ZC#TML   DS     H                       TIMER MESSAGE LENGTH (OS/3 M.T.)
ZC#TDELC DS     XL4                     USER CONTINUOUS OUTPUT CODE
ZC#SFSTC DS     A                             SFS TERMINAL CLASS ENTRY ADDR
ZC#SFSFN DS     CL8                           SFS FORMAT NAME
ZC#SESID DS     F                             SESSION ID
ZC#TTRID DS     CL8                     TRANS ID (INITIAL DATE/TIME)
ZC#TRID  EQU    ZC#TTRID  OS/4 TAG
ZC#DLCNT DS     H           IMC DEADLOCK DETECTION COUNT
ZC#TCB   DS     A           THREAD CONTROL BLOCK ADDR
ZC#TLI   DS     4F          TRANS LOCK INDICATOR
ZC#TAUM  DS     4F          AUDITED UPDATE MAP
*** ZC#TLI AND ZC#TAUM MUST AGREE WITH ZT#TNUMF IN THE THCB
ZC#TTEXT DS     CL5         TRANSLATED TERM CMD/TRANS CODE
ZC#TCODE EQU    ZC#TTEXT    OS/4 TAG
ZC#TDDRC DS     CL1         DDR NAME ID CHAR (HIGH BYTE = X'FD')
*** THE ABOVE FIELD IS DEFINED IN OS/4 BUT NOT TAGGED
ZC#TDDRN DS     CL7         DATA DEF REC NAME
ZC#TDFN  DS     CL7         DEFINED FILE NAME
ZC#TES   DS     F           SUCC ACT RECORD RELATIVE ADDR
*      MULTI-THREAD SYSTEMS USE ZC#ES & ZC#CDC IN PLACE OF ZC#TES
         ORG    ZC#TES
ZC#ES    DS     H           SUCC ACT RECORD RELATIVE ADDR
ZC#CDL   DS     H           CONTINUITY DATA LENGTH
*
ZC#WAI   DS     H           WORK AREA INC
ZC#CDI   DS     H           CONTINUITY DATA AREA INC
ZC#TTTN  DS     XL1         TCT RECORD NUMBER
         DS     XL1         UNUSED
ZC#TINT  DS     H           TOTAL TRANSACTION INPUTS
*      MULTI-THREAD USES ZC#CDR & ZC#CES INSTEAD OF ZC#TTTN & ZC#TINT
         ORG    ZC#TTTN
ZC#CDR   DS     H           TCT RECORD NUMBER
ZC#CES   DS     H           SUCC ACT REL ADDR - ROLLBACK
ZC#SCFR  DS     XL4         COUNT FIELD FOR ROLLBACK
*
ZC#TTIR  DS     XL1         TERM IND FOR ACTION PROG USING ROLLBACK
ZC#TIR   EQU    ZC#TTIR     OS/4 TAG
         ORG    ZC#TIR
ZC#TRWA  DS     F           TRACE WORK AREA
ZC#FBPA  DS     H * FIRST BLOCK OF PARTITION
ZC#CBPA  DS     H * CURRENTLY ACCESSED BLOCK
ZC#LBPA  DS     H * LAST BLOCK OF PARTITION
ZC#NRBCB DS     H *# OF REM.BYTES IN CURR. BLOCK
*
ZC#TLNAM DS     CL4         LINE NAME
*
ZC#TLEN  EQU    *-ZC#DTCT
&SYSECT  CSECT
         END
```

Figure D—2. Single-thread and Multithread Terminal Control Table (TCT) (Part 2 of 2)



```
ZT#DTHCB DSECT
ZT#THQPT DS    F                       . NEXT THREAD IN QUEUE POINTER
ZT#NTHCB DS    F                       . NEXT THREAD FOR SCHEDULING
ZT#THURF DS    X                       . URGENT FLAG   0 - ROUTINE
ZT#THRDF DS    X                       . THREAD READY FLAG  1 - READY
ZT#DWAIT DS    0X        BIT 0 INITIAL THREAD WAIT FLAG - WAIT
ZT#REGRS DS    X         BIT 7 RESTORE REGISTER FLAG  0 - YES
ZT#IECB3 DS    X         BIT 0 CANCEL FLAG  1 - CANCEL
*                        BIT 2 OUTPUT MESSAGE GENERATED BY Z6#MTMSO
*                        BIT 3 INTERNAL CANCEL INITIATED
*                        BIT 7 IECB FLAG     1 - 3WORD
ZT#THSVR DS    F                       . THREAD SAVE AREA REGISTER
ZT#THRAD DS    F                       . THREAD RETURN ADDRESS
ZT#TPIBA DS    A         PROGRAM INFORMATION BLOCK ADDR
ZT#TIMA  DS    A         INPUT MESSAGE AREA ADDR
ZT#TWA   DS    A         WORK AREA ADDR
ZT#TOMA  DS    A         OUTPUT MESSAGE AREA ADDR
ZT#TCDA  DS    A         CONTINUITY DATA AREA ADDR
ZT#TDRMA DS    A         DEFINED RECORD AREA ADDR
ZT#DDREC DS    A         DATA DEFINITION RECORD ADDR
ZT#SUBFL  DS   A              DEFINED FILE SUB-FILE DESC ADDR
ZT#TFAM  DS    4F        FILE ALLOCATION MAP
ZT#TNUMF EQU   *-ZT#TFAM FILE ALLOCATION MAP LENGTH
ZT#TATA  DS    A         ACTION CONTROL TABLE RECORD ADDR
ZT#TPTA  DS    A         PROGRAM CONTROL TABLE RECORD ADDR
ZT#TPTA1 DS    F
ZT#TTTA  DS    A         TERMINAL CONTROL TABLE RECORD ADDR
ZT#TIMB  DS    A         INPUT MSG BUFFER ADDR
ZT#TEDIT DS    A         EDIT TABLE ADDR
ZT#TRID  DS    CL8       TRANSACTION ID
ZT#TIND  DS    XL1       CONTROL INDICATORS
*                        BIT 0     TERMINATION TYPE    0    NORMAL
*                                                      1    ABNORMAL
*                        BIT 2     ERROR RETURN        0    NO
*                                                      1    YES
*                        BIT 3-4   INTERNAL MESSAGE CONTROL:
*                                  00   END ACTION OR END TRANSACTION
*                                  01   EXPLICIT OUTPUT
*                                  10   DELAYED INTERNAL SUCCESSION
*                                  11   CANCELLED
*                        BIT 5     INTERNAL REQUEST INDIC FOR FM
*                                                      0    NO
*                                                      1    YES
*                        BIT 6  OUTPUT IN PROCESS
*                        BIT 7  OUTPUT WAITED
ZT#TER#  DS    X         ERROR CODE NUMBER
ZT#TES   DS    H         RELATIVE ACT RECORD ADDR
* FILE MANAGEMENT ENTRIES
*    PARAMETER LIST FOR SUBTASK
ZT#TBA   DS    A                       BEGIN ADDR
ZT#TRPLA DS    A                       REQUEST PARAM LIST ADDR
ZT#TFC   DS    A                       BYTE 0 - # OF PARAMS IN LIST
*                                      BYTE 3 - FUNCTION CODE
ZT#TUPDA DS    A                       UNPROTECTED DTF ADDR
ZT#TCR   DS    A                       COVER REG
*     OTHER
ZT#TFWA  DS    3A                      WORK AREA
ZT#TSAV1 DS    11A       SAVE AREA 1
ZT#TSAV2 DS    11A
ZT#SAV5  EQU   ZT#TSAV2  SAVE AREA 5
ZT#SAVE6 EQU   ZT#SAV5+40
         DS    7F'0'
ZT#TSAV4 DS    18A       SAVE AREA 4
ZT#TSAV3 DS    11A       SAVE AREA 3
ZA#PSSK  DS    9F
ZT#TFLA  DS    F                        REQUIRED BY IRAM
ZT#TF1   DS    F                        APPL.MANAG.
ZT#TF2   DS    F                        FLAG BYTE
ZT#SYIND EQU   ZT#TF2                   FLAGS
ZT#TOMRD EQU   X'40'                    INDICATES TOM READ
ZT#ZZOPN EQU   X'04'  INDICATES TO WRITE ZZOPN TERM. RECORD
ZT#DMSL  DS    A         TCT ADDR OF DMS RUN-UNIT
ZT#DMCA  DS    A         DMS - DMCA ADDRESS
ZT#SIBA  DS    F         SIB ADDRESS
         DS    0F
ZT#TLEN  EQU   *-ZT#DTHCB              LENGTH OF CONTROL BLOCK
&SYSECT  CSECT
         END
```

Figure D—3. Multithread Thread Control Block (THCB)

Appendix E. Device Independent Control Expressions

E.1. GENERAL

You use device independent control expressions (DICE sequences) to format input and output messages being handled by your action program. These codes are needed to control various operations, such as cursor positioning and carriage return, on a terminal screen.

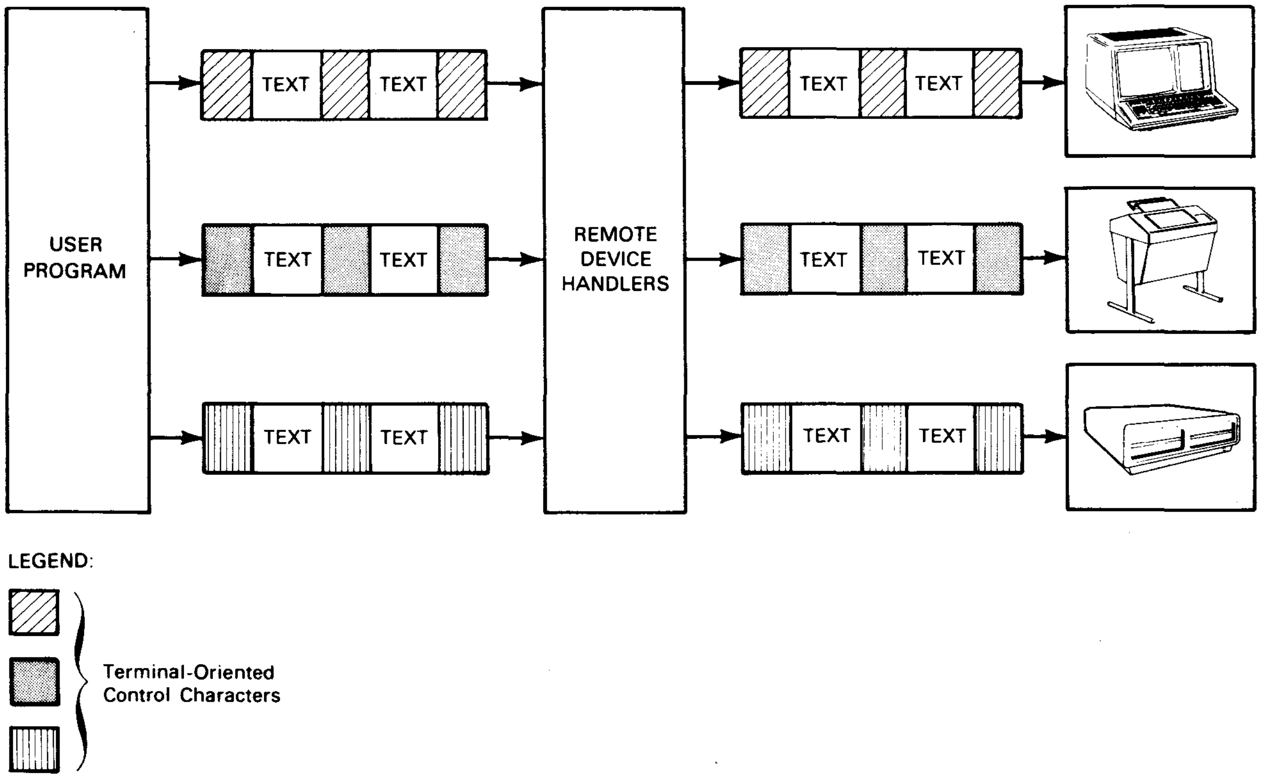
Appendix E supplies all DICE sequences and their interpretations, describes how to use them in formatting your messages in your action programs, and discusses the DICE macroinstructions you can use in BAL user action programs to create the DICE sequences.

E.2. USING DICE TO FORMAT MESSAGES

For output, your action program can use either of two methods to control the format of a message displayed at a terminal.

1. By embedding format control characters, the message text is directed to each specific terminal. Obviously, if you do this, your program must include a different formatting routine for each type of terminal; this is illustrated in the following diagram:

OUTPUT TEXT AND CONTROL CHARACTERS

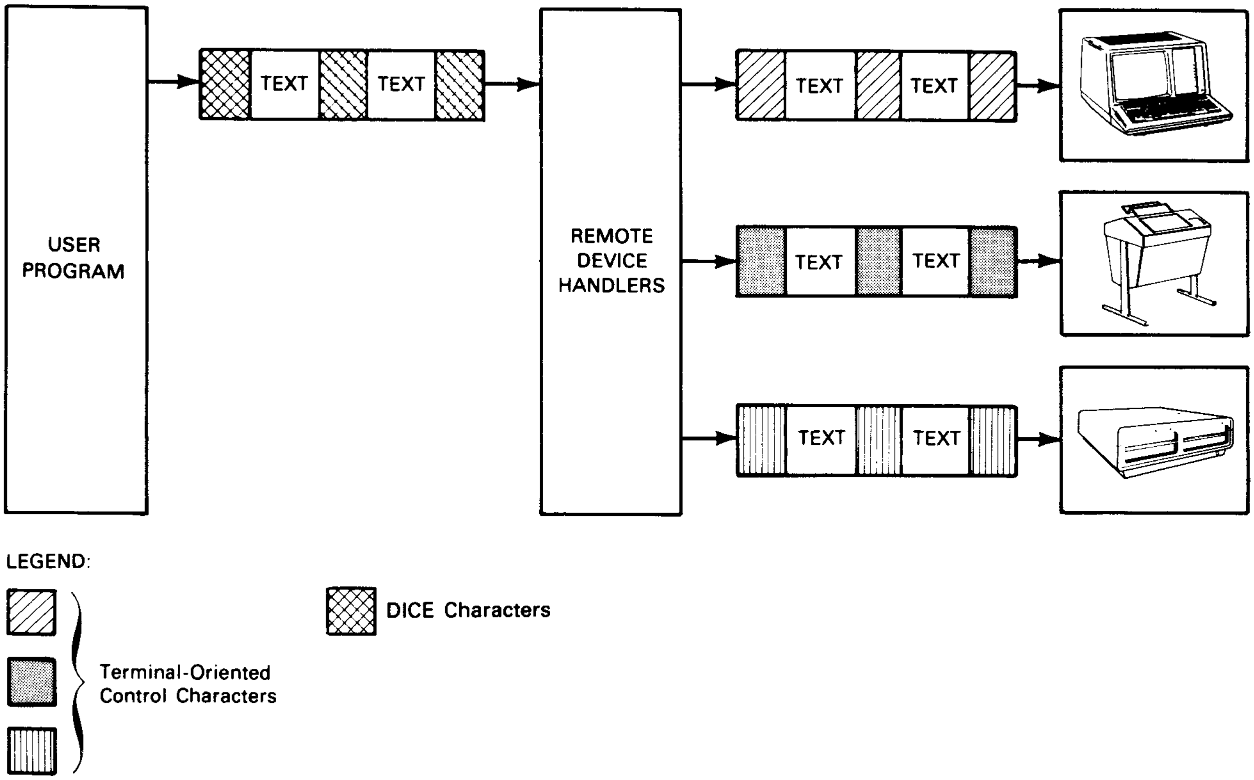


2. By embedding DICE sequences, the control of format for various types of terminals and auxiliary devices is simplified. The remote device handler (RDH) converts DICE sequences to control characters for each destination terminal. Some of the control character functions are:

   - 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; or
   - cursor movement to a specific row and column on a display.

   You can place DICE sequence anywhere in a message to accomplish the control you want. As you can see by the following illustration, formatting is easier when you use DICE.

OUTPUT TEXT AND DICE



For input, control characters received in a message are converted into DICE sequences by the RDH. For certain terminals, your program can analyze these DICE sequences to determine cursor position. In addition, input DICE is handy for message switch application because control characters in each input message are converted to DICE sequences. The RDH converts these sequences into the appropriate control characters for the destination terminal.

You can turn DICE on or off at network definition time with the DICE operand on to the TERM macroinstruction.

```
DICE=({ON })
      {OFF}
```

The default is DICE=(ON).

1. DICE=(ON) tells RDHs to create input DICE according to your input terminal cursor movements; DICE are created automatically.

   For UNISCOPE 100/200, UTS 400, and UTS 4000, the ICAM remote device handler passes the start of text character (STX) to your program as the first byte of data. For example, in a message containing a start of entry character (RS), the following data is received in the text portion of your input work area:

```
S  E  V  Y  X  N  S  R
T  S  T        U  I  S   rest-of-text
X  C           L
```

End of text characters (ETX) are always removed by the remote device handlers; they are never supplied to your program as text.

2. DICE=(OFF) tells the RDHs not to create input DICE.

   For UNISCOPE 100/200, UTS 400, and UTS 4000, the start of text character (STX) is suppressed by the remote device handler, and control character sequences are converted to DICE sequences. For example, in a message containing a start of entry character (RS), the following text is received in the text portion of your work area:

```
4-character-dice-sequence   R   rest-of-text
                            S
```

   End of text characters (ETX) are always removed by the remote device handlers; they are never supplied to your program as text.

See the OS/3 fundamentals of ICAM, UP-8194 (current version), for a detailed example of input DICE creation.

E.2.1. Format of DICE Sequences

The format of a DICE sequence is as follows:

Format:

select character	function code	m field	n field

where:

select character
: Is a hexadecimal character (10_{16}) designating the start of a DICE sequence. This character, a data link escape (DLE) control character in EBCDIC, must be used only to designate the start of a DICE sequence.

function code
: Defines the device control sequence that is recognized by the RDHs on input. On output, this code is a 1-byte field defining the operation to be performed on the text message. DICE function codes are listed in Table E-1.

m field and n field
: 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 applies to horizontal positioning.

These fields may be expressed in absolute values (m_a and n_a) or relative displacement values (m_r and n_r) in Table E-1. 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, e.g., move cursor 2 rows down and to column 1. If you choose to use DICE macroinstructions, these parameters must be specified.

E.2.2. DICE Macroinstructions

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.

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 RDH converts the DICE constants into the corresponding control characters to produce the necessary positioning.

On input, DICE sequences are automatically created by the RDHs unless you specify the DICE=(OFF) parameter in your network definition. Table 2-5 lists the DICE macroinstructions, function code generated, and m and n coordinates as they apply to particular devices on input and output.

You must specify m and n coordinates in your program according to the absolute and relative values expressed in Table E-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 $(m_a, n_n) = (1,1)$. 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 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 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.

E.2.3. DICE Code Generation

Macroinstructions are provided to generate the DICE codes.

Format:

LABEL	△OPERATION△	OPERAND
[symbol]	dice-macroinstruction	m,n

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 E-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 E-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 E-1).

Examples:

```
      1          10     16
   ---------------------------
1. | NEWLINE    ZO#POS   Ø,Ø
2. | COORDI     ZO#COORD  5,1Ø
```

1. This DICE sequence causes movement to a new line.

2. New text will be started at line 5, column 10 due to this DICE.

Table E—1. DICE Input/Ouput Commands, Codes, and Device Interpretation (Part 1 of 4)

DICE Macro-instruction	Function	Function Code Value	I/O	m	n	Character-oriented Devices[1]	CRT Devices	Page Printing Devices (n is Not Interpreted)	Communications Output Printer (COP)[1]
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.	Action is optional.[2]
ZO#FORM	Forms control	02_{16}	INPUT	01	01	Form feed	Not used	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 optional[2]	Action is optional.[2]

Table E—1. DICE Input/Ouput Commands, Codes, and Device Interpretation (Part 2 of 4)

DICE Macro-instruction	Function	Function Code Value	I/O	m	n	Character-oriented Devices[1]	CRT Devices	Page Printing Devices (n is Not Interpreted)	Communications Output Printer (COP)[1]
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
ZO#CUR	Current position control	06_{16}	INPUT	01	00	Line feed	Not used	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]

Table E—1. DICE Input/Ouput 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	Page Printing Devices (n is Not Interpreted)	Communications Output Printer (COP)[1]
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 position 4	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]
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

Table E—1. DICE Input/Ouput Commands, Codes, and Device Interpretation (Part 4 of 4)

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.

The first option should be used if the character-oriented terminals are a part of a message switch environment.

Certain terminals do not have a form feed capability (i.e., some TTY terminals). For these terminals, the DICE expressions that specify form feed will result in line feed instead.

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:

$$\text{TERM . . . , DICE=}\left(\begin{Bmatrix}\text{FORMS}\\ \text{NEWLINE}\end{Bmatrix}\right)\text{, . . .}$$

If FORMS is specified, the set coordinates macroinstruction will be interpreted as the forms control macroinstruction.

If NEWLINE is specified, the set coordinates macroinstruction and the forms control with clear macroinstruction will 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.

If the DICE parameter is not specified, the default option will be to 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 dispaly terminal. The "current position control with clear" is the only DICE macroinstruction that can be used to 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 When using DICE function code 09_{16} for setting a tab stop, m=0 and n=0 will result in a tab stop being placed at the current cursor location (no cursor positioning is performed). This applies to UNISCOPE aand UTS 400 devices only. For TTYs and DCT 500 terminals, a space character is inserted.

If m or n is greater than the maximum allowable m or n, action will vary depending on the remote terminal:

- UNISCOPE display terminals–wraparound will occur on screen.

- Character-oriented terminals–will give different results depending on the characteristics of the device.

E.2.4. Interpretation OF DICE

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 limitations of the terminal.

The interpretation of a DICE by the RDH is controlled by the following factors:

1. DICE function code

2. DICE m and n fields

3. The terminal involved

4. The particular device on the terminal being used.

The ICAM RDHs currently provide device-independent support for three classes of remote terminal 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 1000 (DCT 1000), and TELETYPE* teletypewriter models 28, 32, 33, 35, 37.

2. Hard copy page printer type device, such as the SPERRY UNIVAC 1004 Card Processor System, Data Communications Terminal 2000 (DCT 2000), and 9200/9300 Systems, and the IBM 2780.

3. CRT-type terminals, such as the UNISCOPE 100 and 200 and the UTS 400 Display Terminals.

Table E–2 defines the primary output device and the primary input device for each terminal type.

Table E—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

*Trademark of Teletype Corporation

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 E-3.

Table E—3. DICE Usage for Auxiliary Devices

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	Card reader/card punch Paper tape reader/punch	DICE is applied as if the output/input is to/from the primary device, even through it is for the auxiliary device. ②
DCT 500/TTY	Paper tape reader/punch	
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 If the print transparent option is not used, DICE is applied to the UNISCOPE screen even through 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 E-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.

See 3-3 for more detailed information on the usage of DICE when applied to the COP.

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 E-1.

Glossary

A

action

The basic unit of work in IMS 90. An action consists of message input, the processing of the message by one or more action programs that may access data files, and the output of at least one message.

action program

An independent, relocatable program defined to IMS 90 and conforming to the programming conventions established by IMS 90. One or more action programs may be executed in an action.

action scheduling

A component of application management that initiates and terminates an action program and dynamically allocates required and optional main storage areas used by an action program.

activation record

The working storage area used by a specific action program to process messages and transmit information begween IMS 90 and the action program. The activation record must include the program information block (PIB) and the input message area (IMA) and may optionally include the output message area (OMA), work area (WA), continuity data area (CDA), and defined record area (DRA).

application management

The major component of IMS 90 that controls the execution of action programs and includes action scheduling and file management.

automatic status messages

Messages automatically generated by multithread IMS 90 to notify the terminal operator of the status of the last input message.

auxiliary-device-function

A 1-byte field within the auxiliary-device-ID field of the OMA which specifies auxiliary information about this output message. For example, the message is being sent to an auxiliary device or is handled as continuous output.

auxiliary-device-ID
A 2-byte field in the OMA which indicates auxiliary information about this output message (auxiliary-device-function) and to which device (auxiliary-device-number) the message is to be sent if the first byte indicates it is being sent to an auxiliary device.

auxiliary-device-number
A 1-byte field within the auxiliary-device-ID field of the OMA which specifies the device number that corresponds to the logical device number in the CCA network definition.

auxiliary output device
The two auxiliary output devices supported by IMS 90 for communications are the communications output printer (COP) and terminal printer (TP).

B

batch transaction processing
A method of processing groups of transactions entered from cards or a source library file instead of an online terminal. This method is useful for testing if an online terminal or ICAM is not available, or for performing batches of transactions overnight.

batch transaction processor
An optional component of single and multithread IMS 90 which enables transactions to be entered via cards or source library file instead of a display terminal and transmits output to the printer file.

C

common storage area (CSA)
An area in main storage containing one or more resident ISAM files, called CSA files. Action programs retrieve data from and make updates to the CSA file instead of a disk ISAM file, thus reducing disk access.

communications control area (CCA)
A collection of tables that describe the ICAM communications network.

configuration
The process by which the available IMS 90 software is tailored so that the resultant IMS 90 package satisfies the needs specified by the user and reflects the user's hardware and software environment.

continuity data area (CDA)
An optional main storage area in the activation record in which user-defined data is automatically passed from action to action in a transaction. The continuity data is written to the continuity data file at the termination of an action and read into main storage when the successor action is scheduled.

continuous output
A continuous series of output messages transmitted to a terminal without intervening input messages.

D

data definition language
A COBOL-like language used to describe defined files accessed by user action programs or UNIQUE through defined record management.

data definition processor
That module of IMS 90 that processes the data definition and writes a data definition record into the named record file.

data definition record
A record in the named record (NAMEREC) file that contains the description of a defined file and related subfiles and is used by defined record management to access that defined file or those subfiles.

data name
A word that names an entry in the data division of a data definition or COBOL action program.

defined file
A sequence of defined records in order by their identifiers.

defined file name
Name of defined file.

defined record
A sequence of items obtained from IMS 90 by UNIQUE and other action programs with the execution of a single function call. It is displayed at the terminal by UNIQUE in response to a single operator command. IMS 90 composes the record dynamically from one or more disk records, according to a user-supplied data definition.

defined record area
An optional main storage area in the activation record used by the defined record management portion of IMS 90 if defined files are accessed.

defined record management
That module of IMS 90 which retrieves and/or updates defined records for action programs.

defined record name
That name that identifies a particular type of defined record in a defined file.

defined record supplement
Additional items from logical records or repeating groups, other than the source of the primary part, that complete a defined record.

definition division
The third division in the input to the data definition processor. This division describes the defined files and the defined records.

delayed internal succession
A method of succesion in which one action builds an output message in the OMA and queues it as an input message to a second action without sending the output message to the originating terminal.

delivery notice scheduling
The scheduling of a successor action program to receive a 5-byte IMS 90 acknowledgement message (delivery notice) indicating that an output message has been successfully or unsuccessfully delivered to its destination for continuous printing.

detail line
An output line containing values associated with the item names from the header line. The values are from the file being accessed.

detailed status code
A 2-byte binary value returned by IMS 90 following a request when the status code indicates invalid request, I/O error, or internal message control error. The detailed status code supplements the status code by giving more detailed information.

device-independent-control-expression
A 4-byte control sequence used in formatting messages at a terminal and consisting of the select character (10_{16}), a hexadecimal function code, and two hexadecimal coordinates defining row and column positions on a terminal.

dialog
Interaction between a terminal operator and IMS 90 consisting of a sequence of logically related input and output messages. When UNIQUE is used, it begins with an OPEN and ends with a CLOSE or another OPEN.

dialog transaction
A transaction that consists of two or more actions. See also external succession.

disk overflow
The staging and linking of input messages to disk when the main storage queue has become saturated.

display content
In the LIST or DETAIL command, a list of item names, subrecord names, or arithmetic expressions that indicate to UNIQUE what items or values to display at the terminal.

display format command
An input/output message structure especially suited for a cathode ray tube (CRT) type device.

ditto mechanism

A shorthand means used in UNIQUE commands to identify a defined record by referring to part of the identifier of a previously identified defined record or to specify LIST or DETAIL parameters by referring to the appropriate part of the previous LIST or DETAIL command.

downline load

A process of obtaining programs from a load library and then sending them to the UTS 400 memory, cassette, or diskette.

E

edit table generator

An offline IMS 90 utility program that writes a record in the named record (NAMEREC) file that contains the user's definition of how online freeform terminal input is converted into fixed formats required by action programs. The edit table generator also checks input for data types, value ranges, and the presence of required fields.

explicit output message

Output messages sent via the CALL SEND function.

external succession

A method of succession in which an action program terminates by requesting that an output message be sent to the originating terminal and a specified action program be scheduled to proces the next input message form that terminal. Information in the predecessor's CDA is passed to the successor.

F

file I/O functions

Those functions such as retrieval, insertion, deletion, update, and write, which are supported by the file management component of IMS 90.

file management

That IMS 90 component that issues requests to data management for user logical records required by action programs. File management interfaces with both action programs and data management, and provides services such as record locking, recovery, error processing, etc.

function key

A key on a UNISCOPE display terminal or UTS 400 terminal that is converted to a text input message. It can be used as a transaction code if the terminal is not in interactive mode or as a response to an output message to be passed on to a successive action program if the terminal is in interactive mode.

H

hard copy format command
An input/output dialog structure especially suited for a teletypewriter device.

header line
An output line containing item names that serve as column headers.

hierarchical structure
In IMS 90, a file structure with more than one type of record having one or more levels of subordinate records. The relationships between record types are parent, child, and fraternal.

I

identifier
The string of characters that is keyed in by a terminal operator to select a particular occurrence of a defined record.

immediate internal succession
A method of succession in which the first action program indicates the name of a succeeding action program without issuing an output message or deallocating main storage areas. The second action program then uses the same main storage areas to complete and terminate action processing.

implicit output message
A message in the OMA when the CALL RETURN function is requested at action termination.

IMSCONF
A job procedure which generates and executes control streams that configure IMS 90 using the user configuration specifications as input.

input message area
A required main storage area in the activation record consisting of a 16-byte control header and the variable length input message.

inquiry operation
An operation that retrieves and displays records from a file. The UNIQUE commands DISPLAY, LIST, MORE, and DETAIL are inquiry commands.

integer
A numeric literal that does not include any character positions to the right of the assumed decimal point.

interactive mode
The state of a terminal while a dialog transaction is in progress. Input messages do not contain transaction codes.

interactive processing

A method of information processing that involves a dialog between a terminal operator and IMS 90.

internal message control (IMC)

The IMS 90 component that controls message input/output processing and terminal command processing for action programs and IMS 90.

item

A consecutive string of data bytes that is treated as a single entity by display, validation, and arithmetic functions. It appears in every occurrence of the type of record that contains it, in the same position, and with the same usage characteristics as specified in the data definition. The name of the item is displayed by UNIQUE as a column header when the item appears at a terminal.

K

key

One or more items contained in a record or in a repeating group item occurrence that distinguish that occurrence from all others in the same file or table. The key also determines where a new occurrence is to be inserted in an existing sequence of records or repeating group item occurrence.

keyword parameter

A parameter whose specification is identified by its name, rather than by its position in the operand field.

L

locked record

A logical record which, by specification of the LOCK parameter at configuration time, is protected while being updated from being concurrently updated by another transaction.

lock-rollback-indicator

A 1-byte value in the PIB which is set by the action program to select the record lock and rollback functions of IMS 90 action scheduling.

logical record

The actual record that exists physically on a user data file and accessed by standard access methods (DAM, SAM, MIRAM, IRAM, and ISAM).

M

main storage queueing
The process whereby messages are staged and linked in main storage.

mandatory configuration parameter
A parameter whose specification is required because omission of the specification renders IMS 90 inoperable.

master terminal
A control terminal used in monitoring IMS 90 and in controlling the IMS 90 communications network. In single-thread IMS 90, the master terminal may be the console.

multithread IMS 90
The capability of concurrently processing actions that come from different terminals.

N

named record file (NAMEREC)
An internal IMS 90 file containing configuration tables, data definition records, edit tables, and password definitions.

O

offline processing
The operations that recover damaged or inconsistent user files.

online processing
The operations (such as startup/shutdown, internal message control, UNIQUE or user action program execution, and file management) that process transactions interactively.

output delivery notice status code
The hexadecimal value provided by IMS 90 in the fifth byte of the input message to inform continuous output user of the status of the last output message.

output message area
An optional main storage area in the activation record consisting of a control header and area for output message text.

output-for-input queueing
The operation of generating an output message via the SEND function to be scheduled as an input message on a terminal other than the initiating terminal.

P

password
An installation-defined name that is associated with a specific defined file or subfile for security purposes.

prefix
One or more characters that are not a part of the record key that is stored on a disk, but which the terminal operator must key in as part of the identifier of a defined record. These characters are used to distinguish fraternal record types where the ranges of the values of the record keys overlap.

pre-online processing
The operations of IMS 90 utilities and processors (e.g., the NAMEREC utility, data definition processor, and configurator) that prepare and configure the system for transaction processing.

primary part
Those items of a defined record, including the identifier, that come from the record occurrence or repeating group item occurrence that is located by the identifier. The primary part must exist and is always the first part of the defined record to be read.

print mode
A form of output printing in which what appears on an auxiliary device is the same as what appears on the screen of the primary device.

print transparent mode
A form of output printing to an auxiliary device which is independent of the hardware limitations of the remote terminal screen.

program information block (PIB)
A required, predefined 48-byte main storage area within the activation record used to pass control information between IMS 90 and the user action program.

pseudoterminal
A virtual terminal created by the IMS 90 configurator for use in batch transaction processing.

R

record type
The specific defined record layout when there are several possible record layouts in the same defined file.

reentrant code
Shared code that is not self-modifying allowing concurrent use by several threads.

repeating group item
An item for which an OCCURS clause appears in the data division.

resident subprogram
A user-written program that resides in main storage, is called by an action program or another subprogram, and returns control to the calling program.

S

screen format services
A method of displaying predefined formatted screens at a terminal. IMS 90 places the screen formats into the action program's OMA when a BUILD function call is issued.

selection criteria
In the LIST and DETAIL commands, conditional expressions that determine which records are to be displayed in response to these commands.

serially reusable
A program that, when it is in main storage, can be executed by another thread after the current thread has terminated its use.

shared code
Code whose self-modification observes certain restrictions enabling action scheduling to switch the use of that code from one thread to another whenever a function request is made. This effectively allows concurrent use by several threads.

simple transaction
A transaction that consists of a single action.

single-thread
The capability of processing a single action at a time, from initiation to termination of the action program.

solicited message
A message that is a reply to previous input

standard terminal
A production terminal used for entering commands and receiving messages during transaction processing.

statistical function
Functions provided by the UNIQUE LIST command to derive totals, record counts, averages, minimums, and maximums.

status code
A 2-byte binary integer value in the PIB indicating the completion status of a request. This code is interpreted differently for file I/O functions and explicit message output.

stored record name
The name of a group item in the data division that describes the logical record source of the defined record.

subfile
In a defined file, a subset of defined records or subrecords that provides an alternative and more restrictive access than the defined file.

subprogram
A separately compiled module that is called by an action program or another subprogram. It can either be linked with the calling program or made resident, in which case it can be either reentrant or serially reuseable. It performs functions common to two or more action programs.

subrecord
A variant of a defined record, that allows alternative and more restrictive access.

subrecord name
The name of a subrecord that provides an alternative to defined record name in the content clause of the LIST command.

success unit
The sequence of actions over which record locks are held starting with the input of a message or a rollback point, and ending with the next rollback point or transaction termination.

succession
The mechanism that provides the dynamic sequencing of action programs in a transaction. Immediate internal succession provides the linkage between two action programs in an action with no modification in resource allocation. Delayed internal succession provides the linkage between two actions with the output message from the first action queued as input to the successor action. External succession provides the linkage between actions in a transaction, with messages sent to and from the initiating terminal.

successor-ID
A 6-byte field in the PIB indicating the name of the action program to be activated when the current action program terminates.

suffix
A string of characters that occupies the least significant character positions of a record key as stored in disk. These characters are not included in the identifier keyed in by the terminal operator; therefore, they must be literally added by DRM. They are specified in the FILL KEY statement.

supplement

Those additional items of a defined record that come from a record or a repeating group item occurrence that is different from the source of the primary part. The source of the supplement is located by a pointer that is constructed from one or more items of the primary part or of previously read supplements.

supplement name

The name that identifies a supplement within a data definition

T

terminal commands

There are standard and master terminal commands. Standard terminal commands may be issued from any terminal to control the flow of messages at the terminal. Master terminal commands may be used only at the master terminal to control the overall system and communications network.

terminal control table (TCT)

Describes the terminal and transaction environment and the program status.

termination indicator

A 1-byte value in the PIB which indicates the type of termination for the current action program

test mode

A terminal state in which there is no physical alteration to data files; file updates are simulated.

thread

A unit of control for the sequence of events needed to complete the processing of an action.

total line

An output line containing line names, their requested totals, and other statistical functions.

transaction

A sequence of one or more actions that are related through delayed internal or external succession.

transaction code

One to five alphanumeric characters, with the first character alphabetic, used to identify and initiate a transaction.

transaction control interface (TCI)

An interactive communications interface designed for IMS 90.

transaction terminal table

A table used to pass information between IMS 90 and ICAM.

transaction ID
The unique date-item stamp of the initial input message in a transaction

U

uniform inquiry data element (UNIQUE)
A set of action programs supplied by Sperry Univac that retrieves and updates data in files.

UNIQUE commands
A series of commands that facilitate the manipulation of defined files.

unsolicited output
All messages that are not responses to previous input (e.g., requests from one terminal to be transmitted to another terminal).

update format
A format, displayed by UNIQUE, that reveals to the terminal operator certain characteristics of each item that can be updated. These characteristics include length and decimal placement.

update operation
An operation that causes a record in a file to be deleted, added, or changed. The UNIQUE commands DELETE, ADD, and CHANGE are update commands.

V

validity test
An acceptance test performed by defined record management on new item values when a defined record is added or updated. The new item value is compared to ranges that have been specified in VALUE statements in the data definition.

W

work area (WA)
An optional main storage area within the activation record generally used by sharable or reentrant action programs for file I/O logical record areas and working storage.

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:

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

Cut along line.

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

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:

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

Cut along line.

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

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:

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

Cut along line.

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