SH20-9026-4

Program Product

IMS/VS Version 1 Application Programming Reference Manual

Program Number 5740-XX2

Release 1.2

IBM

Fifth Edition (May 1976)

This edition replaces the previous edition (numbered SH20-9026-2), its technical newsletter (numbered SN20-9110), and the reprint (numbered SH20-9026-3), and makes them obsolete.

This edition applies to Version 1 Release 1.2 of IMS/VS, program number 5740-XX2, and to all subsequent releases unless otherwise indicated in new editions or technical newsletters.

Technical changes are summarized under "Summary of Amendments" following the list of figures. In addition, miscellaneous editorial and technical changes have been made throughout the publication. Each technical change is marked by a vertical line to the left of the change.

Information in this publication is subject to significant change. Any such changes will be published in new editions or technical newsletters. Before using the publication, consult the latest *IBM System/370 Bibliography,* GC20-0001, and the technical newsletters that amend the bibliography, to learn which editions and technical newsletters are applicable and current.

Requests for copies of IBM publications should be made to the IBM branch office that serves you.

Forms for readers' comments are provided at the back of the publication. If the forms have been removed, comments may be addressed to IBM Corporation, General Products Division, Programming Publishing—Department J57, 1501 California Avenue, Palo Alto, California 94304. All comments and suggestions become the property of IBM.

PREFACE

This manual describes the functions of the Information Management System/Virtual Storage (IMS/VS) available to the application programmer using the data base and/or data communication facilities of IMS/VS. The reader should be familiar with the concepts and terminology discussed in prerequisite and associated publications of the IMS/VS reference library (cited below).

This manual contains seven chapters and two appendixes:

- Chapter 1, "IMS/VS Environment for Application Programming," describes the effect of IMS/VS on the application programmers, new application programs, and existing programs, and the major considerations in implementing an IMS/VS application.
- Chapter 2, "Data Base Batch Programming," describes application programming using the IMS/VS data base facility. It covers the details that applications analysts and programmers require to use an IMS/VS logical data structure, and the manner in which a batch application program interfaces with the processing capabilities of IMS/VS.
- Chapter 3, "Data Base Processing: Advanced Functions," describes the more advanced data base processing functions provided by IMS/VS.
- Chapter 4, "Data Communication: Application Programming," describes application programming using the IMS/VS data communication facility. It includes a discussion of the teleprocessing application interface and the logical terminal concept.
- Chapter 5, "Data Communication: Conversational Processing," describes application programming for programs that process transactions defined as conversational.
- Chapter 6, "Application Program Examples," contains examples of IMS/VS application programs.
- Chapter 7, "Application Programming Testing Aids," describes use of the DL/I Test Program (DFSDDLT0) and the requirements (those that pertain, for example, to interfaces, JCL, control statements, and execution in different types of regions) that govern use of this application program. This chapter also describes use of the Message Processing Region Simulation facility to check out a message processing program, in a batch processing region, with a set of data bases designed for testing purposes.
- Appendix A is a quick-reference chart of the DL/I status codes IMS/VS returns to application programs.
- Appendix B contains a description of the status codes described in Appendix A.

PREREQUISITE PUBLICATION:

IMS/VS General Information Manual, GH20-1260

ASSOCIATED PUBLICATIONS:

IMS/VS Installation Guide, SH20-9081
IMS/VS System/Application Design Guide, SH20-9025
IMS/VS System Programming Reference Manual, SH20-9027
IMS/VS Operator's Reference Manual, SH20-9028
IMS/VS Utilities Reference Manual, SH20-9029
IMS/VS Messages and Codes Reference Manual, SH20-9030
IMS/VS Message Format Service User's Guide, SH20-9053
IMS/VS Advanced Function for Communications, SH20-9054
IMS/VS Low Level Code/Continuity Check In Data Language/I Program Reference and Operation Manual, SH20-9047
IMS/VS Program Logic Manual, Volume 1 of 3, Logic, LY20-8004
IMS/VS Program Logic Manual, Volume 2 of 3, LY20-8005
IMS/VS Program Logic Manual, Volume 3 of 3, LY20-8041

GUIDE TO USING IMS/VS SYSTEM PUBLICATIONS

Figure P-1 is a guide to using the IMS/VS system publications. This guide is divided into three parts, each dealing with a specific IMS/VS component -- Data Base System, Data Communication feature, and Interactive Query Facility (IQF) feature. For each component, one or more functional areas is identified. For each functional area, one or more tasks is specified, and the IMS/VS manual or manuals that contain major information regarding this task are noted. The titles of the IMS/VS manuals are abbreviated as follows;

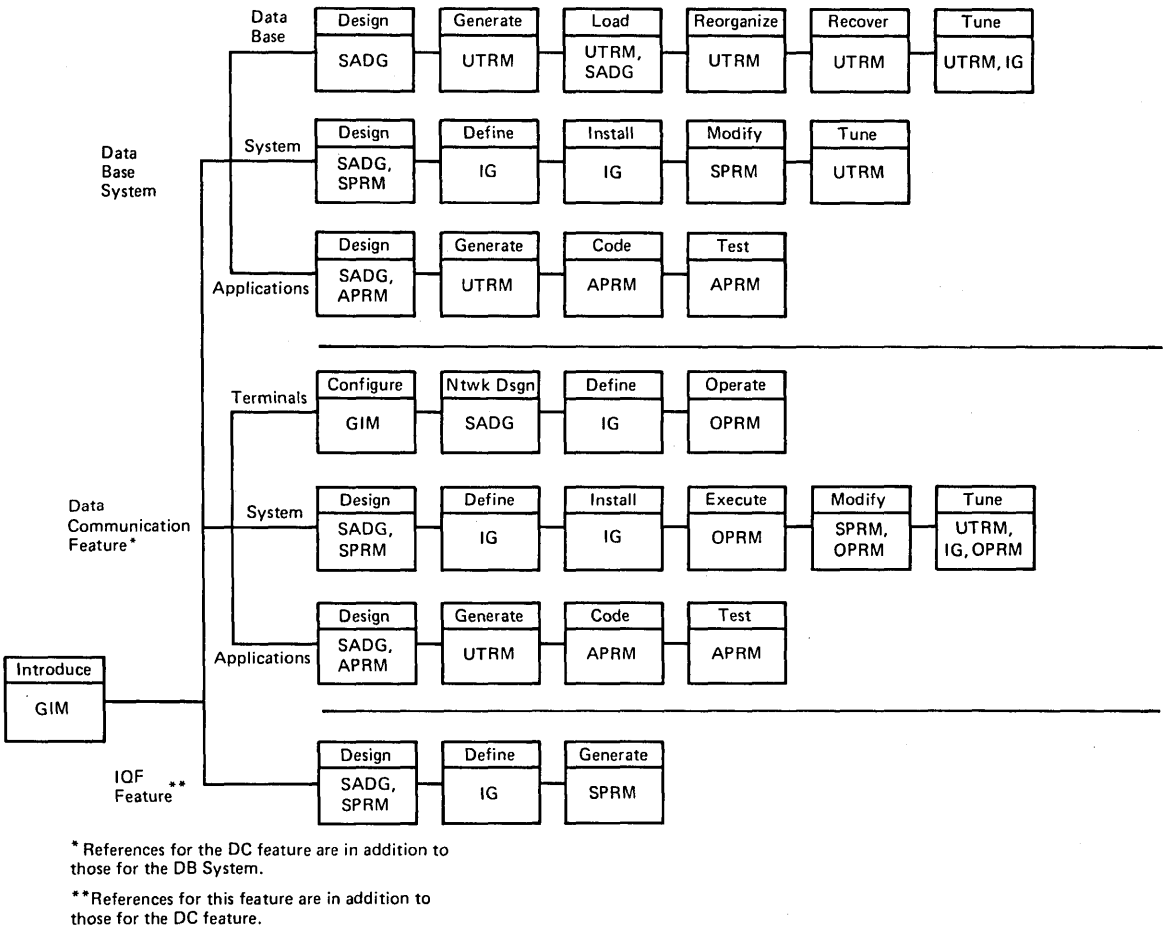
Abbreviation	Full Manual Title
GIM	IMS/VS General Information Manual
SADG	IMS/VS System/Application Design Guide
IG	IMS/VS Installation Guide
SPRM	IMS/VS System Programming Reference Manual
APRM	IMS/VS Application Programming Reference Manual
UTRM	IMS/VS Utilities Reference Manual
OPRM	IMS/VS Operator's Reference Manual

Four IMS/VS manuals are not referred to in Figure P-1:

- IMS/VS Messages and Codes Reference Manual: This manual supports essentially all tasks noted in Figure P-1.
- IMS/VS Low Level Code/Continuity Check in PL/I: Program Reference and Operation Manual: This manual supports the Data Base System when the LLC/CC function is used.
- IMS/VS Message Format Service User's Guide: This manual supports the Data Communication feature when MFS is used.
- IMS/VS Advanced Function for Communications: This manual supports the Data Communication feature when an AFC system is used.

The IQF section of Figure P-1 refers only to IMS/VS system library manuals that contain information on IQF. Additional IQF information can be found in:

- IQF General Information Manual, GH20-1074
- IQF Language Guide, GH20-1222
- IQF Terminal User's Reference Guide, GH20-1223



* References for the DC feature are in addition to those for the DB System.

**References for this feature are in addition to those for the DC feature.

Figure P-1. Guide to Using the IMS/VS System Publications

CONTENTS

FIGURES

SUMMARY OF AMENDMENTS

VERSION 1, RELEASE 1.2

This release reflects technical changes to this publication in support of the following devices:

- 3767 Communication Terminal
- 3770 Data Communication System

OTHER CHANGES

- A symbolic call interface for the extended checkpoint/restart facility has been added. With this facility, COBOL and PL/I application programs can now issue extended CHKP/XRST DL/I calls and also CHKP DL/I calls that specify OS checkpoints.
- Updates have been made to PL/I information, and a revised example is included for the PL/I Optimizing Compiler.
- Chapter 7 of this edition comprises the "DL/I Test Program" that was formerly Appendix C of the IMS/VS Utilities Reference Manual, and "Message Processing Region Simulation" that was formerly Appendix B of the IMS/VS System/Application Design Guide.

VERSION 1, MODIFICATION LEVEL 1.1

- Support has been added for the 3740 Data Entry System. IMS/VS supports the 3741 Data Station, Model 2, and the 3741 Programmable Work Station, Model 4, attached on a switched line using BTAM.
- The restriction against the Utility Control Facility (UCF) has been lifted.

VERSION 1, MODIFICATION LEVEL 1

The following new and/or enhanced IMS/VS functions have been added:

- Generalized Sequential Access Method (GSAM).
- Expanded restart (restart call), GET SCD call, and Statistics call
- Response Alternate PCBs.
- Fixed-length SPAs.
- Program Isolation.
- Application program output limits.
- Message Format Service (MFS) support for additional terminals.

Note: Information in this manual about the Utility Control Facility (UCF) is for planning purposes only until that facility becomes available.

VERSION 1, MODIFICATION LEVEL 0.1

- Support for the IBM 2260 Display Station, Model 1 and 2, and for the IBM 2265 Display Station, Model 1.

CHAPTER 1. IMS/VS ENVIRONMENT FOR APPLICATION PROGRAMMING

The objectives of the IMS/VS Data Base (DB) facility are to enable multi-application use of shared data, with greater integrity of the data itself, and with greater independence from data management for the programs, the application programmers, and the users. For the full Data Base/Data Communications (DB/DC) facility of IMS/VS, these objectives extend to multi-application use of shared terminals, with greater integrity of the transmission, and with greater independence from the mechanics of terminal hardware and teleprocessing procedures.

EFFECT ON APPLICATION PROGRAMMERS

The real effect of IMS/VS on application programming groups occurs in organizational procedures. There will be a significant difference in how a data organization is designed, who does it, and at what point in time. The manner in which data is administered and maintained will change, and a significant change should occur in the interface between an application programming group and the systems function in the central data processing organization.

PRE-DB/DC ORGANIZATIONAL PROCEDURES

In most companies, application programming has been scattered throughout the various functional areas of the company. The central data processing organization provided an interface advisory function, establishing procedures for using the system and determining resource requirements for these functional groups. But each group designed and implemented its independent programs and independent data files, and each group negotiated and programmed for its own teleprocessing terminals.

DB/DC ORGANIZATIONAL PROCEDURES

To obtain the most effective use of an IMS/VS system, users may wish to consider an adjustment in functions and procedures. There must be a central coordination of the data base structures and contents, since these structures are to be shared by multiple functional areas. Accordingly, a new function of "Data Base Administrator" may be found desirable in the central data processing organization, as illustrated in Figure 1-1.

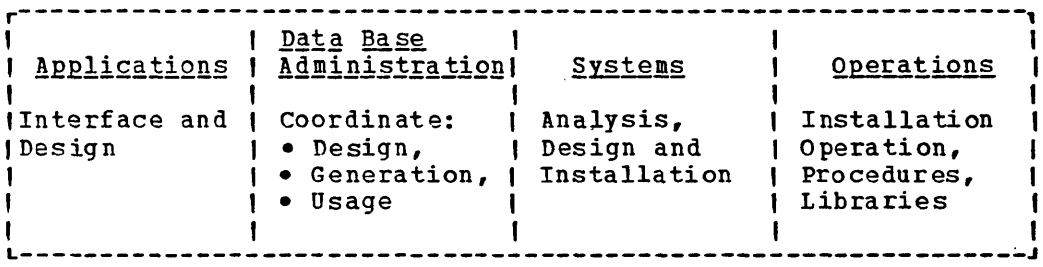


Figure 1-1. Basic Functions of a User Installation

Since the decision to install IMS/VS is actually a decision to make an integrated data organization fulfill the requirements of multiple application programs, a focal management function becomes desirable to:

- coordinate current application requirements;
- anticipate future requirements of current and future applications;
- plan, schedule, and control the design, installation, and access to data bases; generate all data bases;
- inform applications personnel of existing data structures and provide guidelines as to their use;
- analyze and evaluate the effect of current or planned data structures on overall system performance;
- coordinate with systems and operations organizations the development of effective procedures for data protection.

Naturally, the other three functional areas are very much affected by and involved in designing and installing a data base system. But the important interface with the DP organization now becomes, for application programming groups, the Data Base Administration function.

Figure 1-2 exemplifies the functional relationship which develops between application programming groups and data base administration. Whereas application groups used to design both programs and data files, now the design of the data structures referenced by application programs becomes a joint task. Developing the procedures for implementing that joint design function can be one of the most important tasks an installation faces.

Second, and equally important, an equivalent focal point is required to coordinate and control the teleprocessing network; to keep track of the location and use of physical terminals; to map logical users onto the physical network; and to plan, schedule, and control the dynamics of message traffic, and the load on the central data processing system.

Figure 1-2. Application Analysis Joint Interface with Data Base Administration

EFFECT ON NEW PROGRAMS

The changes described above are procedural and organizational. The net effect on conventional, pre-IMS/VS application programming tasks is simplification:

- New applications using IMS/VS will require much simpler data I/O and message I/O procedures.
- Follow-on maintenance of any IMS/VS application should be significantly reduced due to the logical independence from data files and teleprocessing hardware.

IMS/VS DATA BASE VERSUS OS/VS FILE DESIGN AND ACCESS

Application analysts and programmers converting to use of IMS/VS for new applications find their task considerably simplified because all data description and file definition occur externally. The programmer is relieved of the need to build these functions into the application, and can concentrate on the symbolic representation of the application data and their logical interrelationships.

Under the System/370 operating systems and data management services, a "data set" is considered the major unit of data storage and retrieval. A data set is made up of physical records each of which in turn, may contain multiple logical records.

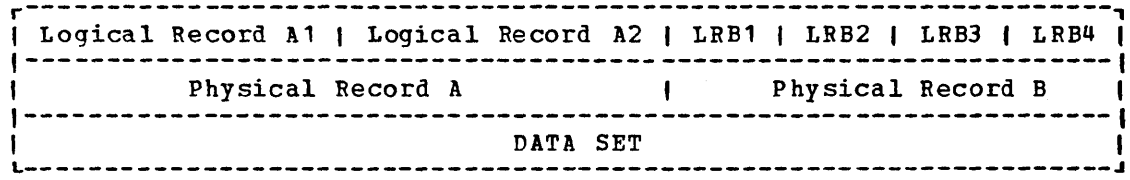


Figure 1-3. OS/VS Data Management Data Structure

This OS/VS structure is shown in Figure 1-3. The application program is constrained by this structure: its definition must be a part of the program, the logical representation of data must be within the bounds of the physical structure, and any change in the structure almost surely will require a change in the program.

Under IMS/VS, logical elements are identifiable and processable by the programmer with no knowledge of or reference to the physical format. Figure 1-4 illustrates the difference.

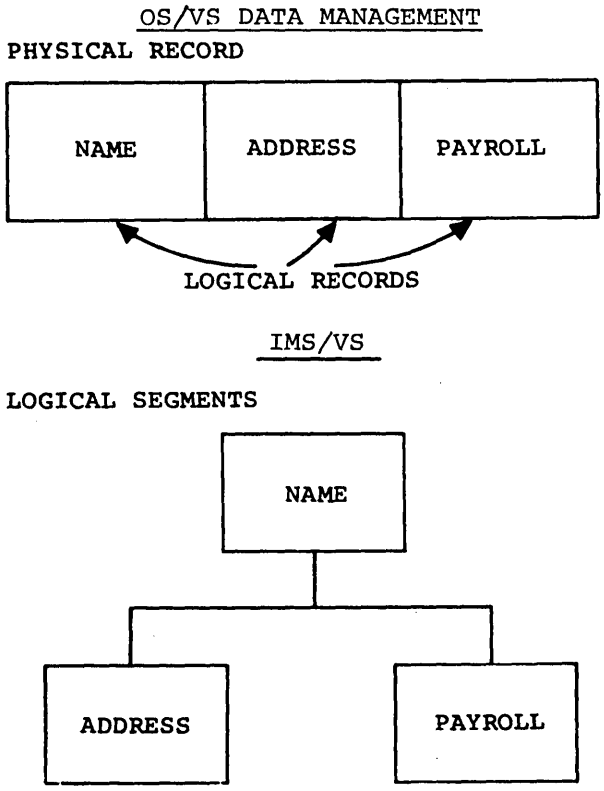


Figure 1-4. Comparison of OS/VS Physical Record and IMS/VS Logical Segment Relationship

IMS/VS DATA COMMUNICATION VERSUS OS/VS TELEPROCESSING

The task of application programmers writing data communications programs is simplified to a large degree by being able to deal just with logical terminals within the program. IMS/VS handles the teleprocessing access method, the correlation between logical and physical terminals, and distinctions between the hardware characteristics of various terminal devices.

IMS/VS VERSUS NON-IMS/VS PROGRAM STRUCTURE

The IMS/VS system capabilities which enable the programmer to deal exclusively at a logical level with data and terminals consist of two principal facilities:

- an offline facility for generating control blocks which accomplish the mapping between logical and actual data and between logical and actual terminals. This facility is intended to be administered by DB/DC administration in the central data processing organization;
- an inline high-level language called Data Language/I (DL/I) which interprets and processes data and/or message input/output requests during program execution. Programmers invoke DL/I via structured CALLs from PL/I, COBOL or Assembler Language programs.

A detailed description of these features with respect to batch data processing and online message processing constitute the remaining chapters.

CONVERTING EXISTING PROGRAMS

The task of converting an existing application program to enable its use of IMS/VS data structures requires analysis by the application group in consultation with the data base administrator and other DP systems personnel. Two factors are important in this analysis: data integrity and program performance.

If data integrity is critical and can be markedly improved by shifting to an IMS/VS structure, and if the present I/O procedures in the application can be located and converted to IMS/VS I/O procedures in a straight-forward manner, then the installation may find that an initial conversion can be accomplished by altering just the program I/O areas. At the same time, program performance should be analyzed so that the effect of this initial change on the system and on the application users can be evaluated.

Where performance is critical, IMS/VS users generally find it desirable to redesign the application so as to take full advantage of the facilities IMS/VS offers. This is particularly true where the application has been accessing sequential files and doing minimal processing.

MAJOR CONSIDERATIONS IN IMPLEMENTING AN IMS/VS APPLICATION

Figure 1-5 describes the major steps required to activate an IMS/VS system. The items to the right are some of the decisions which must be made before any of the center actions can be taken. This figure shows the context in which an IMS/VS application is implemented.

Looking just at the activity of creating an application program, it would appear that aside from the logic of the application itself, an application programmer need be concerned only with selecting the programming language and observing the IMS/VS interface conventions. This can be quite true for individual application programmers.

Concurrently, however, the application programming management and the application analysts must actively participate in the design of the logical data structures and the definition of how the program will use its data bases. The vertical columns on the left show that these tasks:

- must occur earlier than they previously may have been undertaken;
- must be shared, in an organized fashion, between systems and application personnel.

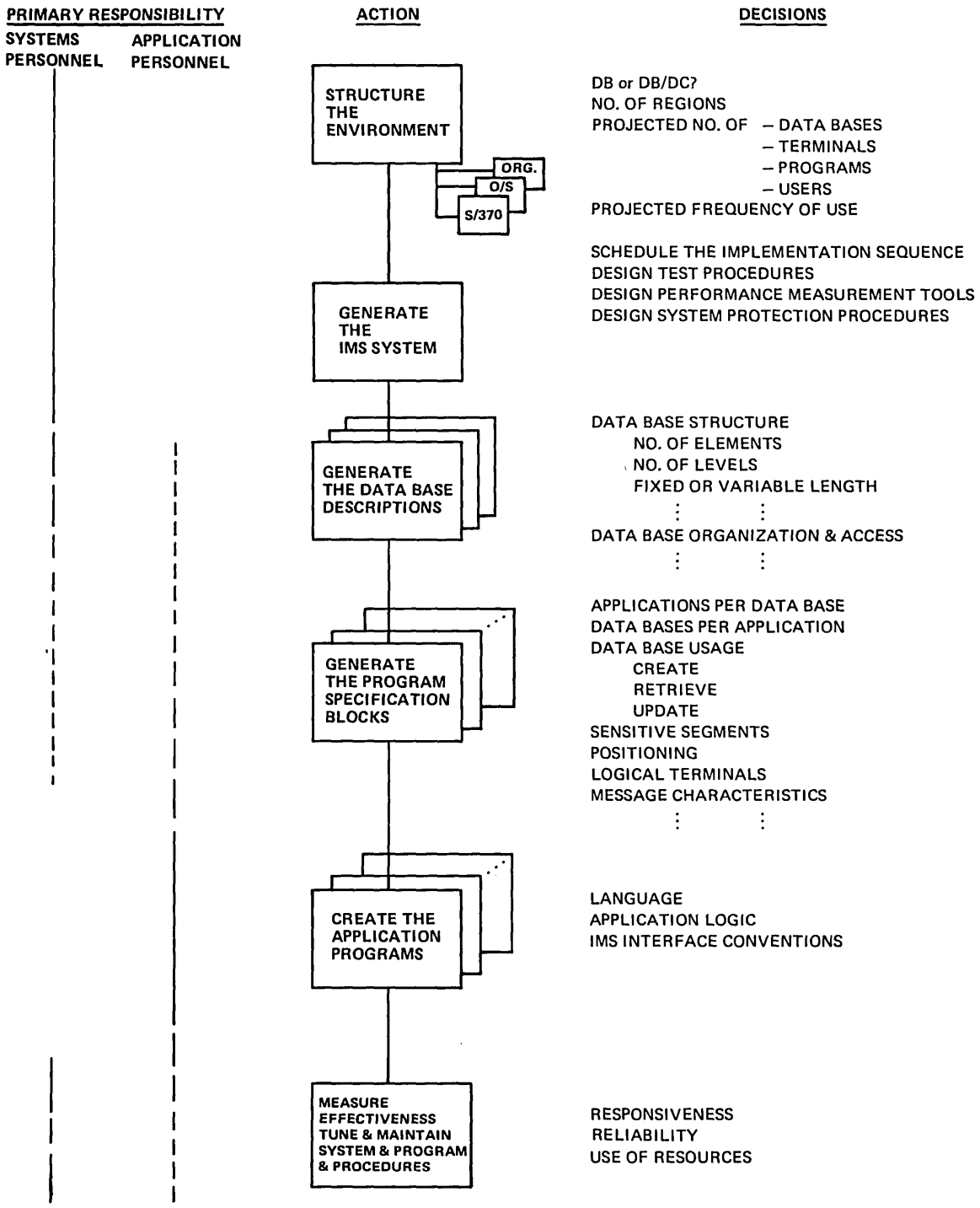


Figure 1-5. Decisions, Actions, and Responsibilities for the Design, Implementation, and Continued Use of an IMS/VS System

CHAPTER 2. DATA BASE BATCH PROGRAMMING

This chapter provides application analysts and programmers with reference material on the basic capabilities of the IMS/VS Data Base (DB) facility. The DB facility is available to and used by all IMS/VS application programs, whether operating in a batch mode, batch message processing mode, or message processing mode. The Data Communications (DC) facility of IMS/VS is required for the latter two modes of operation. Application programming for the DC facility is described in later chapters of this manual.

This chapter concentrates on the DB facility and addresses the two fundamental aspects of the IMS/VS application programming task:

- Designing application views of data (logical data structures)
- Interfacing the application program with IMS/VS

With respect to data design, the application analyst wants to know how to:

- Define each of the elements of data required by the program and the ways each element will be processed
- Organize the data elements and indicate their relationships

With respect to interfacing with IMS/VS, both the application analyst and the programmer want to know:

- How the program can access data
- How the program can identify the basic data elements and the way they can be processed
- How, where, and in what form IMS/VS responds to the program requests

Data design is shared by the application analyst and the data base administrator. Whereas application programming is concerned with logical views of data, data base administration is concerned with providing physical data structures to make those logical views possible while at the same time meeting the requirements of other applications, system performance, and data protection. The details of IMS/VS data structures, from the application analyst's and programmer's viewpoint, are described in the first section of this chapter.

The second concern, the operational interface between the application program and IMS/VS, is covered in the second section. It describes what is contributed to the interface by IMS/VS:

- Data Language/I (DL/I)
- The Program Communication Blocks (PCBs)

and by the programmer:

- DL/I calls
- PCB masks

It is assumed that the reader of this chapter is familiar with the IMS/VS General Information Manual, and has attended an IMS/VS "Concepts and Facilities" class, or the equivalent.

It is suggested that the reader also refer to relevant portions of the IMS/VS System/Application Design Guide and the IMS/VS Utilities Reference Manual.

The IMS/VS Messages and Codes Reference Manual will be a necessary tool, once the programmer begins to develop and check out the application program.

IMS/VS DATA BASE ORGANIZATION

The cornerstone of the IMS/VS Data Base (DB) facility is the capability to overlay multiple "logical" (application-oriented) data structures on non-repetitive "physical" data organizations. It is this concept which enables an application programmer to consider only the data with which the application is concerned, structured in a manner which satisfies the functional requirements of the program logic rather than the interests of physical storage or access methods. The application programmer, and in many instances the application analyst, need not be concerned with any data that is extraneous to the program or the physical organization of data.

In this chapter, four distinct kinds of IMS/VS data structures are identified:

- Physical data bases
- Logical data bases
- Logical data structures
- Application data structures

Physical and logical data bases are "internal", system-oriented structures. Logical and application data structures are "external," application-oriented structures.

Physical data bases define to IMS/VS the format of each actual data element, the relationships between data elements, and how these elements are to be organized in physical storage. Logical data bases (using "logical relationships" specified in physical data bases) define a structural relationship among actual data elements in one or more physical data bases which is different from the structural relationship defined in the physical data base(s). Physical and logical data base structures are designed by the data base administrator to meet the combined requirements of multiple application programs. Definition of these structures to IMS/VS is accomplished via the DBDGEN utility program.

An application data structure specifies to IMS/VS what data the program will process and what structural view the program takes of that data. One and only one application data structure must be defined for each program. An application data structure consists of one or more logical data structures. A logical data structure specifies what data the program will process within a particular logical or physical data base. Application data structures are functionally designed by the application analyst. Logical data structures are designed by the data base administrator often together with the application analyst, using the application data structure. Definition of these structures to IMS/VS is accomplished via the PSBGEN utility program: a logical data

structure is that structure defined in a PCB; an application data structure is that structure defined in the PSB.

These four structures are discussed in the sections "Design and Definition of IMS/VS Data Bases" and "Design and Definition of Application and Logical Data Structures" later in this chapter. The remainder of this section describes aspects which are common to all four structures:

- Structure -- hierarchical
- Basic element -- the segment
- Hierarchical interrelationships
- Limits on the design of data structures

STRUCTURE: HIERARCHICAL

Relationships of Data Elements

In IMS/VS, all data is organized in hierarchical structures. These structures consist of elements of data interconnected to show relationships. The elements of data are called "segments" and are described below. In a hierarchical structure, the relationships indicate either dependency or equivalence. In IMS/VS, dependency is called a "parent-child" relationship; equivalence is called a "twin" relationship. The schematic convention for representing an IMS/VS data structure is shown in Figure 2-1.

In Figure 2-1, B1 is a child of A1 and a parent of C1 through G1, but not of F1 whose parent is E2. Elements D1, D2, and D3 are twins, F1 and F2 are also twins, as are I1 and I2. Elements C1 through G1 (except for F1) are children of B1, and elements I1 through J1 are children of H1. Element K1 is a child of J1. D1 and E1 are not considered twins, even though they have a sibling relationship under B1. Elements G1 and I1 have different parentage and hence are not related. A parent may have 0 to n children; a child may have only one parent; a child may have 0 to n twins.

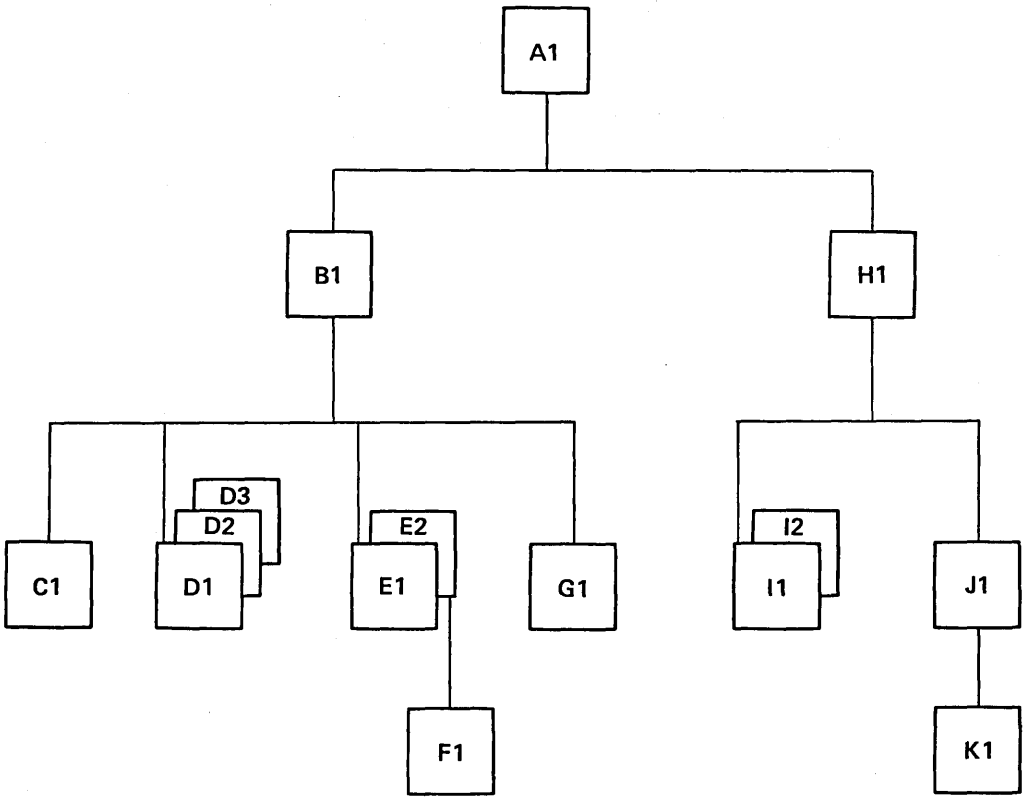


Figure 2-1. Schematic Representation of a Hierarchical Data Structure

Levels

The successive dependencies of a hierarchical structure are called "levels." In Figure 2-1 there are 4 levels: A is the top level, B and H, the second level, C, D, E, G, I, J make up the third level; and F and K are the bottom level. An IMS/VS data base may have a maximum of 15 levels.

Traversal

By convention, IMS/VS traverses a hierarchical structure from top to bottom, front to back, left to right. At every position, it seeks a lower level; if none exists, it seeks the next-right element on the same level; if none exists, it seeks, in the level immediately above, the element which is next-right to the last element it had reached earlier at that level. The data base in Figure 2-1 would be traversed in alphabetic sequence, A1, B1, C1, D1, D2, D3, E1, E2, F1, G1, H1, I1, I2, J1, K1.

When an application retrieval request says to get the next segment, this traversal order is used by IMS/VS.

When the term "position" is used later in this chapter, position along this sequence is meant, and "forward from current position" means forward according to this sequence.

BASIC ELEMENT: THE SEGMENT

The basic element of data in any IMS/VS data structure is called a "segment."

All segment types may be either fixed or variable length.

Segments may comprise one or more "fields." One field per segment in a logical data base may be identified as a "key field." A key field is used by IMS/VS for indexing, searching, and sequencing purposes. Searches can be carried out also on non-key fields. In defining the structure of a data base to IMS/VS, each element of the structure is identified as a "segment type." In Figure 2-1, each of the alphabetic elements, A through K would be defined at data base definition time as "segment types." Later at load time, there can be 0 to n "segment-occurences" of any segment type. In Figure 2-1, D1,D2,D3 are segment-occurences of segment type D. In discussing a data base, it is important to distinguish between the generic term "segment type" and specific "segments" or "segment-occurences."

HIERARCHICAL INTERRELATIONSHIPS

Root Segments

In the hierarchy of an IMS/VS data structure, the highest (top) level segments are called "root segments." A root segment can be only a parent, never a child.

Path

A hierarchical "path" is the sequence of segment occurrences, one per level, leading directly from a segment at one level to a particular segment at a lower level. In figure 2-1, A1-B1-E2-F1 is a path. Paths are used in processing to reach a segment below the root level.

Data Base Record

A single occurrence of a root segment and all of its dependents is defined as a "data base record." The concept of data base record is more useful to systems personnel setting up the physical storage of a data base than to application analysts or programmers.

Figure 2-2 compares a conventional OS/VS data management physical record with an IMS/VS data base structure.

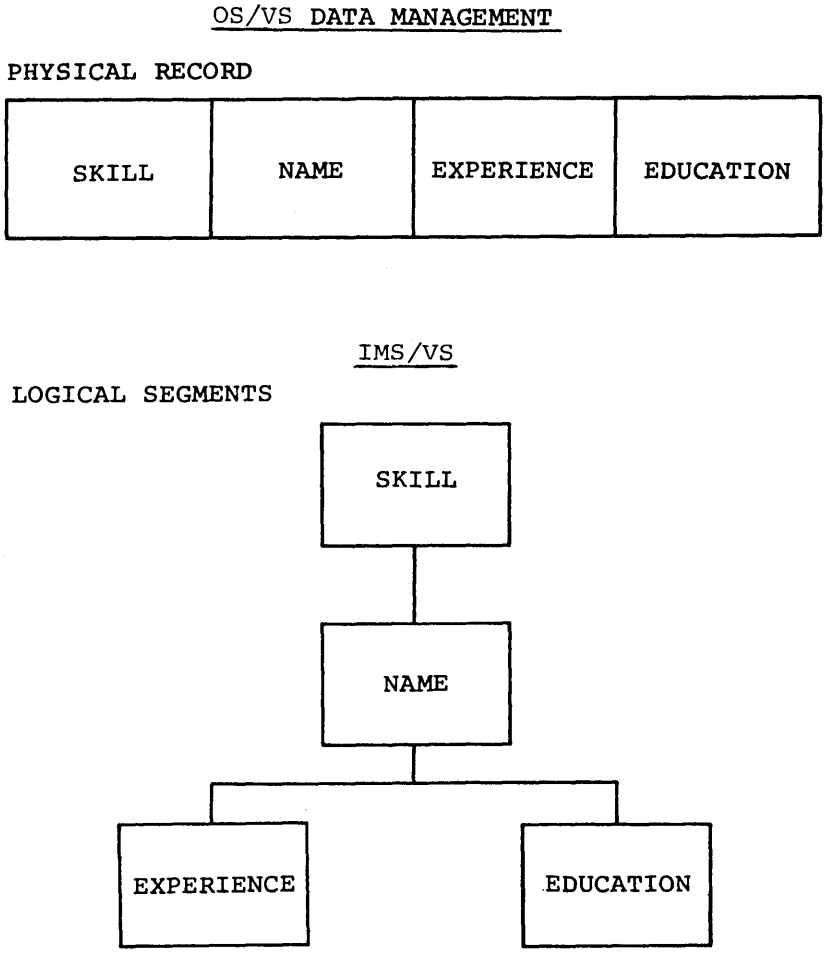


Figure 2-2. OS/VS Data Management -- IMS/VS Data Base Relationship

Figure 2-3 shows a typical data base record which the IMS/VS structure of Figure 2-2 might contain. (Notice the data redundancy implied: If this were an OS/VS record, "Adams" might occur 5 times, Jones 6, and Smith 2. "Skill" (the root segment) might occur 13 times).

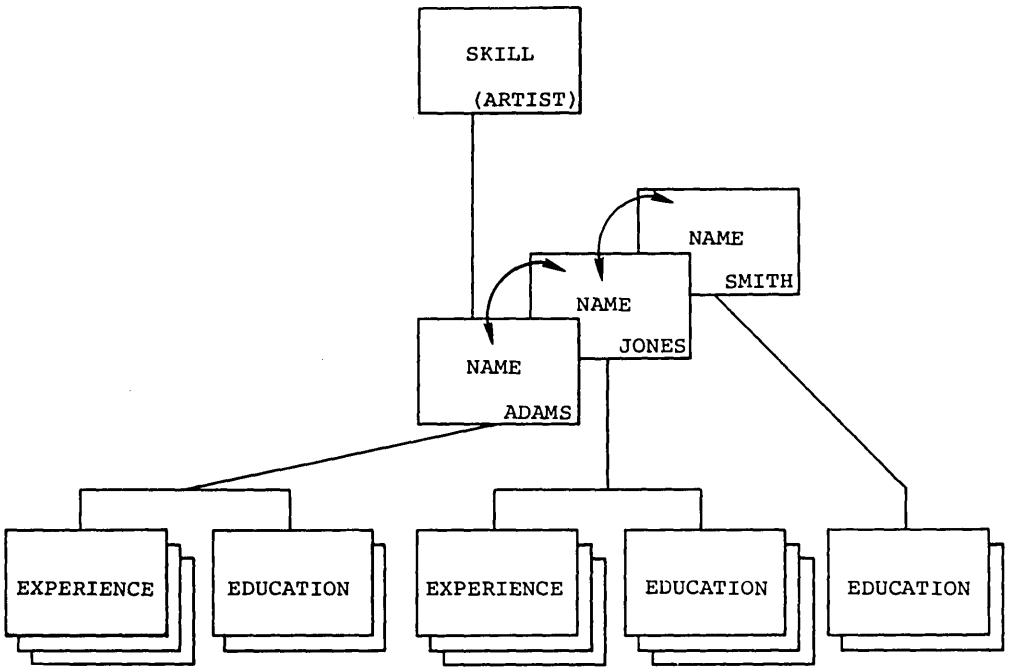


Figure 2-3. IMS/VS Data Base Record

LIMITS ON THE DESIGN OF DATA STRUCTURES

The rules which constrain the size and extent of an IMS/VS data structure are summarized in Figure 2-4.

	No. per Data Base	Dependents per Parent-Segment Type
Levels	1 to 15	0 to 14
Segment types	1 to 255	0 to 254
Segment occurrences	1 to n	0 to n

Figure 2-4. Data Base Structural Limits

DESIGN AND DEFINITION OF IMS/VS DATA BASES

PHYSICAL DATA BASES

Physical data bases represent the organization and access method of actual data on the storage medium. They define the actual format and content of each data segment type, as well as all the physical relationships which exist between segment types. In addition they include all the "logical relationships" by means of which potential alternate paths between segment types can be defined. The existence of these logical relationships enable the definition of logical data bases (see below).

Physical data bases must be designed by the data base administrator, who has the responsibility of coordinating the data requirements of multiple application programs.

Physical data bases are defined to IMS/VS via the "Data Base Definition Generation" (DBDGEN) utility program which is part of the IMS/VS program product package. The definition, like the design, of Physical data bases should be the responsibility of the data base administrator.

LOGICAL DATA BASES

Logical data bases define logical hierarchical structures. These structures are composed of segment types defined in physical data bases, and are implemented by means of the "logical relationships" defined in those data bases for those segments. Any given logical data base is a hierarchical view of segment types selected from one or more physical data bases; segment types from any given physical data base can "belong to" multiple logical data bases.

Logical data bases must be designed by the data base administrator, based on functional specifications of data requirements provided by application analysts.

Logical data bases are defined to IMS/VS via the same "DBDGEN" utility program used to define physical data bases. Conceivably, generating a new logical data base may require multiple "DBDGEN" runs: one to define the logical data base, preceded by one or more additional runs to specify the required logical relationships in the referenced physical data base(s).

DESIGN AND DEFINITION OF APPLICATION AND LOGICAL DATA STRUCTURES

An application data structure defines the complete hierarchy of segment types which is unique to a single application and describes the kind of processing intended by the application against each segment type. An application data structure enables IMS/VS to tailor its DB facility to the requirements of each application as it is executed. An application data structure, once it is designed and defined, consists of one or more logical data structures. The design of these logical data structure subsets evolves during the process of designing the application data structure.

Application Data-Structure Design

The design of an application data structure is based on functional specifications provided by the application analyst to the data base administrator. These functional specifications should describe, at minimum, the data elements (segment types) to be processed, their hierarchical relationships, and the processing intent of the program against each segment type. Additional information to be included in the specifications must be determined by each installation, as well as the manner in which the specifications are communicated.

From these specifications, the data base administrator designs an application data structure (and its logical data structure subsets) which will satisfy the data and processing requirements of the program, will optimize system and program performance, and will protect the integrity of the program, the other programs which share the use of the data, and the data bases themselves.

In most installations, the data base administrator needs the active participation of the application analyst during this design task. The design of an application data structure is essentially a "mapping" process in which the external program "view" of its data is mapped onto portions of existing or proposed internal structure (that is, physical and/or logical data bases). Usually, several alternative mappings are possible, and the effect of each on the design and performance of the program needs to be evaluated by the application analyst.

The final result of the application data structure design process is a set of logical data structures. Each of these structures identifies an IMS/VS data base (physical or logical) the program will access, the segment types the program will use (be "sensitive" to), and the type of processing the application program will perform on each segment type. In addition, the design process may disclose that:

- An existing data base satisfies the requirements.
- A cross-section of one or more existing data bases can be used.
- A new logical data base must be defined.
- A new physical data base must be generated, or an existing physical data base must be reconstructed.
- The program requirements must be redefined -- data cannot be made available as requested.

Application and Logical Data-Structure Definition

Application and logical data structures are defined to IMS/VS by using the Program Specification Block Generation (PSBGEN) utility program. The Program Specification Block (PSB) thus generated consists of Program Communication Blocks (PCBs). Each PCB identifies a physical or logical data base (defined, in turn, by DBDGEN) which the program will access. (PSBGEN also identifies resources associated with the use of the IMS/VS DC facility; this aspect is discussed in a later chapter.)

The basic information provided to IMS/VS by each PCB definition is the identification of each segment type within the physical or logical data base which will be processed, and the type of processing which will be done. (Additional information specifies concatenated key length, and options on the advanced functions "multiple positioning" and "secondary indexing" described in the next chapter). The contents

of the actual PCB generated by the IMS/VS PSBGEN utility program are described in detail later in this chapter.

PROCESSING OPTIONS: The processing options which can be specified are combinations of the DL/I call functions (get, insert, replace, delete) and additional processing logic such as read only and load only. A processing option may be specified for each segment type to which the program is sensitive. If it is not, IMS/VS defaults to the processing option which must be specified for the entire data base.

SEGMENT-TYPE SENSITIVITY: PSBGEN also identifies the specific segment types within a data base which the application program intends to process. An application program can be key-sensitive, data-sensitive, or not sensitive to segment types. If a program is not sensitive to a segment type, then it cannot access occurrences of that segment type or their dependents. Dependents of key-sensitive segments can be accessed if there is data sensitivity to the dependent segments. If the program is key-sensitive to a segment, the program can specify that segment in an SSA but cannot access the segment itself. If it is data-sensitive, it can access the segment. Data sensitivity implies key sensitivity.

DATA BASE IDENTIFICATION: A logical or physical data base may be specified more than once in the PSBGEN for an application program. This can be a useful processing tool: for example, when it is desirable to maintain multiple positions in a data base, or to separate one processing option from another. See the discussion later in this chapter on "Access to Multiple Data Bases."

References

To work effectively with the data base administrator, applications analysts should be familiar with source documents for the PSBGEN and DBDGEN utility programs as described in the IMS/VS System/Application Design Guide and the IMS/VS Utilities Reference Manual. The Program Communication Blocks (PCBs) which make up the PSB for any application are described in further detail in this chapter, particularly in terms of their use to an application programmer.

INITIALLY LOADING A DATA BASE

Once a data base structure is defined, data can be loaded into it. Data base loading is accomplished by a user-written application program as described in the IMS/VS Installation Guide. The program employs one of the data processing Call functions IMS/VS provides for this purpose. All of these Call functions are described in the next section. Certain pointer relationships must be resolved when a data base is initially loaded. IMS/VS utilities are provided for this purpose and are described in the "Data Base Reorganization/Load Processing" chapter of the IMS/VS Utilities Reference Manual. Other considerations of initial load are also discussed there.

ACCESSING A DATA BASE

Application programs for which a PSB has been generated are able to access their relevant data bases by issuing calls to DL/I. The call format names the PCB of the logical structure, identifies the segment(s) desired, and specifies the processing function to be performed.

PROGRAM STRUCTURE AND INTERFACE TO IMS/VS

The operational interface which IMS/VS provides to the application program is composed of two components, DL/I and the Program Communication Blocks (PCBs). They provide communication between IMS/VS and the running program, and enable an application to process data in an IMS/VS data base.

Figure 2-5 illustrates the elements of the interface and their relationships.

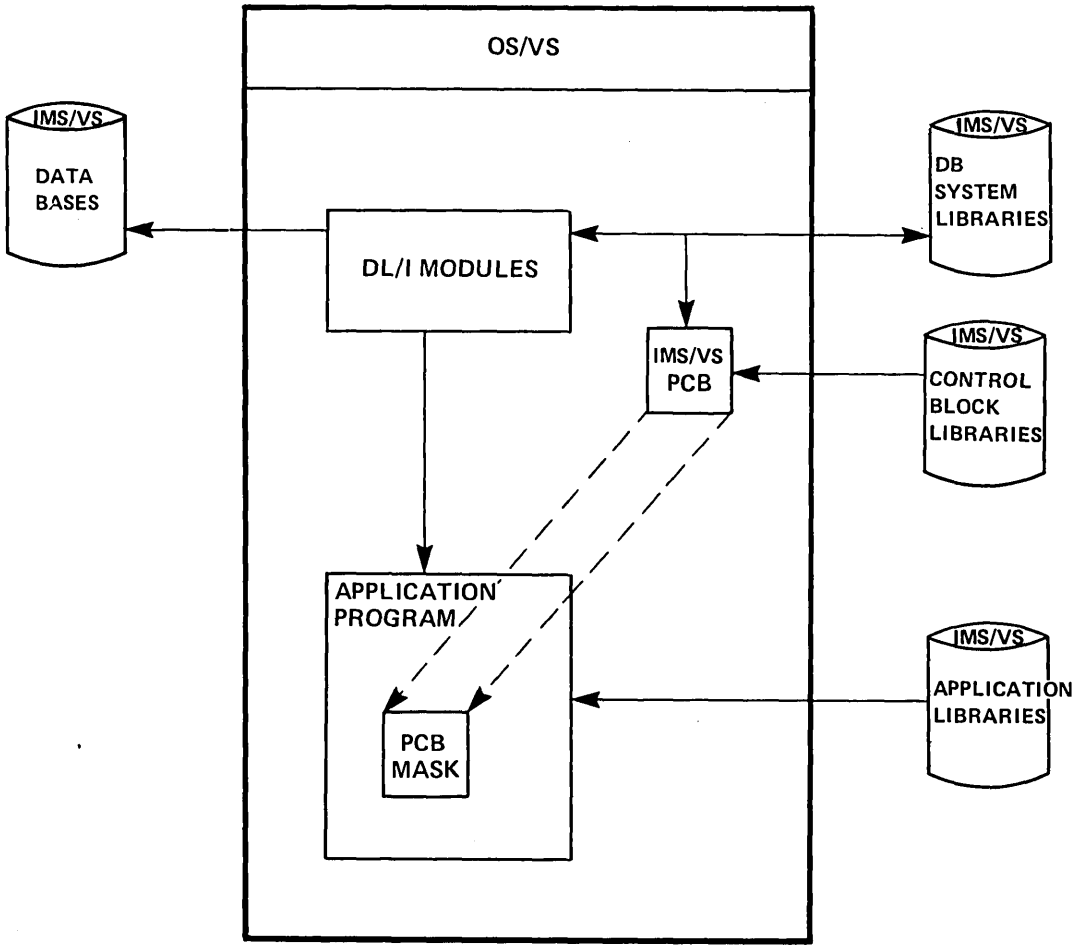


Figure 2-5. IMS/VS Interface with Application Program

Prior to execution of the application program, the data base administrator must execute the IMS/VS Program Specification Block Generation utility program (PSBGEN) to create the PCBs. The PCBs (one for each logical data structure the application will access) are placed in a system library, ready for use by IMS/VS whenever the application is executed.

DL/I is a set of IMS/VS modules which reside in the batch region with the application program. DL/I interprets the data-processing CALL requests issued by the program.

The application program interfaces with these two IMS/VS components via the following program elements:

- An ENTRY statement specifying the PCBs for the program
- A PCB-mask which echoes the information maintained in the pre-constructed IMS/VS PCB and which receives return information from DL/I
- An I/O area for passing data segments to and from IMS/VS data bases
- Calls to DL/I specifying processing functions
- A termination statement

The PCB mask(s) and I/O areas are described in the program's data declaration portion. Program entry, calls to IMS/VS, processing, and program termination are described in the program's procedural portion. Calls to IMS/VS, processing statements, and program termination may reference PCB mask(s) and/or I/O areas. In addition, IMS/VS may reference these data areas.

Figure 2-6 illustrates how these elements are functionally structured in a program and how they relate to DL/I. The elements are discussed in the text that follows.

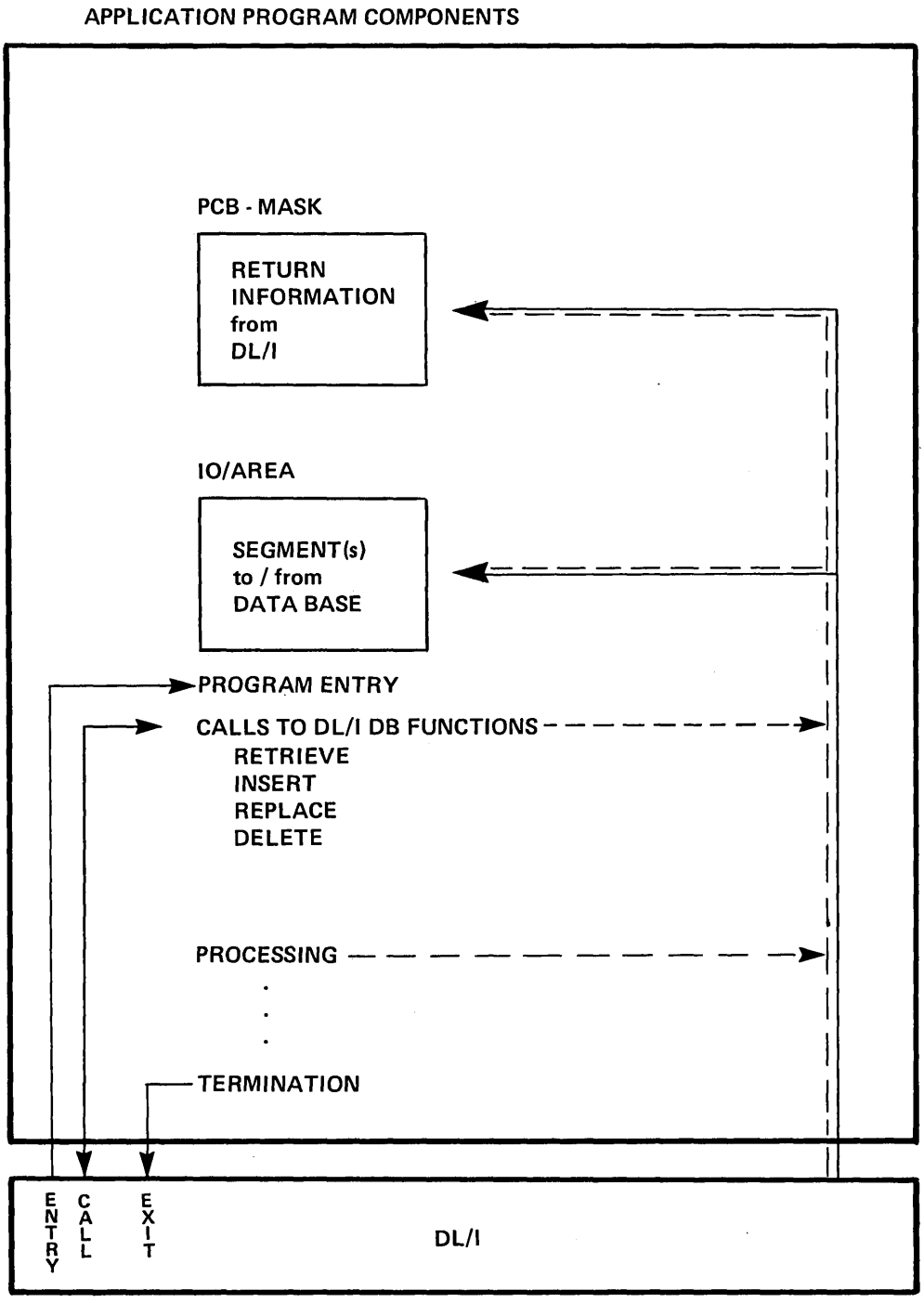


Figure 2-6. Structure of a Batch Application Program

LANGUAGE AND COMPILATION

The application program is written in one of three languages: PL/I, COBOL, or Assembler Language. All of the examples in this manual employ versions of these languages. The program is compiled through the user-selected language compiler and placed in the appropriate program library.

After the PSB for an application program has been generated and the program itself has been compiled, the program can be executed in an IMS/VS batch environment. Figure 2-7 depicts two environments. One is the conventional application program with its embedded file description and its use of the operating system data management directly. The second environment is IMS/VS. Here, under IMS/VS control, both the application program to be executed and its associated PSB are loaded from their respective libraries. The PSB contains the PCBs of the data structures to be used by the application program.

ENTRY POINTS TO APPLICATION PROGRAMS

As illustrated in Figure 2-7, when the operating system gives control to the IMS/VS control facility, the IMS/VS control program in turn passes control to the application program (through the entry points as defined in the following examples) and specifies all the pcb-names used by the application program. The order of the pcb-names in the entry statement must be in the same sequence as specified in the PSB generation run for this application program. The sequence of PCBs in the linkage section or declaration portion of the application program need not be the same as the sequence in the entry statement.

Batch DL/I programs cannot be passed parameter field information from the EXEC statement PARM keywords.

Programs that are OS/VS subtasks of an application program called by IMS/VS must not issue DL/I calls. If they do, the results will be unpredictable.

It should be noted that with PL/I, whenever PL/I multitasking is used, all tasks, even the apparent main task, operate as subtasks to a hidden PL/I control task. PL/I tasking is therefore not allowed in an IMS/VS program.

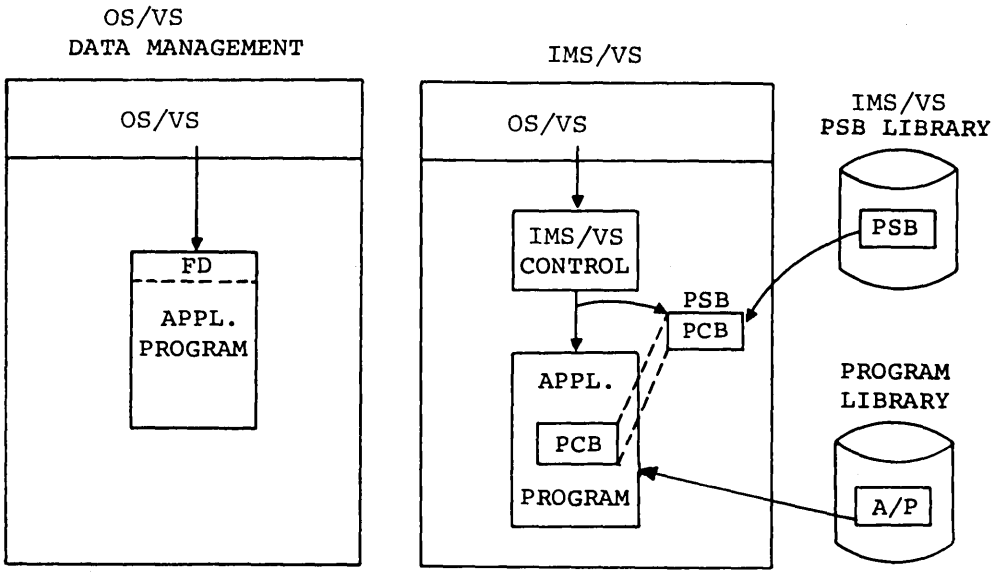


Figure 2-7. IMS/VS Batch Environment Comparison to OS/VS

Initial Invocation of a PL/I Transaction

Programs generated by the OS PL/I optimizing compiler can be entered by one of three entry points -- PLISTART, PLICALLA, and PLICALLB. These entry points differ in the parameter list each expects to receive. PLISTART is the default that is used for entry directly from the OS Scheduler. For this reason, it is not suitable for use by programs running under IMS/VS. Either PLICALLA or PLICALLB can be used under IMS/VS, but the following considerations apply:

- If the PL/I execution options (for example, ISASIZE) are specified through PLIXOPT (see the description of this module in the PL/I Programmer's Guide, SC33-0006) or have satisfactory defaults (specified during installation of PL/I), PLICALLA can be used by including an ENTRY PLICALLA control statement during link-editing.
- If PLIXOPT cannot be used to specify the options (because, for instance, the scanning of PLIXOPT by PL/I initialization routines is time-consuming), and the default options are not suitable for this particular transaction, PLICALLB can be used as the entry point. PLICALLB must be called, however, by a user-written assembler program which passes a parameter list that describes the execution options. The load module entry point must be included in the assembler routine.

Examples

For COBOL, the following entry appears first, in the beginning of the Procedure Division:

```
ENTRY 'DLITCBL' USING pcb-name1,...pcb-namen.
```

For PL/I, the first procedure of a program should be:

```
DLITPLI:  PROCEDURE (pcb_name1,...pcb_namen) OPTIONS(MAIN);
```

The MAIN procedure statement of a PL/I program should be:

```
DLITPLI: PROCEDURE (pcb_ptr1,...,pcb_ptr) OPTIONS(MAIN);
```

Note that the parameters are pointers. The actual PCBs are declared:

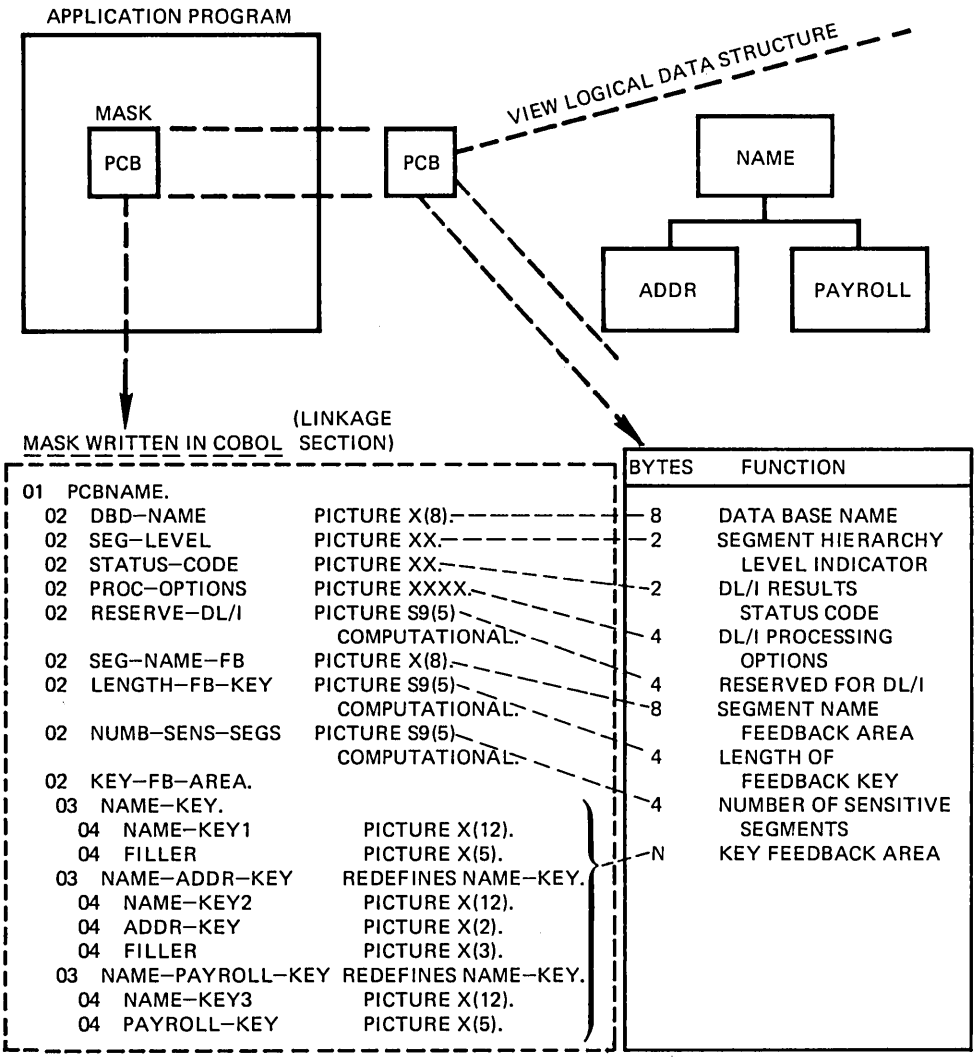
```
DCL 1 pcb_namei BASED(pcb_ptri),
```

Note also that DLITPLI will not be the load module entry point. With IMS/VS, PL/I programs are entered through entry points PLICALLA or PLICALLB.

For an Assembler Language program that utilizes DL/I, the entry point can have any name. However, Register 1, upon entry to the application program, contains the address of a variable length fullword parameter list. Each word in this list contains a control block address which must be saved by the application program. The high-order byte of the last word in the parameter list has the 0 bit set to a value of one to indicate the end of the list. The addresses (PCB control block addresses) in this list are subsequently used by the application program when executing DL/I calls.

DATA BASE PCB MASKS

A data base-PCB mask or skeleton must be provided in the application program through which it views a logical data base. One PCB is required for each data base. The details are shown in Figure 2-8.



MASK WRITTEN IN PL/I FOR THE OPTIMIZING COMPILER

```
DECLARE 1  PCBNAME BASED (PCB_PTR),
             2 DBD_NAME CHAR(8),
             2 SEG_LEVEL CHAR(2),
             2 STATUS_CODE CHAR(2),
             2 PROC_OPTIONS CHAR(4),
             2 RESERVE_DLI FIXED BIN(31,0),
             2 SEG_NAME_FB CHAR(8),
             2 LENGTH_FB_KEY FIXED BIN(31,0),
             2 NUMB_SENS_SEGS FIXED BIN(31,0),
             2 KEY_FB_AREA CHAR(17);
DECLARE    KEY_FB_AREA_PTR POINTER;
DECLARE    NAME_KEY CHAR(12) BASED (KEY_FB_AREA_PTR);
DECLARE 1  NAME_ADDR_KEY BASED (KEY_FB_AREA_PTR);
             2 NAME_KEY2 CHAR(12),
             2 ADDR_KEY CHAR(2);
DECLARE 1  NAME_PAYROLL_KEY BASED(KEY_FB_AREA_PTR),
             2 NAME_KEY3  CHAR(12),
             2 PAYROLL_KEY CHAR(5);
KEY_FB_AREA_PTR=ADDR(KEY_FB_AREA);
```

PCB_PTR is in the parameter list for the PL/I PROCEDURE.

Bytes and function as above

Figure 2-8. Application Program Data Base-PCB Mask

The data base PCB provides specific areas used by DL/I to advise the application program of the results of its calls. At execution time, all PCB entries are controlled by DL/I. Access to the PCB entries by the application program are for read-only purposes.

The following items comprise a PCB for a logical data structure from a data base.

1. Name of the PCB. This is the name of the area which refers to the entire structure of PCB fields and is used in program statements. This name is not a field in the PCB. It is the 01 level name in the COBOL mask in Figure 2-8.

2. Name of Data Base. This is the first field in the PCB and provides the DBD name from the library of Data Base Descriptions associated with a particular data base. It contains character data and is eight bytes long.

3. Segment-Hierarchy-Level Indicator. Data Language/I (DL/I) loads this area with the level number of the last segment encountered which satisfied a level of the call. When a retrieve is successfully completed, the level number of the retrieved segment is placed here. If the retrieve is unsuccessful, the level number returned is that of the last segment that satisfied the search criteria along the path to the desired segment. This field contains character data; it is two bytes long and is a right-justified numeric.

4. DL/I Status Code. A status code that indicates the results of a DL/I call is placed in this field and remains here until another DL/I call uses this PCB. This field contains two bytes of character data. When a successful call is executed, this field is returned blank or with an informative status indication. DL/I status codes are summarized for quick reference in Appendix A, and described in detail in Appendix B.

5. DL/I Processing Options. This area contains a character code which tells DL/I the "processing intent" of the program against this data base, -- the kinds of calls that can be used by the program for processing data in this data base. This field is four bytes long. It is left-justified. It does not change from call to call. It gives the value coded in the PCB PROCOPT parameter.

   Possible values for the processing options are:

   G - Get function
   I - Insert function
   R - Replace function
   D - Delete function
   A - All, includes the above four functions
   P - Required if D command code is to be used on get type or insert function calls
   E - Exclusive use of the data base or segment; to be used in conjunction with G, I, D, R, A, and L
   L - Load function for data base loading (except HIDAM)
   S - Segments in ascending sequence only; to be used in conjunction with G, I, D, R, A, and L
   GS- Get segments in ascending sequence only (HSAM only)
   LS- Segments loaded in ascending sequence only (HIDAM, HDAM); required for HIDAM

Note: The L or LS processing options cannot be used in a single PCB with a processing option of G, I, R, D, A or GS. GSAM supports only G, L, GS, and LS, where S implies large-scale sequential accessing requiring multi-buffering.

6. Reserved Area for DL/I. DL/I uses this area for its own internal linkage related to an application program. This field is one fullword (4 bytes)

7. Segment-Name-Feedback Area. DL/I fills this area with the name of the last segment encountered which satisfied a level of the call. When a retrieve is successful, the name of the retrieved segment is placed here. If a retrieve is unsuccessful, the name returned is that of the last segment, along the path to the desired segment, that satisfied the search criteria. This field contains eight bytes of character data. This field may be useful in GN and GNP calls. If the status code is AI (data management open error), the DD name is returned into this area.

8. Length of Key-Feedback Area. This entry specifies the current active length of the key feedback area described below. This field is four bytes long.

9. Number of Sensitive Segments. This entry specifies the number of segment types in the data base to which the application program is sensitive. This would represent a count of the number of segments in the logical data structure viewed through this PCB. This field is one fullword (4 bytes).

10. Key-Feedback Area. DL/I places in this area the concatenated key of the last segment encountered which satisfied a level of the call. When a retrieve is successful, the key of the requested segment and the key field of each segment along the path to the requested segment are concatenated and placed in this area. The key fields are positioned from left to right, beginning with the root segment key and following the hierarchical path. Keys for both key-sensitive and data-sensitive segments are placed in the key feedback area. When a retrieve is unsuccessful, the keys of all segments along the path to the requested segment for which the search was successful are placed in this area. See Figure 2-9.

The application program contains a mask of the PCB. All of the actual PCBs associated with an application program are contained in a Program Specification Block (PSB) accessible only to IMS/VS. There is normally a one-to-one relationship between PSBs and application programs. A PSB and the PCBs associated with it are created by a PSB generation utility program. The result of PSB generation is to place a compiled PSB in a library called the PSB Library.

Note: A batch program PSB can contain an I/O PCB as well as data base PCBs. See the section "Using a Batch Region to Check Out Online Message Programs" later in this chapter. See also the description of the CMPAT option of the PSBGEN procedure in the IMS/VS Utilities Reference Manual.

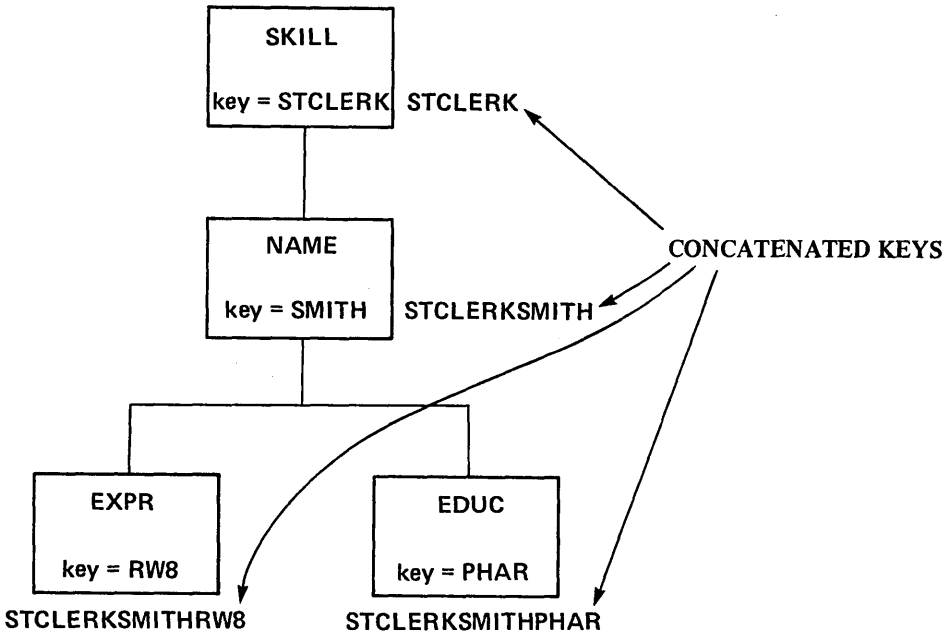


Figure 2-9. Concatenated Keys

CALLS TO DATA LANGUAGE/I (DL/I)

Actual processing of IMS/VS data bases is accomplished using a set of input/output functional call requests.

A call request is composed of a CALL statement with an argument list. The argument list describes a particular processing function and the hierarchic path to the element of data operated upon. One segment or multiple segments along the hierarchical path of segments may be operated upon with a single input/output call.

The arguments contained within any call request include the addresses of the:

- Input/output function to be performed
- PCB
- Segment input/output work area
- Identification of the data segment(s) to be operated upon

Examples

Examples of how these I/O processing calls might appear in COBOL, PL/I, or Assembler Language programs are given below, followed by a list of definitions for all of the arguments. The following sections describe the processing considerations of each argument.

For COBOL:

```
CALL 'CBLTDLI' USING [parmcount,]function, PCB-name,
I/OArea, SSA-1,...SSA-n.
```

For PL/I:

```
CALL PLITDLI (parmcount,function,PCB_ptr,I/OArea,...SSA_n);
```

For Assembler Language:

```
      |CBLTDLI|
CALL  |ASMTDLI|,([,parmcount,]function, PCB-name, I/OArea,...SSA-n) [,VL]
```

parmcount
: a binary fullword which is the address of the parameter count or argument count of the number of arguments following. IMS/VS will accept either of two types of parameter lists. One type is the explicit, in which the first argument in the list is the number of entries in the list. The other type is the implicit in which the end of the list is indicated by the last entry in the list having the leftmost bit on. PL/I must always pass an explicit list. COBOL always passes an implicit list. Either type may be passed by Assembler Language. IMS/VS dynamically determines the type of list for each call. This is done by testing the leftmost byte of the first argument. If zero, the argument is assumed to be a count of the number of entries in the list and therefore explicit. If non-zero, it is assumed to be an IMS/VS function and therefore an implicit list. This means that even though COBOL will set on the leftmost bit in the last entry, it is possible to make the list appear to IMS/VS to be an explicit list merely by providing a count as the first entry in the list. This can be handled conveniently by allowing a common call list of maximum length and adjusting the first entry, the count, to the current number of entries.

function
: is the address of a DL/I CALL input/output function. This argument is the name of a 4-character field which describes the desired I/O operation. The DL/I functions are described briefly below, and in full detail in the following section entitled "Detailed Description of DL/I Processing Functions."

name
: is the address of a data base Program Communication Block (PCB). See the section "PCB-name."

  Note: If the standard form of the OS/VS CALL macro is used, this parameter must be a register which has been loaded with the address of the PCB.

I/O Area
: is the address of an I/O work area name. See the section "I/O Work Area."

SSA-1 to SSA-n (optional)
: are the addresses of Segment Search Arguments. There can be a maximum of 1 SSA per level along the hierarchic path being accessed. See the section "Segment Search Arguments" later in this chapter.

VL
: should be designated in Assembler Language as shown if parmcount is not used. This parameter sets the flag indicating the end of a variable parameter list.

Function

The I/O functions specified in the "function" argument of the call statement are the data services of DL/I. The functions provide a full data processing repertoire of retrieving, updating, adding, and deleting data.

Listed below are all of the DL/I call functions. The righthand column indicates whether the call may be employed against data base segments, message segments, or both. Message segments may be processed in both message and batch message programs. Data base segments may be processed by any program type.

Meaning	Call Function	Type of PCB Which Can Be Used
GET UNIQUE	GUbb	Message or Data Base
GET NEXT	GNbb	Message or Data Base
GET NEXT WITHIN PARENT	GNPb	Data Base only
GET HOLD UNIQUE	GHUb	Data Base only
GET HOLD NEXT	GHNb	Data Base only
GET HOLD NEXT WITHIN PARENT	GHNP	Data Base only
INSERT	ISRT	Message or Data Base
DELETE	DLET	Data Base only
REPLACE	REPL	Data Base only
PURGE	PURG	Message only
SNAP	SNAP	Message or Data Base
CHANGE	CHNG	Message only
CHECKPOINT	CHKP	Message only
RESTART	XRST	Message only
DEQUEUE	DEQb	Message only
ROLLBACK	ROLL	Message only
LOG	LOGb	Message or Data Base
GET SCD	GSCD	Message or Data Base
STATISTICS	STAT	Data Base only

The calls listed above fall into two main categories: (1) Data Base calls comprising GET, Insert (ISRT), Delete (DLET) and Replace (REPL) calls, and (2) System Service calls, the last ten calls in the above list. These calls are discussed separately, near the end of this chapter. The manner in which each of the data processing functions (that is, Get, Insert, Delete, Replace) is executed by DL/I depends on a combination of several factors. These include control options specified at DBDGEN for the data base being called, processing options

specified at PSBGEN for the data base being called, the other arguments in the call (for example, the SSAs), and the processing dynamics (that is, the preceding calls, and the current position of DL/I in the data base). A detailed discussion of the DL/I execution logic of these functions is preceded in this chapter by a description of the remaining arguments in the call: the PCB, I/O Work Area, and SSAs.

PCB-Name

The pcb-name is the third argument in the call statement. It is the name of the block that identifies for DL/I which specific logical data structure the application program wishes to process. This means that the data the application program accesses at this point in the program execution resides in the data structure identified by this PCB-name.

See Figure 2-8 for an example of how to code PCBs in COBOL and PL/I.

I/O Work Area

The I/O work area name is the fourth argument (I/O AREA) in the call statement. The work area is an area in the application program into which DL/I puts a requested segment, or from which DL/I takes a designated segment. Only segments to which the program is data-sensitive will be placed in or taken from the I/O work area by DL/I. If a common area is used to process multiple DL/I calls, it must be as long as the longest path of segments to be processed. The work area name points to the leftmost byte of the area. Segment data is always left-justified within a work area.

When inserting or retrieving a hierarchical path of segments with one call, the I/O work area must be large enough to hold the longest concatenation of segments to be retrieved or inserted.

COBOL Example:

```
IDENTIFICATION DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  INPUT-AREA.
    02 KEY PICTURE X(6).
    02 FIELD PICTURE X(84).
```

When the data base segment is to be placed in this area, the following call statement is used, and the length of this work area is 90 bytes:

```
CALL 'CBLTDLI' USING function, PCB-name, INPUT-AREA, SSA-1.
```

PL/I Example:

In PL/I, the name used to specify the I/O area can be a major structure, an array, a fixed-character string (for example, CHAR(100)), adjustable character string (for example, CHAR(N)), a pointer to any of these, or a pointer to a minor structure. The name cannot be a minor structure or a VARYING character string.

```
DECLARE 1 INPUT_AREA  /* major structure used as I/O area*/
        2 KEY CHAR(6),
        2 FIELD CHAR(84);
.
.
.
CALL PLITDLI (parmcount,function,PCB_name,I_O_AREA,SSA1);
```

Segment Search Arguments

CONCEPT AND FUNCTION: The concept and the basic functions of SSAs (segment search arguments) are described in this section of this chapter. The fully augmented capabilities of SSAs are discussed in the "Advanced Data Base Functions" chapter of this manual. A CALL statement is considered "qualified" or "unqualified" depending on the presence or absence of SSAs within the CALL.

In concept, SSAs answer two processing needs: to relieve the application from as much processing as possible, if the programmer so desires, or to provide IMS/VS with sufficient information to satisfy the call. Hence SSAs are required for INSERT calls, optional for GET calls, and only conditionally allowed for DELETE/REPLACE calls. The rules for usage are described for each type of call function in the section "Detailed Description of DL/I Processing Functions" of this chapter.

The basic function of the SSA permits the application program to apply three different kinds of logic to a call:

- Narrow the field of search to a particular segment type, or to a particular segment occurrence
- Request that one segment or a path of segments (type or occurrence) be processed;
- Alter the traversal procedure for this call or the data base position for a subsequent call

The SSAs represent the fifth through last arguments (SSA-1 thru SSA-n) in the CALL statement. There can be 0 or 1 SSAs per level, and since IMS/VS permits a maximum of 15 levels per data base, a call can contain from 0 to 15 SSAs. They do not appear directly in the CALL statement arguments provided to DL/I, an SSA name is given which points to an area in the user's program which contains the SSA.

STRUCTURE: The SSA may consist of from one to three main elements: the segment name, and (as required) a command code(s), and one or more qualification statements. In this chapter, only SSAs with one qualification statement are considered. The three main elements of an SSA are shown in the following diagram.

SEGMENT NAME	COMMAND CODE	QUALIFICATION STATEMENT (QS)				
		Begin QS	Field Name	R.O.	Value	End QS
8 bytes	vbl.	1	8	2	1 - 255	1

SEGMENT NAME

The segment name must be eight bytes long, left-justified with trailing blanks as required. It is the segment name that pertains to a specific segment type in the hierarchical structure viewed through the associated data base PCB. The segment name must be defined as sensitive to the using application program in the PCB associated with the program. Only the names of segments to which the program is key- or data-sensitive can be specified.

COMMAND CODES

The command codes are optional. They provide functional variations to be applied to the call for that segment type. An asterisk (*) following the segment name indicates the presence of one or more command codes. A blank or a left parenthesis is the ending delimiter for command codes. Blank is used when no qualification statement exists. The designation "vbl" means variable.

QUALIFICATION STATEMENT

The presence of a qualification statement is indicated by a left parenthesis following the segment name or, if present, command codes. Each qualification statement consists of a field name, a relational operator, and a comparative value.

Begin Qualification Character

The left parenthesis, (, indicates the beginning of a qualification statement. If the SSA is unqualified, the 8-byte segment name, or, if used, the command codes should be followed by a blank.

Field Name

is the name of a field which appears in the description of the specified segment type in the DBD. The name is eight characters long, left-justified with trailing blanks as required. The named field may be either the key field or a data field within a segment. The field named is used for searching the data base, and must have been defined in the DBD.

RO = Relational Operator
is a set of two characters which express the manner in which the contents of the field, referred to by the field name, are to be tested against the comparative-value. The choice of relational operator does not affect the starting point of the search nor the order of search.

Operator			Meaning
b=	or	EQ	must be equal to
>=	or	GE	must be greater than or equal to
<=	or	LE	must be less than or equal to
b>	or	GT	must be greater than
b<	or	LT	must be less than
¬=	or	NE	must be not equal to

Note: As used above, the lowercase "b" represents a blank character. The non-alphabetic operators above can be used in the reverse combination, -- the single-character operators can be in the first position followed by a blank, as well as in the second position preceded by a blank.

Comparative value
is the value against which the contents of the field, referred to by the field name, is to be tested. The length of this field must be equal to the length of the named field in the segment of the data base, that is, it includes leading or trailing blanks (for alphameric) or zeros (usually needed for numeric fields) as required. A collating sequence, not an arithmetic, compare is performed.

End Qualification Character
The right parenthesis ') ' indicates the end of the qualification statement.

QUALIFICATION: Just as calls are "qualified" by the presence of an SSA, SSAs are categorized as either "qualified" or "unqualified," depending on the presence or absence of a qualification statement. Command codes may be included in or omitted from either qualified or unqualified SSAs.

In its simplest form, the SSA is unqualified and consists only of the name of a specific segment type as defined in the Data Base Description (DBD). In this form, the SSA provides DL/I with enough information to define the segment type desired by the call.

Example: SEGNAMEb

Qualified SSAs (optional) contain a qualification statement composed of three parts: a field name defined in the DBD, a relational operator, and a comparative value. DL/I uses the information in the qualification statement to test the value of the segment's key or data fields within the data base and thus to determine whether the segment meets the user's specifications. Using this approach, DL/I performs the data base segment searching and the program need process only those segments which precisely meet some logical criteria.

Example: SEGNAMEb(FIELDxxx>=VALUE)

The qualification statement test is terminated either when the test is satisfied by an occurence of the segment type, or when it is determined that the request cannot be satisfied.

COMMAND CODES: Both unqualified and qualified SSAs can contain one or more optional command codes which specify functional variations applicable to either the call function, the segment qualification, or the setting of parentage.

A complete discussion of command codes is presented in the "Data Base Processing: Advanced Functions" chapter in this manual. An example of the D command code is presented for introductory purposes. The D command code has a widespread, basic value in that it enables the issuance of path calls. A path call enables a hierarchical path of segments to be inserted or retrieved with one call. (A "path" was defined earlier in this chapter as the hierarchical sequence of segments, one per level, leading from a segment at one level to a particular segment at a lower level.) The meaning of the D command code is as follows:

- For retrieval calls, move the segment which satisfies this level of the call (if data-sensitive to that segment type) to the user's I/O area. This allows the retrieval of multiple segments in a hierarchical path in a single call. This type of call will subsequently be referred to as a path call. The first through the last segment retrieved are concatenated in the user's I/O area.

  Intermediate SSAs can be present without the D command code. If so, these segments are not moved to the user's I/O area. The segment named in the PCB "segment name feedback area" is the lowest-level segment retrieved, or the last level satisfied in the call in case of a not-found condition.

  Higher-level segments associated with SSAs having the D command code will have been placed in the user's I/O area even in the not-found case. The D command code is not necessary for the last SSA in the call since the segment which satisfies the last level is always moved to the user's I/O area. A processing option of "P" must be specified in the PSBGEN for any segment type for which a D command code will be used.

- For insert calls, the D command code designates the first segment type in the path to insert. The SSAs for lower-level segments in the path need not have the D command code set.

An example of SSA construction using the D command code appears at the end of this chapter.

Both command codes and qualification statements are discussed in the "Data Base Processing: Advanced Functions" chapter of this manual.

GENERAL CHARACTERISTICS OF SEGMENT SEARCH ARGUMENTS (SSAs):

- An SSA may consist of the segment name only (unqualified). It may optionally also include one or more command codes and a qualification statement.

- SSAs following the first SSA must proceed down a hierarchical path. All SSAs in the hierarchical path need not be specified, that is there may be missing levels in the path. DL/I will provide, internally, SSAs for missing levels according to the rules given later in this chapter.

Note: More specific statements which apply to the use of SSAs with a particular function such as GU, or ISRT are provided later in this chapter.

Examples of SSAs with the DL/I call I/O functions are included in the section "Examples of Data Base Processing Using DL/I I/O Functions" later in this chapter.

PL/I Example

For application programs written in PL/I, the SSA can be specified to PL/I as a major structure, an array, a fixed-character string (for example, CHAR(100)), an adjustable character string (for example, CHAR(N)), a pointer to any of these, or a pointer to a minor structure. An example follows:

```
DCL    SSA_PTR POINTER;
DCL  1 SSA,
       2  SEGNAME CHAR(8),
       2  SEGQUAL CHAR(1),
       2  SEGFLDNAME CHAR(8),
       2  SEGFLDVALUE CHAR(23),
       2  SEGENDCHAR CHAR(1);
SSA_PTR=ADDR(SSA);
       .
       .
       .
CALL PLITDLI(THREE,'GUbb',PCB-PTR,SSA-PTR);
```

DETAILED DESCRIPTION OF DL/I PROCESSING FUNCTIONS

Two terms need to be defined prior to discussing how DL/I executes the processing functions of get, insert, delete, and replace. The terms are "current position in the data base" and "segment on which position is established at that level."

The current position in the data base is the starting position which will be used by IMS/VS to satisfy any GN calls and any GNP calls.

The segment on which position is established at that level relates to retrieving or inserting a particular segment occurrence. When a segment occurrence is either retrieved or inserted, position is established on that segment and on all parent levels (if any) of that segment occurrence.

It is assumed that the reader understands the meaning of various IMS/VS terms used to describe data base structures (for example, "physical child," "logical child", and "sensitivity"). Data base structure is discussed in detail in the "Data Base Design Considerations" chapter of the IMS/VS System/Application Design Guide.

GET CALLS

The GET calls are get unique (GUbb), get next (GNbb), get next within parent (GNPb) and all forms of get hold (GHUb, GHNb, and GHNP).

Uses of Get Calls

THE GET UNIQUE CALLs (GUbb or GHUb) - DATA BASE: The get unique call is used to retrieve a segment occurrence independent of current position within the data base. The get unique call can therefore be used for random processing, or it can be used to establish a position in the data base for subsequent sequential processing. See GU rules for exceptions.

THE GET NEXT CALLs (GNbb or GHNb) - DATA BASE: The get next calls are used to retrieve a segment or a path of segments by proceeding forward from a previously established position within the data base until a segment occurrence is found at each level which satisfies the search criteria at that level. The SSAs determine the search criteria.

The basic difference between get next and get unique calls is the initial position used in attempting to satisfy the call. The get unique call will be satisfied by finding the earliest level-one (root) segment in the logical data base which satisfies that level in the call and then attempting to satisfy all lower levels with the first occurrence of that segment type under its parent. The get next call, on the other hand, proceeds forward from the current position in the data base in attempting to satisfy the current call. (An exception to this is the F command code which allows the get next call to move back to the first occurrence of this segment type under its parent.)

The execution of a get next call without SSAs returns the next data sensitive segment occurrence within the data base relative to the positioning of the data base during the previous GU, GN, GNP, ISRT, REPL, or DLET call. An uninterrupted series of get next call statements could be used to retrieve each segment occurrence from the data base, beginning with the first, and proceeding sequentially through the last for all sensitive segments. The parameters for this form of a get next call are the call-function, db-pcb-name, and I/O area.

The execution of a get next call with an unqualified SSA returns the next segment occurrence of the segment type specified in the SSA relative to the current position in the data base. An uninterrupted series of get next calls with unqualified SSAs could be used to retrieve all segment occurrences of a specified type in the data base.

A get next call following an ISRT or DLET call delivers the first sensitive segment hierarchically above or to the right of the inserted or deleted segment. That is, the position established by an ISRT call is the same as if the inserted segment had been retrieved with a get unique or get next call. The position following a delete is immediately following the deleted segment, or if the deleted segment had dependent segments then immediately following those dependent segments (because dependents of a deleted segment are also deleted).

The get next call only progresses forward from the position in the data base established in the preceding call in an attempt to satisfy the current call requirements. (An exception to this rule is the use of the F command code, which allows backing up to the first occurrence of this segment type under its parent. Also, this limitation does not apply when "Multiple Positioning" is in effect. Command codes and multiple positioning are discussed in the "Data Base Processing: Advanced Function" chapter of this manual.)

THE GET NEXT-WITHIN-PARENT CALLs (GNPb or GHNP) - DATA BASE: The GNP call is similar to the GN call except that segments which may satisfy a GNP call are limited to the lower-level dependent segments of the established parent.

Setting of Parentage

Parentage is set by means of a GNP call or a P command code under the following conditions:

- Parentage is set at the issuance of the first GNP call that follows a completely satisfied get next or get unique call. The parentage will be set at the lowest level segment that was retrieved by the preceding get next or get unique call. Parentage that is established by the first GNP call following a get next or get unique call remains constant for successive GNP calls.

- Parentage can be set at other than the lowest level segment that was returned by a get next or get unique call by using the P command code. For additional information, see the description of the P command code in the section "Command Codes" in the "Data Base Processing: Advanced Functions" chapter of this manual.

Processing within Parentage

If a series of GNP calls without SSAs is issued, the calls retrieve all segment occurrences under the segment on which parentage was established going up and down hierarchical levels and crossing boundaries in the structure beneath the parent for all sensitive segments. A "not-found" condition results when DL/I encounters the next segment occurrence that is at the same level as the parent or higher.

If a GNP with SSAs is issued, it also is restricted to occurrences of that segment type named in the SSA, and will return a not-found condition if the requested segment cannot be found within the dependents of the established parent.

Resetting of Parentage

Parentage is only conditioned for reset (actually reset by a GNP) by the issuance of a get unique or get next call. Intervening ISRT, DLET or REPL calls therefore do not affect parentage. A GNP call (qualified or unqualified) which results in a GE status code (not-found condition) does not affect parentage.

Notes:

1. If no parent has been established on the GNP following a GU or GN, a GP status is returned for the GNP call. No parent has been established if the prior GU or GN call was not satisfied and did not contain a P command code, or if the call was partially satisfied but none of the satisfied levels contained a P command code.

2. Although the ISRT call does not affect parentage, it should be noted that position following an ISRT call is established immediately following the inserted segment. For this reason, if the inserted segment is at a level equal or closer to the root than the parent, then succeeding GNP calls following the ISRT cannot be satisfied.

Rules for Get Calls

The following rules apply to GET calls:

1. The call may or may not have SSAs.

2. For any level, the SSA may or may not include command codes or a qualification statement.

3. If an SSA without a qualification statement (unqualified SSA) is specified, any occurrence of that segment type under its parent will satisfy the call.

4. A get unique call with an unqualified SSA at the root level will attempt to satisfy the call by starting to scan from the beginning of the data base.

5. If the application program does not specify SSAs for one or more of the levels above the lowest level specified, then DL/I will process the call with the following implied SSAs used to fill the missing levels.

   a. GET NEXT or GET NEXT WITHIN PARENT CALLS -

      unqualified SSAs are always implied for missing levels.

   b. GET UNIQUE CALLS -

      (1) If the prior call established position on an implied segment type, an SSA qualified with current position is assumed. If a parent level qualified SSA is provided for other than the parent's current position, an unqualified SSA is assumed by DL/I for all missing levels below that parent.

      (2) If the prior call did not establish position on any implied segment type, then DL/I assumes an unqualified SSA at that level.

THE GET HOLD CALLS (GHUb, GHNb, GHNP) - DATA BASE: To change the contents of a segment in a data base, through a DLET or REPL call, the program must first obtain the segment. It then changes its contents and requests DL/I to place the segment back in the data base.

When a segment is to be changed, this must be indicated to DL/I at the time the segment is obtained. This indication is given by using the get hold calls. These function codes are like the standard get function, except the letter "H" immediately follows the letter "G" in the code; that is, the hold form of the standard get next within parent (GNPb) is GHNP. There are three get hold calls:: GHUb, GHNb, and GHNP. They function like the standard get calls. They also indicate to DL/I that the segment can be changed or deleted. (When a hold call is issued in a batch message, or message processing program, the segment retrieved is enqueued single update. No enqueue is issued in a batch mode.)

After DL/I has returned the requested segment to the user, one or more fields, but not the key field, in the segment can be changed.

After the user has changed the segment contents, he is ready to call DL/I to return the segment to the data base. If, after issuing a get hold call, the program determines that it is not necessary to change the retrieved segment, the program may proceed with other processing, and the enqueue will be freed when positioning changes because of a subsequent call to the PCB.

Examples of get calls appear at the end of this chapter.

INSERT CALLS

The DL/I insert call is used for two distinct purposes: It is used to initially load the segments for creation of a data base. It is also used in HISAM, HDAM, and HIDAM organizations to add new occurrences of an existing segment type into an established data base. The processing options field in the PCB indicates whether the data base is being added to or loaded. The format of the insert call is identical for either use.

When loading or inserting (except in a path insert), the last SSA specifies the segment being inserted. To insert a path of segments, the D command code is set for the highest level segment in the path; this SSA must be unqualified.

Lower-level unqualified SSAs designate the other segment types in the path. The segment corresponding to the SSA with the D command code must be the first segment in the I/O area, with the other segments in the path concatenated behind it.

Up to the level to be inserted, the SSA evaluation and positioning for an insert call is exactly the same as for a GU call. For the level to be inserted, the value of the sequence field in the segment in the user I/O area is used to establish the insert position.

If there is no sequence or key field for the segment, or if a non-unique sequence field was defined, then the "first", "last", or "here" insert rules are used. If the "here" insert rule is used, the F or L command code will also be used if specified. See the following chapter for the meaning of these command codes.

Rules for Insert Calls

These rules apply to ISRT calls:

1. The call must have at least one unqualified SSA.
2. If a D command code is not used (that is, it is not a path call), the lowest-level SSA specifies the segment being inserted and this SSA must be unqualified.
3. If a D command code is specified in an SSA, that SSA and all lower level SSAs must be unqualified.
4. Since the positioning for SSAs above the level of the segment(s) to be inserted is identical to GU calls, rules 3, 4 and 5 under GET call apply for inserting SSAs above the level to be inserted just as they apply to GU calls.

Using Insert Calls for Updating

The ISRT call can be used with other DL/I segment processing calls in a message processing program. In this environment, the ISRT call is used to place new occurrences of existing segment types into an established HISAM, HDAM, and HIDAM data base. Of course, the ISRT call can also be used for updating by batch and batch message processing programs.

When inserting segments into an existing data base involving logical relationships, a logical child segment cannot be inserted into a path with its parents and/or dependent segments. A logical child or logical child/logical parent combination cannot be inserted in a path call.

When inserting a segment into an existing data base, qualified SSAs for higher levels are normally provided to establish the position of the segment to insert.

Using Insert Calls for Loading a Data Base

When inserting to a hierarchical sequential (HSAM) data base, ISRT means to load an output data base. The PCB processing option L is used. Option A is invalid for HSAM. Inserts to an established HSAM data base cannot be made without reprocessing the whole data base or by adding to the end, and must be in sequence.

In a message processing program, it is not possible to perform a HDAM, HISAM or HIDAM load. The program to load a HDAM, HISAM and HIDAM data base must be a DL/I batch program.

When loading a data base, higher level qualified SSAs for the parents of the segment being loaded are not necessary, since there is no position to establish. They may, however, be provided and, if provided, the comparative-value in the qualification statement must equal the key field values of the parents of the segment being loaded.

For HISAM and HIDAM organizations, IMS/VS uses the high-values key (X'FF..FF'). A return code of LB will be given on any attempt to insert this key.

Examples of ISRT calls appear at the end of this chapter.

DELETE AND REPLACE CALLS

Use of Delete and Replace Calls

THE DELETE CALL (DLET) - DATA BASE: To delete the occurrence of a segment from a data base, the segment must first be obtained by issuing a GHbb call through DL/I. Once the segment has been acquired, the DLET call may be issued.

If DL/I calls which use the same PCB intervene between the GHbb call and the DLET call, the DLET call is rejected. Quite often a program may want to process a segment prior to deleting it. This is permitted as long as the processing does not involve a DL/I call which refers to the data base PCB used to get the segment. However, other PCBs may be referred to between the GHbb and DLET calls.

DL/I is advised that a segment is to be deleted when the user issues a call that has the function DLET. When the DLET call is executed, the specified segment occurrence may not be physically deleted, but simply flagged as being deleted. The deletion of a parent, in effect,

deletes all the segment occurrences beneath that parent. If the segment being deleted is a root segment, all dependent segments under that root are deleted. All subordinate data set groups must be available for processing prior to the delete call being issued. If they are not, an AI status code is returned. All physical dependents of the deleted segment are deleted, regardless of the logical data structure used by the program. Furthermore, deletion may carry across logical relationships.

If the DLET call follows a GHbb call which retrieved a path of segments, and there is no SSA in the call, then the highest level segment obtained on the prior call and all its children are deleted. One SSA is allowed on DLET calls following path GHbb calls. It must be for one of the segment types retrieved on the prior hold call. The SSA specifies the highest-level segment to be deleted. This segment and its children will be deleted, but higher level segments obtained on the prior GHbb call will not be deleted.

The segment to be deleted must occupy the area referred to by the I/O work area in the DLET call. If the previous GHbb call returned multiple segments, the segment(s) to be deleted should occupy the same relative position in the I/O area as on the retrieve call.

For a program which processes hierarchical sequential (HSAM) data bases where each record is rewritten on a new data base, the DLET call has no meaning and is rejected as an invalid call function. If a segment occurrence is to be deleted, it is simply not written to the output data base.

THE REPLACE CALL (REPL) - DATA BASE: The purpose of the REPL call is to allow a segment, or path of segments, that has been retrieved through a GHbb call and modified through program processing, to be replaced in the data base. The segment or segments to be modified and replaced must first be obtained by a GHbb call. No intervening calls involving the associated data base PCB may be made between the GHbb and the REPL call. If this rule is violated, the REPL call is rejected.

In the modification of a segment to be replaced in the data base, care must be taken not to modify the segment key field. If modification of the key field is attempted, the REPL call is rejected. All data, including fields which are indexed through secondary indexes, but with the exception of the key(s) of the logical child or the concatenated key of the logical parent, may be changed. This subject is treated in greater detail in the IMS/VS System/Application Design Guide.

The segment or segments to be replaced must occupy the area referred to by I/O work area in the REPL call. The segment or segments in the DL/I buffer area is overlaid with the I/O work area in the REPL call.

When a GHbb path call is made, DL/I "remembers" the format of the segments concatenated in the user I/O work area. If a REPL call without SSAs follows a path call, all segments in the I/O area for which the user has replace sensitivity will replace the corresponding segments in the data base. To preclude having segments replaced in the path, SSAs with N command codes can be used to prevent DL/I from replacing corresponding segments in the data base.

For a program which processes hierarchical sequential (HSAM) data bases where each record is rewritten on a new data base, the REPL call has no meaning. If a segment occurrence is to be replaced, it is simply placed in the output data base with an ISRT call.

When a held segment is updated with a REPL or DLET call, the enqueue level is raised from single update to exclusive (batch message or message processing only).

Rules for Delete and Replace Calls

The following rules apply to both the DLET call and the REPL call:

1. The segment, or path of segments, to be deleted or replaced must have been obtained with a get hold call (GHUb, GHNb, or GHNP).

2. Consecutive replace calls to the same segment are allowed, but no intervening calls to the same data base PCB are allowed between the get hold call and the DLET or REPL call.

3. The sequence or key field of the segment, or path of segments, to be deleted or replaced may not be changed in the user's I/O work area. For a logical-child segment, three field types must not be changed:

   a. The physical-twin sequence field

   b. The logical-twin sequence field (the sequence field specified for the virtually-paired logical child, if any)

   c. The portion of the logical child which is the concatenated key of the logical parent.

4. Segment search arguments (SSAs) are only applicable to DLET or REPL calls when the prior get hold call retrieved a path of segments. When this situation applies, one unqualified SSA for one of the segments in the path is allowed for DLET calls, and multiple unqualified SSAs for segments in the path are allowed for REPL calls.

Delete Requests Issued against a Logical Data Base

Delete calls differ from other DL/I calls in that their effects are generally propagated down to the dependent segments of a deleted segment.

For segments participating in logical relationships, DL/I provides various options for propagation. These options are specified in the DBD generation and allow for various degrees of selective deletion of segment types that may be reached from alternative paths. Options are also provided to allow the delete request to be accepted or rejected, depending on the status of segments participating in logical relationships.

The data base administrator should be specifically involved in setting up the delete processing capabilities for programs working with logical data bases. This is of particular importance when multiple logical relationships exist, or when a segment can be reached from its dependent segment types through logical relationships.

Implications of delete propagation and delete rule options are discussed further in the IMS/VS System/Application Design Guide.

Examples of DLET and REPL calls appear at the end of this chapter.

FORMAT OF SEGMENTS IN THE I/O AREA

Fixed-Length Segments

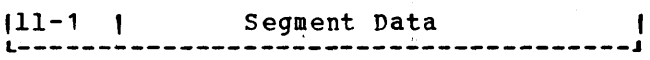
Within the structure of a data base, the IMS/VS segment format consists of a prefix portion and a data portion. The prefix serves a

structural purpose which is transparent to the application program. Within the prefix are all the necessary identity, usage, and pointer data required by IMS/VS for control and traversal purposes. (A detailed discussion can be found in the IMS/VS System/Application Design Guide.) It is only the data portion of the segment which an application program obtains from or returns to a data base. The length of fixed-length segments are defined at DBD generation. This length refers to the length of the data portion.

Variable-Length Segments

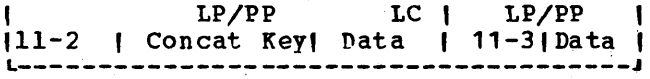
The format of variable-length segments, both in the data base and in the application I/O area, differs from the format of fixed-length segments in only one respect: The first two bytes of the data portion contain the binary value of the length of the data portion of the segment (including the 2-byte length count). Since this 2-byte field describes the segment length as the user sees it, the minimum valid value in this field is two. Specification of a value less than 2 at execution time will be ignored, and a default value of two will be assumed. The following illustrations show the format of variable-length segments in the application I/O area:

Variable-Length Physical Segment



ll-1 = segment length

Variable-Length Concatenated Logical Segment



LC = logical child
LP = logical parent
PP = physical parent
ll-2 = segment length for LC
ll-3 = segment length for LP/PP

Segment retrieval, including path calls, follows normal retrieval rules. After the segment has been accessed, replacement of existing data can occur with a REPL call. If the segment length has not changed, a one-for-one replacement takes place. If the length of a segment is increased or decreased during a replace operation, the new segment length must be placed in the segment-size field by the user. For an insert operation, the user places the segment size in the appropriate field, followed by the corresponding segment data, and the ISRT call is issued.

Since the segment-size field is actually a part of the segment, all starting positions for fields are in reference to the first position of the segment-size field in a segment, not the start of the user data. Except for the required 2-byte binary field describing the segment length, the content and data alignment, as well as the existence of any defined data fields, are the responsibility of the user. Segment-sequence fields, if defined, must always exist in their DBD-defined position and cannot be altered by REPL calls. The length field of a segment can be referenced in an SSA by defining a 2-byte hexadecimal field with a starting position of one.

TERMINATING THE APPLICATION PROGRAM

At the completion of processing of any application program (message or batch), control must be returned to the IMS/VS control facility. The RETURN or GOBACK statement must be given in every program as follows:

ANS COBOL	PL/I	ASSEMBLER
GOBACK.	RETURN;	RETURN (14,12),RC=0

The RETURN or GOBACK statement in a batch program returns control to the IMS/VS control facility. However, the IMS/VS control facility subsequently returns control to the operating system job terminator after DL/I resources are released.

The RETURN or GOBACK statement in a message processing program causes control to return to the IMS/VS control facility in a message processing region. The IMS/VS control facility records accounting information and passes to the IMS/VS scheduling facility a request for rescheduling in the message processing region.

If the program is terminating normally, the RETURN statement from an application program written in Assembler Language must have the contents of Register 15 equal to zero.

Since IMS/VS links to application programs, the return to IMS/VS causes storage occupied by the application program to be released. If non-IMS/VS initiated I/O operations are outstanding against open DCBs, various ABENDS in IOS and POST may occur. Final termination of the job step may also produce abends in CLOSE.

ANS COBOL Batch-Program Structure

Figure 2-10 outlines the fundamental parts of a batch program. Each item should be considered when designing a batch program. This program retrieves data from a detail file to update a master data base. Neither the detail nor the master is a teleprocessing data base. A similar structure must be used to create a teleprocessing or batch processing data base in a batch region.

```
REF
NO.
      ENVIRONMENT DIVISION.
          .
          .
      DATA DIVISION.
      WORKING-STORAGE SECTION.
          77 FUNC-DB-IN1  PICTURE XXXX VALUE 'GUbb'.
 1        77 FUNC-DB-IN2  PICTURE XXXX VALUE 'GHUb'.
          77 FUNC-DB-OUT  PICTURE XXXX VALUE 'REPL'.
          77 FUNC-DB-NEXT PICTURE XXXX VALUE 'GHNb'.
          77 CT PICTURE S9(5) COMPUTATIONAL VALUE +4.
          .
 2        01  SSA-NAME.
 3        01  MAST-SEG-IO-AREA.
          01  DET-SEG-IN-AREA.
      LINKAGE SECTION.
 4        01  DB-PCB-MAST.
          01  DB-PCB-DETAIL.

      PROCEDURE DIVISION.

 5        ENTRY 'DLITCBL' USING DB-PCB-MAST,
              DB-PCB-DETAIL.
          .
 6        CALL 'CBLTDLI' USING FUNC-DB-IN1, DB-PCB-DETAIL,
              DET-SEG-IN-AREA, SSA-NAME.
          .
 7        CALL 'CBLTDLI' USING CT, FUNC-DB-IN2,
              DB-PCB-MAST, MAST-SEG-IO-AREA, SSA-NAME.

 8        CALL 'CBLTDLI' USING FUNC-DB-NEXT, DB-PCB-MAST,
              MAST-SEG-IO-AREA.
          .
 9        CALL 'CBLTDLI' USING FUNC-DB-OUT, DB-PCB-MAST
              MAST-SEG-IO-AREA.
          .
10        GOBACK.

11        COBOL-LANGUAGE INTERFACE
```

Figure 2-10. ANS COBOL Batch-Program Structure

The following explanation relates to the reference numbers along the left side of Figure 2-10.

1. A 77 level or 01 level working storage entry defines each of the CALL functions used by the batch program. Each picture clause is defined as four alphameric characters and has a value assigned for each function (for example, GUbb).

2. An 01 level working storage entry defines each segment search argument used by an application program. An example of an SSA definition, with lowercase "b" representing blank, is:

```
01 SSA-NAME.

   02 SEG-NAME  PICTURE X(8) VALUE 'ROOTbbbb'.
   02 SEG-QUAL  PICTURE X  VALUE '('.
   02 SEG-KEYNAME  PICTURE X(8)  VALUE 'KEYbbbbb'.
   02 SEG-OPERATOR  PICTURE XX  VALUE 'b='.
   02 SEG-KEY-VALUE  PICTURE X(6)  VALUE 'vvvvvv'.
   02 SEG-END-CHAR  PICTURE X  VALUE ')'.
```

   When this COBOL syntax is decoded, it will be in a data string as follows:

```
ROOTbbbb(KEYbbbbbb=vvvvvv)
```

3. An 01 level working storage entry defines the program segment I/O work area.

4. An 01 level linkage section entry describes the DB-PCB entry for every input or output data base. No TP PCBs can be included. It is through this linkage that a COBOL program may access the status codes after a DL/I call.

5. This is the standard entry point in the procedure division of a batch program. After IMS/VS control has loaded and completed the PSB for the program in the batch region, IMS/VS gives control to this entry point. The PSB contains all the PCBs used by the program. The USING statement at the entry point to the batch program must contain the same number of names in the same sequence as there are PCBs in the PSB.

6., 7.
   These are typical CALLs used to retrieve data from a data base using a qualified search argument.

   Item 7 also shows the use of another argument (parm-count) in the call made from COBOL to DL/I. This additional explicit argument is a binary counter (fullword) of the number of remaining arguments in the current DL/I call. This allows the user to set up the parameters of a call in the working storage section of his data division and to truncate or expand this call through the use of the binary counter.

8. This is a typical call used to retrieve data from a data base using an unqualified search. This CALL is also a HOLD call for a subsequent delete or replace.

   ```
   CALL 'CBLTDLI' USING call-function, db-pcbname, work-area.
   ```

9. This is used to update data from a batch program in a data base.

10. GOBACK causes the batch program to return control to IMS/VS control facilities.

11. A language interface module (DFSLI000) is provided by IMS/VS. This module must be link-edited to the batch program after compilation and provides a common interface to IMS/VS and DL/I.

    The language interface function of IMS/VS is reenterable, and compatible with that of IMS/360 Version 2. In order to take advantage of the reenterable capability of the IMS/VS language interface, application modules must be re-link-edited, replacing the IMS/360 Version 2 language interface with the IMS/VS language interface. The IMS/360 Version 1 language interface is not available in IMS/VS. Existing IMS/360 Version 2 programs can be executed without re-link-editing them with the IMS/VS language interface.

PL/I Optimizing Compiler Batch-Program Structure

Figure 2-11 outlines the fundamental parts of a PL/I optimizing compiler batch program. Each item should be considered when designing a batch program. This program retrieves data from a detail file to update a master data base. Both the detail and the master PCBs represent data bases.

```
REF
NO.
        /* ------------------------------------------------ */
        /*                   ENTRY POINT                    */
        /* ------------------------------------------------ */
 1          DLITPLI: PROC(MAST_PTR,DETAIL_PTR) OPTIONS (MAIN);

 2      DCL  FUNC_GU CHAR(4) STATIC INIT ('GU'),
        DCL  FUNC_GHU CHAR(4) STATIC INIT ('GHU'),
        DCL  FUNC_REPL CHAR(4) STATIC INIT ('REPL'),
        DCL  FUNC-GHN CHAR(4) STATIC INIT ('GHN'),

 3      DCL  SSA_NAME

 4      DCL  DET_SEG_IO_AREA...;

 5      DCL  1 DB_PCB_MAST    BASED(MAST_PTR),...;
        DCL  1 DB_PCB_DETAIL  BASED(DETAIL_PTR),...;

 6      DCL  THREE FIXED BINARY(31) STATIC INITIAL(3);
        DCL  FOUR FIXED BINARY(31) STATIC INITIAL(4);

 7      CALL PLITDLI(FOUR,FUNC_GU,DETAIL_PTR,DET_SEG_IO_AREA,
             SSA_NAME);

 8      CALL PLITDLI(FOUR,FUNC_GHU,MAST_PTR,MAST_SEG_IO_AREA,
             SSA_NAME);

 9      CALL PLITDLI(THREE,FUNC_GHN,MAST_PTR,MAST_SEG_IO_AREA);

10      CALL PLITDLI(THREE,FUNC_REPL,MAST_PTR,MAST_SEG_IO_AREA);

11      END DLITPLI;


12      PL/I LANGUAGE INTERFACE
```

Figure 2-11. PL/I Optimizing Compiler Batch-Program Structure

The following explanation relates to the reference numbers along the left side of Figure 2-11:

1. This is the main standard entry point to a PL/I batch program. After the IMS/VS control program has loaded and completed the PSB for the program in the batch region, it gives control to this entry point. The PSB contains all the PCBs used by the program. The entry-point statement of the batch program must contain the same number of names in the same sequence as there are PCBs in the PSB.

2. These declarations define the call functions used by the PL/I batch program. Each character string is defined as four alphameric characters, with a value assigned for each function (for example, GU). Other constants can be defined in the same manner.

3. This declaration defines storage for SSAs. In the following example, the SSA is declared as a structure; other methods can be used (see the example under the section "General Characteristics of Segment Search Arguments" in Chapter 3 of this manual).

Example: (lower case "b" represents blank)

```
DCL 1  SSA_NAME STATIC,
       2   SEG_NAME CHAR(8)         INIT('ROOT'),
       2   SEG_QUAL CHAR(1)         INIT('('),
       2   SEG_KEY_NAME CHAR(8)     INIT('KEY'),
       2   SEG_OPERATOR CHAR(2)     INIT('b='),
       2   SEG_KEY_VALUE CHAR(6)    INIT('vvvvvv'),
       2   SEG_END_CHAR CHAR(1)     INIT(')');
```

When the above PL/I syntax is decoded, it will be in a data string as follows:

```
ROOTbbbb(KEYbbbbbb=vvvvvv)
```

4. The I/O area is most efficiently passed to DL/I as a fixed-length character string or through a pointer variable; other methods can be used, however, (see the PL/I example under the section "I/O Work Area" earlier in this chapter). An example follows.

```
DCL MAST_SEG_IO_AREA CHAR(256);
```

5. A major structure declaration describes the DB-PCB entry for every input or output data base. It is through this description that a PL/I program may access the status codes after a DL/I call.

6. This is a descriptive statement used to identify a binary number (fullword) that represents the parameter count of a call to DL/I. The parameter count value equals the remaining number of arguments following the parameter count set off by commas.

7., 8.
These are typical calls used to retrieve data from a data base using a qualified search argument.

9. This is a typical call used to retrieve data from a data base using an unqualified search argument. This call is also a HOLD call for a subsequent delete or replace.

10. This call is used to replace data from a DL/I batch program on to a data base.

11. This END statement causes the batch program to return control to the IMS/VS control facilities. Another statement that causes the batch program to return control to the IMS/VS control facilities is the RETURN statement. The RETURN statement may or may not immediately precede the END statement.

12. A language interface (DFSLI000) is provided by IMS/VS. This module must be link-edited to the batch program and provides a common interface to IMS/VS and DL/I.

    The language interface function of IMS/VS is reenterable and compatible with that of IMS/360 Version 2. To take advantage of the reenterable capability of the IMS/VS language interface, application modules must be re-link-edited, replacing the IMS/360 Version 2 language interface with the IMS/VS language interface. The IMS/360 Version 1 language interface is not available in IMS/VS. Existing IMS/360 Version 2 programs can be executed without re-linking them with the IMS/VS language interface.

Assembler Language Batch-Program Structure

The entry point to an Assembler Language program which utilizes DL/I may have any desired name. However, Register 1, upon entry to the application program, contains the address of a variable-length fullword parameter list. Each word in this list contains a PCB control block base address which must be saved by the application program. The high-order byte of the last word in the parameter list has the 0 bit set to a value of one to indicate the end of the list. The PCB addresses from this list are subsequently used by the application program when executing DL/I calls.

All DL/I calls from an Assembler Language program should be executed with the CALL macro instruction. Register 1 must be constructed prior to execution of the CALL statement to point to the variable-length fullword parameter list. This may be done through operands of the CALL macro instruction. The parameters in this list are addresses of:

- The input/output function
- The PCB address associated with data base
- Input/output work area
- Zero or more segment search argument identifiers

The entry point for the CALL macro instruction is CBLTDLI. The IMS/VS-supplied language interface module (DFSLI000) must be link-edited with the compiled Assembler Language program.

Application programs used in the batch DL/I environment can use both DL/I for data base processing and standard OS/VS data management for non-data base input/output operation.

STATUS CODES FOR DL/I I/O CALLS

At the completion of a DL/I call, a status code that indicates the results of the call that was made is presented to the application program in the PCB status-code field.

The user should follow each call in his program with statements which examine the status codes returned in the PCB to determine if the requested action was completed properly.

The IMS/VS installation should normally provide application programs with a standardized status code checking procedure to be applied after each call.

Appendix A provides a quick reference of DL/I status codes. The status codes are described in full detail in Appendix B.

STATUS CODES FOR SUCCESSFUL COMPLETION OF GET CALLS

If the GET call was successfully completed, the 2-byte status code is blank or GA, or GK; otherwise, another status code applies.

The GA status code is a warning indication. When a GN or GNP call without SSAs is issued, DL/I may return this status code to indicate the crossing of hierarchical boundaries. This status code indicates that DL/I has passed from one segment in the logical data structure at level X to another segment in the logical data structure at level Y, where Y is less than X. In other words, it has proceeded upward in the hierarchy toward the root segment. This code is not returned to the using application program when a GU, GN with SSAs, or GNP with SSAs is issued, because the user is explicitly asking, through the presence of the SSAs, to traverse a known path in the data base. The GA status code is thus a warning to the user of the GN or GNP call that DL/I has taken him implicitly from a segment at one level of the hierarchy to a segment at another, higher, level of the hierarchy.

Similarly, DL/I returns the GK status code to GN or GNP calls without SSAs to indicate the crossing of a lateral boundary from one segment type to another. GK (like GA) is not returned to the using application program if a GU, GN or GNP with SSAs is issued. The GK status code is a warning that DL/I has proceeded implicitly from the last segment occurence of one segment type to the first segment occurence of the next segment type, at the same level in the hierarchy.

STATUS CODES FOR VALID EXCEPTIONAL CONDITIONS IN THE DATA BASE

When the GET call is not completed due to exceptional but valid conditions in the data base, either the GB or the GE status code is returned. GB indicates that DL/I has encountered the end of the data base. GE means that DL/I has not found the segment occurence specified in the GET call.

POSITION IN THE DATA BASE

PCB AND POSITION FOR "NOT-FOUND" CALLS

The terms "current position in the data base" and "segment in which position is established at that level" were defined earlier in this chapter under "Detailed Description of DL/I Processing Functions." These terms are particularly relevant and require further clarification when discussing the PCB and position in a data base for "not-found" calls.

The segment on which position is established at that level is relevant for GU and ISRT calls which do not specify all levels in the call and also for GU, GN and ISRT when the U and V command codes are used. The position of the parent is relevant when F and L command codes are used, and the position of the parent is relevant if the P command code was specified for the parent and a GNP call follows. The current position in the data base is the same as the segment on which position is established at the lowest level in the call when the call is fully satisfied. They may differ, however, when the call is not fully satisfied.

Earlier in this chapter (see the section "PCB Masks"), it was stated that the segment level, segment name, and key feedback areas of the PCB always reflect the last segment and keys for higher levels which satisfied a level of the call. If the call is completely satisfied, this shows the lowest level segment requested on the call. If the call

is not completely satisfied, it shows the last segment which satisfied a level of the call. The PCB therefore reflects the lowest level segment on which position is established. Position is also established for all parent levels, if any exist for that segment.

No position is established on the segment that could not satisfy the request and on lower level segments.

If level-one (root) SSA could not be satisfied, the segment name is cleared to blank, and the level and key-feedback length are set to zero. The key-feedback area is never cleared. Segment-key field values are concatenated in this area as the segment levels are satisfied. The segment name in the PCB or the key-feedback length field may be used to determine the length of the relevant data in the key-feedback area. Contents of the feedback area beyond the length value is indeterminate as the feedback area is never returned to zero from previous calls.

In considering current position in the data base, it must be remembered that DL/I must first establish a starting position to be used in satisfying the call. This starting position is the current position in the data base for GN calls and is a unique position normally established by the root SSA for GU and ISRT calls. DL/I will then scan segment occurrences in a forward direction based on the logical hierarchical data base structure. For "found" calls without SSAs, the position in the data base is clearly the position of the lowest level segment retrieved in the call. For "not-found" calls, the current position in the data base is immediately preceding the earliest segment encountered in attempting to satisfy this call which could be used by DL/I to determine that the call could not be satisfied. The segment which will be returned if an unqualified get next call is now issued is the segment which indicated the not-found condition, above.

This not-found position, then, is dependent on the SSAs used in the call. If all SSAs are qualified with an equal operator (=) and a key field, then the not-found position is fairly easily determined. If, however, some of the SSAs are qualified on data fields or use greater-than operands (>), the not-found position may be further in the data base than when equal operands (=) are used on key fields. If the application program is depending on the not-found position, then it must realize that this position is based on the CALL function (particularly GNP functions) and on the SSAs used in the CALL.

If an SSA is qualified on a sequence field and the sequence field is defined as non-unique (more than one segment occurence of this type may have the same key value), the retrieval search for a segment to meet an equal qualification will stop when the first occurence with the qualified value is found. If lower level SSAs for this call cannot then be satisfied, the next occurence with the same non-unique sequence field value will be retrieved and an attempt will be made to satisfy the lower level SSAs. When attempting to satisfy a level which is qualified on a non-unique sequence field and a segment with a higher key is encountered, the search will stop. That segment with the higher key will be the not-found position and the next GN call will start from that point. In the above situation, if the end of a segment-twin chain is reached before a higher-key value is found, the position is assumed to be at the next segment hierarchically above or to the right. Therefore, all dependents of the segment type with the non-unique sequence field have been passed and cannot be retrieved with any kind of GN call (*F command code excepted) unless an intervening GU or ISRT call is issued for repositioning.

To reestablish known position in a data base after an unsuccessful GN call which was qualified on a data field or a non-unique sequence field, the application program can issue a fully qualified GU call.

The following clarifications apply to the current position in the data base for special situations:

- If no current position exists in the data base, the assumed current position is the start of the data base. This applies to the first call issued by either a batch or online program.
- If the end of the data base is encountered, the assumed current position to be used by the next call is the start of the data base.

ACCESS TO MULTIPLE DATA BASES

An application program can access data segments in more than one physical or logical data base. The program may also access more than one logical data structure in the same data base. The use of multiple data structures means that the PSB loaded from the PSB library at initiation of an application program has multiple data base PCB blocks within it. Upon entry to the application program, each PCB name is provided to the application program (see Figure 2-12).

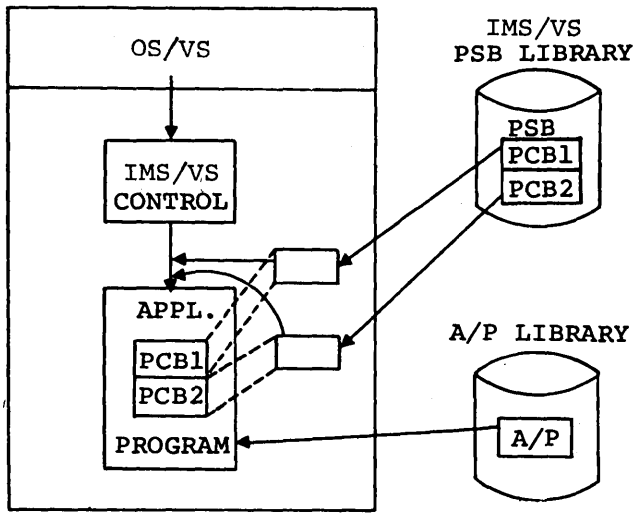


Figure 2-12. Accessing Multiple PCBs in an IMS/VS Batch Environment

The use of more than one data base PCB requires the ENTRY or PROCEDURE statement in the application program to contain multiple PCB names. The sequence of PCB names in the ENTRY or PROCEDURE statement must be the same as their sequence in the PSB associated with the application program.

Access to multiple logical data structures in the same data base (via the specificaiton of multiple PCBs against that data base) enables application programs to:

- Achieve parallel processing
- Simplify replace and delete call sequences when the action could only be determined after other segments have been examined
- Be used by the data base administrator for a proper choice of internal DL/I procedures in using OS/VS data management routines

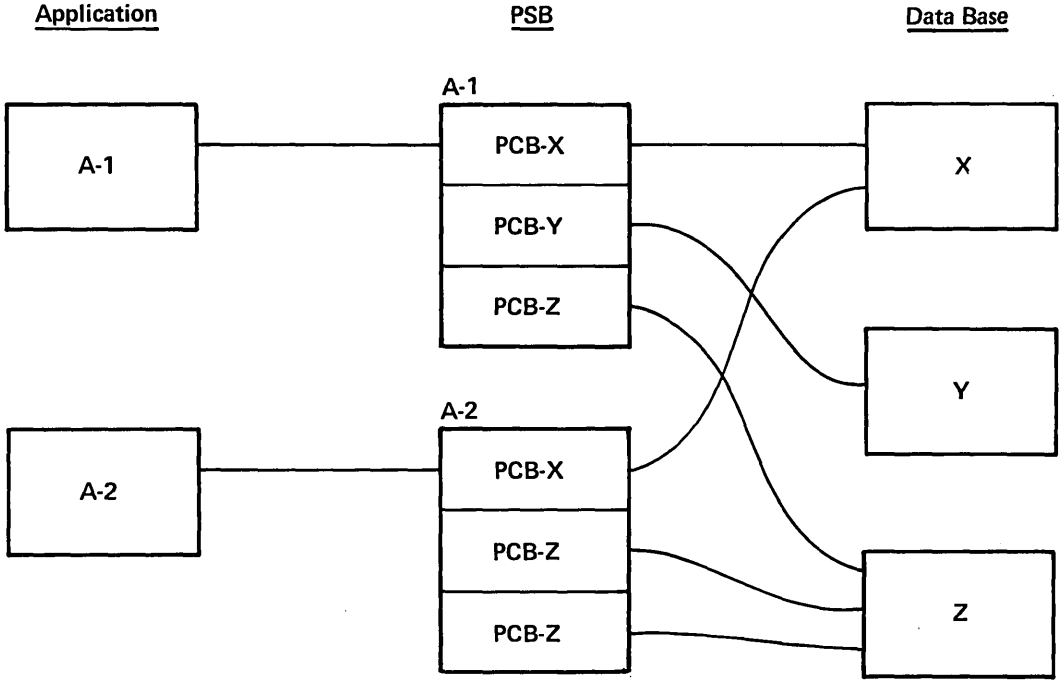


Figure 2-13. Multiple Logical Data Structures for the Same Data Base

In Figure 2-13, the PSB for application program A-1 shows that the application is processing three logical data structures, one from each of three different data bases. The PSB for application program A-2 also contains PCBs for three logical data structures. Two of these logical structures, however, identify the same data base. They may or may not identify different segment types or processing options.

SYSTEM SERVICE CALLS

System service calls control the system rather than transmit data. The following system service calls are available to IMS/VS application programs:

CHECKPOINT (CHKP). The CHKP call informs IMS/VS that the user has reached a logical synchronization point and that the program can be restarted at this point. IMS/VS can optionally invoke an OS checkpoint. The current position is maintained in GSAM data bases.

RESTART (XRST). The XRST call requests IMS/VS to restore checkpointed user areas and reposition GSAM data bases for sequential processing if a checkpoint ID for restarting has been supplied by the call or in the JCL. XRST is only valid for a batch or BMP region.

DEQUEUE (DEQb). The DEQ call is used to make available for general use any segments previously enqueued by the user with the Q command code in an SSA of a data base call.

ROLLBACK (ROLL). The ROLL call is used to request that any data base updates be backed out and output messages generated by the caller not be sent. It is treated as a user program abend, but the program and transaction are not stopped.

LOG (LOGb). The LOG call allows the user to put information on the system log.

GET SCD (GSCD). The GSCD call obtains the address of the IMS/VS System Contents Directory (SCD).

STATISTICS (STAT). The STAT call is used to obtain various statistics from DL/I.

The DEQb and ROLL calls are only valid from a message or batch message processing region. The CHKP call is valid from any IMS/VS user region, but the action taken varies with the type of region. The LOGb call is valid from any IMS/VS region. If issued from a batch region which has no system log, no action is taken.

The CHKP, DEQ, and LOG system service calls must reference the I/O PCB. The I/O PCB for a batch program is defined at PSBGEN time by use of the CMPAT option. For additional information on the CMPAT option, see the IMS/VS Utilities Reference Manual.

CHECKPOINT (CHKP)

When DL/I receives a CHKP call, it writes to the data base all buffers that were modified by the user. A log record is also created which contains the checkpoint identification passed with the call. The position of each data base PCB is set to the beginning of the data base. See the section "Batch Checkpoint Restart Considerations" in the "Application Program Design" chapter of the System/Application Design Guide.

If CHKP is issued from either a message or a batch message processing region, the following additional actions are taken:

- All data base resources enqueued for this user are released.
- If the user program references a transaction code, the message queue for that transaction is checkpointed. After the checkpointing action is completed, a GU call to the input message queue is internally generated, and a new message (if one is available) is returned in the work-area location.

The CHKP call is used in a message processing region in conjunction with multiple-mode scheduling of transactions. It allows the user to determine the grouping of messages for backout and restart purposes. For single mode scheduling or multiple mode scheduling where no grouping is necessary, Program Isolation will handle all checkpoint functions. The grouping IMS/VS uses is either that each message is unique or that all messages read at a given schedule of the program are considered to be connected. The basic CHKP call allows the user to specify groupings in between.

Batch or batch-message programs can use either the basic CHKP call or the symbolic CHKP call to coordinate logical synchronization points with the IMS/VS recovery log. If the basic CHKP call is used, the following rules apply:

- The user can request that IMS/VS issue an OS checkpoint for the user's region.
- The user cannot issue an OS checkpoint.

Examples of the Basic CHKP Call

The format of the basic CHKP call for an ANS COBOL program is:

```
CALL 'CBLTDLI' USING [parmcount,] call-func,
    IOPCB-name, I/O-area [,chkp-func].
```

The format of the basic CHKP call for a PL/I program is:

```
CALL PLITDLI (parmcount,call-func,IOPCB-name,I/O-area
    [,chkp-func]);
```

The format if the basic CHKP call for an Assembler Language program is:

```
CALL ASMTDLI ([parmcount,]call-func,IOPCB-name,
    I/O-area[,{chkp-func}])[,VL]
              [ {chkp-DCB } ]
```

parmcount
: is the address of a binary fullword containing the number of parameters that follow (required for PL/I).

call-func
: is the address of the call function "CHKP".

IOPCB-name
: is the address of the I/O PCB.

I/O-area
: is the address of the I/O area. In applications that access the IMS/VS message queues, the CHKP call implies a message GU, and a message can be returned. In batch or batch-message programs, the I/O area must contain the 8-byte checkpoint identification. This is used for operator or programmer communication and should consist of EBCDIC characters.

chkp-func (optional)
: is the address of an 8-byte area containing the value "OSVSCHKP". If this parameter is specified and DD statements are provided for an OS checkpoint data set, IMS/VS will provide DCBs for the user and issue an OS/VS checkpoint for the user's region before proceeding with the DL/I CHKP call.

Note: The optional parameters "chkp-func" and "chkp-DCB" are mutually exclusive and are valid only for batch or batch-message programs written in Assembler Language.

The symbolic CHKP call is used in conjunction with the XRST call and is valid only if the batch or batch-message program issued a restart (XRST) call. The following functions are provided by the symbolic CHKP call:

- The fully-qualified key of the last record processed by the application program for each IMS/VS data base is recorded on the IMS/VS recovery log.
- User-specified areas (for example, application variables, control tables, and position information for non-IMS/VS data sets) are optionally recorded on the IMS/VS recovery log.

Examples of the Symbolic CHKP Call

The format of the symbolic CHKP call for an ANS COBOL program is:

```
CALL 'CBLTDLI' USING [parmcount,]call-func,IOPCB-name,
    I/O-area-len,I/O-area
    [,1st-area-len,1st-area,...,nth-area-len,nth-area].
```

The format of the symbolic CHKP call for a PL/I program is:

```
CALL PLITDLI (parmcount,call-func,IOPCB-name,I/O-area-len,
    I/O-area[,1st-area-len,1st-area,...,nth-area-len,nth-area]);
```

The format of the symbolic CHKP call for an Assembler Language program is:

```
CALL ASMTDLI ([parmcount,] call-func,IOPCB-name,I/O-area-len,
    I/O-area[,1st-aera-len,1st-area,...,nth-area-len,
    nth-area])[,VL]
```

parmcount, call-func, IOPCB-name, and I/O-area
: are the same as for the basic CHKP call.

I/O-area-len
: is the address of the length of the largest I/O area used by the application program.

1st-area-len (optional)
: is the address of the length of the first area to checkpoint.

1st-area (optional)
: is the address of the first area to checkpoint.

nth-area-len (optional) is the address of the length of the nth area to checkpoint.

: Note: A checkpoint can be taken on a maximum of seven areas (n=7).

nth-area (optional)
: is the address of the nth area to checkpoint.

: Note: A checkpoint can be taken on a maximum of seven areas (n=7).

In addition to the status codes returned from GU calls, the following status code is also returned from the CHKP call:

XD	IMS/VS is terminating; further DL/I calls must not be issued.

The application program should test the status code returned from a DL/I CHKP call. If the status code indicates that IMS/VS is undergoing a checkpoint freeze (code "XD"), the application should terminate without issuing further DL/I calls. (This code will only be returned to a batch-message application.) If another DL/I call is issued, the application program will abend.

The user must re-establish his position in all IMS/VS data bases (except GSAM) after return from the checkpoint.

RESTART (XRST)

Upon receiving this call, IMS/VS checks whether a checkpoint identification (ID) has been supplied in the PARM field of the EXEC card or in the work area pointed to by the XRST call. If no ID has been supplied, the call is treated as a NOP, except that a flag is set to trigger storing of repositioning data and user areas on subsequent CHKP calls.

If the checkpoint at which restart is to occur has been supplied, the IMS/VS batch restart routine reads backward on the log defined in the //IMSLOGR DD statement to locate the checkpoint records.

User-program areas are restored. If the user does not specify main storage locations, IMS/VS obtains storage for him from subpool 0. Addresses and lengths of the areas are returned in the area list specified by the call.

Each GSAM data base that is active at the checkpoint is repositioned for sequential processing by issuing a GU for the last record processed at that point. Data bases being loaded are not repositioned except for GSAM data bases defined to use BSAM accessing. No data is returned from this automatic GU. Key feedback information is provided in the PCB for each data base that is active at the checkpoint. The user program must reposition itself on all non-GSAM data bases, just as it must do after taking a checkpoint.

Examples

The format of the XRST call for an ANS COBOL program is:

```
CALL 'CBLTDLI' USING [parmcount,]call-func,IOPCB-name,
    I/O-area-len,work-area[,1st-area-len,1st-area,...,
    nth-area-len,nth-area].
```

The format of the XRST call for a PL/I program is:

```
CALL PLITDLI (parmcount,call-func,IOPCB-name,I/O-area-len,
    work-area[,1st-area-len,1st-area,...,nth-area-len,
    nth-area]);
```

The format of the XRST call for an Assembler Language program is:

```
CALL ASMTDLI ([parmcount,]call-func,IOPCB-name,I/O-area-len,
    work-area[,1st-area-len,1st-area,...,nth-area-len,
    nth-area])[VL]
```

parmcount
: is the address of a binary fullword containing the number of parameters that follow (required for PL/I).

call-func
: is the address of the call function "XRST".

IOPCB-name
: is the address of a pointer to either the I/O PCB or the "dummy" I/O PCB supplied by the CMPAT option during PSBGEN.

I/O-area-len
: is the address of the length of the largest I/O area used by the user program.

work-area
is the address of a 12-byte work area. This area should be set to blanks (X'40') before the call and tested on return. If the program is being started normally, the area will be unchanged. If the program is being restarted from a checkpoint, the ID supplied by the user and specified in the PARM keyword on the EXEC statement in his CHKP call will be placed in the first eight bytes.

If the user wishes to use his own restart method, the XRST call can be used to reposition GSAM data bases by placing the checkpoint ID in this area before issuing the call. This ID can be either the 8-byte left-aligned user-supplied ID, or the 12-byte YYDDD/HHMMSS ID.

1st-area-len
is the address of the length of the first area to be restored.

1st-area
is the address of the first area to be restored.

nth-area-len
is the address of the length of the nth area to be restored.

Note: The maximum number of areas that can be restored is seven (n=7).

nth-area
is the address of the nth area to be restored.

Notes:

1. The number of areas specified on the XRST call must be equal to the maximum specified on any symbolic CHKP call.

2. The lengths of the areas specified on the XRST call must equal the maximum lengths of the corresponding areas (in sequential order) of any symbolic CHKP call.

3. The XRST call is issued only once and is the first request that is made to IMS/VS.

4. The maximum number of areas that can be restored is seven (n=7).

DEQUEUE (DEQb)

DEQ is used by the user's program to release resources that had previously been reserved with the Q command code in an SSA. If the resource to be freed had been upgraded to the exclusive-use level as a result of being modified since being reserved with the Q call, the resource will be released by the next synchronization point. If, however, the resource to be freed had not been upgraded as described above, it will be released (dequeued) to other users who request it. (The Q command code is described in the following chapter.)

The PCB passed with the DEQ call must be the I/O PCB. It is used only for returning a status code.

The work area must contain the ID of the queue class to be released.

Examples

The format for an ANS COBOL program is:

```
CALL 'CBLTDLI' USING deq-func, pcb-name, work-area.
```

The format for a PL/I program is:

```
CALL PLITDLI (THREE,DEQ_FUNC,PCB_NAME,WORK_AREA);
```

The format for an Assembler Language program is:

```
CALL |ASMTDLI|
     |CBLTDLI| ,(PARMCOUNT,DEQ FUNC,(Rp),(Rw))
```

Rp
is the register pointing to the PCB, and Rw is the register point to the work area

DEQ issues only the "bb" status code.

ROLLBACK (ROLL)

ROLL is issued by a user program when it determines that some invalidity exists in the processing it has done. All data bases and message activity (except EXPRESS) since the last sync point are backed out. This call is recognized and processed in the user region and therefore no parameters other than the function code are required.

A user 0778 abend is generated in the user region. This abend code is recognized in the control region and special abend action is taken. All DL/I activity is backed out for the current message (or group of messages) back to the last synchronization point. No output messages are sent, except those inserted with the express facility. The input message (or group of messages) is dequeued. The transaction will be rescheduled, and processing will continue with the next message. If the 'ROLL' call is issued by a DL/I (independent batch) program, it will cause the user 0778 abend. The DL/I activity must be backed out, however, using the IMS/VS Data Base Backout utility. See the discussion of the "Data Base Backout Utility" in the "Data Base Recovery" chapter of the IMS/VS Utilities Reference Manual.

Examples

The format for an ANS COBOL program is:

```
CALL 'CBLTDLI' USING roll-func.
```

The format for a PL/I program is:

```
CALL PLITDLI (ONE,ROLL_FUNC);
```

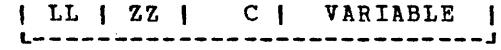
The format for an Assembler Language program is:

```
CALL |ASMTDLI|
     |CBLTDLI| ,(PARMCOUNT,ROLL)
```

No status codes are returned for a ROLL call.

LOG (LOGb)

The LOG call causes a user record to be written to the system log. The record must be of the following format:



LL
is a halfword containing the length of the message.

When PL/I is used, the LL field must be defined as a binary fullword. The PL/I user must place the length of the text to be written in this field. The value must represent the total of:

2 for the count field (even though it is physically 4 bytes in the PL/I environment)

2 for the ZZ field

1 for the C field

n for the variable field

ZZ
is a halfword of zeroes.

C
is a 1-byte user code which must be equal to or greater than X'A0' in value.

The length of the area must be 4 bytes less than the length of the LRECL for the system log.

The PCB passed with the LOG call must be the I/O PCB. It is used only for returning a status code.

Examples

The format for an ANS COBOL program is:

```
CALL    'CBLTDLI' USING log-func, pcb-name, record-area.
```

The format for a PL/I program is:

```
CALL    PLITDLI (THREE,LOG_FUNC,PCB_NAME,RECORD_AREA);
```

The format for an Assembler Language program is:

```
        |ASMTDLI|
CALL    |CBLTDLI|,(PARMCOUNT, LOGFUNC, (Rp), (Rr))
```

RP
is the register pointing to the first PCB in the list of PCBs passed at entry, and Rr is the register pointing to the record area.

There are three possible status codes from the LOG call:

1. "AT" The record length in the LL field is too long. No logging is done.

2. "GL" The log code is not a valid user code.

3. "bb" Everything is fine.

GET SCD (GSCD)

The GSCD call is used to obtain the addresses of the IMS/VS System Contents Directory (SCD) and Partition Specification Table (PST). It is suggested that any program that references these control blocks use the DSECTS provided by macros in the macro library for the IMS/VS system. The macro for the SCD DSECT is ISCD SLDBASE=0; the macro for the PST DSECT is IDLI PSTBASE=0.

Example

An example of a GSCD call format for an Assembler Language program is:

```
        |ASMTDLI|
CALL    |CBLTDLI|,([parmcount,]function,pcb-name,work area)[,VL]
```

parmcount
is 3 if provided

function
is the address of the call function 'GSCD'

pcb-name
is the address of any valid PCB

work area
is the address of an 8-byte area. The call will place the address of the SCD in the first 4 bytes and the address of the PST in the second 4 bytes.

VL
VL must be specified if parmcount is not used.

Note: When running a MSG or BMP region type, using either the VS2 Operating System or the VS1 Operating System with fetch protect specified, the GSCD call functions normally. The operating system, however, does not permit a program in one region (the MSG or BMP region) to access data in another region (the CTL region). Therefore, the addresses returned on the GSCD call cannot be used in either a MSG or BMP region type. An OC4 system abend results if they are used in the above situation. Since the SCD and PST are in the same Operating System region as the application program when running in a DLI or DBB region type, these addresses can be used by a DLI or DBB region.

STATISTICS (STAT)

The STAT call is used to obtain statistics in various forms from the IMS/VS system.

Examples

The format of the STAT call for a COBOL program is:

```
CALL 'CBLTDLI' USING [parmcount,]call-func,pcb-name,
                     I/O-area,stat-func.
```

The format of the STAT call for a PL/I program is:

```
CALL PLITDLI (parmcount,call-func,pcb-name,I/O area,
              stat-func) ;
```

The format of the STAT call for an Assembler Language program is:

```
CALL ASMTDLI, ([parmcount,] call-func,
              pcb-name,I/O-area,stat-func)[,VL]
```

parmcount
: is an optional parameter except for PL/I. If present it is the address of a binary fullword containing the value 4.

call-func
: is the address of the call-function STAT.

pcb-name
: is the address of a data base PCB. This PCB is used to pass status back to the application program. The OS access method used by the data sets associated with this PCB are not related to the type of statistics that will be returned from the STAT call.

I/O-area
: is the address of an area in the application program large enough to hold the statistics requested.

stat-func
: is the statistics function and the address of a 9-byte area whose contents describe the type and form of statistics required. The first 4 bytes define the type of statistics desired and the 5th byte defines the format to be provided. The remaining 4 bytes should contain EBCDIC blanks. If the stat-function provided is not one of the defined functions, then an AC status code is returned to the user.

Stat Functions - ISAM/OSAM Buffer Pool
: For ISAM/OSAM buffer pool statistics, the following are the possible values for the stat-function parameter and the format of the data that will be returned to the application program. If there is no ISAM/OSAM buffer pool present, then a GE status code will be returned.

DBASF

This function value will provide the full ISAM/OSAM data base buffer pool statistics in a formatted form. The application program I/O area must be at least 360 bytes. Three 120 bytes formatted (for printing) records are provided; two heading lines and one line of statistics.

The format of the data is as follows:

```
  BLOCK   FOUND  READS    BUFF    OSAM  BLOCKS    NEW  CHAIN
    REQ IN POOL ISSUED    ALTS  WRITES WRITTEN BLOCKS WRITES
nnnnnnn nnnnnnn  nnnnn nnnnnnn nnnnnnn nnnnnnn  nnnnn  nnnnn

WRITTEN    POOL    BUFF   BUFFS    RET   ISAM   ISAM
ON CHNS COMPACT    COMB   MOVED BY KEY GT NXT  SETLS ERRORS
nnnnnnn nnnnnnn nnnnnnn nnnnnnn  nnnnn  nnnnn  nnnnn  nn/nn
```

BLOCK REQ	=	number of block requests received
FOUND IN POOL	=	number of times the block requested was found in the buffer pool
READS ISSUED	=	number of OSAM reads issued
BUFF ALTS	=	number of buffers altered in the pool
OSAM WRITES	=	number of OSAM writes issued
BLOCKS WRITTEN	=	number of blocks written from the pool
NEW BLOCKS	=	number of new blocks created in the pool
CHAIN WRITES	=	number of chained OSAM writes issued
WRITTEN ON CHNS	=	number of blocks written on OSAM chains
POOL COMPACT	=	number of times the buffer pool was compacted
BUFF COMB	=	number of buffers combined during pool compactions
BUFF MOVED	=	number of buffers moved during pool compactions
RET BY KEY	=	number of ISAM records retrieved by key
ISAM GT NXT	=	number of ISAM get next calls received by the buffer handler
ISAM SETLS	=	number of ISAM SETLs issued by the buffer handler
ERRORS	=	number of write error buffers currently in the pool / the largest number of errors in the pool during this execution

DBASU

This function value will provide the full ISAM/OSAM data base buffer pool statistics in an unformatted form. The application program I/O area must be at least 72 bytes. Eighteen fullwords of binary data are provided. The first word is a count of the number of words that follow; the second through eighteenth words are the statistic values in the same sequence as presented with the DBASF function value above.

DBASS

This function value will provide a summary of the ISAM/OSAM data base buffer pool statistics in a formatted form. The application program I/O area must be at least 180 bytes. Three 60-byte formatted (for printing) records are provided.

The format of the data is:

```
DATA BASE BUFFER POOL: SIZE nnnnnnn
  REQ1 nnnnn REQ2 nnnnn READ nnnnn BISAM nnnnn WRITES nnnnn
  KEYC nnnnn COMP nnnnn COMB nnnnn MOVES nnnnn ERRORS nn/nn
```

SIZE	=	buffer pool size
REQ1	=	number of block requests
REQ2	=	number of block requests satisfied in the pool plus new blocks created
READ	=	number of read requests issued
BISAM	=	number of BISAM reads issued
WRITES	=	number of OSAM writes issued
KEYC	=	number of retrieve by key calls
COMP	=	number of pool compactions
COMB	=	number of buffers combined by compaction
MOVES	=	number of buffers moved by compaction
ERRORS	=	number of permanent errors now in the pool / largest number of permanent errors during this execution

Stat Functions - VSAM Buffer Subpools

Since there may be several buffer subpools for VSAM data bases, the STAT call is iterative when requesting these statistics. The first time the call is issued, the statistics for the subpool with the smallest buffer size will be provided. For each succeeding call (without intervening use of the PCB), the statistics for the subpool with the next larger buffer size will be provided. The final call for the series will return a GA status code in the PCB and the statistics returned will be totals for all subpools. If there are no VSAM buffer subpools present, a GE status code will be returned.

VBASF

This function value will provide the full VSAM data base subpool statistics in a formatted form. The application program I/O area must be at least 360 bytes. Three 120-bytes formatted (for printing) records are provided; two heading lines and one line of statistics. Each successive call will return the statistics for the next subpool.

The format of the data is:

```
      B U F F E R   H A N D L E R   S T A T I S T I C S
BSIZ NBUF RET RBA RET KEY ISRT ES ISRT KS BFR ALT   BGWRT SYN PTS
nnnK  nnn nnnnnnn nnnnnnn nnnnnnn nnnnnnn nnnnnnn nnnnnnn nnnnnnn

          V S A M   S T A T I S T I C S
   GETS  SCHBFR   FOUND   READS USR WTS NUR WTS ERRORS
nnnnnnn nnnnnnn nnnnnnn nnnnnnn nnnnnnn nnnnnnn  nn/nn
```

BSIZ = the size of the buffers in this subpool
In final total this is the total size of all subpools.

NBUF = the number of buffers in this subpool
In final total this is the total number of buffers in all subpools.

RET RBA = the number of retrieve by RBA calls received by the buffer handler

RET KEY = the number of retrieve by key calls received by the buffer handler

ISRT ES = the number of logical records inserted into ESDSs

ISRT KS = the number of logical records inserted into KSDSs

BFR ALT = the number of logical records altered in this subpool

BGWRT = the number of times the Background Write function was invoked by the buffer handler

SYN PTS = the number of synchronization calls received by the buffer handler

GETS = the number of VSAM GET calls issued by the buffer handler

SCHBFR = the number of VSAM SCHBFR calls issued by the buffer handler

FOUND = the number of times VSAM found the control interval requested already in the subpool

READS = the number of times VSAM read a control interval from external storage

USR WTS = the number of VSAM writes initiated by IMS/VS

NUR WTS = the number of VSAM writes initiated in order to make space in the subpool

ERRORS = the number of write error buffers currently in the subpool / the largest number of write errors in the subpool during this execution

VBASU

This function value will provide the full VSAM data base subpool statistics in an unformatted form. The application program I/O area must be at least 72 bytes. Eighteen fullwords of binary data are provided for each subpool. The first word is a count of the number of words that follow; the second through eighteenth words are the statistics values in the same sequence as presented with the VBASF function value above.

VBASS

This function value will provide a summary of the VSAM data base subpool statistics in a formatted form. The application program I/O area must be at least 180 bytes. Three 60-byte formatted (for printing) records are provided.

The format of the data is:

```
DATA BASE BUFFER POOL: BSIZE nnnnnnn
  RRBA nnnnn RKEY nnnnn BFALT nnnnn NREC nnnnn SYN PTS nnnnn
  NMBUFS nnn VRDS nnnnn FOUND nnnnn VWTS nnnnn  ERRORS nn/nn
```

BSIZE	= the size of the buffers in this VSAM subpool
RRBA	= number of retrieval requests by RBA
RKEY	= number of retrieval requests by key
BFALT	= number of logical records altered
NREC	= number of new VSAM logical records created
SYN PTS	= number of synchronization point requests
NMBUFS	= number of buffers in this VSAM subpool
VRDS	= number of VSAM control interval reads
FOUND	= number of control intervals VSAM found in the subpool thru lookaside
VWTS	= number of VSAM control interval writes
ERRORS	= number of permanent write errors now in the subpool / largest number of errors in this execution

EXAMPLES OF DATA BASE PROCESSING USING DL/I I/O FUNCTIONS

In the examples which follow, a very simple form of qualified SSAs is used. SSA qualification is discussed in detail in the following chapter.

Consider Figure 2-14. The name of the skill segment type is SKILLINV, and its key field name is SKILCODE. The SSA for GU of the skill segment with skill code equal to artist appears as:

```
SKILLINV(SKILCODEb=ARTIST)
```

The portion of the SSA within the parentheses is called the qualification statement.

For unique retrieval or addition of a root segment, only one SSA must be provided. The unique retrieval or insertion of a dependent segment normally requires multiple SSAs to be provided in the functional request. Each SSA in the list describes a segment to which the dependent segment to be operated upon is dependent. The SSAs for a given DL/I call must be in proper hierarchical relationship. If the generic name of a name segment type is NAME, its key field name is NAME. Note that there is an employee with a key field value of ADAMS whose parent is a skill segment having a key field value of ARTIST. Unique retrieval is accomplished by two SSAs included within the parameter list of the DL/I call:

```
SKILLINV(SKILCODEb=ARTIST)

NAMEbbbb(NAMEbbbbb=ADAMS)
```

The definition of the data base to be operated upon is provided in each DL/I call by a data base PCB. All data base PCBs used by a particular application program for data base operations are contained within the PSB for that program. At execution time, the base addresses of the PCBs are passed to the application program. Each PCB contains the 1- to 8-byte name of the DBD associated with the data base.

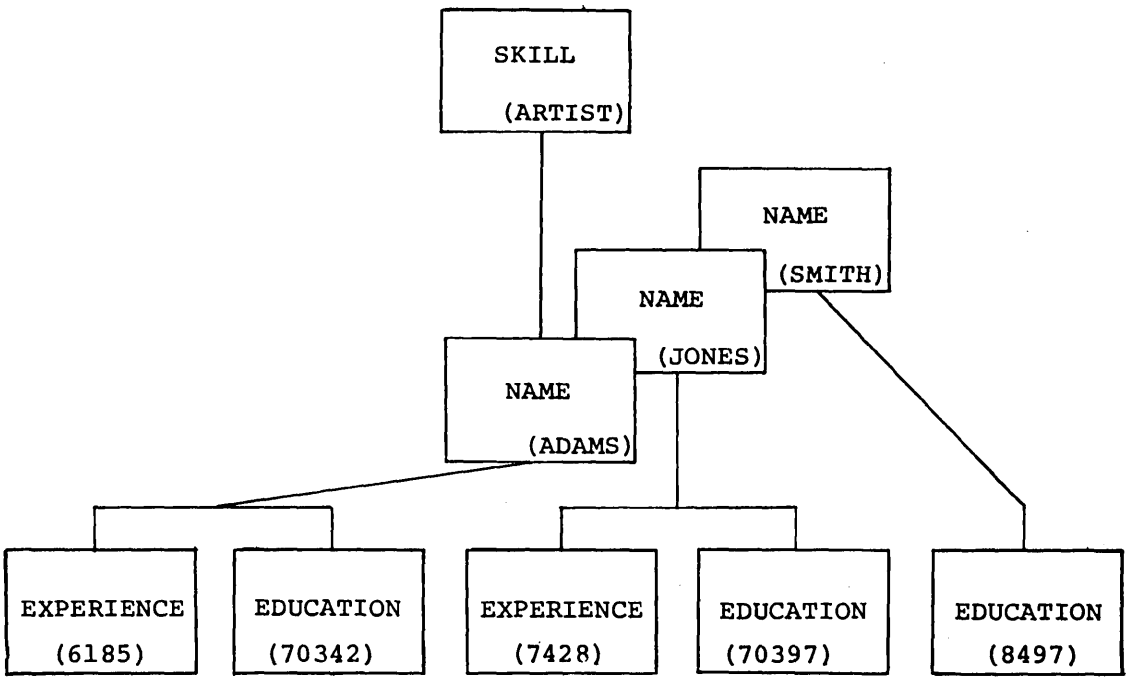


Figure 2-14. Logical Data Base Record Structure

DATA BASE CREATION

A data base is created by an application program issuing DL/I calls to insert data base records presorted by the key field of the root segment. This is a requirement of HSAM, HISAM, and HIDAM databases. An HDAM data base load can receive sorted or unsorted keys of data base records.

In an HSAM, HISAM, and HIDAM data base, when a data base record is composed of more than the root segment, all segments within the data base record must be presorted by their hierarchical relationship and key-field value and must be inserted in their hierarchical order. Consider the process of inserting the segments of a skill inventory data base record described in Figure 2-14. First, the Skill (root) segment is inserted. The name segment for Adams is inserted next. Then the experience segment of Adams is inserted, followed by the education segment of Adams. This continues with the name segment (Jones), its experience segment and education segment, then name segment (Smith) and its education segment. If this data base record represented the segments of data associated with skill X, the segments to be inserted into the data base next would be those associated with SKILLINV X + 1.

The insert function is used to create or load (recreate or reorganize) a data base. Prior to the execution of a DL/I call to cause segment insertion, the segment to be inserted must be moved into a segment input/output work area and the proper list of an SSA or SSAs must be assembled. Let us assume that we are creating the skill inventory data base and we are about to load the segments of data associated with SKILL value ARTIST. The first four segments to be loaded would be skill, name (Adams), experience (Adams), and education (Adams). The associated segment search arguments and work area contents for these four DL/I ISRT calls are as follows. Note that lowercase b's indicate blanks.

Skill Segment Insertion

```
SSA1   -   SKILLINV

Work Area - (containing skill  segment)

       ┌──────────┬──────────┐
       │ Key Field │ Data Field│
       └──────────┴──────────┘
       Key =ARTIST
```

Name Segment Insertion

```
[SSA1 -  SKILLINV(SKILCODEb=ARTIST) ]   OPTIONAL

 SSA2 -  NAMEbbbbb

Work Area - (containing name segment)

       ┌──────────┬──────────┐
       │ Key Field │ Data Field│
       └──────────┴──────────┘
       Key =ADAMS
```

Experience Segment Insertion

```
SSA1 -  SKILLINV(SKILCODEb=ARTIST)      OPTIONAL

SSA2 -  NAMEbbbb(NAMEbbbbb=ADAMS)       OPTIONAL

SSA3 -  EXPERIENb

Work Area - (containing experience segment)

       ┌──────────┬──────────┬──────────┐
       │ Key Field │ Data Field│ Data Field│
       └──────────┴──────────┴──────────┘
       Key =6185
```

Education Segment Insertion

```
SSA1 -  SKILLINV(SKILCODEb=ARTIST)      OPTIONAL

SSA2 -  NAMEbbbb(NAMEbbbbb=ADAMS)       OPTIONAL

SSA3 -  EDUCbbbbb

Work Area - (contains education segment)

       ┌──────────┬──────────┬──────────┐
       │ Key Field │ Data Field│ Data Field│
       └──────────┴──────────┴──────────┘
    Key =70342
```

Notice that the SSAs of a DL/I call for inserting a segment into a data base may describe the complete hierarchical path to the segment. It is not necessary, however, to describe the complete path. DL/I assumes existing position when no SSA is specified. When creating a data base, therefore, it is only necessary to supply the segment name of the segment being inserted. Notice that the last SSA within each ISRT call does not (and must not) include the qualification statement

portion. The qualification information is taken from the image of the segment in the input/output work area.

A hierarchical path of segments may be inserted into the data base with one call by concatenating the segments to be inserted in the I/O area and supplying a corresponding list of unqualified segment search arguments. The SSA for the first segment in the path to be inserted by this call must have the D command code set. The hierarchical path must proceed downward in the hierarchy, with each segment in the I/O area being a child of the segment preceding it in the I/O area. The example shown below illustrates the insertion of the first six segments shown in Figure 2-15 by using a path of insert calls.

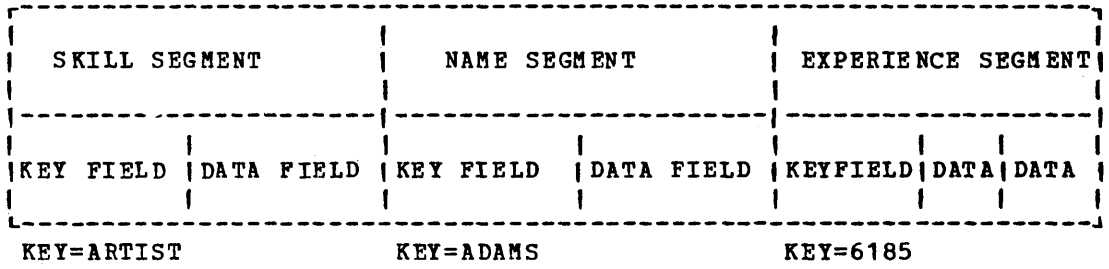
Skill, Name, and Experience Segment Insertion

SSA - SKILLINV*Db

SSA - NAMEbbbbb

SSA - EXPERIENb

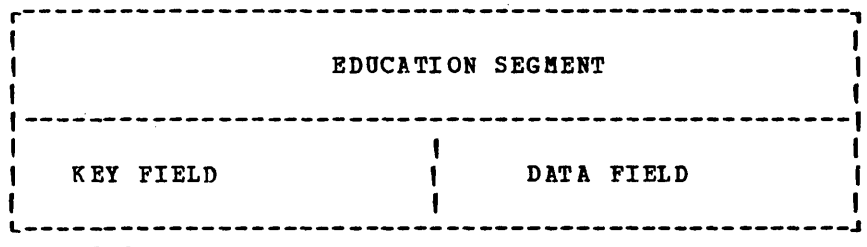
Work Area (containing Skill, Name, and Experience segments)



Education Segment Insertion

SSA - EDUCbbbbb

Work Area (containing Education segment)

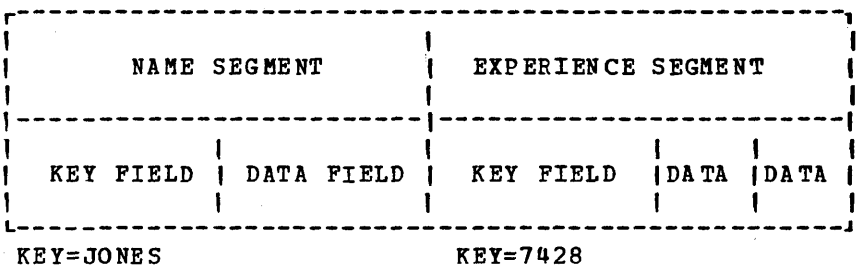


Name and Experience Segment Insertion

SSA - NAMEbbbb*Db

SSA - EXPERIENb

Work Area (containing Name and Experience segments)



All data base creation and reorganization must be performed in a batch processing region of IMS/VS. If the physical organization is HDAM, presorting by key field of the root is not required for data base creation. HISAM, HSAM, and HIDAM all require presorted root segments by key field sequence.

DATA BASE RETRIEVALS

The retrieval of segments within a data base is accomplished by the three GET call functions: get unique, get next, and get-next-within-parent segment. Get unique provides for the retrieval of a specific segment by direct reference into the data base. Get next provides for sequential segment retrieval. Usually the get next function is used after a get unique or get next which has provided "positioning" to a unique segment within the data base. A get next may be used, however, without positioning being supplied by a previous get unique or get next. If DL/I has no position established within a data base when a get next call is issued, the request is satisfied by proceeding from the beginning of the data base. The get-next-within-parent segment allows sequential retrieval of all segments subordinate to a parent segment. An example, using Figure 2-14, would be all experience and education segments within the skill inventory data base for a given skill code and employee number. The parent segment is a unique name segment, and parentage must have been previously established with a get unique or get next request.

Once all the experience and education segments for a given skill code and employee number have been retrieved by a succession of get next within parent requests, an indication is returned to the application program. This indication provides definition of the end of subordinate segments for the particular skill code and employee number.

In addition to direct retrieval of a unique segment and sequential retrieval of segments, an ability to sequentially skip from one segment to another of a common type is provided. Assume that it becomes necessary to retrieve all name segments within a particular skill segment but not those segments subordinate to each name segment (that is, experience and education data segments). The first name segment would be retrieved with a call where the function equals get unique. The SSAs would be:

```
SSA  -  SKILLINV(SKILCODEb=ARTIST)
SSA  -  EMPLOYEEb
```

By changing the function to get next and repeating the above SSAs, all name segments whose SKILL=ARTIST would be retrieved with a not-found status returned when there were no more employee segments for SKILL=ARTIST.

DATA BASE UPDATES

The updating of data within a segment of a data base is performed through the replace input/output function. Before a DL/I call to replace a segment can be executed, the segment to be updated must be retrieved through a call with a GET function. The GET functions which can be specified are those previously discussed; they must, however, include the addition of a Hold definition (get hold unique, get hold next, and get hold next within parent). The replace function must then be executed in the next call by this program against the data base PCB. Any intervening calls against the same data base PCB by this program cause the rejection of the subsequent replace call. No SSAs are permitted with the replace function unless command codes for segment path replacement are employed. The key field of the segment to be updated through the replace function call must not be modified.

The following is an example of how to change the data in the field of the skill segment of artist from COMMERCIAL to COMMERCIAL-CARTOON:

The first PL/I call statement is:

```
CALL PLITDLI (FOUR,FUNC_GHU,DB_PCB,WORK_AREA,SSA1);

SSA1 is SKILLINV(SKILCODEb=ARTIST)

WORK_AREA is then   | ARTIST| COMMERCIAL     |
                    L-----------------------J
```

The second PL/I call statement is:

```
CALL PLITDLI (THREE,FUNC_REPL,DB_PCB,WORK_AREA);

WORK_AREA is now    | ARTIST|   COMMERCIAL-CARTOON  |
                    L-------------------------------J
```

and this is the data that is placed back in the data field of the skill segment.

DATA BASE DELETIONS

The deletion of an entire segment (all fields) within a data base is performed through the delete input/output function. Before a DL/I call to delete a segment may be executed, the segment to be deleted must be retrieved through a get hold call. The delete function must be executed as the next call against the data base, through the same PCB, or the delete function is rejected. The deletion of a parent segment normally causes deletion of all segments subordinate to the deleted segment. All subordinate data set groups must be available for processing prior to the delete call being issued. If they are not, a status code of AI is returned. Subordinate segments that could be accessed are deleted.

The following is an example of how to delete the skill segment data (both key and data fields) of artist:

The first PL/I call statement is:

```
CALL PLITDLI(FOUR,FUNC_GHU,DB_PCB,WORK_AREA,SSA1);

SSA1 is SKILLINV(SKILCODEb=ARTIST)

WORK_AREA is then | ARTIST |   COMMERCIAL-CARTOON   |
                  L--------------------------------J
```

The second PL/I call statement is:

```
CALL PLITDLI(THREE,FUNC_DLET,DB_PCB,WORK_AREA);

WORK_AREA is still | ARTIST |   COMMERCIAL-CARTOON   |
                   L--------------------------------J
```

and dependent segments under this root or parent are deleted. That is, name segment (ADAMS), experience segment (ADAMS), and education segment (ADAMS) are deleted as well as all other name, experience, and education segments under this root.

If a GU call is made to this particular skill segment, a status code of GE (not found) will be returned, but the WORK AREA, if not blanked out, may still contain the above data.

DATA BASE INSERTIONS

The addition or insertion of a new segment (all fields) into an existing data base is performed through the insert input/output function. The techniques used for performing an insert function to add a segment to an existing data base are identical to those used with the insert function when creating a new data base. Remember that the addition of a dependent-level segment is not permitted unless all parent segments in the complete hierarchical path already exist in the data base. An example, using Figure 2-14, would be the addition of an experience segment subordinate to a particular name segment. The name segment must already exist in the data base or be added before any experience segment subordinate to that name segment may be added.

USING A BATCH REGION TO CHECK OUT ONLINE MESSAGE PROGRAMS

A natural stage in the development of online DC programs intended to be executed in BMP or MPP regions is first to test the DB portion of the program in a batch region.

The CMPAT option of the IMS/VS PSBGEN procedure (see the IMS/VS Utilities Reference Manual) circumvents the need to recompile the program between batch and online executions.

When the CMPAT option is exercised, the PSB parameters passed by the DB facility to a program executing in a batch region will contain the I/O PCB and the alternate PCBs specified in the PSBGEN.

The application programmer must be responsible for determining that the parameter list of the application program contains entries in PSBGEN sequence for the teleprocessing and data base PCBs.

EXAMPLES

Examples of teleprocessing programs to be run in a batch region are as follows.

For COBOL:

```
ENTRY 'DLITCBL' USING IO-PCBNAME, ALT-PCBNAME1, ALT-PCBNAMEN,
    DB-PCBNAME1.
```

For PL/I:

```
DLITPLI: PROCEDURE(IO_PCB_PTR,ALT_PCB_PTR1,ALT_PCB,PTRN,DB_PCB_PTR)
    OPTIONS(MAIN);
```

GENERALIZED SEQUENTIAL ACCESS METHOD (GSAM)

The Generalized Sequential Access Method (GSAM) implemented under DL/I provides sequential data base management capabilities. GSAM is intended especially for non-hierarchical sequential data bases.

GSAM supports data sets organized according to the following OS/VS access methods:

- Sequential Access Method (SAM)
- Virtual Storage Access Method (VSAM)

GSAM supports the Basic Sequential Access Method (BSAM) on DASD, unit record, and tape devices, and the ESDS Virtual Storage Access Method (VSAM) on DASD devices.

Record formats can be specified as fixed, or variable (or undefined in BSAM). The terms "segment," "segment type," "hierarchical," and "parentage" are not applicable to GSAM data sets, and the concepts of key and field do not apply.

GSAM DATA BASE RESTRICTIONS

The following restrictions apply to GSAM data bases:

- GSAM data bases can be allocated in a user region only.
- GSAM data bases do not have keys.
- GSAM data bases do not have segments (or segment types).
- VSAM data bases are non-keyed, non-indexed entry sequenced data sets (ESDS).
- Checkpoint cannot be supported during a VSAM data base load.
- VSAM load operations are not restartable.
- VSAM data bases must reside on DASD devices.
- Temporary, SYSIN, SYSOUT, and unit-record files are not supported in VSAM.
- Temporary data sets should not be used if program restartability is desired.

- SYSOUT data set restart provides redundant data output if output occurred after the restart checkpoint.
- Update/delete functions are not supported.
- Records cannot be randomly added to GSAM data bases. The data base can be extended using the ISRT function code (with DISP=MOD for BSAM).

GSAM FUNCTIONS

The functional capabilities of GSAM are the same for BSAM and VSAM data bases. There are three major functions:

- ISRT for file creation or extension only
- GN for sequential accessing
- GU for unique record accessing.

A GSAM data base can be created or extended with ISRT. The data base will be created in the sequence order of the input from the load program.

A GSAM data base can also be a data set previously created by use of O/S BSAM, QSAM, or VSAM. A GSAM data base may conversely later be accessed by other programs using those O/S processing methods.

GN is used for sequential processing of a GSAM data base. The starting point of the sequential retrieval can be established by GU, with the Record Search Argument (RSA).

An RSA can be supplied for data bases in which other than standard sequential retrieval is required. This argument will position the data set at the particular record desired.

GU can also be used for random accessing of any GSAM data base. This would be practical, however, only on DASD and should be limited in tape accessing.

DATA BASE ACCESS

All accessing to GSAM data bases is done with DL/I calls. A check is made by IMS/VS to determine whether a user request is for a GSAM data base. If so, control is passed to GSAM, which will be resident in the user region. If not, control is passed to the IMS/VS control region, or to Batch DL/I, and standard IMS/VS hierarchical processing will result.

Calls to be used for GSAM Accessing are:

- OPEN — Open GSAM data base
- CLSE — Close GSAM data base
- GU — Retrieve a unique record or reset sequential processing base
- GN — Retrieve next sequential record
- ISRT — Insert a new logical record (at end of data base only)

- CHKP — To request a region checkpoint
- XRST — To request a region/program restart
- DUMP, or SNAP — To send GSAM control blocks to IMSERR or secondarily to SYSPRINT. (No return code is used - status codes and control blocks remain the same.)

The open and close call are optional calls to be used to explicitly initiate or terminate data base operations. The data base will automatically be opened by the issuance of the first processing call used and automatically closed at "end-of-data" or at program termination.

Records cannot be randomly added to GSAM data sets. The data set may be extended by opening in the load mode, with DISP=MOD, and using the ISRT function code.

GSAM CALLS

The IMS/VS user communicates with IMS/VS through a DL/I call statement. The IMS/VS calls are generated as follows:

- COBOL — CALL 'CBLTDLI' USING [arg0,]arg1,arg2,arg3[,arg4].
- PL/I — CALL PLITDLI (arg0,arg1,arg2,arg3[,arg4]);
- ASSEMBLER — CALL ASMTCLI,([arg0,]arg1,arg2,arg3[,arg4])

where:

Arguments 0 and 4 are optional.

argument_0 — is the optional address of the parameter count or argument count of the number of arguments following argument 0.

argument_1 — is the address of the function code

argument_2 — is the address of the GSAM PCB

argument_3 — is the address of the I/O area (IOA) for access Calls, or the optional address of the OPEN-option for an OPEN Call

argument_4 — is the optional address of the record search argument (RSA) (It is not a segment search argument -- GSAM has no concept of segments; it is required only for GU.)

The first word contains the:

- BSAM tape relative block address, or
- BSAM DASD TTRZ, or
- VSAM relative byte address

For BSAM, the second word contains the volume-sequence number in the high halfword and the BSAM record displacement in the block in the low halfword.

The OPEN option is either INP, OUT, or, in the case of SYSOUT data sets, OUTA or OUTM to include control characters.

The PL/I description of the RSA is:

```
DCL     1 GSAM RSA,
            2  BLOCK_ID FIXED BIN (31),
            2  VOL_SEQ_NO FIXED BIN (15),
            2  RECORD_DISP FIXED BIN (15);
DCL     1 FIRST_RCD_RSA,
            2 (BLOCK1,DISP0) FIXED BIN (31) INIT (1);
```

Status Codes

Status codes inform the user program of situations that are normally encountered and abnormal situations caused by violations of IMS/VS conventions. No data is transferred from data base to user area, or vice-versa, when a nonblank code is returned.

GSAM initializes the PCB status code to blanks before processing each user request.

The common status codes, used to indicate the status of an I/O request after it has been processed, are passed to the PCB from the GSAM access modules. These status codes are included in Appendix B of this manual.

RECORD FORMATS

Records may be fixed or variable length, blocked or unblocked. Records must be unkeyed. Undefined data set format is supported only for BSAM. The inclusion of carriage control characters may also be indicated in the JCL RECFM subparameter (for example, RECFM=FBA) for all record formats. An optional control character may be used in the first byte of each record.

Fixed-Length Records

With fixed-length-record data sets, the user need not include a record length at the beginning of a data record. User records include only data bytes and are returned to the user in that form. The data set is built in the fixed- or fixed-blocked format by GSAM, with the logical record length coming from the DBD or JCL into the DCB.

The user must specify the record format (RECFM) subparameter as RECFM=F, or FB, in the definition of the data set DBD.

The specification of RECFM=F can be overridden by the JCL specification DCB=RECFM=FB.

Variable-Length Records

Variable length records contain the length field in the first two bytes of the record. When the record is retrieved, the length of the record is inserted into this field by GSAM.

The user requests variable length support by specifying the record format (RECFM) subparameter as RECFM=V or VB in the definition of the data set DBD. A definition of RECFM=V in the DBD can be overridden by specifying RECFM=VB in the JCL.

Undefined-Length Records

Undefined length records are supported only for BSAM data sets. Undefined length records, under O/S BSAM, relieve the user of including a record length at the beginning of a data record. The user records include only data bytes and are returned to the user in that form. However, undefined length records are of variable length. (The number of bytes of data moved cannot be taken from any LRECL constant.) When loading, therefore, the user must specify the record length. When retrieving records, the length of the record retrieved must be returned in this same area. That area is defined as a fullword in the PCB, known as the PCB Length Feedback Area (DBPCBURL). Any length less than or equal to the logical record length, and greater than eleven (by O/S convention) can be loaded to an "undefined" data set. To allow for these undefined records of variable lengths, each block is treated as a record. This is accomplished by specifying RECFM=U.

Records of undefined length have been provided to permit the processing of any records that do not conform to the fixed (F) or variable (V) formation.

Data Set I/O Area

The user area and the information placed on the device are dependent upon whether the data set has fixed or variable length records, and whether there is carriage control information.

User Area

The user's IOAREA (for variable records) is as follows:

Position	Contents
0,1	Halfword length field in fixed binary
2	Carriage control character (specified only if it is a print record or a punch)
2-"n-1"	Data (begins in position 3 if carriage control is specified)

Fixed length, and undefined length records do not include the length field. Data, or control characters, begin in position 0.

The length for undefined length records is passed, in both directions, in the PCB length feedback area as a fullword.

Direct Retrieval by Record Search Argument (RSA)

A Record Search Argument (RSA) accessing facility is supported on all GSAM data bases. This facility allows the user to request particular records via the GU call. The RSA for the particular record desired is supplied by the user by what is normally the SSA address parameter, the fourth in the parameter list.

An RSA parameter is defined under GSAM as two fullwords, on a fullword or doubleword boundary, addressed by the fourth parameter in the CALL parameter list (replacing the standard SSA parameter pointer).

The contents of that doubleword vary according to the access method and device type. The actual contents is irrelevant to the application

program since the program saves and supplies on a GU call whatever had been returned previously by GSAM.

Selective use of RSA can be used to enhance partial retrieval. For instance, at the end of a long string of sequential accessing, retrieval can be resumed in some midpoint of a data base without having to reaccess all preceding records. Any string of such retrievals must be initiated by a GU. GNs can be issued until end-of-data.

The RSA for a particular record can be obtained if the user either loads the data set with the ISRT macro (requesting that the RSA be returned), or issues subsequent GN calls. This argument is returned to the application for each call (provided the RSA pointer exists and is non-zero) for all sequential I/O requests (ISRT and GN).

That argument can then be supplied for a GU request to position the data base at the physical block containing the record, "position" GSAM at the particular logical record, and return that record to the application program.

Subsequent GN calls result in the sequential return of the following logical records, and their RSAs, until end-of-data occurs.

Record Search Argument (RSA)

In VSAM, RBA means the Relative Byte Address (RBA) of the specific record within the data set; it has no bearing on the block number or device position (that is, track number). Since VSAM uses a fixed "blocksize" regardless of the record format the VSAM RBA for a given record in a data set is a constant value which is device-independent. The VSAM RBA is passed in the first word of the IMS/VS GSAM "RSA" parameter.

GSAM provides the flexibility of "direct" accessing by always maintaining the current volume-sequence number.

The BSAM IMS/VS Record Search Argument (RSA) parameter is a doubleword used to locate individual records within a block. The first word contains the SAM RBA, and the second word contains the volume-sequence number in the first halfword and the displacement within the block to the specified record in the second halfword.

Record Search Argument (RSA) Usage

A GU call with the doubleword RSA equal to F'1,0', is interpreted as a request to reposition on a GSAM data base to the first record. OPEN and CLOSE will be issued only if the current accessing is on other than the first volume.

This feature does not apply after end-of-data since that condition causes an immediate CLOSE to be issued. A GN call after end-of-data PCB status code GB obtains the first record on the data set.

RSAs will be returned, or must be supplied, under the following conditions:

- GU — The RSA doubleword must contain the RSA of the record desired. The address of the RSA must be provided by the fourth parameter of the call.
- GN or ISRT — The RSA will be returned if the fourth parameter is provided with a valid address.

BUFFERING

VSAM's high performance is due, in part, to its self-optimizing buffer management and usage of virtual and auxiliary memory. It automatically calculates the optimum sized units in which to store data and the total amount of memory required. It optimizes the use of virtual memory for I/O buffers.

BSAM does not provide such high performance services. Multi-buffered I/O is provided within GSAM for sequential services. Any number of buffers may be requested via the DCB BUFNO parameter. Any direct request (with RSA) will cause only the specified block to be read into one buffer. Subsequent sequential accessing will initiate multi-buffering. Anticipatory READs will be issued to attempt to keep at least half of the buffers full at all times during sequential retrieval.

GSAM dynamically acquires buffers for BSAM data sets when they are opened. An OS OPEN is issued whenever a data base is opened by an IMS/VS access request or OPEN request and an OS CLOSE is issued in response to a user CLSE request or at end-of-data.

For data bases being loaded, GSAM will use the DBD blocksize if the user does not provide blocksize information in the DCB parameter of the DD card for the data set. If the blocksize is given, validity checks are made. If DBDGEN computes the blocksize, the actual length assigned is dependent upon the record format (fixed, variable, blocked, or unblocked) and upon the DBD BLOCKS= blocking factor and LRECL= record length. BLOCKS defaults to 1 or unblocked, with BLKSIZE= LRECL (+4 if Variable).

For loading a DASD SAM file, use of the RSA option will decrease the effectiveness of buffered I/O. This is because every time RSA is requested, a NOTE must be issued to obtain the SAM RBA, if it has not already been done for this block. In an output environment, any WRITEs on the queues must be purged, thereby negating the savings of anticipatory buffering.

Buffered I/O may be specified by: (1) including "S" as one of the PCB processing options or, (2) coding on the JCL DD statement:

DCB= OPTCD=[C / WC] ,BUFNO=m

OPTCD=[C / WC] specifies chained scheduling and is ignored in a V=V region.

Unless specified in the JCL DCB parameter, the number of buffers defaults to twice the number of blocks per track. DCB=(BUFNO=1) overrides the PCB processing specification of 'S'.

CHECKPOINT/RESTART

The IMS/VS extended checkpoint/restart facility allows long-running application programs to be restarted from intermediate synchronization points.

An application program issues a CHKP call to inform IMS/VS that the user has reached a logical synchronization point and that it can be restarted at that point. When a CHKP call is issued, IMS/VS saves certain system information on the IMS/VS recovery log which can be used by the application program to reposition its data bases at restart time.

During a checkpoint operation, RSA information is stored in the system journal. If a subsequent program restart is required, all GSAM data bases in use at that time will be repositioned to their checkpoint locations, without producing either reprocessing of sequential input or redundant sequential output on tape or direct access devices.

A program loading VSAM files cannot be restarted, although checkpoints may be issued to release system resources.

SYSIN and SYSOUT data sets can also be used through GSAM. If output unit-record devices are repositioned, however, duplicate output or separated output is produced. SYSIN data sets are repositioned.

Checkpoint Restrictions

IMS/VS cannot completely determine whether a program is capable of being restarted from any checkpoint. For instance, the program may have non-IMS/VS files or transient data sets which IMS/VS cannot reconstruct. The programmer must be aware of such a condition before issuing a checkpoint.

The following JCL restrictions are recommended:

- Temporary data sets cannot be reset.
- Volume requests for new files must be specific.
- Data set disposition cannot be DELETE or UNCATLG.
- Data sets cannot be used if they have been passed.
- Backward references to data sets in previous steps cannot be used.
- DISP=NEW must be used for all output data sets.

Note: GSAM does not enforce these guidelines explicitly. They should be established as conventions.

JCL

JCL guidelines for initializing a BMP region is very similar except for the inclusion of //IMS...DD statements and GSAM DD statements.

For example:

```
//STEP     EXEC        PGM=DFSRRC00,PARM='BMP,.............'
//STEPLIB  DD          DSN=reslib-name,DISP=SHR
//         DD          DSN=pgmlib-name,DISP=SHR
//IMS      DD          DSN=psblib-name,DISP=SHR
//         DD          DSN=dbdlib-name,DISP=SHR
//SYSPRINT DD          SYSOUT=A
//SYSUDUMP DD          SYSOUT=A
//ddnamex  DD          (add DD statements for required GSAM data bases)
           .
           .
           .
           .
/*
```

IMSBATCH JCL PROC

- Two additional DD statements are required for PSB lookup and GSAM control block building. The DD statements are:

```
//IMS/VS     DD  DSN=IMS/VSVS.PSBLIB,DISP=SHR
//            DD  DSN=IMS/VSVS.DBDLIB,DISP=SHR
```

GSAM data base JCL DCB and RECFM parameters will override the DBD parameters. Thus a DBD indicating RECFM=F,RECORD=80, SIZE=80 may be overridden by JCL...RECFM=FB,DCB=(BLKSIZE=400). Refer to the OS/VS JCL Reference Manual for BSAM and VSAM details.

The GSAM Control Block Dump module will, if an error occurs, provide a formatted dump of the GSAM control blocks on the device specified by the //SYSPRINT or //IMSERR DD card.

Some JCL restrictions are indicated in the sections following on checkpoint-restart.

In BSAM usage, the following DCB parameters can be used.

BLKSIZE to specify block size if it is not in the DBD, or to override the DBD block size

LRECL in the same manner

CODE, DEN, TRTCH, MODE, and STACK

BUFNO and/or OPTCD to invoke BUFFIO, although BUFNO is sufficient

DSORG=PS, although it is unnecessary

PRTSP if RECFM does not include A or M

RECFM, if not in the DBD, using either F, FB, V, VB, or (for BSAM) U

The RECFM parameter can also include A or M (that is, FBA) for unit record out put devices

The following should not be used:

BFALN,BUFL, BUFOFF, FUNC, NCP, or KEYLEN

CHAPTER 3. DATA BASE PROCESSING: ADVANCED FUNCTIONS

This chapter describes the following additional data base capabilities that IMS/VS makes available to application programs:

- Segment Search Argument (SSA) advanced functions
  - Command codes
  - Boolean qualification statements
- Multiple positioning
- Secondary indexing

These optional facilities provide the experienced user with more powerful and sophisticated techniques for organizing and processing data base structures.

The application programmer and data base administrator should jointly evaluate tradeoffs before making a decision to use these features in an application, since the candidate application, other applications, and the overall IMS/VS system may be affected. Multiple positioning will require earlier PSBGEN planning; secondary indexing will require both PSBGEN and DBDGEN planning and implementation, and can have significant performance considerations.

The reader of this chapter is assumed to be familiar with the immediately preceding chapter. Before approaching the topic of secondary indexing, the reader should become acquainted with the "Data Base Design" chapter in the IMS/VS System/Application Design Guide.

SEGMENT SEARCH ARGUMENTS USING ADVANCED FUNCTIONS

In the previous chapter, the basic function of the SSA was defined as identifying a specific data base segment called by an application program. The rules pertaining to the use of SSAs by each DL/I functional call are enumerated in that discussion.

Frequently, however, an application wishes to retrieve a segment based on some conditional retrieval logic or on some qualification of the segment, or on some variation in the call function.

To accommodate this requirement and to remove the need for incorporating such conditional logic in the application program, IMS/VS provides the fully expanded SSA capability described below.

The SSA can consist of from one to three main elements: a segment name, command code(s), and a Boolean qualification statement. The segment name alone provides DL/I with enough information to define simply the segment type desired by the program, thus the segment name may itself be the total SSA, as described in the previous chapter. In its complete form, the SSA may be augmented by command codes and/or a set of field qualifications logically related by Boolean logic elements.

The command codes are optional and provide specification of functional variations applicable to either the call function, the segment qualification, or the setting of parentage.

The qualification statement is also optional and contains information which DL/I uses to test the value of the segment's key or data fields within the data base to determine whether the segment meets the user's

specifications. Using this approach, DL/I performs the data base segment searching and the program need process only those segments in which it is interested.

Each qualification statement is composed of three parts: a field name, a relational-operator, and a comparative-value. Boolean qualification may be performed by connecting qualification statements together with the AND and OR Boolean operators. The complete set of qualifications for each segment is contained between the left and right parentheses. In a segment search argument, there may be a maximum of eight qualification statements connected by Boolean operators.

The SSA structure is shown in Figure 3-1:

		(optional)		(optional)												
elements	Segment Name	Com-mand Codes		BOOLEAN STATEMENT →												
				Begin Qualif.	Qualification Statement #1			Operator	Qualification Statement #2			Operator	Qualification Statement #n			End Qualif.
contents	Name of Segment-type	*	Code Char-acters	'('	Field Name	R.O.	Compar. Value	'#' '*' or '+'	Field Name	R.O.	Comp. Val.	'#' '*' or '+'	Field Name	R.O.	Compar. Value	')'
no. of bytes	8	1	VBL.	1	8	2	1 to 255	1	8	2		1	8	2	1 to 255	1

Figure 3-1. SSA Structure

SEGMENT NAME

The segment name must be left-justified in the field and padded on the right with blanks to make eight bytes. It is the segment name that pertains to a specific segment type in the hierarchical structure of a data base record and which is established in the Data Base Description.

COMMAND CODES

The command codes are optional. They provide functional variations to be applied to the CALL for that segment type. An asterisk (*) following the segment name indicates the presence of one or more command codes. A blank or a left parenthesis is the ending delimiter for command codes. The functions of the command codes are documented later in this chapter.

BEGIN QUALIFICATION CHARACTER

The left parenthesis, '(' , indicates the beginning of a qualification statement. Any character other than a '(' implies an unqualified SSA. If the SSA is unqualified, the eight-byte segment name, or, if used, the command codes must be followed by a blank.

QUALIFICATION STATEMENT

The presence of a qualification statement is indicated by a left parenthesis following either the segment name or, if present, command codes. Each qualification statement consists of a field name, a relational operator, and a comparative value.

Field Name
is the name of a segment search field which appears in the description of that segment type in the Data Base Description. The name must be left-justified in the 8-character field and padded on the right with blanks. The named field can be either the key field or a data field within a segment.

RO = Relational Operator
is a set of two characters which express the manner in which the contents of the field, referred to by the field name, are to be tested against the comparative-value. The choice of relational operator does not affect the starting point of the search or the order of search.

Operator	Meaning
b= or EQ	must be equal to
>= or GE	must be greater than or equal to
<= or LE	must be less than or equal to
b> or GT	must be greater than
b< or LT	must be less than
¬= or NE	must be not equal to

Note: As used above, the lowercase "b" represents a blank character. The symbols in the non-alphabetic relational operators can be reversed without changing the meaning (that is, "GE" is equivalent to ">=" or "=>").

Comparative-value
is the value against which the contents of the field, referred to by the field name, is to be tested. The length of this field must be equal to the length of the named field in the segment of the data base, that is, it includes leading or trailing blanks (for alphameric) or zeros (usually needed for numeric fields) as required.

END QUALIFICATION CHARACTER OR BOOLEAN OPERATOR
Following the comparative-value is either a Boolean operator, relating this qualification statement to the next qualification statement, or a right parenthesis as the ending delimiter indicating the last qualification statement for this segment. The Boolean operators are documented later in this chapter.

The qualification statement test is terminated either when an occurrence of the requested segment type is found, or when it is determined that the request cannot be satisfied.

Examples of SSAs with the DL/I calls are contained in the previous chapter.

GENERAL CHARACTERISTICS OF SEGMENT SEARCH ARGUMENTS

- An SSA may consist of the segment name only (unqualified). It may optionally also include one or more command codes (unqualified) and/or a qualification statement for that segment (qualified).
- SSAs following the first SSA must proceed down a hierarchical path. All SSAs in the hierarchical path need not be specified; that is, there may be missing levels in the path. DL/I will provide, internally, SSAs for missing levels according to the rules specified in the section on each functional call in the previous chapter.

- A search field specified as a "field name" in an SSA must be defined for the segment during DBD generation.

- Any of the valid relational operators may be specified. All comparisons on key or data fields are logical bit-for-bit comparisons.

Note: More specific SSA statements which apply to a specific function such as GU or ISRT are provided in the discussion unique to that function in the previous chapter of this manual.

COMMAND CODES

Command codes can be divided into three categories: those which modify the call function, the segment qualification, and the setting of parentage.

Command
Code Meaning

Call Function

F Start with the first occurrence of this segment type under its parent in attempting to satisfy this level of the call. It is possible to either back up to the first occurrence of the segment type on which position is established or to back up to the first occurrence of a segment defined earlier in the hierarchy than the one on which position is established.

For GN type calls, this command allows backing up at this level within a data base record. This command applies only to GN type calls, since GU calls operate this way, normally.

For ISRT calls, this command says that segments having non-unique or no sequence fields and RULE=(,HERE) are to be inserted as the first segment on the twin chain.

The F command code used at the root level is disregarded.

L Retrieval:

Retrieve the last occurrence of this segment type under its parent which satisfies the qualification statement; or, if unqualified, retrieve the last occurrence of this segment type under its parent.

Insert:

Only applies for segments with non-unique or no sequence field. Otherwise the key field in the segment determines the insert position.

Used to insert "last" segment in "twin" chain for segments defined with non-unique or no key fields and RULE=(,HERE).

For example, suppose a data base has an insert rule of HERE, and that "HERE" happens to be just past the last segment in a twin chain. Suppose that at that point the application program wishes to insert a segment with no key field defined at the end of the twin chain. Without the L command code, DL/I would position itself on the following segment type, recognize that it could not insert HERE because it is the wrong segment type, and default to the first segment occurrence of the desired

Command
Code Meaning

segment type. Using the L command code on the segment being inserted allows DL/I to position for insert such that the segment is inserted following all other segment occurrences.

If the L command code is used at the root level, it is disregarded.

The interaction of insert rules with the F and L command codes is summarized in the following table.

Command Codes	Insert Rules		
	FIRST	HERE	LAST
F	(N.A.)	CC overrides	Rule(**) overrides
L	CC overrides	CC overrides	(N.A.)

(**) This combination would be poor programming practice since it forces extra processing by telling DL/I, in effect, to start at the first and insert at the last.

D For retrieval calls, move the segment which satisfies this level of the call to the user's I/O area. This allows the retrieval of multiple segments in a hierarchical path in a single call. This type of call will subsequently be referred to as a path call. The first through the last segment retrieved are concatenated in the user's I/O area. Intermediate SSAs may be present without the D command code. If they are present, these segments are not moved to the user's I/O area. The segment name in the PCB is the lowest level segment retrieved, or the last level satisfied in the call in case of a not-found condition. Higher level segments having the D command code will have been placed in the user's I/O area even in the not-found case. The "D" is not necessary for the last SSA in the call, since the segment which satisfies the last level is always moved to the user's I/O area. The processing option of "P" must be specified in PSBGEN for any segment for which the D command code will be used in the PSB associated with the application program. It should be noted that the retrieval search logic is not affected by the D command code. The only effect is to move all segments with the D command code into the I/O work area.

For insert calls, the D command code designates the first segment type in the path to insert. The SSAs for lower level segments in the path need not have the D command code set.

N When a replace call follows a path retrieval call, it is assumed that all segments in the path are being replaced. If any of the segments have not been changed, and, therefore, need not be replaced, the N command code may be set at those levels, telling DL/I not to attempt to replace the segment at this level of the path.

Q The Q command code causes DL/I to enqueue the segment described by the SSA for single update. If the segment is a root segment, no other user will be able to obtain any position in the data base record. If it is a dependent segment, other users can retrieve the segment with a non-hold call, but it cannot be obtained using a hold call.

Command
Code Meaning

The purpose of the Q command code is to provide a facility for users to cause segments to be enqueued and also control the duration of the enqueue. One case where this could be useful is when the application needs to examine a number of segments and none of them may change while the others are being examined. The application can obtain the segments using the Q command code and then retrieve them again with the assurance that none of them can be modified until the application issues a DEQ call or reaches a sync point.

If no DEQ is issued by the application program, the enqueued segments will be dequeued when a synchronization point is reached. A synchronization point is reached when any one of the following occurs:

1. A GU to the message queue is issued and the scheduled transaction MODE=SNGL.
2. A CHKP call is issued.
3. The application program terminates.

The DEQ call is described in the previous chapter.

In order to provide a degree of flexibility in selectively dequeueing the enqueued resource, the Q command code must be followed by a single byte in the range of "A" through "J" which specifies the class. This same class identifier is specified on the dequeue call which dequeues all resources enqueued by this user using the Q command code and that class. The sole usage of the class identifier is for selective dequeue; it does not allow one user to obtain a resource of a particular class and a different user to obtain the same resource of a different class.

Note: By definition the Q command is always followed by a one character class. This means that the second byte after the 'Q' command code must be another command code, left parenthesis, or blank.

Segment Qualification

C The data enclosed in parentheses immediately following the command code is the concatenated key of the named segment (for example, '*C' (concatenated key)). Qualification to this level is treated identically to a call specifying all SSAs of all parents of the named segment qualified on their respective sequence fields.

Using this command code may be more convenient than using separate SSAs, when the concatenated key is available and can be used as is, rather than moving each portion of the concatenated key into a separate SSA.

Only one SSA with a C command code is allowed per call and it must be the first SSA in the call.

Command
Code Meaning

U The U command code indicates that no occurrence of the segment type specified in the SSA (other than the segment type upon which position is already established) under the parent of the segment type will be used to satisfy the call. If position is not currently established for the named parent, this code has no effect.

The U code prevents position being moved from a segment during a search of its hierarchical dependents. If the segment has a unique sequence field, use of this code is equivalent to qualifying the SSA such that it is equal to the current value of the key field. When a call is being satisfied, if position is moved to a level above that at which the U code is issued the code has no effect for the segment type whose parent changed position.

The U code is especially useful when dependents that are unkeyed or non-unique keyed segments are being processed. The position on a specific occurrence of an unkeyed or non-unique keyed segment can only be held by use of this code.

The U command code is disregarded if it is used at the lowest level or if the SSA is qualified, or if used in conjunction with command code F or L.

V The V command code is the same as the U command code, except that the command code is automatically set at all higher levels in the call. This means that DL/I, in attempting to satisfy this call, cannot move from the existing position at the level at which the V is specified, unless the command code is disregarded. See the U command code for the condition under which it will be disregarded.

Setting of Parentage

P Set parentage at this level. Succeeding GNP-type calls will treat this level as the parent level rather than the lowest level segment returned on this call. The parentage will remain in effect for succeeding GNP, ISRT, DLET, and REPL calls. The parentage will be destroyed whenever a GU or GN call is executed.

If the P command code is used at multiple levels in the same call, the lowest level is set.

If the call is not fully satisfied (GE status code) but the level at which the P code is used is satisfied, parentage is set by the P command code.

If the call is not fully satisfied and the level at which the P code is used is not satisfied, parentage is not established and a GP status will be returned on succeeding GNP calls.

If the P command code is specified on a GNP call, the call is processed based on the parentage that was in effect or established by preceding calls. The parentage is set based on the P command code that was used at the completion of the GNP call.

In addition to the codes in the three categories above, there is also a null (hyphen "-") command code. Its purpose is to simplify the building of SSAs using command codes since the program can set aside a fixed number of bytes for command codes and turn them on and off by means of the hyphen.

The following table indicates which command codes are applicable to which functions. If the command code is used with a function where it is not applicable, the command code has no effect.

Use of Command Codes by Function

Command Code	GU GHU	GN GHN	GNP GHNP	DLET	REPL	ISRT
C	A	A	A	D	D	A
D	A	A	A	D	D	A
F	A	A	A	D	D	A
L	A	A	A	D	D	A
N	D	D	D	D	A	D
P	A	A	A	D	D	D
Q	A	A	A	D	D	A
U	A	A	A	D	D	A
V	A	A	A	D	D	A

A = Applicable
D = Disregarded

No combinations of command codes are declared invalid by returning an error status code. However, when F or L is used in conjunction with U or V, the U or V is disregarded.

BOOLEAN QUALIFICATION STATEMENTS

Boolean logic qualifications can be performed on each segment by specifying up to eight qualification statements for each segment. The qualification statements can be logically related to each other by using the Boolean AND and OR operators between the qualification statements.

All Boolean statements connected by AND operators are considered a "set" of qualification statements. An OR operator between two qualification statements begins a new set of qualification statements. A set can consist of one or more statements. To satisfy an SSA, a segment can satisfy any set of qualification statements. To satisfy any set, the segment must satisfy all statements within the set.

If a GU call for a root segment has one or more Boolean qualification statements, and if any set of qualification statements does not contain at least one statement qualified on the key field of the root segment, then the initial position that will be used in attempting to satisfy the call will be the beginning of the data base.

If all sets have at least one statement qualified on the key field of the root segment, then the lowest key field value will be the initial position used in attempting to satisfy the call.

The qualification scan will be made sequentially in a forward direction similar to a GN call. Each root encountered will be examined to see if the search can continue. It should be noted that in an HDAM data base the roots are not stored in key sequence and therefore using Boolean statements for root qualification may not produce the desired results.

Example:

```
SEGMENTA(FIELDAAAb>099*FIELDAAAb<201+FIELDBBBb=0)
```

For the above SSA, those segments called SEGMENTA whose FIELDAAA is in the range between 100 and 200, or whose FIELDBBB is equal to zero, will satisfy the SSA.

The logical "And" is expressed by the EBCDIC character "*" or "&". The logical "Or" is expressed by the EBCDIC "+" or "|". A special "independent" AND operator, expressed by the "#" character is described later in this chapter in the section concerning secondary indexing.

USE OF FIELD NAMES IN SEGMENT SEARCH ARGUMENTS FOR CONCATENATED SEGMENTS

The field names used in the qualification statement of SSAs for concatenated segments may be fields defined for either of the two segments making up the concatenated segment. In other words, if the concatenated segment consists of the logical child and the logical parent, then fields defined for either of these two segments may be used. If the sequence field of the logical parent is used, however, it is treated as a data field, not as a sequence field. An example illustrates why this must be so. Suppose that DL/I is positioned at the beginning of the chain of logical children illustrated in Figure 3-2. If the application program issues a call:

```
GN      FLC=4
```

DL/I gets, sequentially, the first, second, and third logical child; recognizes that the sequence field of 5 is greater than 4, and stops.

If, however, the application program issues a call:

```
GN      FLP=8
```

DL/I will traverse the entire chain of logical children because the logical parents are not examined sequentially by their sequence fields. DL/I cannot assume, in this instance, for example, that because the first logical child pointed to a logical parent whose sequence field is 40 that it has passed (or that there does not exist) a logical parent with a sequence field of 8. Hence, DL/I must treat logical parent sequence fields as if they were data fields.

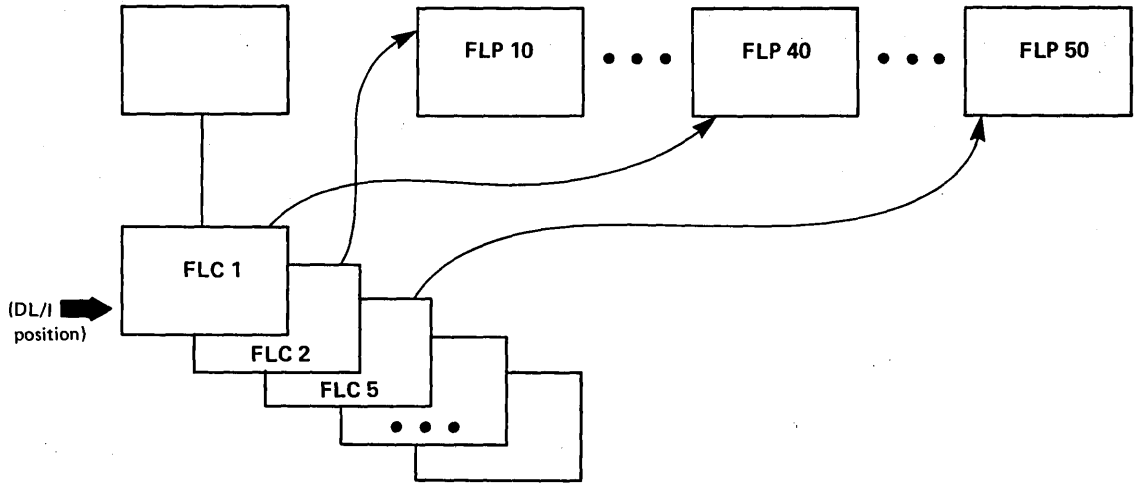


Figure 3-2. Effect of Using Logical-Parent Sequence Fields

If the concatenated segment consists of the virtual segment and its logical parent, then the fields used may be any field defined for the virtual segment or any field defined for the logical parent. Fields defined for the pair of the virtual segment may be used also, as long as no part of the field falls within the part of the segment which is the concatenated key of the paired segment's logical parent.

When the concatenation is the virtual segment and its logical parent, the only field which is treated as a sequence field is the sequence field defined for the virtual segment.

The data base administration function should make available logical data structure definitions and segment layouts. The segment layout should indicate the field names for the sequence field and other searchable fields. Thus it would be immaterial to the application programmer whether or not the segment is a concatenation of two physical segments.

MULTIPLE POSITIONING

Two alternatives are provided by DL/I regarding the current position in the data base. These are single or multiple positioning. This option is specified in the PCB statement at PSB generation.

- When single positioning is specified for a PCB, DL/I maintains only one position in that data base for that PCB. This is the position which will be used in attempting to satisfy all subsequent GN calls.
- If multiple positioning is specified, DL/I will maintain a unique position in each hierarchical path in the data base.

An example of how multiple positioning might be used is illustrated in Figure 3-3.

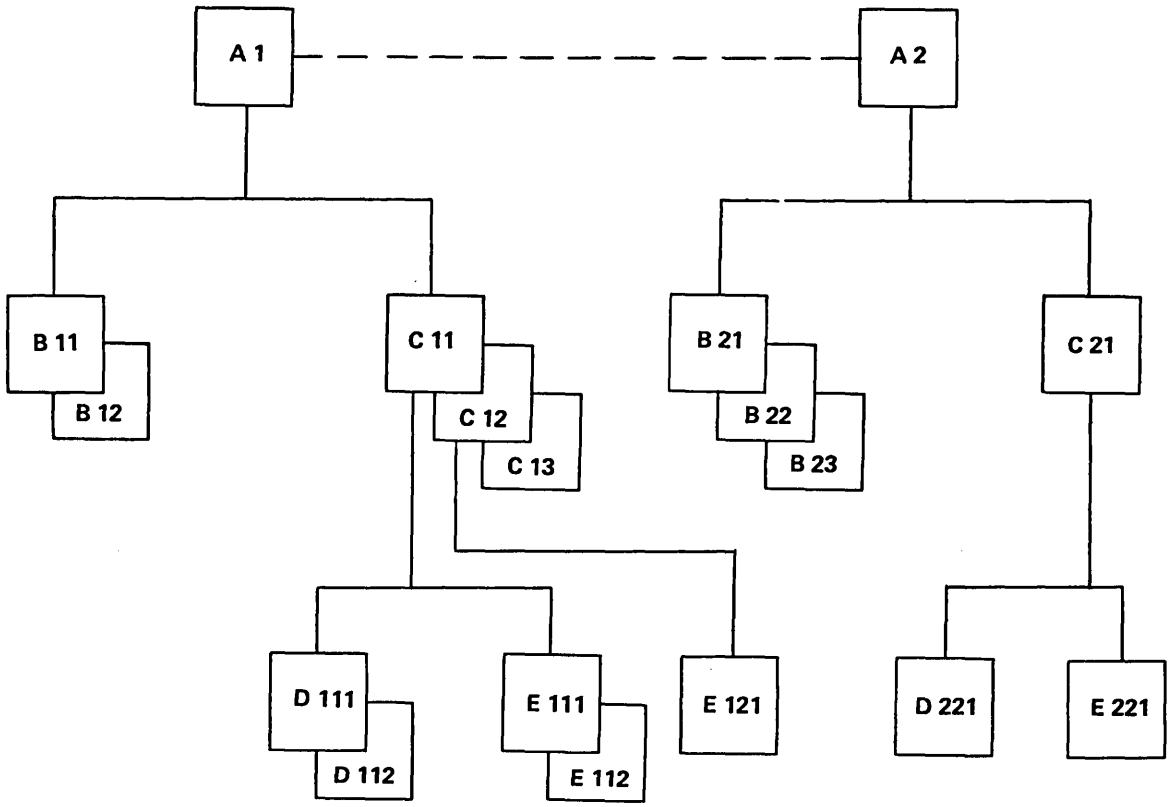


Figure 3-3. Assumed Data Base to Illustrate Single and Multiple Positioning

In Figure 3-3, assume that under each A segment an application program desires to examine every C segment based on each B segment. Using multiple positioning, the following sequence of calls would suffice:

CALL		Result
GN	A	get A1
GNP	B	get B11
GNP	C	get C11
GNP	C	get C12
GNP	C	get C13
GNP	C	get not found
GNP	B	get B12
GNP	C * F	get C11
GNP	C	get C12
etc.		

As can be seen from the example above, multiple positioning provides a capability of processing, in parallel, different segment types under the same parent.

EFFECT OF MULTIPLE POSITIONING ON DL/I CALL FUNCTIONS

GN and GNP Calls Using Multiple Positioning

IMS/VS attempts to satisfy GN calls from the existing position by analyzing segments in a forward direction only. Since multiple positioning allows position to be maintained at each level in all hierarchical paths rather than at each level in only one hierarchical path, the get next call will be satisfied using the existing position established on the path of the hierarchy in which the get next call is qualified. If the get next call is not qualified, IMS/VS will use the position established by the prior call. The position can be reset by a GU call to a new root or to the same root; position cannot be reset by a path call under the previously accessed root-segment occurence.

GU and ISRT Calls Using Multiple Positioning

The only time multiple positioning has an effect on GU and ISRT calls is when these calls have missing SSAs in the hierarchical path. These missing levels are internally completed by the system according to the rules for GET calls described earlier in this chapter.

Since this internal completion is based on current position, multiple positioning allows a completion to be made independent of current positions established for other segment types under the same parent occurrence.

DLET and REPL Calls Using Multiple Positioning

These calls are not affected by single or multiple positioning. However the necessary preceeding GET HOLD calls are as described previously.

EXAMPLES OF CALL SEQUENCES USING SINGLE AND MULTIPLE POSITIONING

The following examples compare the results of single and multiple positioning, using the data base of Figure 3-3.

Call Sequence	Result of Single Positioning	Result of Multiple Positioning
Example 1		
GU A(KEY=A1)	A1	A1
GNP B	B11	B11
GNP C	C11	C11
GNP B	Not found	B12
GNP C	C12	C12
GNP B	Not found	Not found

Call Sequence	Result of Single Positioning	Result of Multiple Positioning
GNP C	C13	C13
GNP B	Not found	Not found
GNP C	Not found	Not found

Note: Segment types B and C are processed in parallel.

Example 2

GU A(KEY=A1)	A1	A1
GN B	B11	B11
GN C	C11	C11
GN B	B21	B12
GN C	C21	C12

Example 3

GU A(KEY=A1)	A1	A1
GN C	C11	C11
GN B	B21	B11
GN B	B22	B12
GN C	C21	C12

Example 4

GU A(KEY=A1)	A1	A1
GN B	B11	B11
GN C	C11	C11
GN D	D111	D111
GN E	E111	E111
GN B	B21	B12
GN D	D221	D112
GN C	C under next A	C12
GN E	E under next A	E121

USE OF MULTIPLE POSITIONING

By specifying multiple positioning, a user may be able to design application programs with greater data independence. Multiple positioning also makes it possible to achieve parallel proceesing of dependent segment types.

Increased Data Independence

Multiple positioning allows a user to develop application programs using GN and GNP calls and ISRT and GU calls with missing levels in a manner independent of the relative order of segment types defined at the same level in the logical DB structure.

Hence, if performance could be improved by changing the relative order of segment types, and all application programs which access those segment types use multiple positioning, then the change could be made with no impact on previously produced application programs. It should be noted, however, that this ability depends on the proper use of the calls relevant to multiple positioning (GN, GNP and incompletely specified ISRT and GU calls). It also presents an increased responsibility for the application programmer to keep track of all positions maintained by DL/I. There are other alternatives to decrease an application program's exposure to future changes as for instance increased use of explicitly given call specifications when possible. These alternatives may require additional application program coding. Such trade-offs must be determined in the user's own environment.

Parallel Processing of Dependent Segment Types

When an application program needs to process dependent segment occurrences in parallel (that is, to switch alternately from one dependent segment type to another under a parent), the program may specify multiple positioning to accomplish such processing. An alternative parallel processing technique would be to give the program two or more PCBs using the same data base. Under this alternative, the program processes the data base as though it were two or more different data bases. This approach may be more useful if the update of a segment depends on the analysis of other subsequent segments. The use of multiple PCBs may decrease the number of get hold calls required but increase the number of other calls required to maintain proper positioning in two or more data base structures. Internal control block requirements will also increase with each added PCB. However, there are circumstances when the use of multiple PCBs for a single data base will increase performance. Multiple PCBs may be of particular value when an application desires to compare information in many segments of two or more data base records. The selection of multiple positioning or multiple PCBs for a single data base must be evaluated in the user's environment.

It should be emphasized that multiple positioning uses position differently from single positioning. If an application program changes from one option to another, the user must not assume the same results will be produced. An application program must be developed for one alternative or the other.

MIXING CALLS WITH AND WITHOUT SEGMENT SEARCH ARGUMENTS AND MULTIPLE POSITIONING

The multiple positioning feature is intended to be used for DL/I requests which specify SSAs, thereby providing for parallel processing and increased data independence. Retrieval calls without SSAs can also be used, however, when multiple positioning is specified to accomplish a sequential retrieval of segment occurrances independent of segment types.

Certain restrictions apply if retrieval calls without SSAs are mixed with DL/I requests that specify SSAs in processing a single logical data base record.

1. No position may previously have been established on segment types which retrieval calls without SSA specifications may encounter within the processing of that logical data base record.

   Example (using Figure 3-3)

CALL	Result (with multiple positioning)
GU A(KEY=A1)	gets A1
GN C	gets C11
GN B	gets B11
GN B	gets B12
GN	Unpredictable

   The GN calls may not attempt to retrieve occurrences of the C segment type because a position has already been established on this segment type using the multiple positioning feature. The result of the call is unpredictable.

2. When segment types have previously been processed with retrieval calls not specifying SSAs, a position is established on the last retrieved segment type and its parent (hierarchical path). Multiple positions are no longer maintained.

CALL	Result (with multiple positioning)
GU A(KEY=A1)	gets A1
GN C	gets C11
GN B	gets B11
GN C	gets C12
GN	gets E121
GN B	unpredictable

   Multiple positions on B are no longer maintained. The result of the GN B call is unpredictable.

It should be noted that although mixed use of retrieved calls with and without SSAs in processing a single logical DB record may be valid for some types of parallel processing, it may decrease the degree of data independence created by the use of multiple positioning. The implications of the two restrictions stated above should be carefully considered before application programming is based upon mixed use of retrieval with and without SSAs within a single DB record. If possible retrieval calls without SSAs should be limited to GNP calls to avoid potentially inconsistent retrieval situations.

SUMMARY

The essential difference is that with multiple positioning, position can be maintained on different segment types under the same parent, while with single positioning a single position is maintained for different segment types under the same parent. The difference in the internal operation of DL/I is as follows.

With single positioning, whenever a segment is obtained, position for all dependent segments and all segments on the same level is cleared. With multiple positioning, whenever a segment is obtained, position for all dependent segments is cleared but position for segments at the same level is maintained. The blocks in either case are the same (multiple positioning does not require more storage). There is no significant performance difference, even though in some cases multiple positioning will require slightly more CPU time. Multiple positioning is not supported for the HSAM Access Method.

The use of multiple PCBs by an application program to process a single data base essentially allows the multiple positioning concept within a data base record to be expanded to multiple data base records. Thus, an application can process segments from one data base record in parallel with segments from one or more other data base records. DL/I can maintain a position on segments within a data base record with each PCB. Since DL/I can maintain position in multiple data base records, increased performance may be obtained in certain circumstances relative to a single PCB for all accesses to the data base.

SECONDARY INDEXING

One of the features of IMS/VS is a secondary indexing facility. This facility is a data base structuring technique which ordinarily would concern only the data base administrator of an IMS/VS installation and be transparent to the majority of the application programmers.

However, in those installations which employ secondary indexing, two factors make it desirable that the experienced application programmers have some familiarity with the secondary indexing facility. First, secondary indexes are used to establish alternate entries to physical or logical data bases for application programs. The existence of a secondary index on a segment can affect the manner in which DL/I processes the SSAs for that segment. Second, secondary indexes can be processed as data bases themselves.

A complete discussion of secondary indexing can be found in the IMS/VS System/Application Design Guide which addresses the IMS/VS access methods and the design and implementation (as opposed to processing) aspects of data base structures. This is the necessary context for a discussion of secondary indexing, and the application analyst or programmer who is interested in this facility is referred to that document for adequate familiarization. The discussion which follows simply summarizes the characteristics of secondary indexing and describes the effect of secondary indexing on data base processing.

The application analyst or programmer who has read the IMS/VS System/Application Design Guide or attended a formal IMS/VS education course is aware that physical data bases are organized in either hierarchical sequential or hierarchical direct organization and employ one of the four access methods called HSAM, HISAM, HIDAM, or HDAM. These are considered basic access methods. Physical data bases can be connected to form logical data bases, the "access method" of which would be an appropriate combination of these basic access methods.

An INDEX data base is an auxiliary data base used to locate data in an HISAM, HDAM, or HIDAM type of data base. HIDAM always has one INDEX data base which is called a primary index and which indexes only on the sequence field of the root segment. All other Indexes are secondary indexes, and they may index segment types at any level of the data base structure including root segments. HSAM and INDEX data bases cannot be indexed.

Logical data bases can have secondary indexes, that is, secondary indexes existing for a physical data base that participates in a logical relationship often can be used when accessing the logical data base.

Unlike primary indexes as used with HISAM and HIDAM, secondary indexes can:

- Index any field or combination of fields (not necessarily contiguous) in a segment of a HIDAM, HDAM, or HISAM data base at any level
- Index non-unique data, which means that different occurrences of a segment type with identical values in the indexed fields are allowed
- Be processed as data bases themselves, in addition to serving as alternative access paths to a data base
- Carry, in addition to the indexed data and pointers, other source data which are system-maintained replications of data from the indexed data base
- Include user-maintained data in addition to the system-maintained data
- Be created as sparse indexes through system provided means to suppress creation of an index entry for certain data base records by allowing user options and/or exits

A secondary index can be used:

- To sequentially process all or a part of a data base in an order which is different from its primary processing sequence
- To sequentially process a data base as if its structure had been inverted, that is, the data base appears to be a differently structured data base
- To randomly retrieve and process single segments faster than with the primary addressing scheme, if the secondary index provides a unique identification of the requested segment
- As a data base itself in order to do scan-type processing in the index rather than in the indexed data base.
- To access a segment in a data base based on data located in one of its dependent segments in the same physical data base

If several indexes exist for a segment type, it may be possible to use indexes in a preparatory step as data bases themselves in order to merge or match index entries before access to the indexed data base is attempted. Time consuming accesses to not-qualifying segments could thus be avoided.

INDEXED SEGMENTS -- INDEXED FIELDS

To provide an adequate basis for describing and discussing the concepts of secondary indexing, a specific nomenclature has been adopted. This nomenclature distinguishes between the segment type being indexed, the segment type used to access the indexed segment, and the segment type containing the indexed fields. These three segmen types are called, respectively:

- index target segment
- index pointer segment
- index source segment

Index Target Segment

A segment type being indexed is called the index target segment if it is 'pointed to' by the index pointer segment. In a secondary index, the indexed data (indexed field) can be contained in the index target segment, or it can come from any segment hierarchically below the index target segment in the same physical data base.

Index Pointer Segment

The segment in the secondary index data base which is used to access the index target segment is called the index pointer segment. It is composed of up to four classes of system maintained data: constant, search, subsequence, and duplicate data. Of the four, only search data is required for index pointer segments; the other three are optional. In addition, there is a direct or symbolic pointer to the index target segment. A more comprehensive description of each of these field classes is contained in the IMS/VS System/Application Design Guide,

Index Source Segment (ISS)

The segment type which contains the indexed field or fields is called an index source segment (ISS). In secondary indexing, as contrasted with primary indexing, the ISS can be the index target segment itself or it can be any one of the target segment's dependent segments.

There is only one index source segment type for each index relationship. If a combination of several fields is to be used to form the search data of an index pointer segment, all these fields must be contained in the same index source segment.

Whenever a segment type is being updated which has been designated as an index source segment, all indexes with which it is associated as an ISS must be available to IMS, whether sensitive or not sensitive.

An example is shown in Figure 3-4, where the index data base X1 indexes a segment on the second level based on data from the third level. SEGMB is the index target segment; SEGMC is the index source segment. For the index data base X2, SEGME is both index target segment and index source segment. Secondary index X1 is based on the two non-contiguous fields FLDC2 and FLDC1. Note that there are two non-unique index entries for the same SEGMB because two of its SEGMC children have the same indexed data.

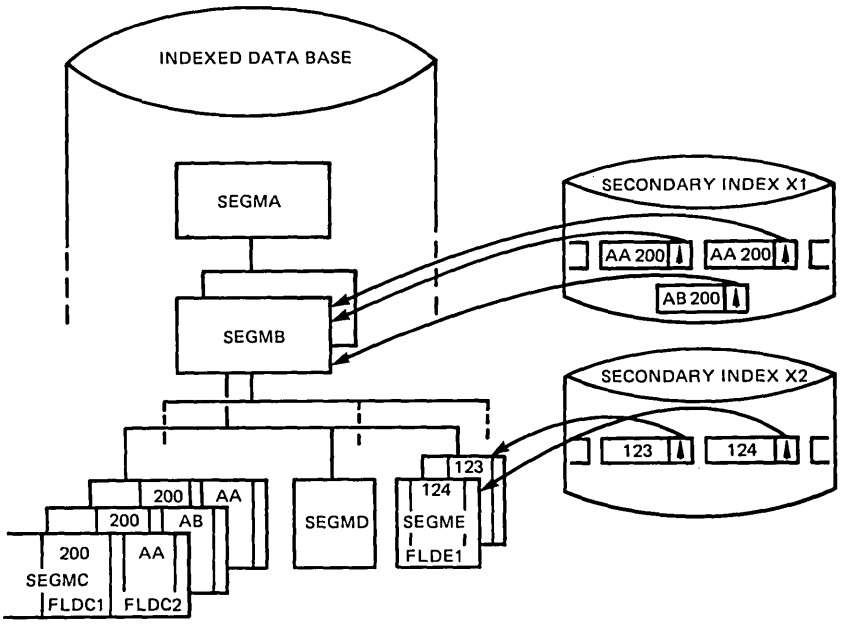


Figure 3-4. Indexing a Data Base with Secondary Indexes

SECONDARY PROCESSING SEQUENCES

A secondary processing sequence is an alternate sequence that is normally not based on a root key. This alternate sequence allows data access in a sequence that is not related to the primary sequence field. The alternate processing sequence can be based on a field or fields in a root segment or a dependent segment. For those secondary indexes which have a dependent segment as their index target segment, the index relationship introduces a new data base processing capability. It provides the possibility of processing the indexed data base as if it were a differently structured data base.

If a secondary index is chosen as the main addressing algorithm for a data base, that is, if a secondary index determines the processing sequence for that data base, then the data base is treated as if its structure had been changed to a secondary data structure.

The secondary data structure, a definition of which follows, is established by the PSBGEN and needs no additional specification by the user. It offers some of the advantages provided by the logical data base concept without requiring the logical DBDGEN and prefix resolution necessary for the creation of logical data bases. In addition, it offers the advantage of processing a structure in which the logical root segment is not a root in any physical structure.

Secondary Data Base Structure Made Possible by Secondary Indexes

If a secondary index is selected to determine the processing sequence of a data base, then the data base appears to have a structure with the following characteristics:

1. The index target segment is the root segment of the secondary structure.

2. Parents, if any, of the index target segment in the physical structure become dependents in reverse order in the secondary structure.

3. The segment which was hierarchically first below the index target segment, if any, becomes the next second level segment.

4. Hierarchical relations existent among the index target segment and its dependents, if any, are taken over into the secondary structure without change.

5. No other segments occur in the secondary structure.

6. The root of a secondary structure may not be a logical child segment, nor may it be a concatenated segment.

Figure 3-5 illustrates these rules. It shows the secondary structures of the data base from Figure 3-4, as indexed by indexes X1 and X2. Note that the structure caused by X2 no longer contains segments SEGMC and SEGMD, since the index target segment in this case, SEGME, was not a dependent of those, nor were they dependents of SEGME.

Note that the rules for secondary structures, when applied to an indexed root, do not change the structure of the data base.

The user specifies in his program specification block generation which, if any, secondary index is to be used as processing sequence for his data base; he also defines the structure in which the data base appears to the application program. For every indexing relationship, the index target segment and index source segment have been, of course, previously determined when the DBD parameters were specified.

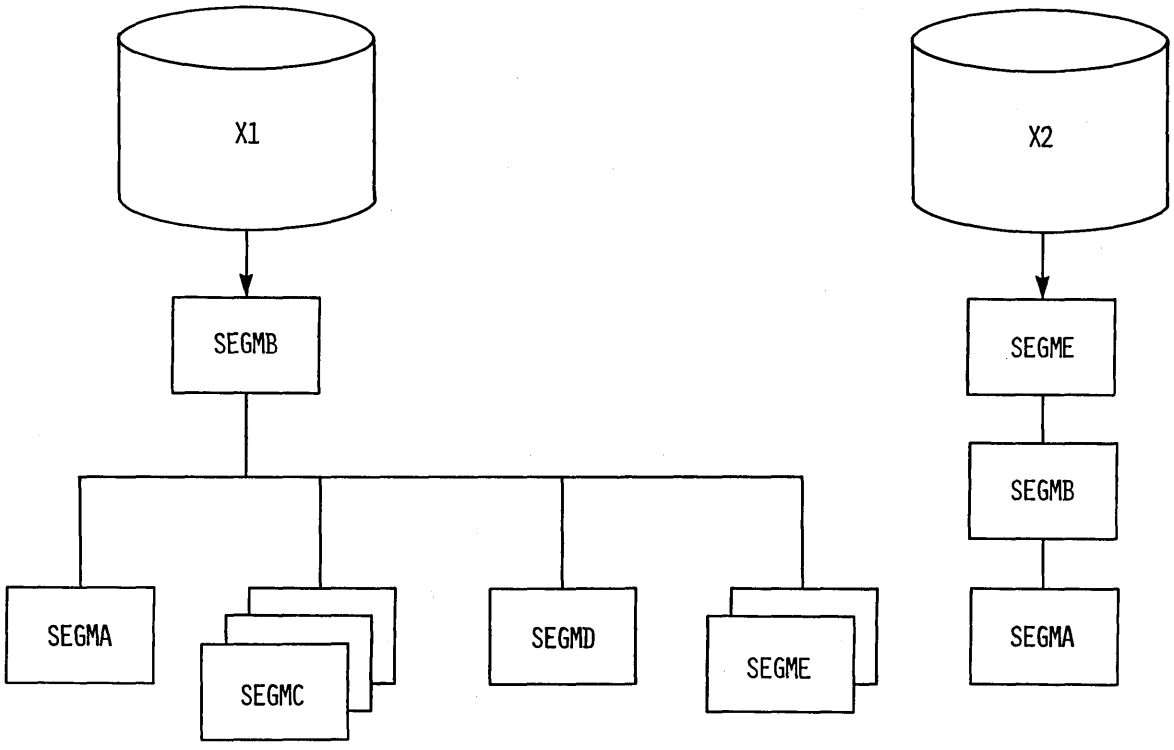


Figure 3-5. Secondary Structures by Secondary Indexes

Some restrictions exist when processing a data base in its secondary structure through the secondary index. These restrictions are:

1. No attempt must be made to insert or delete occurrences of segments of the index target segment's type or of the segment types of which the index target segment was dependent in the original structure.

2. Any data fields, except the sequence key fields but including all fields designated as source fields for secondary indexes, can be changed. The replacement of any ISS associated with the index being used as the secondary processing sequence may result in an anomaly in processing. If the search fields in the index source segment are changed, then the index will be updated to reflect that change and the user processing sequentially could encounter the index entry for the associated ISS again with a different search key value. (The search and other classes of fields are described in the IMS/VS Utilities Reference Manual.)

Options and Rules for Secondary Indexing

A secondary index can be defined using:

- Up to five non-contiguous fields of unique or non-unique data in the index source segment type as the search field of a secondary index

- Up to five non-contiguous fields from the index source segment or from system-related data as the subsequence field of the index pointer segment

To enhance the usefulness of processing a secondary index as a data base:

- The user can specify that up to five fields of the index source segments be duplicated in the index pointer segment generated from each index source segment.

- Index pointer segments can contain any additional user data desired.

- Protection of system-maintained data from modification is an option.

A secondary index can be used to:

- Access only significant or representative segments through sparse indexing by using an option and/or exit provided to enable suppressing the creation of index entries for desired index source segments;

- Access segment types in a single hierarchic path of a data base using the index target segment type as the root for all segment types in that path without having to use logical relationships;

- Selectively access a given segment, through data contained in that segment or a dependent of that segment;

- Directly access a non-root segment type in an HDAM or HIDAM data base in less time than is normally required through the primary accessing method.

Following are the rules that must be observed in secondary indexing:

1. In a physical data base, a logical child, or a dependent of a logical child cannot be an index target segment type.

2. A concatenated segment type, or a dependent of a concatenated segment type cannot be used as a root segment in a secondary data structure for a logical data base.

3. When using a secondary processing sequence, the application cannot insert or delete an index target segment, or any segment on which an index target segment is dependent in its physical data base.

4. Data in any fields of segments can be changed except for data in sequence fields. If data in fields of an index source segment is changed and those fields are used in the search or subsequence fields of an index pointer segment, the index pointer segment is deleted from the position determined by its old key, and reinserted into the position determined by its new key.

Considerations

In using secondary indexing, consideration should be given to the following:

- When an index source segment is inserted into or deleted from a data base, a respective index pointer segment is inserted into or deleted from the respective secondary index. This maintenance occurs in all cases, regardless of whether or not the application program doing the updating actually uses the secondary index.

- When replacing data in an index source segment that is used in the search, subsequence or data fields of an index, the index is updated by IMS/VS to reflect the change. When data used in the ddata field of an index pointer segment is replaced in an index source segment, the index pointer segment is updated with the new data. When data used in the search or subsequence fields of an index pointer segment is replaced in an index source segment, the index pointer segment is updated with the new data, and in addition, the position of the index pointer segment within the secondary index is changed. The position is changed since a change to the content of the search or subsequence field of an index pointer segment changes the key of that segment. The secondary index is updated by deleting the segment from the position determined by the old key and inserting the index pointer segment in the position determined by the new key.

- The use of secondary indexes will increase storage requirements of all steps which include within the PSB: 1. a PCB for the indexed data base, and 2. the processing option which allows the index source segment to be updated. The additional storage requirements for each index data base will range from 6K to 10K. A percentage of this additional storage will be fixed in real memory by VSAM. For additional information on storage requirements, refer to the topic "VSAM Data Base Buffer Pools" in the section on VSAM support for IMS/VS in this manual.

- The use of a secondary index must be considered relative to alternate means of achieving the same function. As an example, it may be desired to produce a report from an HDAM data base in root key sequence. A secondary index will conveniently provide this capability. However, the access of each sequential root will, in most cases, be a random operation. It would be a very time consuming operation to fully scan a large data base where each root access is random. It may be more efficient to scan the data base in physical sequence (GET NEXT not using a secondary index), and then sort the results by root key so that the final report can be produced in root key sequence.

- A secondary index uses only a key sequenced data set if all index pointer segment keys are unique, and both a key sequenced and entry sequenced data set when index pointer segment keys are non-unique. Whenever possible, the data used for keys should be unique to eliminate the need for the entry sequenced data set, which in turn, eliminates the additional I/O operations required to search the entry sequenced data set.

- When calls for an index target segment type are qualified on the search field of a secondary index, additional I/O operations are required since the index must be accessed each time an occurrence of the index target segment type is inspected to see if that occurrence satisfies the call. Since the data contained in the search field of a secondary index is a duplication of data in a source segment, the user should determine whether or not an inspection of source segments in their data base might yield the same result faster.

- When reorganizing an indexed data base, maintenance of data in the user data portion of index pointer segments is the responsibility of the user. During reorganization, IMS/VS maintains data in all portions of index pointer segments in a secondary index data base except the user data portion. To carry user data forward through reorganization, the user must retrieve the data from the old index and replace it in the new index.

PROCESSING A SECONDARY INDEX AS A DATA BASE

A secondary index may be processed as a data base by providing a PCB which references the DBD of the index. The purpose of processing an index alone could be to scan the subsequence or duplicated data fields; to perform logical comparisons or data reduction between two or more indexes; or to add to or change the user maintained data area. Whatever the purpose of processing an index separately, the following guidelines and restrictions apply.

- No changes to system-maintained data fields in the index pointer segment will be allowed unless NOPROT is specified in the ACCESS=operand in the index DBD. Attempts to change system-maintained data without the NOPROT option will result in an AM status code.

- Inserts will not be permitted to any data base in which ACCESS=INDEX is specified.

- Any changes to system-maintained data in an index may render the index as unusable and unmaintainable.

- Deletion of index pointer entries by the user when the associated index source segments (ISS) exist will result in 'NE' status codes if the user makes updates to the ISS which will result in index maintenance.

- Qualification on the key of index pointer segments in SSA's must supply a value which includes not only the search portion of the key but also the constant and subsequence data if supplied. This is the only case in secondary indexing that the user is aware of the constant and subsequence data in the key.

- In processing a secondary index which is a member of a shared index it must still be regarded as a separate index data base. A series of GN calls will not violate the boundaries of the index data base for which it was intended. Each index in the shared index has its unique DBD name and root segment name.

Although the secondary data structure feature as provided by secondary indexes may be a very convenient way to process a data base for some applications, especially because it allows immediate retrieval of the index target segment, it is nevertheless possible to utilize secondary indexes for qualification only at any segment level without changing the apparent data base structure.

If a segment type on any level is indexed, the SSA for this segment may contain field names which have been defined as indexed fields and thereby make use of the existing index. The use of a field of the segment defined during Data Base Definition by an XDFLD statement, rather than a FIELD statement, specifies the use of an index to qualify that part of the call. If the SSA specifies that an index should be used, DL/I will obtain a candidate segment using the processing sequence, or prior positioning (that is, in the same way as if indexing had not been specified in the SSA), and then interrogate the index within the range specified in the SSAs to search for an index entry which points to the candidate segment. This may be more efficient and can save time-consuming data base accesses if the content of the indexed field is from a segment at a lower level than the index target segment. However, a warning might be appropriate here. Satisfying a SSA by inspecting the index means that the index is checked to ascertain whether or not the pointer provided through the index entry matches the segment currently under consideration. This may or may not be efficient. If there are several segments with the same indexed field value, there will be several index entries for this SSA. It may then be necessary to compare all their pointers until it is found that this segment does not satisfy the call, while a single access to the data base would have yielded the same result much faster. An SSA for instance, of the type "field ≠ value" will normally be unsuitable for an index search because several complete index data base scans might result to fulfill one data base call.

Independent and Dependent AND Boolean Operators

Allowing XDFLD field names to be used in SSAs has added another degree of freedom to qualifications in SSAs using Boolean operators. In order to fully utilize this degree of freedom, another Boolean AND operator has been added to the system specifications. The symbol for this new AND is the "#" and it is called the independent AND. The "*" as used with indexed fields is now called the dependent AND.

The distinction is made for the purpose of setting limits on the scan of an index while attempting to satisfy an SSA qualification. The following examples explain the difference between the dependent and independent AND operators.

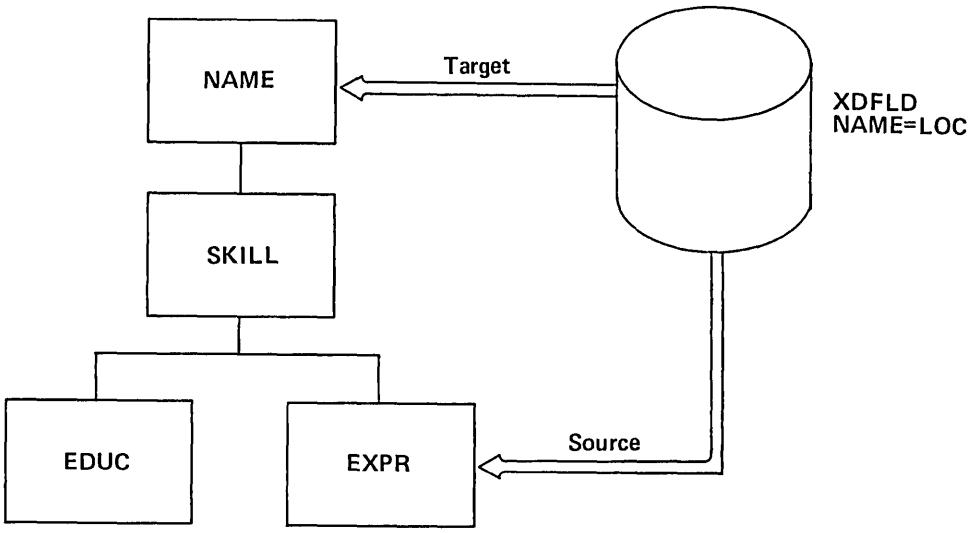


Figure 3-6. Example of Independent AND

Example 1 - To illustrate the need for and use of the independent AND, assume a requirement to search the data structure of Figure 3-6 for personnel having had experience in specific locations. For example, to request a person with experience in both Greenland and Mexico, the following call would be used:

```
GU  NAME(LOCb=GREENLAND#LOCb=MEXICO)
```

This would retrieve the first person who had had experience in both places. Internally, this means that DL/I will independently search through the index entries for Mexico, looking for those which point to the same target segments as are being pointed to by entries for Greenland.

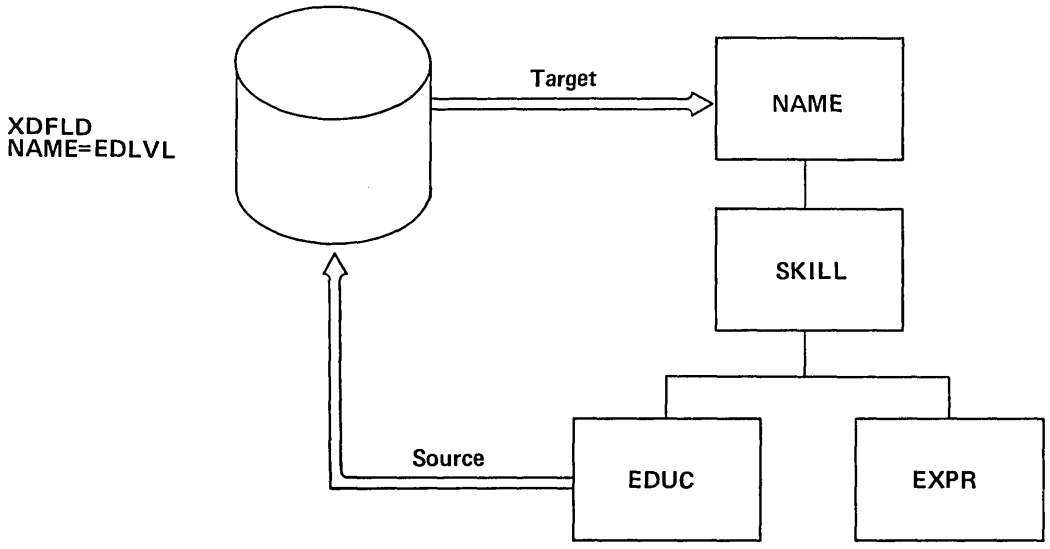


Figure 3-7. Example of Dependent AND

Example 2 - To illustrate the need for and use of the dependent AND, assume the data structure of Figure 3-7 and an index which indexes the name segment based on Education level. To retrieve the first name segment having an education level between 3 and 7 but excluding level 5, the qualification would use the dependent AND as follows:

```
GU  NAME(EDLEVELb>3*EDLEVELb<7*EDLEVEL¬=5)
```

This call would retrieve the Name segment pointed to by the first index pointer segment whose search field met all three conditions.

The difference (as illustrated in the previous examples) between the dependent and independent ANDs is as follows: with the independent AND, the conditions can be satisfied by two or more different pointer segments having the same target; with the dependent AND, all conditions must be met by one index pointer segment. Additional examples follow.

Assume an index data base in which two index entries point to the index target segment being considered. The two index pointer segments have key values of 5 and 20 and the XDFLD has the name of XF1. (This implies that the indexed field content comes from a dependent of the indexed segment.)

Example 3 - Use of the Dependent AND. The SSA qualification is:

```
(XF1b>2*XF1b<10*XF1¬=5)
```

Since the dependent AND is used, these three qualifications are considered to be a dependent group and all three qualifications will be used to set the limits for a single scan of the index. The scan will begin with XF1=3 and stop at XF1=9, but any index pointer segment(s) with XF1=5 will be skipped over. The result is that none of the index pointer segment(s) in the range will satisfy the SSA since the only associated index pointer segment(s) have XF1=5 and XF1=20, and it is not possible to satisfy all three qualifications with any one of them.

Example 4 - Use of the Independent AND. The SSA qualification is:

```
(XF1b>2*XF1b<10#XF1¬=5)
```

Notice now that there are only two qualifications in the dependent group. The conditional XF1¬=5 is an independent qualification. There will be two separate scans of the index conducted. The first scan will start at XF1=3 and proceed sequentially until the index pointer segment(s) with XF1=5 is found. Since this is associated with the index target segment the qualification of the first AND group is satisfied. The next step is to consider the qualification XF1¬=5. In order to satisfy the independent qualification, the scan is begun at the beginning of the index data base and every index pointer segment, except those with XF1=5, can satisfy the qualification. Since there is one with XF1=20 it will be found and the index target segment will have met this qualification. That is, there is an index entry which will satisfy the first AND group and there is an index entry which will satisfy the independent qualification.

The reader should verify for himself that

XF1b>18#XF1b<8 or XF1b=5#XF1¬=5

would be satisified in the above situation whereas

XF1b>18*XF1b<8 and XF1b=5*XF1¬=5

can never be satisified because they have contradictions.

In summary, qualifications connected by an independent AND are satisfied if there exists an index entry which satisfies the first and there exists an index entry which satisfies the second. Two qualifications connected by a dependent AND are satisfied if there exists an index entry which satisfies both.

The use of the "*" for XDFLD field-qualifications can form dependent AND groups only for like field names since all limits thus specified apply only to a single index scan. Two unlike field names connected with "*" will therefore be treated as "#".

The distinction between independent and dependent ANDs is made only in the case of XDFLD field qualifications. Unless both the connected qualifications involve XDFLD field-qualifications, the "*" and the "#" have exactly the same function that the "*" previously had.

CHAPTER 4. DATA COMMUNICATION APPLICATION PROGRAMMING

This chapter describes application programming for a teleprocesing environment using the Data Communication facility of IMS/VS. The emphasis is on the interface between a teleprocessing application program and the Data Communication facility. Figure 4-1 provides an overview of the IMS/VS teleprocessing environment. Both major IMS/VS components, the Data Base facility and the Data Communication facility, are used in this environment. The interface between the Data Base facility and its related application program (as described in the previous chapter) is the same in both the batch and teleprocessing environments.

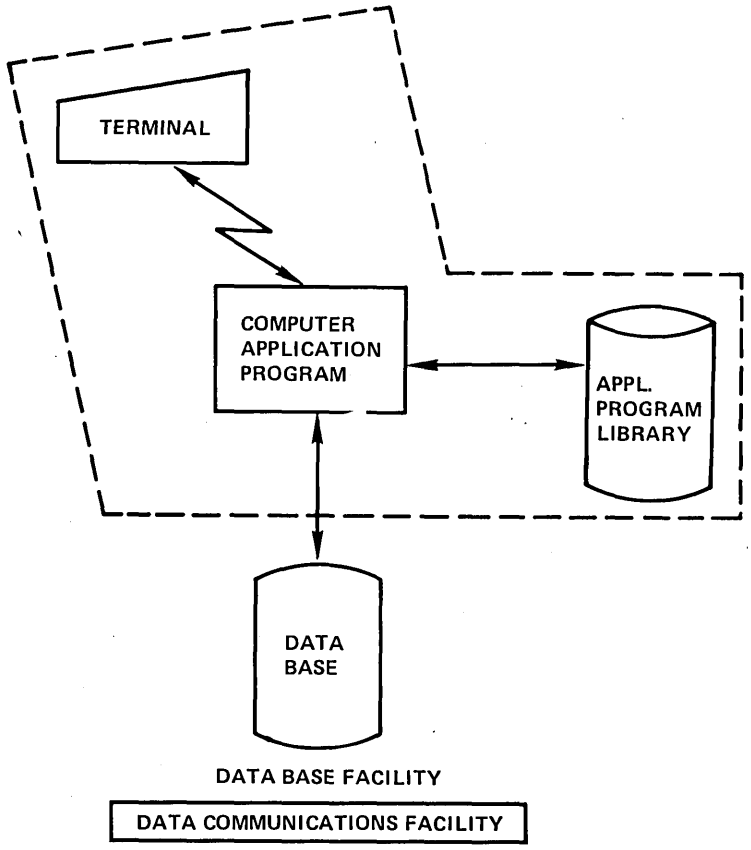


Figure 4-1. IMS/VS Data Communication Facility

Application programmers who use the IMS/VS Data Communication facility should be aware of two major options available to the Data Communication user:

- Message Format Service
- Conversational processing

Message Format Service (MFS) is a facility available to users of the 2740 Data Communication Terminal, 2741 Data Communication Terminal, 3270 Information Display System, 3600 Finance Communication System, 3767 Communication Terminal, and 3770 Data Communication System. Application programming information for MFS users is contained in the IMS/VS Message Format Service User's Guide.

Conversational processing is described in the next chapter of this manual. When conversational processing is used, an application program can retain information acquired through multiple interchanges with a terminal even though the program leaves the message region between interchanges.

TELEPROCESSING APPLICATION PROGRAM INTERFACE TO IMS/VS

When the IMS/VS Data Communication feature is used, application programs can communicate with teleprocessing (TP) devices as well as access data bases. The program communicates with a device logically through IMS/VS rather than directly to the device. This type of communications is made possible by the IMS/VS concept of logical terminals. A logical terminal is a name related to the actual device, the physical terminal. One physical terminal can have one or more associated logical terminals. The logical terminal name or names for each physical terminal are defined by the IMS/VS system programmer during IMS/VS system definition. The IMS/VS System/Application Design Guide contains a complete description of logical terminals.

The logical terminal concept allows an application program to be independent of a particular physical terminal. Generally, the application programmer need not be concerned about the actual location or address of the device. If a physical terminal becomes inoperative, its associated logical terminal(s) can be reassigned to another physical terminal, thereby causing output messages to be sent to another physical terminal. Also, each logical terminal can have unique security checking associated with it.

To an application program, therefore, a logical terminal can be viewed as just another sequential data input source or output destination. The application program interface to the logical terminal is through essentially the same call interface mechanics as that described for data base access. Access to a data base requires the use of a data base Program Communication Block (DB-PCB). Accordingly, communications with a TP device requires the use of a teleprocessing PCB (TP-PCB).

Application programs that operate in a teleprocessing environment normally reference both DB-PCBs and TP-PCBs, and must contain a mask to handle each PCB type. Figure 4-2 shows that the TP application program views terminals and data from a logical view point. Any changes to the physical terminal configuration or to actual data structures have a minimum effect on the application program.

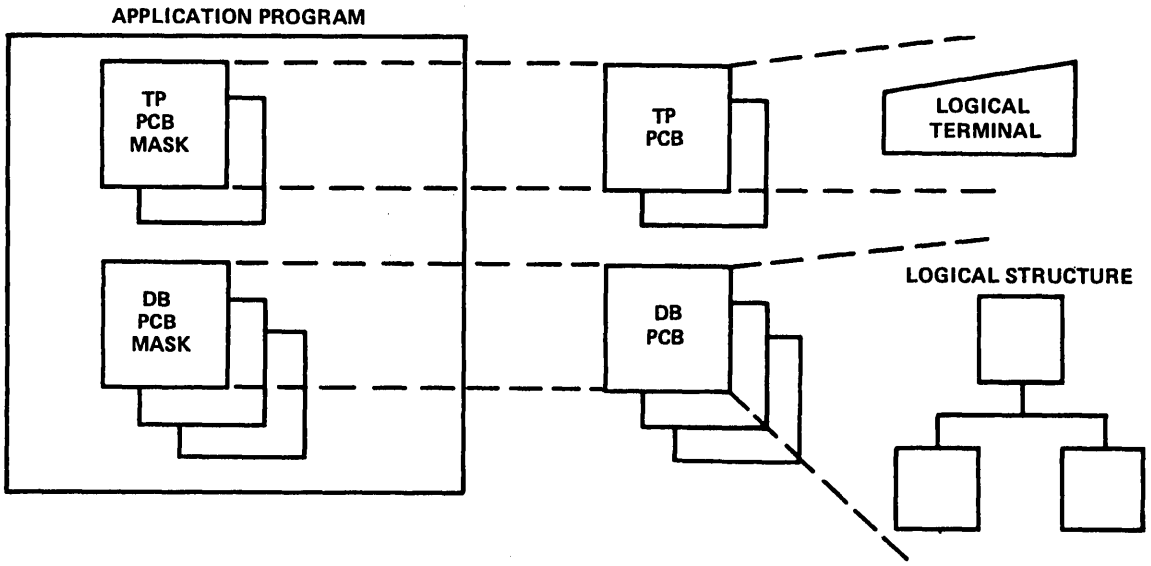


Figure 4-2. Relationship of Teleprocessing Application Program to DB PCBs and TP PCBs

TP PCBs

There are two types of TP PSBs -- the I/O PCB and the alternate PCB. An I/O PCB is always provided by IMS/VS to the application program that executes in a TP environment. Alternate PCBs are optional and are created as part of a Program Specification Block (PSB). A PSB is created by the IMS/VS PSB Generation Utility Program (PSBGEN) and resides in the IMS/VS PSB library (IMSVS.PSBLIB). Both the I/O and alternate PCBs are read into and retained in main storage during execution of the application program. See Figure 4-3.

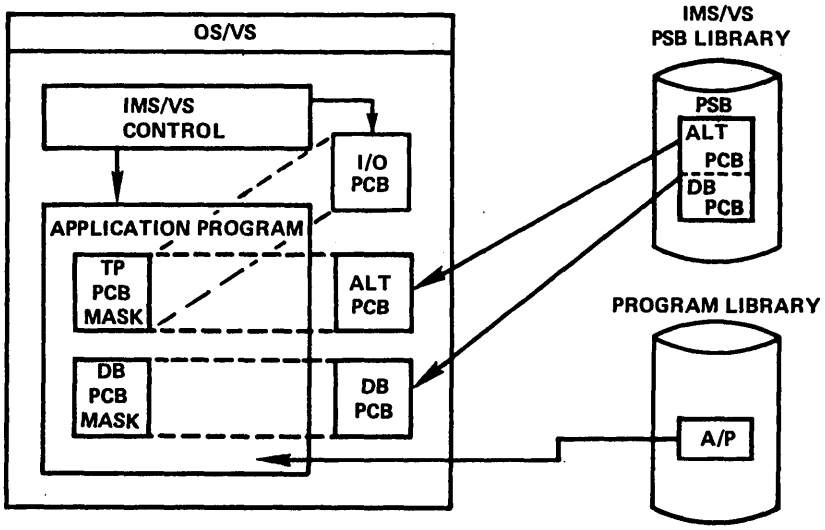


Figure 4-3. Teleprocessing Application Program Execution

To obtain an input message and to reply to it, the application program must reference the I/O PCB. To send a reply to a terminal other than the terminal that originated the input message, or to another application program, the program references an alternate PCB.

To be able to test TP application programs in a batch region without having to recompile before online testing, specify CMPAT=YES in the PSBGEN statement of the PSBGEN utility program. When CMPAT=YES is specified, IMS/VS provides PCBs to the program as if a message region was being used.

I/O PCB

An I/O PCB is the mechanism required by a TP application program to:

- Obtain an input message from a terminal.
- Return a reply to the terminal that originated the input message. Application programs returning replies to terminals operating in response mode, conversational mode, or exclusive mode must direct such replies to the I/O PCB or an alternate PCB that has been defined as ALTRESP=YES.

When IMS/VS receives an input message, it queues the message according to transaction code and schedules the application program that processes the transaction. When scheduling the program, IMS/VS passes to the program the address of its I/O PCB plus the alternate PCB(s) (if any) and the DB-PCB(s) (if any) defined in its PSB. The I/O PCB contains the name of the logical terminal that entered the message (source) and can receive the reply (destination).

Alternate PCB

An alternate PCB is the mechanism required by a TP application program to send an output message to a destination other than the TP device that originated the input message. An alternate PCB specifies a destination of either a logical terminal or a transaction code defined during IMS/VS system definition. The destination can be specified during PSB generation or during program execution. When an alternate PCB specifies a transaction code as a destination, IMS/VS routes the message built using that alternate PCB to the application program that processes the specified transaction code (this is known as a program-to-program message switch).

To be able to specify a destination during program execution, the alternate PCB must be defined as modifiable during PSB generation. When an application program uses modifiable alternate PCBs, the program must specify the output message destination before beginning to build the output message.

Alternate PCBs can also be defined with the express message option, EXPRESS=YES. Messages destined for alternate PCBs so defined are considered complete and sent even if the application program abends. Express alternate PCBs should be used judiciously--they are primarily intended for a program when it detects that some invalid processing occurred and that it must issue a rollback (ROLL) call to resume processing at its most recent synchronization point. The express message PCB provides a way for the program to notify the terminal operator of the situation. If used in circumstances other than the above, express alternate PCBs can cause duplicate transaction or output message processing by an application program if an IMS/VS control region abends.

Alternate PCBs can also be defined with the response option ALTRESP=YES. When so defined, a response to a terminal in response mode, conversational mode, or exclusive mode can be directed to the alternate PCB instead of to the I/O PCB. An alternate PCB so defined meets the I/O PCB requirements for these operating modes and is known as a response alterante PCB. Such a response alternate PCB must have as its destination a logical terminal. Using response alternate PCBs allows the application program to send output to a logical terminal other than the one that originated the input message and still satisfy the requirements of these operating modes. If specified during PSB generation (SAMETRM=YES), IMS/VS will verify that the logical terminal named in this response alternate PCB is assigned to the same physical terminal as the logical terminal that originated the input message. (This check is always made for conversational programs and response mode transactions.)

TP-PCB MASK

To support communication with IMS/VS, the TP application program must contain a TP-PCB mask. As shown in Figure 4-4, a TP-PCB mask must provide for seven fields of information.

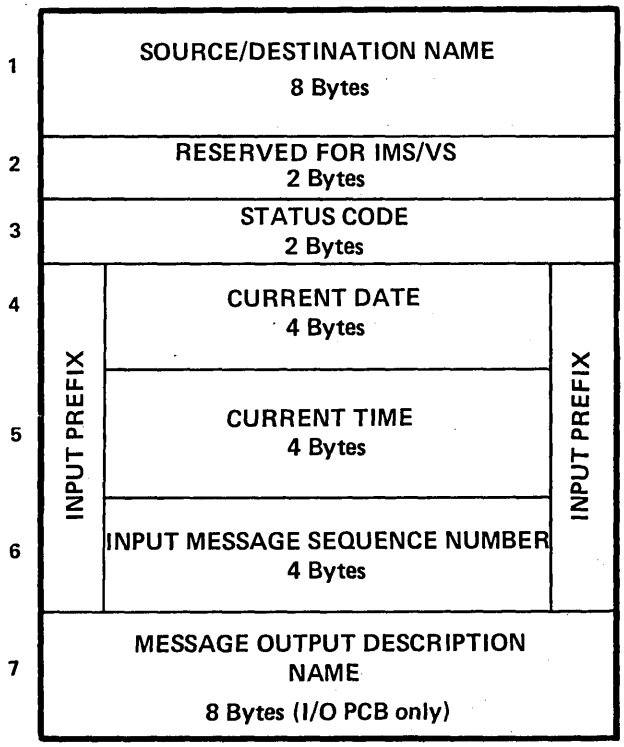


Figure 4-4. Layout of a TP-PCB Mask

1. SOURCE/DESTINATION NAME - For input, this field contains the name of the logical terminal that entered the message, or blanks. For output, this field contains the name of a logical terminal or a transaction code. The name is 1 to 8 bytes long, left-justified, and padded with blanks.

2. RESERVED AREA - a 2-byte area reserved for IMS/VS.

3. STATUS CODE - a status code that is the result of a TP call is placed in this 2-byte field. When a successful call is executed, this field is returned blank. A non-blank status indicator is returned on an unsuccessful call.

4., 6.
INPUT PREFIX - is available only for the I/O PCB. The length of the input prefix is 12 bytes:

4. 4 bytes - Julian date (YYDDD-packed decimal) when the input message was completely received from the physical terminal.

5. 4 bytes - time (HHMMSS.S-packed decimal) when the input message was completely received from the physical terminal.

6. 4 bytes - sequence number (binary) of the input message.

7. MESSAGE OUTPUT DESCRIPTION NAME - is available only for the I/O PCB. This field has meaning only when output messages are sent to terminals that use the IMS/VS Message Format Service (MFS).

When IMS/VS supplies the first segment of an input message, it fills this field with either the name of a message output description or blanks. The contents of this field can be changed by using the output MOD name parameter of the TP output call that contains the first segment of an output message. Further information on the use of the output MOD name parameter is contained in the IMS/VS Message Format Service User's Guide.

COBOL Example of a TP-PCB Mask

The following example is an I/O PCB mask for an American National Standard (ANS) COBOL message processing program. This mask would be found in the linkage section of the program. A mask for an alternate PCB would be similar but without the IN-PREFIX and MOD-NAME fields.

```
DATA DIVISION.

LINKAGE SECTION.

01      IO-PCB.
    02  LTERM-NAME  PICTURE X(8).
    02  DLI-RESERVE PICTURE XX.
    02  STATUS-CODE PICTURE XX.
    02  IN-PREFIX.
        03  JULIAN-DATE PICTURE S9(7)       COMPUTATIONAL-3.
        03  PCB-TIME-OF-DAY PICTURE S9(7)   COMPUTATIONAL-3.
        03  MSG-SEQ     PICTURE S9(7)       COMPUTATIONAL.
    02  MOD-NAME PICTURE X(8).
```

PL/I Example of a TP-PCB Mask

The following is an example for PL/I Optimizing Compiler message processing programs. A mask for an alternate PCB would be similar but without the IN_PREFIX and MOD_NAME fields.

```
DECLARE     1   IO_PCB BASED (IO_PCB_PTR),
                2   IO_TERMINAL CHARACTER(8),
                2   IO_RESERVE  CHARACTER(2),
                2   IO_STATUS   CHARACTER(2),
                2   IN_PREFIX,
                    3   PRE_DATE        FIXED DECIMAL(7),
                    3   PRE_TIME        FIXED DECIMAL(7),
                    3   PRE_MSG_COUNT   FIXED BINARY(31),
                2   MOD_NAME CHARACTER(8);
```

ENTRY TO THE TELEPROCESSING APPLICATION PROGRAM

The entry statement to a TP application program names the TP-PCBs and the DB-PCBs. The TP-PCBs must precede the DB-PCBs, and at least one TP-PCB must be specified to provide for the I/O PCB.

- The format for an ANS COBOL program is:

  ```
  ENTRY 'DLITCBL' USING IO-PCBNAME, ALT-PCBNAME1, ALT-PCBNAMEn,
  DB-PCBNAME1, DB-PCBNAMEn.
  ```

- The format for a PL/I optimizing compiler program is:

  ```
  DLITPLI: PROCEDURE (IO_PCB_PTR,ALT1_PCB_PTR,ALT2_PCB_PTR,
  DB1PCB_PTR,DB2_PCB_PTR) OPTIONS(MAIN);
  ```

Programs that are OS/VS subtasks of an application program called by IMS/VS must not issue DL/I calls. If they do, the results will be unpredictable. With PL/I, whenever PL/I multitasking is used, all tasks, even the apparent main task, operate as subtasks to a hidden PL/I control task. PL/I tasking is therefore not allowed in an IMS/VS program.

TP CALLS

This section describes the call functions available to IMS/VS application programs in a TP environment. These TP calls relate to messages. A message is comprised of one or more segments. Figure 4-5 shows two messages: Message A is made up of segments A1, A2, and A3; Message B is made up of segments B4 and B5.

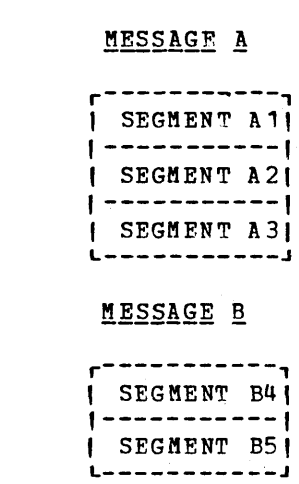


Figure 4-5. Message Relationships to Its Segments

The messages received from terminals and placed in the message queues are accessible to a message program by TP calls. The TP call functions available are:

- GUbb (get unique)
- GNbb (get next)
- ISRT (insert)
- PURG (purge)
- CHNG (change)

The TP call format is slightly different from DL/I calls because there is no hierarchical structure with which to be concerned. SSAs (Segment Search Arguments) are not used for TP calls.

- The format for an ANS COBOL program is:

```
CALL 'CBLTDLI' USING CALL_FUNC, TP_PCB, WORK_AREA.
```

- The format for a PL/I program is:

```
CALL PLITDLI (PARM_COUNT,CALL_FUNC,TP_PCB,WORK_AREA);
```

When a transaction or input message is available for processing, the associated application program is scheduled into a message processing region. After being loaded, the program should issue a get unique (GU) call to obtain the first segment of its input message. Each subsequent segment of that message is obtained with a get next (GN) call. GU and GN calls can be made only to the I/O PCB.

If the program is serially-reusable or reentrable between GU calls, GU calls can be issued for subsequent input messages until all messages are retrieved. If a program is not serially-reusable or reentrable between GU calls, the program must terminate after each GU call so that it will be reloaded and re-initialized.

An insert (ISRT) call is used to build output messages. Each segment of an output message can have appropriate terminal control characters embedded in the text. The purge (PURG) call can be used to delimit output messages being inserted. The output message may be sent as a reply to the terminal that originated the input message or to other terminals.

Insert and purge calls can be issued to any TP-PCB. Message replies to input should be directed to the I/O PCB for return to the inputting terminal. Messages destined for any terminal other than the inputting terminal, or for an application program, must be directed to an alternate PCB. Replies to a terminal in either response mode, conversational mode, or exclusive mode must be made to either the I/O PCB or an alternate PCB defined during PSB generation as ALTRESP=YES.

Messages placed into the queue by an application program are not transmitted to their destination simultaneously with the insertion of the segments. Unless the message destination is an alternate PCB defined as EXPRESS=YES during PSB generation, IMS/VS places the output message in a temporary destination queue until the program reaches a synchronization (sync) point. When the sync point occurs, IMS/VS moves the complete message to its final destination queue. If the application program abends, all activity is backed out as if it never occurred. Activity backed out includes all messages and transactions created, and all data base updates. Backout occurs prior to the termination sync point.

Note: A DC application program sync point is program termination. The checkpoint call, if used, is also a sync point. If the transaction to be processed by the program was defined as MODE=SNGL during IMS/VS system definition, each request for a new message (get unique) is also a sync point.

If the message destination is an alternate PCB defined as EXPRESS=YES, IMS/VS bypasses the temporary queue and moves completed messages to the final destination queue. These messages will be considered complete and sent if the application program abends. The only condition that will prevent these messages from being sent is a deadlock situation that occurs prior to completion of these messages.

The change (CHNG) call is used during program execution to set an output message destination, to an alternate PCB.

At the completion of a TP call, IMS/VS returns a status code, indicating the results of the call, in the 2-byte status code field of the TP-PCB associated with the call. The application program should interrogate the status code after each TP call.

If the call is completed successfully, the status code consists of two EBCDIC blanks; otherwise it is one of the codes shown in Appendix A (quick reference) or B (status code descriptions).

INPUT MESSAGE SEGMENT CALLS

Get Calls (GU, GN)

The get calls are used to retrieve segments of an input message. For each get unique (GU) or get next (GN) call, one segment is returned to the application program. IMS/VS returns the retrieved segment to a work area defined in the application program. Since the length of a message segment is variable, the work area must be large enough to contain the longest segment expected by the program.

The first segment of an input message is obtained with a GU call against the I/O PCB. In response to a GU call, IMS/VS returns the first message segment and fills in the following I/O PCB fields: 1) source name (name of the logical terminal that originated the message), 2) status code, 3) input prefix, and 4) message output description name (when present and message is from a MFS-supported device).

- The format for an ANS COBOL program is:

  ```
  CALL 'CBLTDLI' USING GET-UNIQUE-FUNC, IO-PCB, WORK-AREA.
  ```

- The format for a PL/I program is:

  ```
  CALL PLITDLI (PARM_COUNT,GET_UNIQUE_FUNC,IO_PCB,WORK_AREA);
  ```

For programs that process multiple transaction codes, the text of the input message can be examined to determine the transaction code.

The second and subsequent segments of an input message are retrieved with a GN call.

- The format for an ANS COBOL program is:

  ```
  CALL 'CBLTDLI' USING GET-NEXT-FUNC, IO-PCB, WORK-AREA.
  ```

- The format for a PL/I program is:

  ```
  CALL PLITDLI (PARM_COUNT,GET_NEXT_FUNC,IO_PCB,WORK_AREA);
  ```

Figure 4-6 shows the call functions used to obtain the various segments of messages A and B.

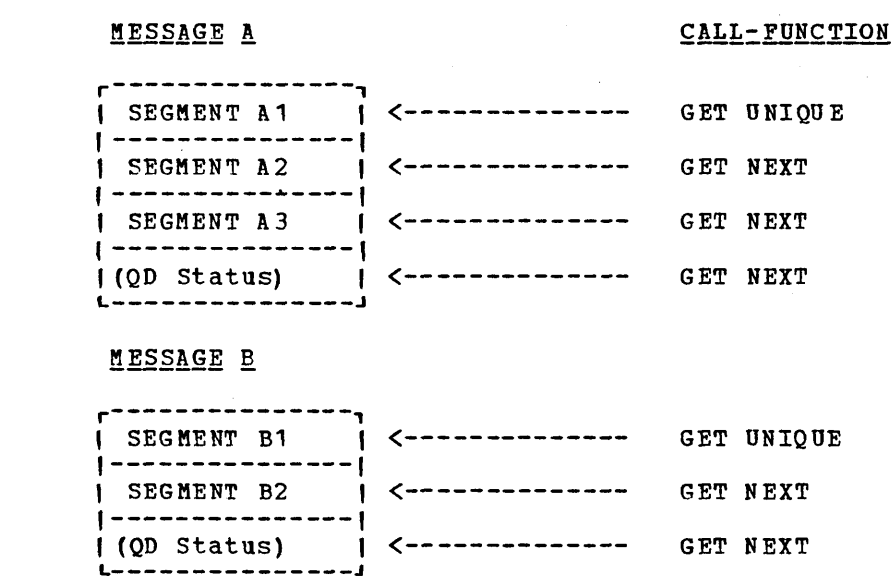


Figure 4-6. Call Functions for Segments of Messages A and B

The application program should inspect the status code field of the I/O PCB after each get call. A blank status indicates a segment was retrieved successfully. A QD status after a GN call indicates there are no more segments to retrieve.

To retrieve all the segments of a message, a GU call and successive GN calls must be issued until a QD status is returned. It is not necessary to retrieve all the segments of a message. A GU call can be issued at any time to retrieve the first segment of the next message. A QC status after a GU call indicates there are no more messages in the queue. If a batch message processing program issues a GU after receiving a QC status, a message will be returned if there is one.

In addition to the status code, IMS/VS fills in the source/destination name field of the I/O PCB after each successful GU call. When the message source is a logical terminal, IMS/VS places the logical terminal name in the name field. When the source is unknown, IMS/VS sets the field to blanks. This occurs in the following instances:

Message Source	Reason
Message processing program (MPP)	• MPP did a program-to-program message switch after a GU of an input message without a source
	• MPP inserted a message before issuing a successful GU call.
Batch message program (BMP)	• BMP has not issued a successful GU call

OUTPUT MESSAGE SEGMENT CALLS

Insert Call (ISRT)

The insert call is used to build output messages. To build an output message in reply to the terminal that originated the input message, output message segments must be inserted to the I/O PCB. Output message segments can also be inserted to alternate PCBs. If an alternate PCB has been defined as modifiable, a change call must be used before the first insert call. The change call sets the destination of the output message (refer to "Change Call" in the section "Additional TP Calls" later in this chapter).

The ISRT call format is similar to that for message get calls.

- The format for an ANS COBOL program is:

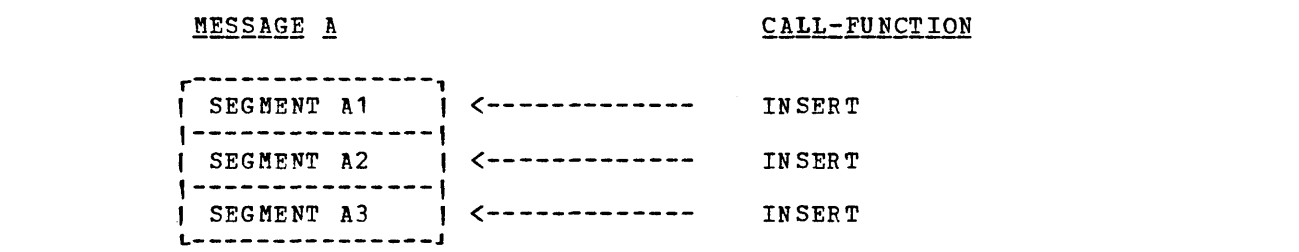
```
CALL 'CBLTDLI' USING ISRT-FUNC, TP-PCB, MSG-SGMT-IO-AREA,
OUTPUT-MOD-NAME.
```

- The format for a PL/I program is:

```
CALL PLITDLI (PARM_COUNT,ISRT_FUNC,TP_PCB,MSG_SGMT_IO_AREA,
OUTPUT_MOD_NAME);
```

The output MOD name parameter is optional and only meaningful to IMS/VS systems that include MFS. It can be specified only on the insert call that provides the first segment of the output message. OUTPUT MOD NAME is the label of an 8-byte field containing the name of the message output description; the name must be left-justified and padded with blanks.

Figure 4-7 shows an ANS COBOL example of output message segment calls for output message A. These three call statements create one output message.




```
CALL 'CBLTDLI' USING INSERT-FUNC, TP-PCB, SEGMENT 1, OUTPUT-MOD-NAME.

CALL 'CBLTDLI' USING INSERT-FUNC, TP-PCB, SEGMENT 2.

CALL 'CBLTDLI' USING INSERT-FUNC, TP-PCB, SEGMENT 3.
```

Figure 4-7. Call Functions for Segments of an Output Message and Call Statements

Figure 4-8 shows an example of an output message as one message segment called Message A. The insert call function creates one message segment which would produce three lines on a 1050. If the segment was sent to a MFS-supported device, it would be edited based on the message output description named MSGFMTX7 (carrier-return characters perform no function other than occupying a position in the output segment).

WORK AREA NAME	CONTENTS OF WORK AREA
MESSAGE A	NO STOCK ON HANDnl* BACK ORDERS ARE PRESENTcr THE NEXT SCHEDULED ARRIVAL IS XX-XX-XX.cr
FMT-A	MSGFMTX7

```
CALL 'CBLTDLI' USING INSERT-FUNC, TP-PCB, MESSAGE A, FMT-A.
```

*nl = new line

Figure 4-8. Output Message As One Segment and Its Call Statement

Output message segments cannot be distinguished as first and subsequent segments by the insert call. Any required distinction must be made by the programmer. All message segments inserted to a given TP-PCB during the processing of a single input message are treated by IMS/VS as a single message unless the PURG call is used. Output message segments may be created by insert calls prior to retrieval of an input message.

If an application program processes a transaction code for which the maximum number and size of output segments has been limited (through system definition or the /ASSIGN command), the program must be prepared to accept the status codes IMS/VS returns if the limits are violated. When an application program inserts a segment that violates the size or number limit, the insert call is not honored and an error status code is returned. If the program attempts to insert an excess number

of segments more than once for the same input message, IMS/VS abends the application program.

Output messages sent in reply to terminals in conversational mode, response mode, or exclusive mode must be inserted to either the I/O PCB or an alternate PCB defined as ALTRESP=YES. All segments of one message must be inserted to the same PCB. If specified during PSB generation, (SAMETRM=YES), and if the application program is conversational, or if the physical terminal is in response mode, IMS/VS will verify that the logical terminal named by that response alternate PCB is assigned to the same physical terminal as the logical terminal that originated the message.

If the TP application program has DB-PCBs defined, one or more data base calls may be executed. The normal sequence of operation may be to obtain the input message, issue data base calls based upon input message content, and create an output message based upon input message content and data base calls.

ADDITIONAL TP CALLS

Purge Call (PURG)

The purge call causes all message segments that have been inserted to a TP-PCB to be collected together as a message and enqueued on the TP-PCB's destination. Completed messages are handled as described in the introduction to this section.

The purge call can be specified with an I/O area. In this format, the purge call performs the purge function and then treats the data contained in the I/O area as the first segment of a new message.

Message segments are normally grouped into a message and made ready for transmission at the time the program issues a GU for a new input message or the program terminates. The purge call allows the program to insert multiple messages to the same destination while processing a single input message.

- The format for an ANS COBOL call is:

```
CALL 'CBLTDLI' USING PURG-FUNC, TP-PCB.

or

CALL 'CBLTDLI' USING PURG-FUNC, TP-PCB, MSG-IO-AREA,
OUTPUT-MOD-NAME.
```

- The format for a PL/I call is:

```
CALL PLITDLI (PARM_COUNT,PURG_FUNC,TP_PCB_PTR)

or

CALL PLITDLI (PARM_COUNT,PURG_FUNC,TP_PCB,IOAREA,
OUTPUT_MOD_NAME);
```

The output MOD name parameter is optional and is only meaningful to IMS/VS systems that include MFS. It can be specified only on the purge call that provides the first segment of an output message. OUTPUT MOD NAME is the label of an 8-byte field containing the name of the message output description; the name must be left-justified and padded with blanks.

Figure 4-9 shows how message segments may be grouped into messages by the application program.

The purge call causes IMS/VS to consider the output message complete even if the application program that issued it subsequently abends. If the message destination was an alternate PCB defined during PSB generation as EXPRESS=YES, this could result in a terminal receiving a response from the application program even though it abended without fully processing this transaction. If the destination of the alternate PCB was a transaction (program-to-program message switch), the program for that transaction will be scheduled even though the originating program abended.

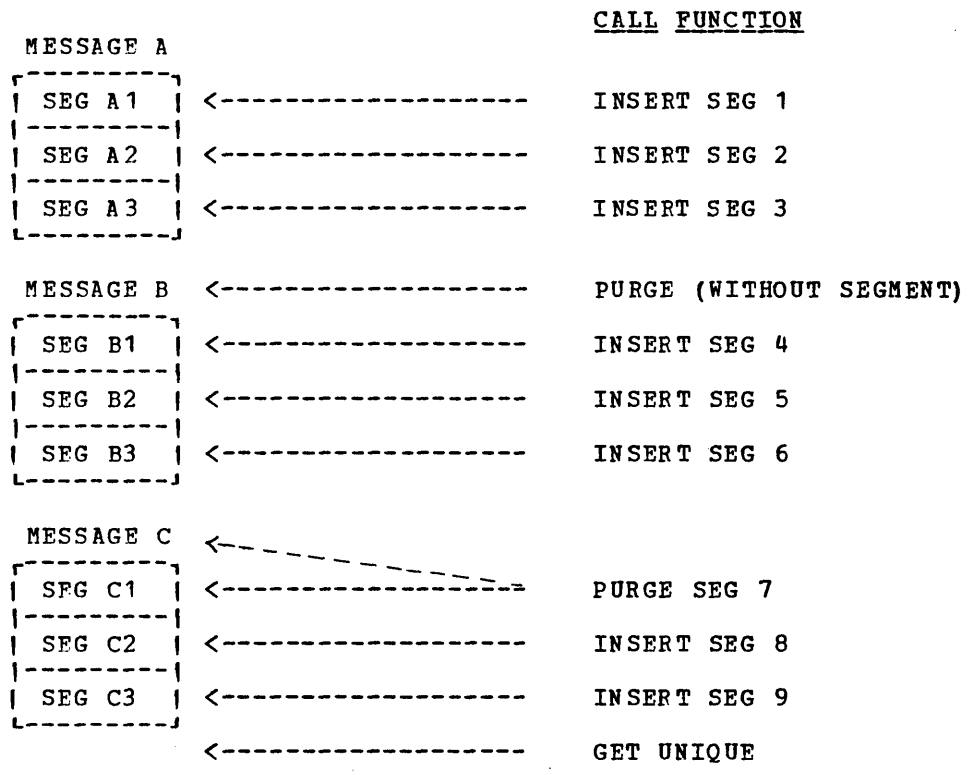


Figure 4-9. Grouping of Message Segments (PURG Call)

If a purge call is issued for a message to a terminal in response mode, output messages may be transmitted out of sequence.

Conversational programs are not allowed to utilize the purge call.

The purge call must not reference an alternate PCB defined as ALTRESP=YES.

If a purge call is used with no parameters, or with the I/O area parameter missing and the optional MFS parameter specified, the results are unpredictable.

Change Call (CHNG)

The change call is used to set the destination of an alternate PCB to any valid logical terminal or transaction code in the system. To use the change call, the alternate PCB must have been defined as modifiable during PSB generation. The destination of the modifiable PCB must be set with the change call before any segments are inserted. CHNG can be used to set the PCB destination to a conversational transaction only by a conversational program.

When used for program-to-program message switching, the terminal from which the message is entered must pass the security check for the new transaction code. If the source terminal is not known to IMS/VS, and the destination has security, the call is rejected with an error status code.

The new destination remains set until either the application program issues another CHNG, issues a GU, or terminates. At that time, IMS/VS resets the destination to blanks.

A change call for an alternate PCB cannot be issued while that PCB is being used to form a message. Therefore, unless PURG is issued, multiple modifiable PCBs must be defined if messages are to be sent to several destinations while processing a single input message.

- The format for an ANS COBOL call is:

```
CALL 'CBLTDLI' USING CHNG-FUNC, ALT-PCBNAME, DEST-NAME.
```

- The format for a PL/I call is:

```
CALL PLITDLI (THREE,CHNG_FUNC,ALT_PCBNAME,DEST_NAME);
```

The destination name parameter (DEST NAME) specifies the label of an 8-byte field containing the name of the logical terminal or transaction code to be assigned as the destination for this PCB. The name must be 1 to 8 bytes, uppercase EBCDIC, left-justified, and padded with blanks.

MESSAGE FORMATS

Three message formats are used within IMS/VS:

- Input message
- Output message
- Program-to-program message switch

The formats shown represent message segments as they would be received or constructed in the message segment I/O work area. A message segment and a single message line are synonymous.

The formats are different when either conversational processing or Message Format Service is used. Formats for conversational processing are described in the next chapter. MFS formats are described in the IMS/VS Message Format User's Guide.

INPUT MESSAGE FORMAT

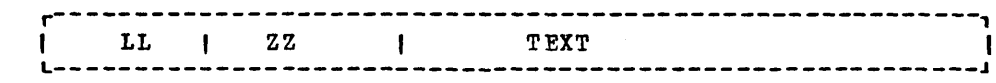
Input message segments originate at a communications terminal and are delivered to the application program's message segment I/O work area by means of a GU or GN call. The length of the input message segment (text portion) is directly related to the line length of the specific communications terminal that originated the message.

The first segment of an input message may not contain a transaction code if the message was switched from another program.

The maximum number of bytes allowed by each terminal supported by IMS/VS is shown below.

Terminal	Number of Bytes
1050, 2740-1, 2740-2, 2741	130. If MFS is used for the 2740/2741, see the IMS/VS Message Format Service User's Guide.
2260-1, 2265-1 with 12/80 screen	80
2260-2	40
2770	Variable, depending on component.
2780	80
2980	Variable, depending on user edit.
3270	Variable; see the IMS/VS Message Format Service User's Guide.
3600, 3790	Variable; refer to the IMS/VS Advanced Function for Communications manual, or, if MFS is used, to the IMS/VS Message Format Service User's Guide.
3741	128
3767, 3770	512. If Message Format Service (MFS) is used, the length of the message segment is defined by the user to MFS.
7770	Variable, depending on user edit.
33/35 Teletypewriter	80
Local Card Reader (2501, 2520, 2540, 1442)	80
System/3 System/7	Variable, dependent on user's program in the System/3 or System/7.
System/370 console	122

The format of each input message segment is:



LL
: is a 2-byte binary field representing the total length of the message segment, including LL and ZZ. The value of LL equals the number of bytes in text plus 4. The LL value is provided by IMS/VS for input messages.

  When PL/I is used, the LL field must be defined as a binary fullword. The value contained in the LL field is the actual segment length minus 2 bytes. For example, if the input message segment is 16 bytes, LL is equal to 14 and represents the sum of the lengths of LL (4 bytes minus 2 bytes), ZZ (2 bytes), and TEXT (10 bytes).

ZZ
: is a 2-byte field reserved for IMS/VS.

TEXT
: is the message segment in EBCDIC as it was entered at the terminal. IMS/VS edits a message segment before passing it to the application program.

- First or only segment

  A single-segment message, or the first segment of a multisegment message, contains a transaction code and the segment text. A transaction code does not exceed 8 bytes, and is followed by a blank. IMS/VS has removed any of the following items if they appeared in the terminal input stream:

  - Leading control characters
  - Leading blanks
  - Backspaces
  - Trailing control characters

  If a password is present, IMS/VS:

  - Removes the password
  - Replaces the password with a blank unless the first byte following the password is a blank
  - Left-justifies the segment text

- Non-first segments

  The text of the second and subsequent segments of a multisegment message contains message text only. The IMS/VS edit functions are the same as above except IMS/VS does not remove leading blanks.

- Preset Mode Segment Edit

  For the first or only segment of a message from a terminal in preset mode, IMS/VS inserts the transaction code. IMS/VS also inserts a blank following the transaction code unless the first byte of message text is a blank. The second and subsequent segments are treated as described above for non-first segments.

The input of segment 1 of Message A at a 1050 or 2740 Model 1 terminal may be:

```
                              b     b             CE
ORDER(PURCH)bNUMBERb42746k5bPAWkRTbNBRb576325RO
                              s     s              B
                              p     p
<---------------47 CHARACTERS----------------->
```

But the received segment 1 of Message A in the input segment I/O work area of the application program (specified in the CALL statement) is:

```
1  2  3     4         5   6           7

LLZZORDERbNUMBERb42745bPARTbNBRb576325
<----------38 CHARACTERS------------>
```

1

LL is the 2 bytes containing the length of the message segment. This message segment is 38 bytes long.

2

ZZ is the 2 bytes reserved by IMS/VS.

3

ORDERb is now the transaction code and a blank where before there was also a password which is edited out before being received at the application program.

4

NUMBER is the first 6 bytes of the text of this message segment.

5

45b shows that the incorrect character (6) and the backspace have been edited out by IMS/VS, leaving the next character (5).

6

PART shows that the incorrect character (W) and the backspace have been edited out by IMS/VS, leaving the next character (R).

7

shows that the CR (carrier return) and the EOB (end-of-block) have been edited out.

DEVICE DEPENDENT INPUT MESSAGE CONSIDERATIONS

The IMS/VS Operator's Reference Manual describes the input message format and operating characteristics for each terminal type supported by IMS/VS. The remainder of this section on input message formats lists input message considerations that should be reviewed by the application programmer responsible for supporting the 2260 Display Station Models 1 and 2, the 2265 Display Station Model 1, components of the 2770 Data Communication System, and the 2972/2980 General Banking Terminal System.

2260-1, 2260-2, 2265-1

- The input message is broken into segments whose length is variable:

Display Station	Number of Bytes per Segment (Screen Line)
2260-1,2265-1	1 to 80 (12 segments per screen)
2260-2(2848-1)	1 to 40 (6 segments per screen)
2260-2(2848-2)	1 to 40 (12 segments per screen)

  A segment contains the number of bytes on one screen line unless a New Line (NL) symbol is entered; when a NL symbol is used, the segment is truncated at the NL symbol.

- A START MI symbol must precede entry of an input message. Only one START MI symbol per screen is allowed. The START MI symbol can be entered by the operator from the keyboard or can be placed on the screen by the application program (when placed by the application program, the START MI must be one character, multipunched X'4A' or C'¢').

  If ENTER is pressed when no START MI is displayed, no data is sent to IMS/VS; IMS/VS displays a START MI and flags the screen as reserved for an input message

- An input message is considered to be that data contained between the START MI and the position of the cursor when ENTER is pressed. Any data outside these bounds when ENTER is pressed is ignored and not transmitted to IMS/VS.

2770 System Components

2265-2

- The input message is broken into segments whose length is variable:

Screen Size	Number of Bytes per Segment (Screen Line)
12x80	1 to 80 (12 segments per screen)
15x64	1 to 64 (15 segments per screen)

  A segment contains the number of bytes on one screen line unless a New Line (NL) symbol is entered; when a NL symbol is used, the segment is truncated at the NL symbol.

- A SMM symbol should precede any entry of an input message. Only one SMM symbol is allowed per screen. The 2770 system defaults to a beginning of screen read to cursor if no SMM symbol is present on the screen, unless the screen has been erased before the ENTER key is depressed. A SMM symbol is not recognized if it is placed on the screen following an NL character on the same line.

- The terminal operator can enter the SMM symbol from his keyboard or the application program can place it on the display screen. (The symbol must be one character, multipunched X'4A', or a C'¢'). IMS/VS also provides that if the operator presses KEYBOARD REQUEST, ERASE FULL, ENTER, IMS/VS displays a SMM symbol on the screen and flags the screen as reserved for an input message.

- The input message is considered to be that data contained between the SMM symbol and the position of the cursor symbol at the time the ENTER key is pressed.

Card Reader

- The input segments may be a maximum of 80 bytes. The segment may be smaller for one of two conditions. If the 2772 control unit has the Buffer Expansion Feature installed, any trailing blanks in an individual card are deleted. If the "/." sequence is used by the terminal operator to indicate the end of a transaction input text, the "/." sequence is deleted from the message segment. If the terminal operator punches an end of media (EM) character in the card, the card is truncated from the position before the EM character.

Keyboard

- The maximum input segment size is dependent on the size of the 2772 line buffer or the size of the IMS/VS queue message buffer minus the prefix length, whichever is smaller.

Paper Tape Reader

- The maximum input segment size is dependent on the value specified in the PTSEG= keyword of the TERMINAL statement during IMS/VS system definition.

Magnetic Ink Character Reader (MICR)

- A message is considered to be all documents read from the component until an End of File Document is detected.
- The input message segment size is dependent upon the MICR features and terminal operator field selections.
- Each document from the MICR is treated as a segment.

Magnetic Data Inscriber (MDI)

- The maximum input segment size is dependent upon the value specified in the MDISEG= operand of the TERMINAL statement during IMS/VS system definition.
- The input message is considered to be all data records processed from the first input until an End of Data Code is detected from the MDI.
- If the ERROPT=ACCEPT option of the TERMINAL statement is selected during IMS/VS system definition, the contents of an input segment for which an error occurred are undefined.

2972/2980 Components

When a 2980 Model 1 or Model 4 teller station is used as the input terminal, there are characters that can be entered which are not translatable into EBCDIC. Figures 4.10 and 4.11 list the hexadecimal values used by IMS/VS to represent the non-EBCDIC characters that can be entered from a teller station. Figure 4-12 identifies 2980 Model 4 function key entries.

Figure 4-10 represents the IMS/VS translation of numeric entry data from a 2980 Model 1 teller terminal. Alphabetic entry data is presented to the application program in the standard EBCDIC character set.

KEY NUMBER	GRAPHIC SYMBOL PRINTED	HEX VALUE	KEY NUMBER	GRAPHIC SYMBOL PRINTED	HEX VALUE	KEY NUMBER	GRAPHIC SYMBOL PRINTED	HEX VALUE
0	I	41	18	S	54	36	3	F3
1	R	67	19	C	55	37	+	4E
2	C	73	20	.	4B	38	-	71
3	H	9A	21	D	56	39	F	75
4	V	B7	22	@	57	40	T	70
5	Q	69	23	MO	58	41	$	63
6	TF	42	24	0	F0	42	$	64
7	CF	43	25	7	F7			
8	MJ	44	26	4	F4			
9	X	B2	27	I	72			
10	NB	45	28	1	F1			
11	M	46	29	8	F8			
12	‡	47	30	5	F5			
13	A	48	31	Ç	62			
14	QB	49	32	2	F2			
15	L	51	33	9	F9			
16	SF	52	34	6	F6			
17	B	53	35	U	9B			

Figure 4-10. 2980 Model 1 Special Character Set

Figure 4-11 represents the IMS/VS translation of numeric entry data from a 2980 Model 4 teller terminal. Except alphabetic entry of keys 11, 15 and 40 as indicated on the above chart, alphabetic entry data is presented to the application program in the standard EBCDIC character set.

KEY NUMBER	GRAPHIC SYMBOL PRINTED	HEX VALUE	KEY NUMBER	GRAPHIC SYMBOL PRINTED	HEX VALUE	KEY NUMBER	GRAPHIC SYMBOL PRINTED	HEX VALUE
0	¢	62	18	G	76	36	0	F0
1	L	67	19	$	64	37	5	F5
2	A	68	20	B	77	38	2	F2
3	C̄	55	21	/	61	39	9	F9
4	.	B7	22	P	78	40	‡	70
5	*	69	23	$\frac{Q}{B}$	49	41	6	F6
6	$	71	24	N	79	42	3	F3
7	I	4F	25	M̄	46			
8	$\frac{N}{B}$	45	26	J	B1	11	2	85
9	E	72	27	#	B9	15	3	B6
10	?	6F	28	X	B2	40	•	90
11	M	B8	29	O	B3			
12	C	73	30	K	B4			
13	-	74	31	7	F7			
14	F	75	32	,	80			
15	I	59	33	4	F4			
16	▽	65	34	1	F1			
17	△	66	35	8	F8			

Figure 4-11. 2980 Model 4 Special Character Set

Figure 4-12 illustrates the character presented to the application program when the corresponding function key is entered. Because of hardware design, IMS/VS cannot distinguish function key entry from a corresponding keyboard entry; the application programmer must therefore be warned of possible conflicts.

FUNC KEY	HEX VALUE	FUNC KEY	HEX VALUE	FUNC KEY	HEX VALUE	FUNC KEY	HEX VALUE
1	61	7	6F	13	B3	19	67
2	66	8	65	14	78	20	68
3	74	9	71	15	75	21	77
4	72	10	69	16	76	22	73
5	46	11	79	17	B1	23	F2
6	B2	12	45	18	B4	24	F3

Figure 4-12. 2980 Model 4 Function Key Translate Table

OUTPUT MESSAGE FORMAT

This section describes the output message formats supported by IMS/VS. IMS/VS supports two output destinations -- terminals and programs. The output formats are essentially the same for both but each has unique application program considerations.

Terminal Destination Output

Terminal destination output message segments originate in the application program and are sent to a logical terminal defined by a TP PCB. Each output message segment is enqueued to be sent by means of an insert call. The format of each segment is:

LL
is a 2-byte binary field representing the total length of the message segment, including LL, Z1, and Z2. The value of LL equals the number of bytes in text plus 4. The application program must fill in this count. If a size limit was defined for output segments of a transaction being processed, LL must not exceed the defined limit.

When PL/I is used, the LL field must be defined as a binary fullword. The value provided by the PL/I application program must represent the actual segment length minus 2 bytes. For example, if an output message segment is 16 bytes, LL is equal to 14 and represents the

sum of the length of LL (4 bytes minus 2 bytes, Z1 (1 byte), Z2 (1 byte), and TEXT (10 bytes).

Z1
is a 1-byte field that must contain binary zeros and is reserved for IMS/VS.

Z2
is a 1-byte field that must contain binary zeros and is reserved for IMS/VS. This Z2 definition applies to all terminals except terminals that use the Message Format Service (MFS), switched terminals, the 2260, the 2265-1, the 2265-2, some 2770 components, and the 2980.

- Z2 for Terminals Using MFS

For terminals supported by MFS, the field is used to denote the beginning of a logical page. See the IMS/VS Message Format Service User's Guide for a complete description of MFS formats.

- Z2 for Switched Devices

For switched devices, Z2 is a 1-byte binary field that can be used by the application program to request that the destination terminal be disconnected from the line after the message containing this request is written to the terminal. This disconnect request is recognized if present in any segment of the output message and is indicated by a value of X'80' in the Z2 field. This feature is not supported for a switched 3275 or 3741.

- Z2 for 2260/2265

For the 2260 and 2265, Z2 is a 1-byte binary field that denotes the type of WRITE command to be effected to the display screen. These types of WRITE commands affect the format of the display screen. For 2260 operation, the IBM 2848 Display Control must have the Line Addressing feature (4787) to accomplish Items 2 and 3 below.

	WRITE Command	Description	Designation
1.	WRITE	Indicates that it will begin writing output segment at the current cursor position	Binary zeros
2.	WRITE AT LINE ADDRESS (WLA)	Indicates that it will begin writing at the line specified (from one through fifteen depending on model)	X'01' through X'0F' for lines 1 through 15. Values above X'06' depend on the type of display station and/or its features.
3.	ERASE SCREEN START AT LINE	Indicates that the screen will be erased first; the output segment will be written at line address specified (line one through fifteen depending on model)	X'11' through X'1F' for lines 1 through 15. Values above X'06' depend on the type of display station and/or its features
4.	WRITE ERASE (WE)	Indicates the screen will be erased first; the output segment will be written starting on the upper left corner of the screen	X'20'

Any code not the same as that designated for the WRITE commands above defaults to binary zeros. No error messages are given. Since the screen may have up to 15 lines, line addresses may range from X'01' to X'0F' depending on model.

If video-paging is included in the system, multiple-page output messages may be designated by inserting an X'40' in the Z2 field of the segment representing the first segment of each page. This flag can be in addition to other video-screen format characters (for example, X'60' for first segment of page and write erase). To page forward and backward within a series of pages, these flags must be contained within a single message; no purge calls or get unique calls to the I/O PCB may be issued while building a multiple-page message. If a page flag is not found in the first segment of a message, subsequent page flags are ignored.

Example:

```
                    Z1 Z2   TEXT

Insert      |LL|00|60_|_SEG_1|
Insert      |LL|00|00_|_SEG_2|  Page 1

Insert      |LL|00|40_|_SEG_3|
Insert      |LL|00|00_|_SEG_4|  Page 2      Message 1
Insert      |LL|00|05_|_SEG_5|

Insert      |LL|00|52_|_SEG_6|  Page 3
```

These three screens can be displayed by the operator multiple times or not at all and may be displayed either in or out of sequence as the operator chooses.

```
                    Z1 Z2   TEXT

Insert      |LL|00|20 | SEG 1|
Insert      |LL|00|00 | SEG 2|  Page 1      Message 1

Purge       |LL|00|00 | SEG 3|
Insert      |LL|00|00 | SEG 4|  Page 1      Message 2
Insert      |LL|00|05 | SEG 5|

Purge       |LL|00|12 | SEG 6|  Page 1      Message 3
```

The above sequences would produce the same images to the terminal as the paged example above and would not require the paging feature. However, these images would be displayed once and only once and must be displayed in sequence.

- Z2 for 2980

Output messages requiring a passbook on a 2980 Model 1 or a 2980 Model 4, or requiring the insertion of the auditor's key on a 2980 Model 2 must contain a X'10' in the Z2 field of each output message segment. If the terminal PCB is the common buffer of the 2972 control field, the Z2 field value is ignored.

If the required passbook is not properly inserted in the output terminal when IMS/VS attempts transmission of a passbook message segment, the segment will be prefixed with two carrier returns, a FEED-OPEN (if 2980 Model 4), a MESSAGE LIGHT (if 2980 Model 1), or a TURN PAGE (if 2980 Model 4) indicator, and the required number of tab characters to position the type element to the passbook area of the output terminal. This allows the teller operator to insert the passbook to the proper print line. When the indicator is turned off (MESSAGE LIGHT or TURN PAGE), the type element tabs to the passbook area and begins printing the output message segment. IMS/VS positions the type element whenever the required passbook is not properly positioned in the output terminal, or if the passbook has been indexed beyond the last printable line when the passbook message segment was first transmitted. For these reasons, output message segments should not contain data for both the journal/audit tape area and the passbook area, since this may cause undesirable results. Output messages requiring the auditor's key on a 2980 Model 2 are not transmitted to the output terminal unless the auditor's key is inserted. Refer to the IMS/VS Operator's Reference Manual for procedures on receiving auditor key messages.

TEXT

is the output message segment in EBCDIC as it is transmitted to a specific logical terminal. The length of an output message segment is governed by the specific communication terminal receiving the output message. The maximum number of bytes for each message segment text is:

Terminal	Number of Bytes
1050, 2740-1, 2740-2, and 2741	130 (can be larger if CRs are embedded at 130 bytes or less). If Message Format Service (MFS) is used for the 2740/2741, refer to the IMS/VS Message Format Service User's Guide.
2260-1, 2265-1	960/screen*
2260-2 with 2848-1	240/screen*
2260-2 with 2848-2	480/screen*

Terminal	Number of Bytes (Continued)
	*Anything over will wrap the screen and overlay the first part of the message.
1053/2848	960; anything over will truncate.
1053/2845	240; anything over will truncate.
2770	Variable, based on component.
2265-2	960; anything over will wrap the screen and overlay the first part of the message.
card punch	80; anything over will truncate.
printer & paper tape punch	less than 32768.
2780	Variable
printer	80 or 120, or 144, based on 2780 printer specifications; anything over will truncate.
punch	80; anything over will truncate.
2972/2980	The following applies:
Common buffer	23
Terminal buffer	47
with buffer expansion	95
3270	Refer to the IMS/VS Message Format Service User's Guide.
3600, 3790	Variable; refer to the IMS/VS Advanced Function for Communications manual, or, if MFS is used, to the IMS/VS Message Format Service User's Guide.
3741	128 or less, based on 3741 specification; segments will be padded with blanks or truncated to this value.
3767, 3770 console, printers	Up to the message size. If Message Format Service (MFS) is used, the length of the message segment is defined by the user to MFS and is limited by the MSGQUEUE macro statement specification at system definition.
3770 punch	80; anything over will truncate.
7770	Any length.
33/35 Teletypewriter (ASR)	80
System/3 System/7	Variable, dependent on user's program in the System/3 or System/7.
System/370 console	126; anything over will truncate.

Terminal	Number of Bytes (Continued)
SYSOUT Print Direct	Variable, based on device; the segment is truncated to the record length specified for the particular device. When the output device is a printer, default segment maximum lengths are: 120* for 1443, 1403 132* for 3211
Spooled	Default segment size is 120*.

*These sizes do not include carriage-return characters as specified later in the section "Online Message Format Considerations." If carriage control is present, these maximums can be increased by 2.

2980 Optional Features

The reader should refer to Component Description: IBM 2972 Models 8 and 11 General Banking Terminal Systems, GL27-3020 for a complete discussion of the optional features available on a 2980 Model 4 and how an application program might make use of them. The discussion following is limited to the use of those features in the IMS/VS environment.

• 2980 Message Lights

The 2980 Model 1 and Model 4 teller terminals incorporate a message light feature that prevents the printing of an output message at the terminal until some operator action is taken. An application program can utilize this message light feature on a 2980 Model 1 by inserting a X'17' in the text of the output message segment. The data following the message light character will not be printed at the terminal until the terminal operator presses the message light key. Any combination of six message lights at a 2980 Model 4 teller terminal can be caused to turn on by the insertion of a two-character message light sequence as the first two (or only) characters of an output message segment. The data following the message light sequence will not be printed until the terminal operator presses the message light key. The message light sequence for a 2980 Model 4 consists of an X'17' followed by any character whose hexadecimal value is greater than X'3F'; an X'40' will be substituted for invalid values. Refer to the above mentioned SRL for detailed information on the use of and setting of message lights on the 2980 Model 4. IMS/VS precedes all system-generated messages with an X'1740' if the message is for a 2980 Model 1 or 2980 Model 4.

• 2980 Function Keys

IMS/VS cannot distinguish a function key entry from a data key entry that causes transmission of the same character to the CPU. Figure 4-13 lists the character received by the application program when the corresponding function key is entered. The application programmer must be aware that, since function keys are an optional feature, in each instance there is a corresponding keyboard entry which results in the same character being received. No direct facility is provided which would give a unique distinction to the application program between entry of function keys 23 and 24 and the graphic numeric characters 2 and 3, respectively. To do so would require the terminal operator to enter alpha shift to enter these numbers. (The application programmer

may require operator entry of keyboard keys 11 and 15 in alpha shift for those numbers if such distinction is necessary.)

Online Message Format Considerations -- MFS Not Used

When Message Format Service (MFS) is not used, it is the application programmer's responsibility to provide all horizontal and vertical format control required to properly display an output message. An output message can contain multiple message segments. It is not necessary to include a logical terminal name in an output message since the destination is determined by the logical terminal PCB.

Certain device control characters must be inserted into an output message when it is desired to format a message at a terminal output device. Output message formatting for the devices supported by IMS/VS may be accomplished as follows:

- When output is to be printed on a typewriter-like device (for example, 2740), the following hexadecimal characters found within the output text function as indicated:

  X'05' Skip to tab stop (HT), but stay on same line.

  X'15' Start new line (NL) at left margin (carriage return).

  X'25' Skip to new line (LF), but stay at same print position horizontally.

  The most efficient way to skip multiple lines is by including a combination of one NL character and multiple LF characters.

  Forms feed control can be provided for a 1052 or 1053 printer by including the forms control characters as the first two bytes of output message segment text. Output message segments may contain multiple typed lines (carriage returns should be embedded at 130 characters or less).

- When output is to be printed on a 1050 printer and vertical forms control is used, the forms control sequence must be the first two characters in a segment.

- When output is to be printed on a 2780 or local printer, a message segment is considered to be a print line, and message text over the designated printer's capability is truncated on output. NL and LF characters are ignored. Control other than single line spacing (which is default) may be achieved by inserting an ESC character (X'27') as the first character of the output message segment text, followed by one of the following carriage control characters (the X'27' and the carriage control characters are not considered part of the message text for truncation purposes):

  S -- Double space after this line is printed.

  T -- Triple space after this line is printed.

  A through L -- Skip to channel 1 through 12 after this line is printed (local print).

  A through H -- Skip to channel 1 through 8 after line is printed (2780).

  M -- Suppress spacing after printing (local print only).

- When output is to be written to the OS/VS system console, a message segment is considered to be a print line. If the output message segment text does not begin with the characters DFS followed by three numeric characters, IMS/VS inserts a prefix of DFS000I. All embedded NL characters are replaced by blanks (X'40') as required by OS/VS WTO. Output message segment text (including DFS000I, if inserted by IMS/VS) in excess of 126 characters is truncated as required by OS/VS WTO.

- When output is to be punched (with, for example, the 2780 terminal or the 3770 card punch), a message segment is considered to be a card, and message text over 80 characters is truncated upon output.

- When output is to be displayed on a 2260-1, 2260-2, or 2265-1, the following are output message considerations:

  - An output message can be composed of multiple segments that make up a single screen. Total segment and message length is variable:

Device	Lines per Screen	Bytes per Segment	Bytes per Message
2260-1, 2265-1	12	80	960
2260-2 (2848-1)	6	40	240
2260-2 (2848-2)	12	40	480

  - If the length of the message exceeds the capacity of the screen, the screen will wrap, destroying the data previously displayed. New Line (NL) characters are honored; Line Feed (LF) characters are ignored.

  - Multiple screen output is allowed.

  - Each segment can specify a write-type request (Z2 field bits). IMS/VS ignores WRITE-ERASE requests except on the first segment of an output message.

- When output is to be displayed on the 2265-2 component of a 2770 system, the following are output message considerations:

  - An output message can be composed of multiple segments that make up a single screen (960 bytes).

  - If the length of the message exceeds the capacity of the screen, the screen will wrap, destroying data previously displayed. NL characters are honored except as described below. LF characters are not honored.

  - Multiple screen output is allowed.

  - An NL character in text that is being written on the last line of the display screen does not cause a screen wrap operation to occur. The NL character(s) is displayed on the last line of the screen.

  - An SMM symbol on the screen after an NL symbol does not transfer data if the ENTER key is pressed.

  - Each output message segment may specify its write-type request.

- When output is to be printed on a 2770 printer component, the following are output message considerations:

  - Segments over the printer line length cause an automatic hardware carriage return before printing of the remainder of the segment.

- If no control operations are embedded in the message segment, the printer is single spaced by the insertion of an IRS character.

- If a trailing NL character is in the segment, the printer component double spaces after printing the line.

- Explicit carriage control can be accomplished by limiting segment length to the length of a print line (this depends on the printer component type and features) and inserting an ESC character (X'27') as the first character of the output message segment text, followed by one of the carriage control characters for the 2770 printer component. See System Components: IBM 2770 Data Communications System, GA27-3013, for a description of these codes.

- When output is to be punched on the 2770 paper tape punch component, the following are output message considerations:

  - IMS/VS inserts an end of media character at the end of each output message to the paper tape punch.

  - If segments whose size is larger than the value specified on the PTSFG= operand of the TERMINAL statement during system definition are sent to this component, the segment will not be properly deblocked on subsequent reentry to IMS/VS.

- When output is to be printed on a 2980 terminal, the following hexadecimal characters function as indicated:

  X'05' Skip to tab stop (HT), but stay on same line.

  X'15' Start new line (NL) at left margin, if the present position of the type element is within the audit/journal tape area; or the type element will be repositioned at the intermediate carriage stop, if the present position of the type element is within the passbook area. In the latter instance, printing will resume on the same print line.

  X'25' When the output message segment is destined for the passbook area of the terminal, this character will cause the start of a new line at the intermediate carriage stop. IMS/VS will ensure that the passbook is properly inserted at a printable line on all transmissions to the passbook area.

  Output message segments may contain multiple print lines. Care should be taken to insert carriage returns (X'15') and/or passbook index (X'25') characters in long message segments to prevent typing past the audit/journal tape or passbook.

- When the output device is a 7770-3 line, it is the responsibility of the application programmer to format the output message with 7770 vocabulary Drum Address characters as required for the application.

Output device independence may be achieved by generating output message segment text no greater than 80 bytes, including a trailing NL character. Output message segment text should not contain any forms or carriage control characters. If video terminals are included in a system, no more data than will fit on a single screen should be generated per output message. It should be noted that the output device independence described above may restrict efficient use of certain output devices, and may restrict use of special output device functions.

Program-to-Program Message Switching

An output message destined to another application program is a program-to-program message switch. The message switch destination can be specified during PSB generation or during program execution using the change call. The destination must be a transaction code defined during system definition. The receiving program must contain an I/O area large enough to hold the largest segment sent by the transmitting program.

Insert calls are used to create the segments of a program-to-program message. When inserting a segment, an alternate PCB must be used. The destination of the alternate PCB must be set prior to the first insert call.

Message security procedures may or may not be invoked during program-to-program message switching. They are invoked when a change call is used to set the destination; the current entering terminal must be authorized to enter the transaction code set by the change call. No checking is performed on insert calls.

The format of a message switch segment is:

The format is essentially the same as for output messages to logical terminals. The following areas should be noted:

- Z1 and Z2 are one-byte fields that must contain binary zeroes; the use of Z1 and Z2 is reserved for IMS/VS.
- TEXT is the message segment that is to be sent to the specified destination.

Since IMS/VS does not prefix a switched message with a transaction code, the application program can put the transaction code at the beginning of the first segment. This assures that messages arriving at the destination are in the same format, whether originating from a program or from a terminal.

TELEPROCESSING OR BATCH/TELEPROCESSING ENVIRONMENTS

ANS COBOL MESSAGE PROGRAM STRUCTURE

Figure 4-13 outlines the fundamental parts of an ANS COBOL message processing program. Each item should be considered when designing a message program. This program processes an inquiry from a terminal, makes a reference to a data base for information, and sends a message to a different terminal or to an application program.

```
REF
NO.
       ENVIRONMENT DIVISION.
            •
       DATA DIVISION.
       WORKING-STORAGE SECTION.
 1         77 GU-CALL PICTURE XXXX VALUE 'GU  '.
           77 ISRT-CALL PICTURE XXXX VALUE 'ISRT'.
           77 CT PICTURE S9(5) COMPUTATIONAL VALUE +4
            •
 2         01  SSA-NAME.
            •
 3         01  MSG-SEG-IO-AREA.
           01  DB-SEG-IO-AREA.
           01  ALT-MSG-SEG-OUT.
       LINKAGE SECTION.
 4         01  IO-PCB.
           01  ALT-PCB.
           01  DB-PCB.

       PROCEDURE DIVISION.

 5         ENTRY 'DLITCBL' USING IO-PCB, ALT-PCB, DB-PCB.
            •
 6         CALL 'CBLTDLI' USING GU-CALL, IO-PCB,
               MSG-SEG-IO-AREA.
            •
 7         CALL 'CBLTDLI' USING GU-CALL, DB-PCB,
               DB-SEG-IO-AREA, SSA-NAME.
            •
 8         CALL 'CBLTDLI' USING ISRT-CALL, ALT-PCB,
               ALT-MSG-SEG-OUT.
            •
 9         GOBACK.

10         COBOL - LANGUAGE INTERFACE
```

Figure 4-13. COBOL Message Program Structure

The following explanations are keyed to the numbers along the left side of Figure 4-13.

1. A 77 level or 01 level working storage statement defines each of the call functions used by the message program. Each picture clause is defined as four alphameric characters and has a value assigned for each function (for example, ISRT).

2. An 01 level working storage statement describes each segment search argument (SSA) used for a data base call. An example of an SSA definition, with a lowercase b representing a blank and a lowercase v representing the symbolic value in the field, is:

```
01  SSA-NAME.
    02  SEG-NAME     PICTURE X(8)  VALUE 'ROOTbbbb'.
    02  SEG-QUAL     PICTURE X     VALUE '('.
    02  SEG-KEYNAME  PICTURE X(8)  VALUE 'KEYbbbbb'.
    02  SEG-OPERATOR PICTURE XX    VALUE 'b='.
    02  SEG-KEY-VALUE PICTURE X(6) VALUE 'vvvvvv'.
    02  SEG-END-CHAR  PICTURE X    VALUE ')'.
```

When the above COBOL syntax is decoded and placed in storage, it will be in a data string as follows:

```
ROOTbbbb(KEYbbbbbb=vvvvvv)
```

(For further discussion, see the section "Segment Search Arguments" in the "Data Base Batch Programming" chapter of this manual.)

3. An 01 level working storage statement describes each segment I/O work area within the message program.

4. An 01 level linkage section entry describes the PCB statement, first for the input terminal for the current message being processed (the I/O PCB), second for each output destination other than the input terminal (alternate PCBs), and third for each data base. It is through this linkage description that a COBOL program can access the status codes after a DL/I call.

5. This is the message program entry point and must be the first executable COBOL statement in the procedure division. There must be a name for every PCB used by the message program. The first PCB name must be for the I/O PCB. The remaining PCB names must be specified in the same order, following the I/O PCB, as they are presented in the program's associated PSB generation. The PCB names could be specified in the linkage section in the same order, but this is not a requirement.

6. This is a typical call used to read the input (source) logical terminal. The first time this call is executed with function equal to get unique, the first segment of the message that caused the message program to be scheduled is brought into this program. If the input message consists of more than one segment, subsequent segments can be obtained with a similar call but with the function equal to get next.

7. This call is used to access data from a data base. The format is the same as that in Item 6 above, except that the PCB refers to a data base and the segment search arguments define a particular data base segment.

8. This call is used to reply to an output destination other than the terminal representing the source of the input message. If the output destination is the input terminal, this call must utilize the I/O PCB.

9. This operation causes the message program to return control to the IMS/VS control facilities.

10. A language interface (DFSLI000) is provided by IMS/VS for all COBOL programs. This module must be link-edited to the message processing program after compilation and provides a common interface to IMS/VS and DL/I for all call statements.

    The language interface function of IMS/VS is reenterable and compatible with that of IMS/360 Version 2. To take advantage of the reenterable capability, application modules from IMS/360 must be re-linkedited, replacing the IMS/360 Version 2 language interface with that of IMS/VS. The IMS/360 Version 1 language interface is <u>not</u> compatible with IMS/VS.

PL/I OPTIMIZING COMPILER MESSAGE PROGRAM STRUCTURE

Figure 4-14 outlines the fundamental parts of a PL/I optimizing compiler message processing program. Each item should be considered when designing a message program. This program processes an inquiry from a terminal, makes a reference to a data base for information, and sends a message to a different terminal or to an application program.

```
REF
NO.

          /* ------------------------------------------------ */
          /*                    ENTRY POINT                   */
          /* ------------------------------------------------ */
 1            DLITPLI:  PROCEDURE(IO_PTR,ALT_PTR,DB_PTR)
              OPTIONS (MAIN);

 2          DECLARE FUNC_GU CHARACTER(4) STATIC INITIAL('GU');
            DECLARE FUNC_ISRT CHARACTER(4) STATIC INITIAL('ISRT');

 3          DECLARE SSA_NAME...

 4          DECLARE MSG_SEG_IO_AREA CHAR(24);
            DECLARE DB_SEG_IO_AREA CHAR(180);
            DECLARE ALT_MSG_SEG_OUT CHAR(24);

 5          DECLARE 1 IO_PCB BASED(IO_PTR),...;
            DECLARE 1 ALT_PCB BASED(ALT_PTR),...;
            DECLARE 1 DB_PCB BASED(DB_PTR),...;

 6          DECLARE THREE FIXED BINARY(31) STATIC INITIAL(3);
            DECLARE FOUR FIXED BINARY(31) STATIC INITIAL(4);

 7          CALL PLITDLI(THREE,FUNC_GU,IO_PTR,MSG_SEG_IO_AREA);

 8          CALL PLITDLI(FOUR,FUNC_GU,DB_PTR,DB_SEG_IO_AREA);

 9          CALL PLITDLI(THREE,FUNC_ISRT,ALT_PTR,ALT_MSG_SEG_OUT);

10          END DLITPLI;


11          PL/I - LANGUAGE INTERFACE
```

Figure 4-14. General PL/I Optimizing Compiler Message Program Structure

The following explanations are keyed to the numbers along the left side of Figure 4-14:

1. This is the main standard entry point to a PL/I optimizing compiler message program. There must be a pointer for every PCB used by the message program. The first PCB pointer must be for the I/O PCB. The remaining PCB pointers must be specified in the same order, following the I/O PCB, as they are presented in the program's associated PSB generation.

2. These declarations define the call functions used by the PL/I message program. Each character string is defined as four alphameric characters and a value assigned for each function (for example, ISRT). Other constants and working areas may be defined in this manner.

3. This declaration defines storage for SSAs. In the following example, the SSA is declared as a structure; other methods can be used (see the section "General Characteristics of Segment Search Arguments" in Chapter 3 of this manual).

   Example (lower case "b" represents a blank and lower case "v" represents the symbolic value in the field):

```
DCL 1  SSA_NAME,
       2    SEG_NAME CHAR(8)         INIT('ROOT'),
       2    SEG_QUAL CHAR(1)         INIT('('),
       2    SEG_KEY_NAME CHAR(8)     INIT('KEY'),
       2    SEG_OPERATOR CHAR(2)     INIT('b='),
       2    SEG_KEY_VALUE CHAR(6)    INIT('vvvvvv'),
       2    SEG_END_CHAR CHAR(1)     INIT(')');
```

4. The I/O area is most efficiently passed to DL/I as a fixed-length-character string or through a pointer variable; other methods, however, can be used (see the PL/I example under the section "I/O Work Area" in Chapter 2 of this manual). An example follows:

```
    DCL MAST_SEG_IO_AREA CHAR(100);
```

5. A level 1 declarative describes the PCB statement first for the input terminal for the current message being processed (the I/O PCB), second for each output destination other than the input terminal (alternate PCBs), and third for each data base. It is through this description that a PL/I program can access the status codes after a DL/I call. (For the PL/I optimizing compiler, the PCBs must be BASED structures.)

6. This is a descriptive statement used to identify a binary number (fullword) that represents the "parameter count" of a call to DL/I. The parameter count value equals the remaining parameters following the parameter count set off by commas.

7. This is a typical call used to read the input (source) logical terminal. The first time this call is executed with function equal to get unique, the first segment of the message that caused the message program to be scheduled will be brought into this program. If the input message consists of more than one segment, subsequent segments can be obtained with a similar call but with the function equal to get next.

8. This call is used to access data from a data base. The format is the same as the one in Item 7 above, except that the PCB refers to a data base and the segment search argument defines a particular data base segment.

9. This call is used to reply to an output destination other than the terminal representing the source of the input message. If the output destination is the input terminal, this call must utilize the I/O PCB.

10. This END statement causes the message program to return control to the IMS/VS control facilities. Another statement that causes the message program to return control to the IMS/VS control facilities is the RETURN statement. The RETURN statement may or may not immediately precede the END statement.

11. A language interface (DFSLI000) is provided by IMS/VS for all programming languages. This module must be link-edited to the compiled message program and provides a common interface to IMS/VS and DL/I.

    The language interface function of IMS/VS is reenterable and compatible with that of IMS/360 Version 2. To take advantage of the reenterable capability, application modules from IMS/360 must be re-linkedited, replacing the IMS/360 Version 2 language interface with that of IMS/VS. The IMS/360 Version 1 language interface is not compatible with IMS/VS.

ASSEMBLER LANGUAGE MESSAGE PROGRAM STRUCTURE

The structure of an Assembler Language message program is the same as for the Assembler Language batch program described in the section "Assembler Language Batch Program Structure" in the "Data Base Batch Programming" chapter of this manual. In addition, the user should remember that an Assembler Language message program receives, upon entry, a PCB parameter list address in register 1. The first address in this list is a pointer to the I/O PCB. Any alternate PCB addresses follow, and finally any data base PCB addresses. Bit 0 of the last address parameter is set to 1 in accordance with operating system conventions for variable parameter lists.

ABENDS ISSUED BY APPLICATION PROGRAMS

Actions taken by IMS/VS on all types of application program abends are described in the IMS/VS System/Application Design Guide.

If an application program is going to issue the ABEND macro, the STEP parameter must not be used. The use of the STEP parameter prevents the message or batch message region from notifying the IMS/VS control region that an application program has abended. This in turn may prevent the release of resources or a normal checkpoint shutdown.

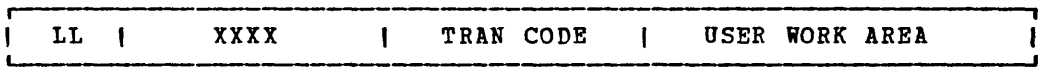
CHAPTER 5. DATA COMMUNICATION: CONVERSATIONAL PROCESSING

Conversational processing allows a user's application program to retain information acquired through interchanges with a terminal even though the application program leaves the message region between interchanges. Special facilities are provided in IMS/VS to allow the retention of information. Data base facilities are not required for information retention.

The conversational option is specified during IMS/VS system definition so that IMS/VS can relate to transaction codes that utilize the conversational mode. When an application program that processes a conversational transaction type is scheduled, a get unique (GU) call against the I/O PCB causes the contents of a Scratchpad Area (SPA) of user-defined length to be passed from IMS/VS to an I/O area defined in the user's application program. Subsequent get next (GN) calls cause the message segments entered from the terminal to be passed to another I/O area defined in the user's application program. Data saved in a SPA can be in any form: bit string, character, binary numbers, or packed decimal.

SCRATCHPAD AREA FORMAT

The SPA format is:



where:

LL
: is a halfword binary field containing the total number of characters in the SPA, including LL, XXXX, TRAN CODE, and USER WORK AREA. This field should not be modified by the user.

  When PL/I is used, the LL field must be defined as a binary fullword. The value contained in the LL field is the actual scratchpad area length minus 2 bytes. For example, if the scratchpad area is 26 bytes, LL is equal to 24 and represents the sum of LL (4 bytes minus 2 bytes), XXXX (2 bytes), TRAN CODE (8 bytes), and text (10 bytes).

XXXX
: is a 4-byte area reserved for IMS/VS. XXXX must not be modified by the user.

TRAN CODE
: is an 8-byte field containing the transaction code that caused the program to be scheduled. The transaction code can be from 1 to 8 bytes, left-justified, and padded with blanks.

  If this code is changed by the user, a different program is scheduled for the next message input from the terminal.

  The transaction code does not appear in the message segment. (When option 3 of the Message Format Service is used, the transaction code is not removed. Refer to the IMS/VS Message Format Service User's Guide.)

USER WORK AREA
is a variable-length area 14 bytes less than that defined by the user during IMS/VS system definition for each conversational transaction code and cleared to binary zeros on first entry to the application program for this conversation. This area is for retaining user information (for example, intermediate calculations or data retrieved through one or more data base calls) required by an application program.

INPUT MESSAGE FORMAT

From a terminal operator's viewpoint, the format of the input message segment that starts the conversation is the same as any nonconversational transaction-type message. IMS/VS removes the transaction code from the first message segment (except as noted above) and always places it in the scratchpad area. The first message segment is left-justified to remove the transaction code. (Transaction code formats are described under "Message Formats" in the chapter "DC: Application Programming".) It is retrieved by the first GN call issued after the GU call that retreived the scratchpad. Additional message segments of an input message are formatted the same as for nonconversational processing.

EXAMPLE

1. First conversational message segment entered at input terminal:

   ```
   CONV +32546.12-1235.27
   ```

2. First CALL statement using PL/I:

   ```
   CALL PLITDLI (THREE,GET_UNIQUE_FUNC,IO_PCB,SPA_AREA);
   ```

3. The SPA_AREA now looks like this after the first GU call:

		TRAN CODE	USER WORK AREA
			binary zeros
LL	XXXX	CONVbbbb	0----------0

4. The first segment of the conversational message now looks like this:

   ```
   +32546.12-1235.27
   ```

   Thus, to bring this text into the application program I/O work area, a GN call must be made.

5. Second PL/I CALL statement using a GN call function to obtain the text of the first message segment:

   ```
   CALL PLITDLI (THREE,GET_NEXT_FUNC,IO_PCB,WORK_AREA):
   ```

   This brings the text as shown in item 4 above into the I/O work area of the application program.

6. To get subsequent message segments, the CALL statement is the same as in item 5 above.

SAVING INFORMATION IN THE SPA

After the input scratchpad area and input message have been obtained, one or more data base calls may be made and one output message may be built. The application program may wish to retain data entered from the terminal or obtained from data bases. This data is saved in the user work area portion of the scratchpad.

If the application program modifies or initializes any SPA fields, it must return the SPA to IMS/VS before issuing another GU or terminating. An SPA is returned to IMS/VS by inserting it to the I/O PCB.

The insert (ISRT) call for PL/I is handled as follows:

```
CALL PLITDLI (THREE,ISRT_FUNC,IO_PCB,SPA_NAME);
```

or, in ANS COBOL:

```
CALL 'CBLTDLI' USING ISRT, IO-PCB, SPA-NAME.
```

OUTPUT MESSAGE FORMAT

A response to the originating terminal is required to allow the conversation to continue. The terminal operator is prevented from entering more data to be processed (except IMS/VS commands) until he has received this response.

The response is accomplished in one of two ways:

1. The conversational program can issue ISRT calls to the I/O PCB or an alternate PCB defined as ALTRESP=YES prior to the next GU call or program termination.

2. Control may be passed to another conversational program by inserting the SPA and a message to an alternate PCB.

   The switched-to-conversational program may then perform 1 above (which will wait for terminal input) or perform 2 again (program switch).

The output message segment format for a conversational application program is the same as for any nonconversational output message format.

PASSING CONVERSATIONAL CONTROL TO ANOTHER CONVERSATIONAL PROGRAM

Conversational message processing programs can pass control of a conversation to another conversational program. Two methods of passing control are supported:

- The program in control can change the transaction name in the SPA before returning the SPA to IMS/VS. IMS/VS will route the next terminal input to the program that handles the specified transaction code. Any intervening program switches can change the transaction name in the SPA.

- For a program-to-program switch, the program in control can insert a message to an alternate PCB that has its destination set to another conversational program. The SPA must be the first segment inserted to the alternate PCB; neither the SPA nor a response can be returned to IMS/VS through the I/O PCB or response alternate PCB if this is done.

If the new program requires a larger or smaller SPA, and the conversation did not start with a fixed-length SPA, IMS/VS will intercept the SPA and extend or truncate it for the new program, while preserving the data that may have been truncated.

If differing sizes for SPAs have been defined at system definition for disk and incore SPAs, care must be exercised by the user to prevent scheduling conversational programs within a series of programs which require SPAs larger than the maximum SPA size allowed by the original program to be scheduled. The first program scheduled sets the type of SPA that will be used for the duration of the conversation.

Example: Main storage maximum defined as 100 bytes; disk maximum defined as 1000 bytes.

TRAN A - main storage 50 SPA bytes TRAN C - disk 100 SPA bytes

TRAN B - main storage 75 SPA bytes TRAN D - disk 1000 SPA bytes

If TRAN A or TRAN B is the first conversational program called by a terminal operator, the conversation can switch control to TRAN A, B, or C, but not to TRAN D, since D requires a larger SPA than the maximum allowed for incore SPAs.

If TRAN C or TRAN D is the first conversational program called by the terminal operator, control can switch to any other transaction.

TERMINATING A CONVERSATION

A conversation is terminated by either the conversational program, terminal operator, master terminal operator, or IMS/VS. A conversational program terminates a conversation by:

- Blanking the transaction code in the SPA and returning the SPA to IMS/VS through an ISRT call. This terminates the conversation as soon as the terminal has received the response.

- Inserting the name of a nonconversational transaction code in the transaction code field of the SPA and returning the SPA to IMS/VS through an ISRT call to the I/O PCB. This causes the conversation to remain active until the next message is entered by the terminal. Except for MFS formatting option 3 messages, the transaction code will be inserted into the input message from the SPA. This message will then be routed to the named transaction code prior to terminating the conversation; the nonconversational program will not get the SPA.

The terminal operator terminates a conversation by:

- Entering a /EXIT command or /EXIT CONVnnn from the terminal that is participating in the conversation.

- Entering the /HOLD command from the terminal that is participating in the conversation. This action temporarily suspends operation and allows the terminal operator to enter other transactions while the first conversation is being "held" inactive. The response to a /HOLD command furnishes the terminal operator with an identifier

of his conversation so that he can reactivate it later by means of the /RELEASE command. A held conversation is considered to be active when the number of current conversations is calculated.

The master terminal operator terminates a conversation by:

- Entering a /START LINE (no PTERM specified) for a terminal in conversation.

IMS/VS terminates a conversation if, after a successful GU or insertion of the SPA, a conversational application program fails to insert a message. When this situation occurs, IMS/VS sends the message DFS3272I NO RESPONSE, CONVERSATION TERMINATED to the terminal, terminates the conversation, and completes synchronization point processing.

RULES FOR WRITING CONVERSATIONAL PROGRAMS

GENERAL

- The first 6 bytes of the SPA cannot be modified in any way by the application program. (IMS/VS uses these 6 bytes to identify the SPA.)
- If a conversation is started for a transaction with a fixed-length SPA, all succeeding transactions used for the duration of the conversation must be defined with and use fixed-length SPAs of the same length.
- The SPA transaction code (beginning in position 7) can be changed by the application program to switch control to a new transaction upon receipt of the next input from the terminal. The conversation is terminated if this transaction is a nonconversational transaction or if it is blanked.
- If modified by an application program, the SPA must be returned to IMS/VS through an ISRT call or the SPA against which a GU call was issued will be reused.
- The SPA cannot be returned to IMS/VS more than once. (Example: ISRT to I/O PCB, then ISRT to alternate PCB for program-to-program message switch.)
- The SPA cannot be inserted to an alternate PCB representing a nonconversational transaction or logical terminal. A response alternate PCB is permissible if it represents the input PTERM.
- If control is being given to another conversational program through a program-to-program message switch, the SPA must be the first segment inserted. (Example: ISRT to alternate PCB defined as a conversational transaction.)

MESSAGE RESPONSE

- An output message response to the I/O PCB or to an alternate PCB defined as ALTRESP=YES is required, unless the SPA has been passed to another conversational program through an insert to an alternate PCB, in which case the response must be given by that program. For additional information, see the section "Alternate PCB" in the "Data Communication: Application Programming" chapter of this manual.

- Only one message response is allowed for each conversational entry. This message can consist of as many segments as required; however, a PURG call cannot be issued to generate multiple output messages. If a PURG call is issued, the synchronization-point processor returns the AZ status code and does not process the call.

- Conversational programs must be designed to handle the condition in which the first GU call to the I/O PCB may produce no message to process. This condition can occur if the operator cancels the conversation through an /EXIT command, prior to the program issuing a GU call, if this was the only message in the queue to be processed.

- It is not permissible to use a PURG call for an I/O PCB, response alternate PCB, or an alternate PCB that represents another conversational transaction.

CHAPTER 6. APPLICATION PROGRAM EXAMPLES

The examples of application programs included in this chapter represent application programs that normally operate in an IMS/VS environment. At least one of the programming languages (COBOL, PL/I, or Assembler) has been selected for each type of application program. Most of the application programs represent source programs used in the sample problem included in the IMS/VS Installation Guide.

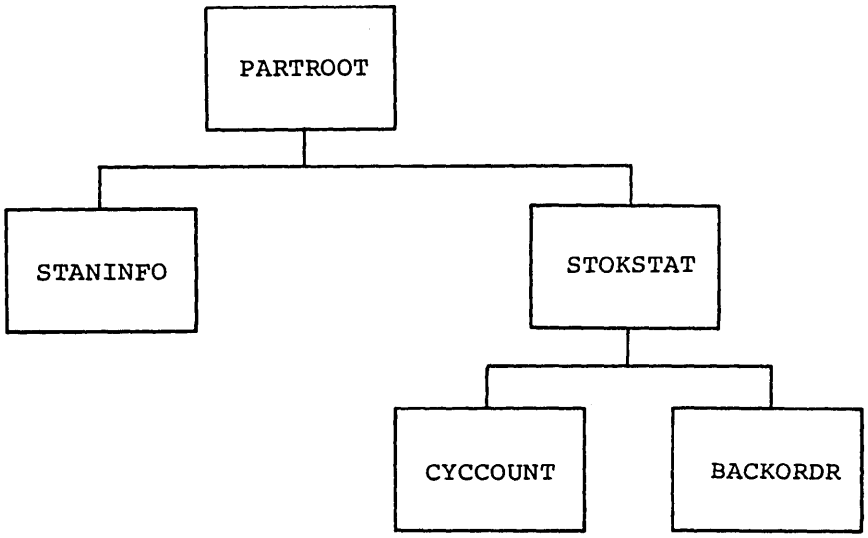
The following types of programs are presented:

Type	Language
Data Base Load Program	COBOL
Data Base Dump Program	Assembler
Batch Processing Program	COBOL and Assembler
Message Processing Program	COBOL
Conversational Processing Program	PL/I

DATA BASE LOAD PROGRAM EXAMPLE

ANS COBOL APPLICATION PROGRAM

In this example, the batch application program DFSSAM01 uses the SYSIN data to load a data base, named DI21PART, whose hierarchical logical data structure is:



```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    'DFSSAM01'
       AUTHOR.          DON TRUDELL.
          REMARKS.  DATA BASE LOAD PROGRAM.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.     IBM-360-H50.
       OBJECT-COMPUTER.     IBM-360-H50.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT  INPUT-FILE       ASSIGN TO UT-S-INPUT.
       DATA DIVISION.
       FILE SECTION.
       FD  INPUT-FILE
           RECORD CONTAINS 80 CHARACTERS
           BLOCK CONTAINS 0 RECORDS
           RECORDING MODE IS F
           LABEL RECORDS ARE OMITTED
           DATA RECORD IS INPUT-RECORD.
       01  INPUT-RECORD.
           02 INP-SEG-NAME          PICTURE X(08).
           02 FILLER                PICTURE X(01).
           02 INP-DATA              PICTURE X(67).
           02 INP-SEQUENCE-NO       PICTURE X(04).
       WORKING-STORAGE SECTION.
       01  DL1-FUNCTION             PICTURE X(04).
       01  PREV-SEG-NAME            PICTURE X(08)            VALUE SPACE.
       01  PREV-SEQUENCE-NO         PICTURE X(04)            VALUE SPACE.
       01  BUILD-SEGMENT-AREA.
           02 BUILD-DATA-AREA       OCCURS 14 TIMES
                                    PICTURE X(67).
       01  MISC-ARITHMETIC-FIELDS   USAGE COMPUTATIONAL.
           02 SUB-1                 PICTURE S9(02)           VALUE ZEROS.
       01  SEG00010-SSA.
           02 SEG-NAME-00010        PICTURE X(08)  VALUE 'PARTROOT'.
           02 BEGIN-OP-00010        PICTURE X(01)  VALUE '('.
           02 KEY-NAME-00010        PICTURE X(08)  VALUE 'PARTKEY '.
           02 REL-OPER-00010        PICTURE X(02)  VALUE ' ='.
           02 KEY-VALUE-00010       PICTURE X(17).
           02 END-OP-00010          PICTURE X(01)  VALUE ')'.
       01  SEG00060-SSA.
           02 SEG-NAME-00060        PICTURE X(08)  VALUE 'STANINFO'.
           02 BEGIN-OP-00060        PICTURE X(01)  VALUE '('.
           02 KEY-NAME-00060        PICTURE X(08)  VALUE 'STANKEY '.
           02 REL-OPER-00060        PICTURE X(02)  VALUE ' ='.
           02 KEY-VALUE-00060       PICTURE X(02).
           02 END-OP-00060          PICTURE X(01)  VALUE ')'.
       01  SEG02000-SSA.
           02 SEG-NAME-02000        PICTURE X(08)  VALUE 'STOKSTAT'.
           02 BEGIN-OP-02000        PICTURE X(01)  VALUE '('.
           02 KEY-NAME-02000        PICTURE X(08)  VALUE 'STOCKEY '.
           02 REL-OPER-02000        PICTURE X(02)  VALUE ' ='.
           02 KEY-VALUE-02000       PICTURE X(16).
           02 END-OP-02000          PICTURE X(01)  VALUE ')'.
       01  SEG02200-SSA.
```

```
           02 SEG-NAME-02200          PICTURE X(08)  VALUE 'CYCCOUNT'.
           02 BEGIN-OP-02200          PICTURE X(01)  VALUE '('.
           02 KEY-NAME-02200          PICTURE X(08)  VALUE 'CYCLKEY '.
           02 REL-OPER-02200          PICTURE X(02)  VALUE ' ='.
           02 KEY-VALUE-02200         PICTURE X(02).
           02 END-OP-02200            PICTURE X(01)  VALUE ')'.
       01  SEG02300-SSA.
           02 SEG-NAME-02300          PICTURE X(08)  VALUE 'BACKORDR'.
           02 BEGIN-OP-02300          PICTURE X(01)  VALUE '('.
           02 KEY-NAME-02300          PICTURE X(08)  VALUE 'BACKKEY '.
           02 REL-OPER-02300          PICTURE X(02)  VALUE ' ='.
           02 KEY-VALUE-02300         PICTURE X(10).
           02 END-OP-02300            PICTURE X(01)  VALUE ')'.
       01  SEG00010-INSERT-AREA.
           02 FILLER                  PICTURE X(050).
       01  SEG00060-INSERT-AREA.
           02 FILLER                  PICTURE X(61).
           02 RIGHT-MAKE-SPAN         PICTURE S9(03).
           02 FILLER                  PICTURE X(06).
           02 WRONG-MAKE-SPAN         PICTURE 9(03).
           02 FILLER                  PICTURE X(12).
       01  SEG02000-INSERT-AREA.
           02 FILLER                  PICTURE X(160).
       01  SEG02200-INSERT-AREA.
           02 FILLER                  PICTURE X(025).
       01  SEG02300-INSERT-AREA.
           02 FILLER                  PICTURE X(075).
       LINKAGE SECTION.
       01  PCB-AREA-1.
           02 DBD-NAME                PICTURE  X(08).
           02 SEGMENT-LEVEL           PICTURE  X(02).
           02 STATUS-CODES            PICTURE  X(02).
           02 PROCESS-OPTIONS         PICTURE X(04).
           02 FILLER                  PICTURE S9(05)  COMPUTATIONAL.
           02 SEG-NAME-FEEDBACK       PICTURE  X(08).
       PROCEDURE DIVISION.
           ENTRY 'DLITCBL' USING  PCB-AREA-1.
           DISPLAY 'START DB LOAD'  UPON CONSOLE.
           OPEN  INPUT  INPUT-FILE.
           MOVE 'ISRT'  TO DL1-FUNCTION.
       READ-INPUT-FILE.
           READ INPUT-FILE           AT END
                                     GO TO END-INP-FILE.
       BUILD-SEGMENT.
           IF INP-SEG-NAME NOT EQUAL TO SPACES
               PERFORM WRITE-BUILT-SEGMENT THRU WRITE-SEGMENT-EXIT
               MOVE ZEROS TO SUB-1
               MOVE SPACES TO BUILD-SEGMENT-AREA
               MOVE INP-SEG-NAME TO PREV-SEG-NAME.
           ADD 1 TO SUB-1.
           IF SUB-1 IS GREATER THAN 14
               DISPLAY 'MORE THAN 14 CARDS PER SEGMENT'  UPON CONSOLE
               DISPLAY 'SEGMENT IS   '  PREV-SEG-NAME   UPON CONSOLE
               GO TO LOCKED-HALT.
           MOVE INP-DATA TO BUILD-DATA-AREA (SUB-1).
```

```
        GO TO READ-INPUT-FILE.
    WRITE-BUILT-SEGMENT.
        IF PREV-SEG-NAME EQUAL TO SPACES
            GO TO WRITE-SEGMENT-EXIT.
        IF PREV-SEG-NAME = 'PARTROOT'  GO TO SEGMENT-IS-SEG00010.
        IF PREV-SEG-NAME = 'STANINFO'  GO TO SEGMENT-IS-SEG00060.
        IF PREV-SEG-NAME = 'STOKSTAT'  GO TO SEGMENT-IS-SEG02000.
        IF PREV-SEG-NAME = 'CYCCOUNT'  GO TO SEGMENT-IS-SEG02200.
        IF PREV-SEG-NAME = 'BACKORDR'  GO TO SEGMENT-IS-SEG02300.
    INVALID-SEGMENT-NAME.
        DISPLAY 'INVALID SEGMENT NAME =  ' PREV-SEG-NAME.
        GO TO LOCKED-HALT.
    SEGMENT-IS-SEG00010.
        MOVE BUILD-SEGMENT-AREA TO SEG00010-INSERT-AREA.
        MOVE BUILD-SEGMENT-AREA TO KEY-VALUE-00010.
        MOVE SPACE TO BEGIN-OP-00010.
        CALL  'CBLTDLI' USING DL1-FUNCTION, PCB-AREA-1,
                              SEG00010-INSERT-AREA, SEG00010-SSA.
        MOVE '(' TO BEGIN-OP-00010.
        IF STATUS-CODES NOT = SPACES, GO TO SEGMENT-INSERT-ERROR.
        GO TO WRITE-SEGMENT-EXIT.
    SEGMENT-IS-SEG00060.
        MOVE BUILD-SEGMENT-AREA TO SEG00060-INSERT-AREA.
        MOVE WRONG-MAKE-SPAN TO RIGHT-MAKE-SPAN.
        MOVE BUILD-SEGMENT-AREA TO KEY-VALUE-00060.
        MOVE SPACE TO BEGIN-OP-00060.
        CALL 'CBLTDLI' USING DL1-FUNCTION, PCB-AREA-1,
                             SEG00060-INSERT-AREA, SEG00010-SSA,
                                                   SEG00060-SSA.
        MOVE '(' TO BEGIN-OP-00060.
        IF STATUS-CODES NOT = SPACES, GO TO SEGMENT-INSERT-ERROR.
        GO TO WRITE-SEGMENT-EXIT.
    SEGMENT-IS-SEG02000.
        MOVE BUILD-SEGMENT-AREA TO SEG02000-INSERT-AREA.
        MOVE BUILD-SEGMENT-AREA TO KEY-VALUE-02000.
        MOVE SPACE TO BEGIN-OP-02000.
        CALL 'CBLTDLI' USING DL1-FUNCTION, PCB-AREA-1,
                             SEG02000-INSERT-AREA, SEG00010-SSA,
                                                   SEG02000-SSA.
        MOVE '(' TO BEGIN-OP-02000.
        IF STATUS-CODES NOT = SPACES, GO TO SEGMENT-INSERT-ERROR.
        GO TO WRITE-SEGMENT-EXIT.
    SEGMENT-IS-SEG02200.
        MOVE BUILD-SEGMENT-AREA TO SEG02200-INSERT-AREA.
        MOVE BUILD-SEGMENT-AREA TO KEY-VALUE-02200.
        MOVE SPACE TO BEGIN-OP-02200.
        CALL 'CBLTDLI' USING DL1-FUNCTION, PCB-AREA-1,
                             SEG02200-INSERT-AREA, SEG00010-SSA,
                                                   SEG02000-SSA,
                                                   SEG02200-SSA.
        MOVE '(' TO BEGIN-OP-02200.
        IF STATUS-CODES NOT = SPACES, GO TO SEGMENT-INSERT-ERROR.
        GO TO WRITE-SEGMENT-EXIT.
    SEGMENT-IS-SEG02300.
        MOVE BUILD-SEGMENT-AREA TO SEG02300-INSERT-AREA.
```

```
           MOVE BUILD-SEGMENT-AREA TO KEY-VALUE-02300.
           MOVE SPACE TO BEGIN-OP-02300.
           CALL 'CBLTDLI' USING DLI-FUNCTION, PCB-AREA-1,
                                SEG02300-INSERT-AREA, SEG00010-SSA,
                                                      SEG02000-SSA,
                                                      SEG02300-SSA.
           MOVE '(' TO BEGIN-OP-02300.
           IF STATUS-CODES NOT = SPACES, GO TO SEGMENT-INSERT-ERROR.
           GO TO WRITE-SEGMENT-EXIT.
       WRITE-SEGMENT-EXIT.   EXIT.
       SEGMENT-INSERT-ERROR.
           DISPLAY   'SEGMENT '
                     PREV-SEG-NAME
                     ' INSERT ERROR, '
                     ' STATUS CODE= '
                     STATUS-CODES                    UPON CONSOLE.
           GO TO WRITE-SEGMENT-EXIT.
       END-INP-FILE.
           CLOSE INPUT-FILE.
           PERFORM WRITE-BUILT-SEGMENT THRU WRITE-SEGMENT-EXIT.
           DISPLAY 'END   DB LOAD'  UPON CONSOLE.
       LOCKED-HALT.
           GOBACK.
```

DATA BASE DUMP PROGRAM

ASSEMBLER LANGUAGE APPLICATION PROGRAM EXAMPLE

In this example, the application program DFSSAM08 is a program used to dump a data base named DI21PART. This is a batch processing program that is the reverse of the data base load program, DFSSAM01, shown previously. The procedure MFDBDUMP (in conjunction with the sample problem in the IMS/VS Installation Guide) uses DFSSAM08 as the source program. The listing follows.

```
FILE: DFSSAM08 PT01138  A   PALO ALTO DEVELOPMENT CENTER


./        REPL  NAME=DFSSAM08
          TITLE 'DFSSAM08 - DUMP SAMPLE DATABASE IMS/VS'
          PRINT NOGEN
DFSSAM08  CSECT
          SPACE 1
PCBREG    EQU   4
BASE1     EQU   12
          ENTRY DLITCBL
          SPACE 1
          USING *,BASE1
DLITCBL   SAVE  (14,12),,SAM08-120
          LR    12,15                LOAD BASE REGISTER WITH EP
          ST    13,SAVEREGS+4        FORWARD CHAIN SAVE AREAS
          LA    15,SAVEREGS          A(SAVE AREA)
          ST    15,8(,13)            BACK CHAIN SAVE AREAS
          LR    13,15                A(SAVE AREA)
          SPACE 1
          L     PCBREG,0(1)          A(PCB) PASSED BY CALLER
          ST    PCBREG,PCBADDR       PUT A(PCB) IN CALL LIST
          MVI   PCBADDR,X'00'        CLEAR HI BYTE
          USING DLIPCB,PCBREG
          OPEN  (OUTFILE,(OUTPUT))
GETDISK   DS    0H
          CALL  CBLTDLI,MF=(E,DLILINK)   ISSUE DL/I CALL
          CLC   DLISTAT,=C'  '       WAS CALL OK ?
          BE    CALLOK               YES, THEN PRINT SEGMENT
          CLC   DLISTAT,=C'GA'       DID CALL CROSS BOUNDARY ?
          BE    CALLOK               YES, THEN PRINT SEGMENT
          CLC   DLISTAT,=C'GK'       IS THIS SIBLING SEGMENT ?
          BE    CALLOK               YES, THEN PRINT SEGMENT
          CLC   DLISTAT,=C'GB'       IS THIS END OF DATA BASE ?
          BE    ENDDISK              YES, THEN RETURN
          WTO   'ERROR IN GET NEXT DL/I CALL'
          B     ABEND
*...BUILD OUTPUT RECORD
CALLOK    DS    0H
          MVC   OUTREC(8),DLISEGFB
          MVC   OUTREC+9(100),SEGRETRN
          PUT   OUTFILE,OUTREC
          MVC   OUTREC(8),=CL8' '
          MVC   OUTREC+9(100),SEGRETRN+100
          PUT   OUTFILE,OUTREC
          MVI   SEGRETRN,X'40'           BLANK
          MVC   SEGRETRN+1(L'SEGBETRN-1),SEGRETRN
          B     GETDISK
          EJECT
ABEND     EQU   *
ENDDISK   CLOSE  (OUTFILE)
          L     13,SAVEREGS+4
          RETURN (14,12),,RC=0
          EJECT
*... CONSTANTS AND DSECTS
DLIFUNC   DC    CL4'GN  '            GET NEXT CALL FUNCTION
*...DLI CALL LIST
DLILINK   DC    A(DLIFUNC)           A(FUNCTION)
```

```
FILE: DPSSAM08 PT01138  A    PALO ALTO DEVELOPMENT CENTER


PCBADDR   DC     A(0)                    A(PCB)
          DC     X'80'                   END OF LIST FLAG
          DC     AL3(SEGRETRN)           A(I/O AREA)
          SPACE  1
SAVEREGS  DC     18F'0'                  REGISTER SAVE AREA
OUTREC    DC     CL110' '                OUTPUT RECORD
SEGRETRN  DC     CL200' '                I/O AREA
          DC     CL100' '
          SPACE  1
          LTORG
          SPACE  1
OUTFILE   DCB    DSORG=PS,MACRF=(PM),                                          X
                 LRECL=110,BLKSIZE=110,RECFM=FB,DDNAME=OUTPUT
          SPACE  1
IMSPCB    DSECT
DLIPCB    DS     0H
DLIFILE   DS     CL8
DLISGLEV  DS     CL2
DLISTAT   DS     CL2
DLIPROC   DC     CL4'G
          DC     F'0'
DLISEGFB  DS     CL8
          END
```

BATCH PROCESSING PROGRAM EXAMPLE

The two programs previously shown, DFSSAM01 and DFSSAM08, are batch processing programs, written in COBOL and Assembler Language, respectively. Refer to them for details as they are not repeated here. Instead, the SYSIN data for DFSSAM01 is provided. Refer to the listing of DFSSAM01 to interpret the format and content of this data.

```
PARTROOT 02AN960C10                    WASHER                                    0001
STANINFO 02                742                              1200  14             0002
         06C                                                                      0003
STOKSTAT 00 AA16511             000000000      EACH00000000000000000000          0004
            512     0000000 0000131 0000015 0000020 0000126 0000104 000000005
         C0000J00000 0000000 0 Z512N                                              0006
STOKSTAT 00 AK2877F             M0C0000000     EACH0000000000000027J0000         0007
            360     0000000 0000000 0000000 0000000 0000000000000370000010008
         0000C00000 0000000 0 Z360N                                               0009
STOKSTAT 0028009126             000000000      EACH                 000000000000000000010
         0000513517517S0C0000000000630 0000000.0000000 0000680 0001053 000000137
         04 000000000000000 0  494Y                                               0138
PARTROOT 02CK05CW181K                  CAPACITOR                                 0142
STANINFO 02                742                              1200  82             0143
         06C                                                                      0144
STOKSTAT 00 VF52906             000001000      EACH00000000000000400000          0169
            245     0000000 0000000 0000000 0000000 0000000000000020000000170
         0800000000 0000000 0 H245N                                               0171
STOKSTAT 0025900326             000000340                        000000          0172
            510510S0C0000000001320 000000000000660 0000660 000000000000000173
         0000000J000000000000                                                     0174
STOKSTAT 0025910926             000000340                        000000          0175
            510510S000000000000008 000000000000000 0000008 000000000000000176
         0000000000000000000                                                      0177
PARTROOT 02CSR13G104KL                 KR1J50KS                                  0178
STANINFO 02                742                              1200  82             0179
         06C                                                                      0180
STOKSTAT 00 DB7455R             M000002710     EACH00000000000000000000          0181
            495     0000000 0000014 0000000 0000000 0000014000000060000000182
         000000C0000 0000000 0 V485N                                              0183
STOKSTAT 00 SK21713             M000002710     EACH00000000000000000000          0184
            260     0000000 0000004 0000000 0000000 0000004000000020000000185
         0000000C000 0000000 0 V260N                                              0186
STOKSTAT 0025502526             000000000                        008000          0187
            472472S000000000000014 0000000 000000000000014 0000050 0000000188
         04 000000000000000000                                                    0189
PARTROOT 02JAN1N976B                   DIODE CODE-A                              0202
STANINFO 02                742                              1200  72             0203
         06C                                                                      0204
STOKSTAT 0025509126             000000000                        004000          0208
            513515S000000000000017 0000000 000020000000017 0000068 000000209
         03 000000000000000000  513                                              0210
BACKORDR 30PR237942                                                              00000211
         2000                                                                    0212
PARTROOT 02MS16995-28                  SCREW                                     0217
STANINFO 02                742                              1200  14             0218
         06C                                                                      0219
STOKSTAT 00 AA16511             000000152      EACH00000000000000000000          0220
            489495N0000000 0000026 0000000 0000000 0000030 0000003 000000221
         0000000000 0000000 0 V489N                                               0222
STOKSTAT 00 BA16515             0C0000069      EACH00000000000000000000          0223
            455     0000000 0000006 0C00000 0000000 0000008000000000000000224
         0000000000 0000000 0 V455N                                               0225
STOKSTAT 00 FF5546D             M000000061     EACH00000000000000000000          0226
            448     0000000 0000044 0000000 0000000 0000043000000000000000227
         0000000000 0000006 0 V448N                                               0228
STOKSTAT 0025910926             C00006980                        000000          0229
            497498S000000000000095 0000000 0000000 0000100 000000000000000230
```

```
        00300C000C00000000000      Y                                                    0231
PARTROOT 02N51P3003F000                SCREW                                          0232
STANINFO 02                  742                              1200  14                0233
         03                                                                           0234
STOKSTAT 0025906026            000000000 00000000                 0000000000000000000241
         0000   404404S000000000000313 0000000 0000000 0000360 0001209 000000242
         00 0000000000000000 0                                                        0243
CYCCOUNT 2000003600     00003600                                                      0244
PARTROOT 02RC07GF273J                  RESISTOR                                       0247
STANINFO 02                  742                              1200  02                0248
         06C                                                                          0249
STOKSTAT 00 AK24527            000000240     EACH000000000000000000                  0250
              213     0000000 0000033 0000000 0000000 000003300000001000000000251
         0000000000 0000000 0 V213N                                                   0252
STOKSTAT 0028C09126            000000000 00000000                 0000000000000000000364
         0000516517517S000000000000017 0000000 0000000 0000017 0000057 000000365
         00 0000000000000000 1   485Y                                                 0366
CYCCOUNT 2000000190     0000017                                                       0367
STOKSTAT 0028011126            000000000                          012000             0368
              459459S000000000000026 0000000 000000000000026 0000240 000000369
         29 00000C00000000000                                                         0370
PARTROOT 021068129JP009                RESISTOR                                       0371
STANINFO 02                  742                              1200  02                0372
         10                                                                           0373
STOKSTAT 0025900326            000000000 00000000                 007005 0000000000000374
         0000393491494S000000000001055 0000200 0000000 0001055 0004780 000000375
         22 0000000000000000 0                                                        0376
STOKSTAT 0025906026            000000000 00000000                 0000000000000000000380
         0000   293     000000000000000 000000000000000000000000 0001808 000000381
         01 0000000000000104 0                                                        0382
STOKSTAT 0025910926            000001820                          000000             0383
              482483S000000000000320 0000000 0000020 0000320 0000000000000000384
         0000000000000000C0000                                                        0385
PARTROOT 02250236-001                  CAPACITOR                                      0386
STANINFO 02                  742                              1200  82                0387
         04                                                                           0388
STOKSTAT 0025900326            000000000 00000000                 007010 0000000000000389
         0000393493884885000000000000410 0000200 0000600 0000010 0000567 000000390
         38 0000000000000000 0   488Y                                                0391
BACKORDR 30PR265943                                                                  00000390
         1000                                                                        0390
BACKORDR 30PR347921                                                                  00000390
         2000                                                                        0390
BACKORDR 30PR426134                                                                  00000390
         3000                                                                        0390
STOKSTAT 0025906026            000000000 00000000                 0010000000000000000395
         0000   448440S000000000000000 000000000000000000000000 0000551 000000396
         10 0000000000000000 0                                                       0397
STOKSTAT 0025910926            000003670                          000000             0398
              517518S000000000000072 0000000 0000000 0000072 0000045 000000399
         0000C0000000000000000   517Y                                                0400
PARTROOT 02250239                      TRANSISTOR                                     0401
STANINFO 02                  742                              1200  02                0402
         05R                                                                         0403
STOKSTAT 0025910926            000006500                          004000             0413
              517518S000000000000068 0000000 0000001 0000067 0000045 000000414
         02 0000000000000000000  517Y                                                0415
PARTROOT 02250241-001                  CONNECTOR                                      0416
```

```
STANINFO 02                742                              1200  42            0417
         04                                                                      0418
PARTROOT 02250794                     RESISTOR                                  0431
STANINFO 02                742                              1200  02            0432
         10                                                                      0433
STOKSTAT 0025900326          000000000 00000000              005005 000000000000434
         0000393348848850000030000000003 0000000 0000000 0000003 0001176 000000435
         64 0000000000000000 0  488Y                                            0436
STOKSTAT 0025906026          000000000 00000000              000000000000000000440
         0000   44844400000000000000000 00000000000000000000000 0001229 000000441
         80 0000000000000000 0                                                  0442
STOKSTAT 0025910926          000001740                       000000            0443
                5175185000000000000390 0000000 0000000 0000381 0000180 000000444
         0000C000C000000000000  517Y                                            0445
PARTROOT 02250796                     SWITCH                                    0446
STANINFO 02                222                              1200  54            0447
         06                                                                      0448
STOKSTAT 0025906026          000000000 00000000              000000000000000000449
         0000   44644650000000000000001 00000000000000000000001 0000062 000000450
         02 0000000000000023 0                                                  0451
STOKSTAT 0025910926          000015350                       000000            0452
                5125135000000000000020 0000000 0000010 0000005 000000000000453
         0000000000000000000                                                    0454
PARTROOT 02250891                     SERVO VALVE                               0455
STANINFO 02                742                              1200  16            0456
         06C                                                                    0457
STOKSTAT 0025906026          0C0000000 00000000              014000000000000000458
         0010   44644600000000C0000004 0000000 0000004 0000000 0000536 000000459
         73 0C000030000000029 0                                                 0460
STOKSTAT 0025910926          0C0395060                       000000            0461
                509440K000000000030235 0000000 0000180 0000055 0000005 000000462
         0000000000000000000  509                                               0463
PARTROOT 02252252-003                 COUPLING                                  0464
STANINFO 02                742                              1200  16            0465
         06C                                                                    0466
STOKSTAT 0025900326          0C0000000                       000000            0467
                485485000000000000092 0000005 0000092 000000000000000000000468
         0000000000000000000      Y                                            0469
STOKSTAT 0025906026          000000000 00000000              007000000000000000470
         0000   44844045000000000000000 0000000 0000000 0000010 0000832 000000471
         87 0000053000000460 0                                                  0472
STOKSTAT 0025910926          000016450                       000000            0473
                5075075000000000000076 0000005 0000010 0000076 0000008 000000474
         0000000000000000000  503                                               0475
PARTROOT 023003802                    CHASSIS                                   0476
STANINFO 02                222                              1200  34            0477
         06                                                                     0478
STOKSTAT 0025900326          000007900                       000000            0479
                494494500000000000005 0000000 0000000 0000005 0000173 000000480
         0000000000C000000000  494                                              0481
STOKSTAT 0025906026          000000000 00000000              017000000000000000482
         0000   293    000000000000000 00000000000000000000000 0001198 000020483
         04 0000000000000000 0                                                  0484
STOKSTAT 0025910926          000007900                       000000            0485
                5175185000000000C0004 0000000 0000000 0000004 0000036 000000486
         0000000C000000000000  517                                              0487
PARTROOT 023003806                    SWITCH                                    0488
STANINFO 02                742                              1200  54            0489
```

```
          06C                                                                0490
STOKSTAT 0025900326          000011263                    024000             0491
         518518S000000000000090 0000005 0000012 0000041 0000300 000000492
         72 0000127 000000000  515Y                                          0493
BACKORDR 30S0536609                    83404        36609                   00000494
         1110                                                               0495
BACKORDR 3030536610                    83404        36610                   00000496
         0160                                                               0497
STOKSTAT 0025906026          00000000  00000000           00200000000000000498
         0000   404    0000000000000000 000000000000000000000000 0001754 000000499
         52 0000155000000455 0                                              0500
STOKSTAT 0025910926          000006620                    100000             0501
         517518S000000000000004 0000002 0000000 0000004 0000036 000000502
         36 0000000000000000  517                                           0503
PARTROOT 023007228                   HOUSING                                0504
STANINFO 02                222                          1200  34           0505
          04                                                                0506
STOKSTAT 0025906026          000000000 00000000           00000000000000000507
         0000   448448N000000000000010 000000000000000000000010 0000125 000000508
         11 0000000000000013 0                                              0509
STOKSTAT 0025910926          00C012000                    000000             0510
         498498S000000000000013 0000000 0000000 0000013 0000006 000000511
         000000000000000000000  498                                        0512
PARTROOT 023009027                   CARD FRONT                             0513
STANINFO 02                46A                          7246  84           0514
          02F                                                               0515
STOKSTAT 0025906026          000000000 00000000           01600000000000000516
         0000   346    0000000000000001 000000000000000000000001 0000044 000000517
         07 0000000000000029 0                                              0518
STOKSTAT 0025910926          000000000                    000000             0519
         459459K000000000000003 0000000 0000003 000000000000000000000520
         000000000000000000000                                              0521
PARTROOT 023009278                   CAPACITOR                              0531
STANINFO 02                742                          1200  82           0532
          06C                                                               0533
STOKSTAT 0025900326          0C0000000                    000000             0534
         500500D000000000000001 0000009 000001000000001 0000014 000000535
         00 0000000000000000000  Y                                          0536
STOKSTAT 0025906026          000000000 00000000           01300000000000000537
         0000   476476N000000000000011 000000000000000000000001 0000083 000000538
         11 0000000000000002 0                                              0539
PARTROOT 023009270                   HOUSING                                0540
STANINFO 02                222                          1200  18           0541
          04                                                                0542
STOKSTAT 0025906026          000000000 00000000           00500000000000000543
         0000   448    0000000000000000 000000000000000000000000 0000044 000000544
         04 0000000000000000 0                                              0545
STOKSTAT 0025910926          000000000                    000000             0546
         448448K000000000000002 0000002 0000002000000002 0000000000000547
         000000000000000000000                                              0548
PARTROOT 023009280                   HOUSING CONV                           0549
STANINFO 02                222                          1200  18           0550
          04                                                                0551
STOKSTAT 0025910926          000293500                    000000             0552
         513452K000000000000002 0000000 0000000 0000002 0000000000000553
         000000000000000000000                                              0554
PARTROOT 023013405-002               MOUNTING                               0555
STANINFO 02                22                                646           0556
```

```
          000                                                                    055,
STOKSTAT 0025900326          000000000 00000000          0040000000000000000558
         0000393509484S000J00000000000 0000000 0000000 0000000 0000720 000000559
         27 000000000000047 0                                                   0560
STOKSTAT 0025906026          000000000 00000000          0030000000000000000561
         0000   329    0000000000000000 00000000000000000000000 0000560 000000562
         14 0000000000000540 0                                                  0563
STOKSTAT 0025910926          000029650                   000000                 0564
               518518S000000000000001 0000004 0000003 0000001 0000008 000000565
         00 0000000000000000  518Y                                              0566
-PARTROOT 023013412                COVER                                       0567
 STANINFO 02                222                          1200  66              0568
          06                                                                    0569
 STOKSTAT 0025906026          000000000 00000000          0020000000000000000570
          0000   448440S000J00000000012 0000005 0000020 0000000 0000400 000000571
          36 000002300000426 0                                                  0572
 STOKSTAT 0025910926          000012100                   000000                 0573
               512512S000000000000047 0000010 0000005 0000047 0000017 000000574
          0000000000000000000  512                                              0575
 PARTROOT 023013429-001             COVER ASSY                                  0576
 STANINFO 02                222                          1200  66              0577
          06                                                                    0578
 STOKSTAT 0025906026          000000000 00000000          0010000000000000000579
          0000380448440N000J00000000000 0000000000000000000000 0000394 000000580
          03 000000000000005 0                                                  0581
 STOKSTAT 0025910926          000003700                   000000                 0582
               513513S000000000000000 0000000 0000075 0000083 0000009 000000583
          0000000000000000000  513                                              0584
-PARTROOT 023013460-001             CAPACITOR                                  0585
 STANINFO 02                742                          1200  82              0586
          04                                                                    0587
 STOKSTAT 0025900326          000000000 00000000          006005 00000000000588
          0000393478478D000000000000004 0000005 000000000000004 0002915 000010589
          73 0000000000000000 0                                                 0590
 STOKSTAT 0025906026          000000000 00000000          0000000000000000000594
          0000   448440N000000000000010 0000000 0000000 0000000 0002248 0000  595
          27 000000000000007 0                                                  0596
 STOKSTAT 0025910926          000001530                   000000                 0597
               514515000000000000349 0000000 0000255 0000094 0000108 000000598
          0000000000000000000  514Y                                             0599
-PARTROOT 023013548-002             CHASSIS                                    0600
 STANINFO 02                222                          1200  34              0601
          09                                                                    0602
 STOKSTAT 0025900326          000000000 00000000          001005 00000000000603
          0000393493460D00000000000000 0000000 0000000 0000000 0001186 000000604
          11 0000000000000000 0      Y                                          0605
 STOKSTAT 0025906026          000000000 00000000          0000000000000000000609
          0000   293    0000000000000020 00000000000000000000000 0000498 000000610
          01 0000000000000505 0                                                 0611
 STOKSTAT 0025910926          000000000                   000000                 0612
               512512K000000000000002 0000000 0000002 0000000000000000000000613
          0000000000000000000                                                   0614
 PARTROOT 0256134-016               NAS671C1 NUT                               0615
 STANINFO 02                742                          1200  14              0616
          03                                                                    0617
 STOKSTAT 0025900326          000003033 00000000          004010 00000000000618
          0000393493495500000000000886 0000200 0000000 0000886 0002376 000000619
          86 0000000000000000 0                                                 0620
```

```
STOKSTAT 0025906026          000000000 00000000            0040000000000000000624
         0000   447447S000000000000014 000000000000000000000014 0000904 000000625
         46 0000000000000000 0                                                0626
STOKSTAT 0025910926          CC0000000                     000000            0627
               461461S000000000000115 0000010 0000000 0000115 00000000000000628
         C000000000000000000000                                               0629
CYCCOUNT 2000001100     00001150                                              0629
PARTROOT 0260003-118                 7734304P8661TO RES                        0630
STANINFO 02                  742                          1200  02            0631
            10                                                                0632
STOKSTAT 0025900326          000000000 00000000            020010 000000000000633
         0070393488488S000C000000000006 0000000 0000000 0000006 0000644 000010634
         26 0C0000000000000 0  488Y                                           0635
STOKSTAT 0025906026          000000000 00000000            000000000000000000639
         0000   448    000000000000C0 000000000000300000000000 0002100 000000640
         39 0000000000000081 0                                                0641
STOKSTAT 0025910926          CC0000130                     000000            0642
               5195185000000000000648 0000000 0000021 0000627 0000091 000000643
         0000C0C0000C0000000  518Y                                           0644
PARTROOT 02552540-002                WIRE WRAP                                0645
STANINFO 02                  222                          1200  42            0646
            04                                                                0647
STOKSTAT 0027909126          000000000 00000000            000000000000000000648
         0000   460    000000000000060 000002000000000000000000 0000012 000000649
         00 0000000000000000 0                                                0650
STOKSTAT 0028009126          000000000                     000000            0651
               5145150000000000000012 0000000 0000012 0000000 00000000000000652
         12 0000000000000000000      Y                                        0653
PARTROOT 02652799                    PULSE TRANSFORMER                        0654
STANINFO 02                  101                          8800              0655
            04                                                                0656
STOKSTAT 0028009126          000000000 00000000            000000000000000000657
         0000   5145155000000000000004 0000000 0000001 0000003 0000008 000000658
         00 0000000000000000 0  493Y                                          0659
PARTROOT 0268663-102                 CM05C100K03                              0660
STANINFO 02                  742                          1200  82            0661
            06C                                                               0662
STOKSTAT 0025906026          000000000 00000000            001000000000000000663
         0000384444440S000000000000000 0000000000000050 0000000 0001186 000000664
         09 0000000000000056 0                                                0665
STOKSTAT 0025910926          CC0000000                     000000            0666
               5185185000000000000232 0000052 0000075 0000232 0000031 000000667
         0000000000000000000  518                                            0668
PARTROOT 0268663-104                 CM05020QJ03                              0669
STANINFO 02                  742                          1200  82            0670
            06C                                                               0671
STOKSTAT 0025906026          000000000 00000000            001000000000000000672
         0000   448    000000000000015 0C0000700000023 0000000 0000394 000000673
         05 0000000000000000 0                                                0674
STOKSTAT 0025910926          000000000                     000000            0675
               5135135000000000000095 0000015 0000000 0000095 0000009 000000676
         0000000000000000000  513                                            0677
PARTROOT 0269857-635                 CP09A1KE153K3 CAPAC                      0678
STANINFO 02                  742                          1200  82            0679
            06C                                                               0680
STOKSTAT 0025906026          000000000 00000000            002000000000000000681
         0000   448400S000000000000000 0000000 0000000 0000200 0000394 000000682
         16 0000000000000011 0                                                0683
```

```
STOKSTAT 0025910926          000000500                      000000          0684
         515515000000000000093 0000000 0000050 0000052 0000009 00000685
         0000C000000000000000  515                                        0686
PARTROOT 027060654P001              ELE TUBE                              0687
STANINFO 02                742                          1200  10          0688
         06C                                                              0689
STOKSTAT 0025906026          000000000 00000000            00300000000000000690
         0000   446    000000000000000 000000000000000000000000 0000400 00000691
         12 0000000000000003 0                                            0692
STOKSTAT 0025910926          0C0100000                      000000          0693
         515517000000000000000038 0000000 0000002 0000036 0000004 00000694
         0000000000000000000J00  515                                      0695
PARTROOT 027438995P002              NUT                                   0696
STANINFO 02                742                          1200  14          0697
         03                                                               0698
STOKSTAT 0025900326          000000000 00000000            000005 00000000000699
         0000   44044050000000000001736 00000000000000000002512 0000443 00000700
         00 0000000000000000 0                                            0701
STOKSTAT 0025906026          000000000 00000000            00000000000000000702
         0000   296    0000000000000017 000000000000000000000033 0000880 00000703
         00 0000000000000015 0                                            0704
PARTROOT 027454949P001              LAMP HOLDER                           0705
STANINFO 02                742                          1200  82          0706
         06C                                                              0707
STOKSTAT 0025900326          000006168                      046000          0708
         518518S000000000000061 0000000 0000040 0000024 0000173 00000709
         80 0C0000000000000J00  505                                       0710
STOKSTAT 0025906026          000000000 00000000            02200000000000000711
         0000   293    0000000000000000 000000000000000000000003 0001301 00020712
         88 000C10900000065 0                                             0713
STOKSTAT 0025910926          000005360                      000000          0714
         517518S000J00000000004 0000000 0000000 0000004 0000036 00000715
         0000000000000000000000  517                                      0716
PARTROOT 027618032P101              CAPACITOR                             0717
STANINFO 02                742                          1200  82          0718
         04                                                               0719
STOKSTAT 0025900326          000001001 00000000            006005 00000000000720
         0000393491492S000000000000013 0000009 0000000 0000013 0002601 00001721
         49 0C0C000000000000 0  485Y                                      0722
BACKORDR 30PR1493291303603      0485   B2446              3013609-001     00000723
         0010                                                             0724
BACKORDR 30PR1493761303603      0485   B2446              3013609-001     00000725
         0010                                                             0726
BACKORDR 30PR1530961303603      0485   B2449              3013609-001     00000727
         0010                                                             0728
BACKORDR 30PR1530981303603      0485   B2451              3013609-001     00000729
         0010                                                             0730
BACKORDR 30PR1695661303603      0485   B2484              3013609-001     00000731
         0050                                                             0732
STOKSTAT 0025906026          000000000 00000000            00200000000000000736
         0000   363    0000000000000000 000000000000000000000000 0000952 00000737
         18 0000000C00000119 0                                            0738
STOKSTAT 0025910926          000000000                      000000          0739
         454454000000000000000022 0000000 0000022 00000000000000000000000740
         0000000000000000000J                                             0741
PARTROOT 027618289P049              CIRCUIT BREA                          0742
STANINFO 02                742                          1200  06          0743
         06C                                                              0744
```

```
STOKSTAT 0025906026          000000000 00000000            00000000000000000745
         0000   450    0000000000000015 0000000000000000000017 0000033 000000746
         03 0000000000000001 0                                              0747
STOKSTAT 0025910926          0C0000000                     000000           0748
               498498U0000000000000002 0000000 0000001 0000001 0000002 000000749
         0000000000000000000000  498                                        0750
PARTROOT 027630843P513              RESISTOR                                0751
STANINFO 02               742                          1200  02            0752
            06C                                                             0753
STOKSTAT 0025900326          000000000 00000000            00000000000000000754
         0000   338    0000000000000002 0000000000000000000000002 0000000 000000755
         00 0CC000000000002 0                                               0756
STOKSTAT 0025906026          000000000 00000000            00100000000000000757
         0000   443442S000000000000000 0000000000000000 0000000 0009555 000000758
         99 0000000000000080 0                                              0759
STOKSTAT 0025910926          0C0000000                     000000           0760
               518518000000000001403 0000000 0000300 0001203 0000858 000000761
         0000000000000000000C  518Y                                         0762
PARTROOT 027736847P001              TRANSFORMER                             0763
STANINFO 02               742                          1200  98            0764
            10                                                              0765
STOKSTAT 0025900326          000000000 00000000            000003 000000000000766
         0000393511511S000000000000179 0000001 0000150 0000040 0001417 000000767
         05 0000089 0000128 0                                               0768
BACKORDR 30PR135640             0485   B3323   0000A3564 448506-100      000000769
         1500                                                               0770
STOKSTAT 0025906026          000000000 00000000            00500000000000000774
         0000   357    0000000000000005 0000000000000000000000005 0000430 000000775
         20 0CC0003000000040 0                                              0776
STOKSTAT 0025910926          000015100                     000000           0777
               495497K000000000000010 0000000 0000000 0000010 000000000000000778
         00 000000000000000000                                              0779
PARTROOT 02303008035                GASKET                                  0780
STANINFO 02               222                          1200  84            0781
            04                                                              0782
STOKSTAT 0025906026          000000000 00000000            00600000000000000783
         0000   293    0000000000000049 0000000000002000000019 0000176 000000784
         11 0000000000000000 0                                              0785
STOKSTAT 0025910926          000002580                     000000           0786
               510510S0000000C0000012 0000000 0000000 0000012 0000008 000000787
         0000000000J000000000  510                                          0788
PARTROOT 0282124-056                RN60C3161F                              0789
STANINFO 02               742                          1200  02            0790
            10                                                              0791
STOKSTAT 0025900326          000000000 00000000            000007 000000000000792
         J000393488488S000000000000008 0000000 0000000 0000008 0001176 000000793
         02 0000000000000028 0  488Y                                        0794
STOKSTAT 0025910926          0C0000000                     000000           0801
               517518S000000000000322 0000000 0000000 0000340 0000190 000000802
         0000000000000000000000  517Y                                       0803
PARTROOT 0282124-640                RN65C9092F                              0804
STANINFO 02               742                          1200  02            0805
            048                                                             0806
STOKSTAT 0025900326          000000000                     000000           0807
               494494D000000000000000 0000000 0000000 0000000 0000008 000000808
         0000000000000000000000  494Y                                       0809
STOKSTAT 0025906026          000000000 00000000            00000000000000000810
         0000   402    0000000000000000 0000000000000000000000000 0000075 000000811
```

```
        03 000000000000000 0                                                    0812
PARTROOT 0292125-869                RN75C8252F                                  0813
STANINFO 02                742                          1200  02              0814
           06C                                                                  0815
STOKSTAT 0025906026           G00000000 00000000              00100000000000000816
         0000   446    000000000000100 000000000000000000000050 0000424 000000817
         03 000000000000013 0                                                   0818
STOKSTAT 0025910926           000000780                       000000            0819
                 513513500000000000090 0000000 0000000 0000090 0000060 000000820
         000000000000000000000  513                                            0821
PARTROOT 0284353-456                RW67V472                                    0822
STANINFO 02                742                          1200                  0823
           06C                                                                  0824
STOKSTAT 0025900326           G00000000 00000000              00200000000000000825
         0000   47947950000000000280 0000000 0000200 0000110 0000165 000000826
         03 000000000000004 0                                                   0827
STOKSTAT 0025910926           000000000                       000000            0831
                 509517S000000000000009 0000000 0000000 0000009 0000012 000000832
         00 000000000000000000  509Y                                           0833
PARTROOT 0290-3033334               BONDED ASSY                                 0834
STANINFO 02                41A                          7236                  0835
           01D                                                                  0836
STOKSTAT 0028009126           GC0C00000                       003000            0837
                 514515S000000000000010 0000000 0000014 0000001 0000032 000000838
         01 000C0000000000000  503Y                                             0839
PARTROOT 0290-3033665               BONDED ASSY                                 0840
STANINFO 02                41A                          7219                  0841
           015                                                                  0842
STOKSTAT 0028009126           CC0000000                       000000            0843
                 514515000000000000024 0000000 0000024 0000000 0000048 000000844
         0C000000000C00000000  482Y                                             0845
PARTROOT 02905537-384               CAPACITOR                                   0846
STANINFO 02                742                          1200                  0847
           06C                                                                  0848
STOKSTAT 0028002526           0C0000000                       002000            0849
                 497461461S000000000000004 0000000 0000000 0000004 0000050 000000850
         01 000C00C00000000100                                                  0851
STOKSTAT 0028C09126           000000000                       000000            0852
                 514444D0000000000000 0000000 000000000000000000000000000000853
         000000000000000000000      Y                                          0854
PARTROOT 02906029-040               CAPACITOR                                   0855
STANINFO 02                742                          1200  82              0856
           06C                                                                  0857
STCKSTAT 0028009126           000000000 00000000              00700000000000000858
         0000   513515S000000000000024 0000000 0000020 0000004 0000028 000000859
         02 000000000000000 0      Y                                           0860
PARTROOT 02907021-782               CAPACITOR                                   0861
STANINFO 02                742                          1200  82              0862
           06C                                                                  0863
STOKSTAT 0028009126           000001700 00000000              00600000000000000864
         0000   517517000000000000008 0000000 0000005 0000003 0000018 000000865
         01 000000000000000 0  503Y                                            0866
PARTROOT 02922294-002               CONNECTOR                                   0879
STANINFO 02                742                          1200  42              0880
           06C                                                                  0881
STOKSTAT 00 TF3467A          M0000003390E    FACH000000000000000000            0882
                 467    0000000 0000000 0000000 0000000 000000000000000000000883
         0100000000 0000000 0 V467N                                             0884
```

```
STOKSTAT 00 TF5877N          M000012540      EACH000000000000000000            0885
              467    0000000 0000001 0000015 0000000 0000001 000000000000000886
         0000000000 0000000 0 V467N                                             0887
STOKSTAT 0028009026           0C0000000 00000000             020000000000000000888
         0000452476476S000000000000001 000000000000000000000001 0000015 000000889
         03 0000000000000003 0                                                  0890
STOKSTAT 0028009126           000000000 00000000             000000000000000000891
         0000   514515S000000000000017 0000000 0000000 0000007 0000030 000000892
         00 0000000000000000 0      Y                                          0893
PARTROOT 02922399-001              CONNECTOR                                   0894
STANINFO 02                 742                          1200                 0895
           06C                                                                  0896
STOKSTAT 0028009126           000011430                       005000           0897
              517517S000000000000006 0000000 0000005 0000005 0000019 000000898
         01 00000000000000000  507Y                                            0899
PARTROOT 02925363-136              DIODE ZENER                                 0900
STANINFO 02                 742                          1200  72             0901
           06C                                                                  0902
STOKSTAT 0028009126           000000000 00000000             007000000000000000903
         0000   514515000000000000005 0000030 0000004 0000001 0000030 000000904
         02 0000000000000000 0      Y                                          0905
BACKORDR 30PR729437                                                           00000905
         0040                                                                 0905
PARTROOT 02925380-101              DIODE                                       0906
STANINFO 02                 74                                  172          0907
           000                                                                  0908
PARTROOT 02930331-102              FILTER                                      0910
STANINFO 02                 816                          0000
           00                                                                   0913
STOKSTAT 0028009126           000000000 00000000             000000000000000000914
         0000   514515S000000000000008 0000000 0000004 0000004 0000025 000000915
         00 0000000000000000 0  488Y                                          0916
PARTROOT 02930331-123              FILTER                                      0917
STANINFO 02                 815                          0000                 0918
           00                                                                   0919
STOKSTAT 0028009126           000000000 00000000             000000000000000000920
         0000   514515S000000000000008 0000000 0000005 0000003 0000025 000000921
         00 0000000000000000 0  487Y                                          0922
PARTROOT C2930333-001              DISCRIMINATO                                0923
STANINFO 02                 816                          0000                 0924
           00                                                                   0925
STOKSTAT 0028009126           C00000000 00000000             013000000000000000926
         0000   514515000000000000002 0000000 0000002 0000000 0000008 000000927
         01 0000000000000001 0      Y                                         0928
PARTROOT 02946325-086              PIN                                         0929
STANINFO 02                 742                          1200  16             0930
           06C                                                                  0931
STOKSTAT 00 RF34674          M0C0000890      EACH000000000000000000            0932
              376    0000000 0001313 0000000 0000000 000111300000000000000933
         0000C00000 0000000 0 V376N                                             0934
STOKSTAT 00 VF34610          M0C0000000      EACH000000000000000000            0935
              170    0000000 0000057500000000 0000000 000005900000000000000936
         0000000000 0000000 0 V170N                                             0937
STOKSTAT C028C09026           000000000 00000000             850000000000000000938
         0000452481452S000000000000000 000000000000000000000000 0000004 000000939
         74 0C0000000001010 0                                                   0940
STOKSTAT 0028009126           000000000 00000000             050000000000000000941
         0000   514515S000300000000008 0000000 0000000 0000008 0000016 000000942
```

```
        C8 000000000000000 0  493Y                                                0943
PARTROOT 02950060-006                RELAY                                        0944
STANINFO 02                  742                              1200  96            0945
        06C                                                                       0946
STOKSTAT C028CC9126           000015300 00000000              00000000000000000947
        0000   517518S000000000003009 0000000 0000000 0000009 0000027 000000948
        00 000000000000000 0  483Y                                                0949
PARTROOT 02954017-001                RESISTOR                                     0950
STANINFO 02                  742                              1200  02            0951
        06C                                                                       0952
STOKSTAT 00 JF3467A           M000002525      EACH000000000000000000              0953
             907     0000000 0000006 0000000 0000000 0000006000000030000000954
        0000000000 0000000 0 V907N                                                0955
STOKSTAT 00 TF3467A           M0000100006     EACH000000000000000000              0956
             401     0000000 0000000 0000000 0000000 00000000000000000000000957
        0000000000 0000003 0 V401N                                                0958
STOKSTAT 00 TF5877N           M000002525      EACH000000000000000000              0959
             474     0000000 0000002 0000000 0000000 0000002000000010000000960
        0000000000 0000000 0 V474N                                                0961
STOKSTAT 0028009126            000000000 00000000              00000000000000000962
        0000   514515S000000000000004 0000000 0000003 0000001 0000008 000000963
        00 0000000000000000 0  486Y                                               0964
PARTROOT 02958007-180                RESISTOR                                     0965
STANINFO 02                  742                              1200  02            0966
        06C                                                                       0967
STOKSTAT 0028009126            0C0000650 00000000              00500000000000000968
        0000   517517S000000000000046 0000000 0000000 0000039 0000021 000000969
        01 0000000000000000 0      Y                                              0970
PARTROOT 02960528-067                RESISTOR                                     0971
STANINFO 02                  742                              1200  02            0972
        06C                                                                       0973
STOKSTAT 00 DF34671           M000007000      EACH000000000000030000              0974
             140     0000000 0000000 0000000 0000000 00000000000001000000000975
        0300000000 0000000 0 V140N                                                0976
STOKSTAT 0028009026            000000000 00000000              10000000000000000977
        0000452481479N000000000000000 0000000000000000 0000090 0000003 000000978
        03 0000000000000003 0                                                     0979
STOKSTAT 0028009126            000009230 00000000              10400000000000000980
        0000   517517000000000000009 0000000 0000005 0000004 0000027 000000981
        28 0000000000000000 0  505Y                                               0982
PARTROOT 02968534-001                SOCKET                                       0983
STANINFO 02                  742                              1200  16            0984
        06C                                                                       0985
STOKSTAT 0028009126            000050000 00000000              02900000000000000986
        0000   514515S000000000000038 0000000 0000003 0000005 0000007 000000987
        02 0C0000000000000 0      Y                                               0988
PARTROOT 02974810-010                THERMOSTAT                                   0989
STANINFO 02                  742                              1200  16            0990
        06C                                                                       0991
STOKSTAT 0028002526            000013250 00000000              00700000000000000992
        0000495516517S000000000000006 0000000 0000000 0000006 0000057 000000993
        04 0000000000000000 0  516                                                0994
STOKSTAT 0028009126            000009750                        007000            0995
             517517S000000000000021 0000000 0000005 0000016 0000014 000000996
        C1 0000000000000000      Y                                                0997
PARTROOT 02975105-001                TRANSFORMER                                   998
STANINFO 02                  742                              1200  16            0999
        06C                                                                       1000
STOKSTAT 0028009126            000106000                        024000            1001
             514515S000000000000029 0000000 0000001 0000028 0000021 000001002
        05 0000000000000000      Y                                                1003
PARTROOT 02989036-001                TRANSFORMER                                  1004
STANINFO 02                  742                              1200  96            1005
        06C                                                                       1006
STOKSTAT 0028009126            000019300                        112000
             517517000000000000007 0000000 0000004 0000003 0000017 00000
        19 00000000000000000      Y
```

MESSAGE PROCESSING PROGRAM EXAMPLE

ANS COBOL APPLICATION PROGRAM

This message processing program, DFSSAM03, provides you with the ability to inquire about the total inventory of a part in all locations. This program is one of several message processing programs used in the Sample Problem, included in the IMS/VS Installation Guide.

The transaction code DSPINV retrieves the data from the data base, DI21PART, loaded by a previous program. Assume that it wishes to display, on a communication terminal, only the third inventory entry listed in the above output. The inventory location key is obtained by concatenating AREA, INVDEPT, PROJCD, and DIV.

The input format for this transaction is:

transaction code	part number	inventory key
despinv	an960c10,	28009126

The output is:

```
PART=AM960C10       ; DESC=WASHER          ; PROC CODE=74
AREA=2; INV DEPT=80; PRJ=091; DIV=26; PRICE=      .000; STK CT DATE=513; UNIT=EACH
CURR REQMTS=    630 ; ON ORDER=       0 ; TOTAL STOCK=    680
DISB PLANNED=   1053 ; DISB UNPLANNED=      4 ; STK CT VARIANCE=      0
```

The program listing is:

```
FILE: DFSSAM03 ASSEMBLE A    PALC ALTO DEVELOPMENT CENTER

       IDENTIFICATION DIVISION.
       PROGRAM-ID. 'DFSSAM03'
       AUTHOR.         DON TRUDELL.
       REMARKS.  SINGLE-LOCATION INVENTORY DISPLAY PROGRAM.
                 THE TRANSACTION CODE WHICH ACTIVATES THE PROGRAM IS
                 DSPINV.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.  IBM-360.
       OBJECT-COMPUTER.  IBM-360.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  NEXT-FUNC                  PICTURE X(04)  VALUE 'GN  '.
       01  UNIQ-FUNC                  PICTURE X(04)  VALUE 'GU  '.
       01  ISRT-FUNC                  PICTURE X(04)  VALUE 'ISRT'.
       01  STOKSTAT-WRITE-SW          PICTURE X(02) VALUE SPACES.
       01  PARTROOT-SSA.
           02  ROOT-NAME              PICTURE X(8)   VALUE 'PARTROOT'.
           02  BEGIN-OP               PICTURE X      VALUE '('.
           02  KEY-NAME               PICTURE X(8)   VALUE 'PARTKEY '.
           02  RELATION-OP            PICTURE XX     VALUE ' ='.
           02  KEY-VALUE              PICTURE X(17).
           02  END-OP                 PICTURE X      VALUE ')'.
       01  STOKSTAT-SSA.
           02  FILLER                 PICTURE X(08)   VALUE 'STOKSTAT'.
           02  FILLER                 PICTURE X(01)   VALUE '('.
           02  FILLER                 PICTURE X(08)   VALUE 'STOCKEY'.
           02  FILLER                 PICTURE X(02)   VALUE ' ='.
           02  SS-SSA-KEY.
               03  FILLER             PICTURE X(02)   VALUE ZEROS.
               03  SS-SSA-KEY-VALUE   PICTURE X(08).
               03  FILLER             PICTURE X(06)   VALUE SPACES.
           02  FILLER                 PICTURE X(01)   VALUE ')'.
       01  TERM-IN-AREA.
           02  FILLER                 PICTURE X(140)   VALUE SPACES.
       01  REFORM-MESSAGE.
           02  REFORM-TRANS-CD        PICTURE X(8).
           02  PART-NO                PICTURE X(15).
           02  INPUT-SS-KEY           PICTURE X(08).
           02  FILLER                 PICTURE X(109).
       01  WORK-AREAS.
           02  ROOT-KEY-WA.
               04  ROOT-PREFIX        PICTURE XX  VALUE '02'.
               04  PN-WORK            PICTURE X(15).
           02  MSG-SEG-CNT            PICTURE S9 COMPUTATIONAL VALUE ZERO.
       01  PARAM-TABLE.
           02  FILLER                 PICTURE S9(2) VALUE +15 COMP.
           02  FILLER                 PICTURE XX    VALUE 'L '.
           02  FILLER                 PICTURE S99 VALUE +8 COMP.
           02  FILLER                 PICTURE X(02)  VALUE 'L '.
           02  END-TABLE              PICTURE S99 VALUE ZERO COMPUTATIONAL.
       01  PART-LINK.
           02  PART-NO-EDIT           PICTURE X(17).
           02  FILLER                 PICTURE XXXX.
           02  REJECT-CODE            PICTURE X.
```

```
01  SEG-RET-AREA.
    02 FILLER                  PICTURE X(02).
    02 PART-NO                 PICTURE X(15).
    02 FILLER                  PICTURE X(09).
    02 DESC                    PICTURE X(15).
    02 FILLER                  PICTURE X(119).
01  STAN-INFO-RET  REDEFINES SEG-RET-AREA.
    02 FILLER                  PICTURE X(18).
    02 PROC-CODE               PICTURE XX.
01  STOCK-STATUS-RET  REDEFINES STAN-INFO-RET.
    02 FILLER                  PICTURE XX.
    02 SS-AREA                 PICTURE X.
    02 SS-DEPT                 PICTURE XX.
    02 SS-PROJ                 PICTURE XXX.
    02 SS-DIV                  PICTURE XX.
    02 FILLER                  PICTURE X(10).
    02 SS-UNIT-PRICE           PICTURE 9(6)V999.
    02 FILLER                  PICTURE X(05).
    02 SS-UNIT-OF-MEAS         PICTURE X(04).
    02 FILLER                  PICTURE X(33).
    02 SS-STOCK-DATE           PICTURE X(03).
    02 FILLER                  PICTURE X(15).
    02 SS-CUR-REQMTS           PICTURE S9(7)V9.
    02 SS-UNPL-REQMTS          PICTURE S9(7)V9.
    02 SS-ON-ORDER             PICTURE S9(7)V9.
    02 SS-IN-STOCK             PICTURE S9(7)V9.
    02 SS-PLAN-DISB            PICTURE S9(7)V9.
    02 SS-UNPL-DISB            PICTURE S9(7)V9.
    02 FILLER                  PICTURE X(23).
01  BACK-ORDER-RET             REDEFINES STOCK-STATUS-RET.
    02 FILLER                  PICTURE X(02).
    02 WORK-ORDER              PICTURE X(08).
    02 FILLER                  PICTURE X(53).
    02 WO-QTY                  PICTURE S9(07)V9.
01  CYCLE-COUNT-RET            REDEFINES BACK-ORDER-RET.
    02 FILLER                  PICTURE X(02).
    02 PHYSICAL-COUNT          PICTURE S9(07)V9.
    02 FILLER                  PICTURE X(04).
    02 TOTAL-STOCK             PICTURE S9(07)V9.
01  LINE-1-AREA.
    02 FILLER                  PICTURE S99 COMPUTATIONAL VALUE +62.
    02  FILLER                 PICTURE S99 VALUE ZERO
                                          COMPUTATIONAL.
    02 FILLER                  PICTURE X(01)  VALUE ' '.
    02 FILLER                  PICTURE X(05)  VALUE  'PART='.
    02 PART-NO                 PICTURE X(15).
    02 FILLER                  PICTURE X(7)  VALUE '; DESC='.
    02 DESC                    PICTURE X(15).
    02 FILLER                  PICTURE X(12) VALUE '; PROC CODE='.
    02 PROC-CODE               PICTURE XX.
    02 CARR-RET                PICTURE X(01)   VALUE ' '.
01  LINE-2-AREA.
    02 FILLER                  PICTURE S9(02)   VALUE +88
                                          COMPUTATIONAL.
    02 FILLER                  PICTURE S9(02)   VALUE ZERO
```

```
                                          COMPUTATIONAL.
    02 FILLER              PICTURE X(01)   VALUE ' '.
    02 FILLER              PICTURE X(05)   VALUE  'AREA='.
    02 SS-AREA             PICTURE X(01).
    02 FILLER              PICTURE X(11)   VALUE '; INV DEPT='.
    02 SS-DEPT             PICTURE X(02).
    02 FILLER              PICTURE X(06)   VALUE '; PRJ='.
    02 SS-PROJ             PICTURE X(03).
    02 FILLER              PICTURE X(06)   VALUE '; DIV='.
    02 SS-DIV              PICTURE X(02).
    02 FILLER              PICTURE X(08)   VALUE '; PRICE='.
    02 SS-UNIT-PRICE       PICTURE Z(6).999.
    02 FILLER              PICTURE X(14) VALUE '; STK CT DATE='.
    02 SS-STOCK-DATE       PICTURE X(03).
    02 FILLER              PICTURE X(07) VALUE '; UNIT='.
    02 SS-UNIT-OF-MEAS     PICTURE X(04).
    02 CARR-RET            PICTURE X(01)   VALUE ' '.
01  LINE-3-AREA.
    02 FILLER              PICTURE S9(02)    VALUE +67
                                             COMPUTATIONAL.
    02 FILLER              PICTURE S9(02)    VALUE ZERO
                                             COMPUTATIONAL.
    02 FILLER              PICTURE X(01)   VALUE ' '.
    02 FILLER              PICTURE X(12) VALUE  'CURR REQMTS='.
    02 SS-CUR-REQMTS       PICTURE Z(06)9-.
    02 FILLER              PICTURE X(11) VALUE '; ON ORDER='.
    02 SS-ON-ORDER         PICTURE Z(06)9-.
    02 FILLER              PICTURE X(14) VALUE '; TOTAL STOCK='.
    02 SS-IN-STOCK         PICTURE Z(06)9-.
    02 CARR-RET            PICTUBE X(01)   VALUE ' '.
01  LINE-4-AREA.
    02 FILLER              PICTURE S9(02)    VALUE +79
                                             COMPUTATIONAL.
    02 FILLER              PICTURE S9(02)   VALUE ZERO
                                             COMPUTATIONAL.
    02 FILLER              PICTURE X(01)   VALUE ' '.
    02 FILLER              PICTURE X(13) VALUE  'DISB PLANNED='.
    02 SS-PLAN-DISB        PICTURE Z(06)9-.
    02 FILLER              PICTURE X(17) VALUE
                                        '; DISB UNPLANNED='.
    02 SS-UNPL-DISB        PICTURE Z(06)9-.
    02 FILLER              PICTURE X(18) VALUE
                                        '; STK CT VARIANCE='.
    02  STOCK-VAR          PICTURE Z(07)9-.
    02 CARR-RET            PICTURE X(01)   VALUE ' '.
01  LINE-5-AREA.
    02 FILLER              PICTURE S9(02)    VALUE +57
                                             COMPUTATIONAL.
    02 FILLER              PICTURE S9(02)   VALUE ZERO
                                             COMPUTATIONAL.
    02 FILLER              PICTURE X(01)     VALUE ' '.
    02 DESC-1              PICTURE X(24).
    02 WORK-ORDER          PICTURE X(08).
    02 DESC-2              PICTURE X(11).
    02 WO-QTY              PICTURE Z(06)9-.
```

```
    02 CARR-RET              PICTURE X(01)  VALUE ' '.
01  NO-PARTROOT-MSG.
    02 FILLER                PICTURE S9(02)   VALUE +48
                                              COMPUTATIONAL.
    02 FILLER                PICTURE S9(02)  VALUE ZERO
                                              COMPUTATIONAL.
    02 FILLER                PICTURE X(01)  VALUE ' '.
    02 FILLER                PICTURE X(10) VALUE 'PART NO.  '.
    02 PART-NO               PICTURE X(15)
    02 FILLER                PICTURE X(17)  VALUE
                                              ' NOT IN DATA BASE'.
    02 CARR-RET              PICTURE X(01)  VALUE ' '.
01  NO-STOKSTAT-MSG.
    02 FILLER                PICTURE S9(02)   VALUE +45
                                              COMPUTATIONAL.
    02 FILLER                PICTURE S9(02)  VALUE ZERO
                                              COMPUTATIONAL.
    02 FILLER                PICTURE X(01)  VALUE ' '.
    02 FILLER                PICTURE X(14) VALUE 'STOCK RECORD  '.
    02 STOCK-KEY             PICTURE X(08).
    02 FILLER                PICTURE X(17)  VALUE
                                             ' NOT IN DATA BASE'.
    02 CARR-RET              PICTURE X(01)  VALUE ' '.
LINKAGE SECTION.
01  IO-TERM-PCB.
    02 IO-TERMINAL           PICTURE X(8).
    02 IO-RESERVE            PICTURE XX.
    02 IO-STATUS             PICTURE XX.
    02 INPUT-PREFIX          PICTURE X(12).
01  PARTFILE-PCB.
    02 PN-DBD-NAME           PICTURE X(8).
    02 PN-SEG-LEVEL          PICTURE XX.
    02 PN-STATUS-CODE        PICTURE XX.
    02 PN-PROC-OPTIONS       PICTURE XXXX.
    02 RESERVE-DLI           PICTURE S9(5) COMPUTATIONAL.
    02 PN-SEG-NAME-FB        PICTURE X(8).
PROCEDURE DIVISION.
    ENTRY 'DLITCBL' USING IO-TERM-PCB, PARTFILE-PCB.
INITIALIZE.
    MOVE SPACES TO STOKSTAT-WRITE-SW.
    MOVE 'OUTSTANDING WORK ORDERS=' TO DESC-1 OF LINE-5-AREA.
    MOVE '; QUANTITY=' TO              DESC-2 OF LINE-5-AREA.
GET-TRANSACTION.
    CALL 'CBLTDLI' USING UNIQ-FUNC, IO-TERM-PCB, TERM-IN-AREA.
CALL-INPUT-ANALYZER.
    CALL 'INPANAL' USING PARAM-TABLE, TERM-IN-AREA,
                         REFORM-MESSAGE, MSG-SEG-CNT.
CALL-PART-EDIT.
    MOVE PART-NO OF REFORM-MESSAGE   TO PART-NO-EDIT.
    CALL 'PNEDIT' USING PART-LINK.
FIND-PART-IN-DATA-BASE.
    MOVE PART-NO-EDIT        TO   PN-WORK.
    MOVE ROOT-KEY-WA         TO  KEY-VALUE.
    CALL 'CBLTDLI' USING UNIQ-FUNC,  PARTFILE-PCB, SEG-RET-AREA,
                         PARTROOT-SSA.
```

```
           IF PN-STATUS-CODE NOT EQUAL TO SPACES,
               GO TO PARTROOT-NOT-FOUND.
       PARTROOT-FOUND.
           MOVE CORRESPONDING SEG-RET-AREA TO LINE-1-AREA.
       FIND-STANINFO-IF-PRESENT.
           CALL 'CBLTDLI' USING NEXT-FUNC, PARTFILE-PCB, SEG-RET-AREA.
           IF (PN-STATUS-CODE EQUAL TO 'GB')
                            OR
              (PN-SEG-NAME-FB NOT EQUAL TO 'STANINFO')
               MOVE SPACES TO PROC-CODE OF LINE-1-AREA
                             ELSE
               MOVE CORRESPONDING STAN-INFO-RET TO LINE-1-AREA.
           PERFORM WRITE-LINE-1 THRU WRITE-LINE-1-EXIT.
       GET-UNIQUE-STOKSTAT.
           MOVE INPUT-SS-KEY TO SS-SSA-KEY-VALUE.
           CALL 'CBLTDLI' USING UNIQ-FUNC, PARTFILE-PCB, SEG-RET-AREA,
                                  PARTROOT-SSA, STOKSTAT-SSA.
           IF PN-STATUS-CODE EQUAL TO 'GE'
               GO TO STOKSTAT-NOT-FOUND..
       STOKSTAT-FOUND.
           MOVE CORRESPONDING STOCK-STATUS-RET TO LINE-2-AREA.
           PERFORM WRITE-LINE-2 THRU WRITE-LINE-2-EXIT.
           MOVE CORRESPONDING STOCK-STATUS-RET TO LINE-3-AREA.
           PERFORM WRITE-LINE-3 THRU WRITE-LINE-3-EXIT.
           MOVE CORRESPONDING STOCK-STATUS-RET TO LINE-4-AREA.
           MOVE 'ON' TO STOKSTAT-WRITE-SW.
           MOVE ZEROS TO STOCK-VAR OF LINE-4-AREA.
       GET-NEXT.
           CALL 'CBLTDLI' USING NEXT-FUNC, PARTFILE-PCB, SEG-RET-AREA.
           IF PN-STATUS-CODE EQUAL TO 'GB'
               GO TO END-CURR-ROOT.
           IF PN-SEG-NAME-FB EQUAL TO 'PARTROOT'  GO TO END-CURR-ROOT.
           IF PN-SEG-NAME-FB EQUAL TO 'STOKSTAT'  GO TO END-CURR-ROOT.
           IF PN-SEG-NAME-FB EQUAL TO 'CYCCOUNT'  GO TO CYCCOUNT-FOUND.
           IF PN-SEG-NAME-FB EQUAL TO 'BACKORDR'  GO TO BACKORDR-FOUND.
           GO TO GET-NEXT.
       CYCCOUNT-FOUND.
           COMPUTE STOCK-VAR OF LINE-4-AREA = PHYSICAL-COUNT  OF
                                              CYCLE-COUNT-RET     -
                                              TOTAL-STOCK OF
                                              CYCLE-COUNT-RET.
           PERFORM WRITE-LINE-4 THRU WRITE-LINE-4-EXIT.
           GO TO GET-NEXT.
       BACKORDR-FOUND.
           IF STOKSTAT-WRITE-SW EQUAL TO 'ON'
               PERFORM WRITE-LINE-4 THRU WRITE-LINE-4-EXIT.
           MOVE CORRESPONDING BACK-ORDER-RET TO LINE-5-AREA.
           PERFORM WRITE-LINE-5 THRU WRITE-LINE-5-EXIT.
           MOVE SPACES TO DESC-1 OF LINE-5-AREA.
           MOVE SPACES TO DESC-2 OF LINE-5-AREA.
           GO TO GET-NEXT.
       END-CURR-ROOT.
           IF STOKSTAT-WRITE-SW EQUAL TO 'ON'
               PERFORM WRITE-LINE-4 THRU WRITE-LINE-4-EXIT.
           GO TO END-IT.
```

```
PARTROOT-NOT-FOUND.
    MOVE PN-WORK TO PART-NO OF NO-PARTROOT-MSG.
    CALL 'CBLTDLI' USING ISRT-FUNC, IO-TERM-PCB, NO-PARTROOT-MSG.
    GO TO END-IT.
STOKSTAT-NOT-FOUND.
    MOVE INPUT-SS-KEY TO STOCK-KEY OF NO-STOKSTAT-MSG.
    CALL 'CBLTDLI' USING ISRT-FUNC, IO-TERM-PCB, NO-STOKSTAT-MSG.
    GO TO END-IT.
WRITE-LINE-1.
    CALL 'CBLTDLI' USING ISRT-FUNC, IO-TERM-PCB, LINE-1-AREA.
WRITE-LINE-1-EXIT.  EXIT.
WRITE-LINE-2.
    CALL 'CBLTDLI' USING ISRT-FUNC, IO-TERM-PCB, LINE-2-AREA.
WRITE-LINE-2-EXIT.  EXIT.
WRITE-LINE-3.
    CALL 'CBLTDLI' USING ISRT-FUNC, IO-TERM-PCB, LINE-3-AREA.
WRITE-LINE-3-EXIT.  EXIT.
WRITE-LINE-4.
    CALL 'CBLTDLI' USING ISRT-FUNC, IO-TERM-PCB, LINE-4-AREA.
    MOVE SPACES TO STOKSTAT-WRITE-SW.
WRITE-LINE-4-EXIT.  EXIT.
WRITE-LINE-5.
    CALL 'CBLTDLI' USING ISRT-FUNC, IO-TERM-PCB, LINE-5-AREA.
WRITE-LINE-5-EXIT.  EXIT.
END-IT.
    GOBACK.
```

CONVERSATIONAL APPLICATION PROGRAM EXAMPLES USING PL/I

This application program illustrates use of the 3270 Model 2 as a simple calculator. The program provides for addition, subtraction, multiplication, and division.

A sample problem for this transaction (whose PSB=HIMAJC03) is provided in the IMS/VS Installation Guide. The examples that follow, however, are entirely independent of the sample problem. No data base is used, and only input to and output from the application program are illustrated.

Example Number 1:

```
/FOR DFSMO2    (for the 3270, Model 1)

/FOR TUBFMT    (for the 3270, Model 2)
```

The first entry is the MOD name (/FOR DFSMO2). Tube is the transaction code.

Display back says:

```
START INPUT HERE.¢
```

You enter one number, the sign (+,-,*,/), and the second number.

```
START INPUT HERE.¢ 555+444.55
```

Display back is the answer, followed by two questions; these are to be answered either YY, YN, or NN. The fourth possibility is NY, which is not correct in this program;

```
YOUR ANSWER IS          999.55

   TWO QUESTIONS.  DO YOU WISH TO CONTINUE?
   AND SHOULD THIS RESULT BE USED AS SUBTOTAL?
   ANS QUESTIONS BY YY OR YN OR NN.  ANSWER HERE. ¢ NY
```

Display back, and the application program ends the conversation:

```
NOT CORRECT ANSWER.  WILL ASSUME ANS=NN.  PROBLEM END.
```

Example Number 2:

Entry to 3270:

```
/FOR TUBE
```

Display back asks for input.

```
START INPUT HERE. ¢ 1234.34+1234
```

Display back gives answer to the problem and asks two questions.

```
YOUR ANSWER IS           2468.34

   TWO QUESTIONS.  DO YOU WISH TO CONTINUE?
   AND SHOULD THIS RESULT BE USED AS SUBTOTAL?
   ANS QUESTIONS BY YY OR YN OR NN.  ANSWER HERE. ¢ YY
```

Because you want the answer to be used as a subtotal, internally in the scratchpad user work area, this is stored:

```
SPA.IN_TEXT=000000002468344;
```

The display returned, and the new subtraction problem is entered:

```
RESULT WILL BE USED AS SUBTOTAL.  START INPUT HERE. ¢ 1234.34-2468.34
```

The display returned is the answer to the above subtraction problem added to the subtotal stored in the scratchpad work area, and the two questions are asked again. This time you want to continue the conversation, but do not want to have a subtotal carried over to the next problem:

```
YOUR ANSWER IS          1234.00

   TWO QUESTIONS.  DO YOU WISH TO CONTINUE?
   AND SHOULD THIS RESULT BE USED AS A SUBTOTAL?
   ANS QUESTIONS BY YY OR YN OR NN. ANSWER HERE. ¢ YN
```

The display returned a message, after which you entered a multiplication problem:

```
CONTINUE,START INPUT HERE. ¢ 4444*44
```

The display returned the answer to the multiplication problem and the two questions. The answer to the questions was YN:

```
YOUR ANSWER IS          195536.00

   TWO QUESTIONS.  DO YOU WISH TO CONTINUE?
   AND SHOULD THIS RESULT BE USED AS SUBTOTAL?
   ANS QUESTIONS BY YY OR YN OR NN.  ANSWER HERE. ¢ YN
```

The display returned a message, after which you entered a division problem:

```
CONTINUE, START INPUT HERE. ¢ 335567.56/33
```

The display returned the answer to the division problem and the two questions. The answer to the questions was NN:

```
YOUR ANSWER IS          10168.71

   TWO QUESTIONS.  DO YOU WISH TO CONTINUE?
   AND SHOULD THIS RESULT BE USED AS SUBTOTAL?
   ANS QUESTIONS BY YY OR YN OR NN. ANSWER HERE. ¢ NN
```

The message displayed then was:

```
ANS WAS NN.  CONVERSATION ENDED.
```

The conversation is over.

PL/I OPTIMIZING COMPILER EXAMPLE

FILE: PLIPROG1 TEST A GPD COMMON CMS

```
 /*****  PL/I  EXAMPLE OF A CONVERSATIONAL PROGRAM  *****/
 /*******************************************************/

 DLITPLI: PROCEDURE (TERMINAL) OPTIONS (MAIN,REENTRANT) REORDER;

 /*********************************************************************/
 /*                                                                   */
 /*    THIS PROGRAM IS AN EXAMPLE OF CONVERSATIONAL PROCESSING.  IT   */
 /*    IS WRITTEN IN PL/I FOR THE PL/I OPTIMIZING COMPILER.           */
 /*                                                                   */
 /*    THE PROGRAM WILL ACCEPT A SIMPLE EXPRESSION CONSISTING OF TWO  */
 /*    OPERANDS SEPARATED BY AN OPERATOR, WILL COMPUTE THE VALUE OF   */
 /*    THE EXPRESSION AND RETURN THE ANSWER.  THE EXPRESSION MUST BE  */
 /*    IN THE FORM:  NNNOMMM, WHERE NNN AND MMM ARE NUMBERS WITH NO   */
 /*    MORE THAN 7 DIGITS, AND O IS ONE OF THE OPERATORS +,-,* OR /.  */
 /*    A MAXIMUM OF SEVEN CHARACTERS CAN PRECEDE OR FOLLOW THE        */
 /*    OPERATOR.  IF ONE OR BOTH OF THE OPERANDS IS OMITTED, IT WILL  */
 /*    BE ASSUMED TO BE ZERO.  IF MORE THAN ONE OPERATOR IS ENTERED,  */
 /*    ALL BUT THE LAST WILL BE CONVERTED TO ZERO AND THE COMPUTATION */
 /*    WILL PROCEED.  ANY BLANKS OR NON-DIGITS EMBEDDED IN EITHER     */
 /*    OPERAND WILL BE CONVERTED TO ZERO AND THE COMPUTATION WILL     */
 /*    PROCEED.   OPTIONALLY YOU CAN REQUEST THAT THE ANSWER BE ADDED */
 /*    TO A SUBTOTAL MAINTAINED OF PRECEDING COMPUTATIONS.            */
 /*                                                                   */
 /*********************************************************************/

1/*********************************/
 /*  DECLARE LOGICAL TERMINAL PCB   */
 /*********************************/

      DECLARE TERMINAL POINTER;
      DECLARE 1 IOPCB BASED (TERMINAL),
                2 IO_TERMINAL CHARACTER (8),
                2 IO_RESERVED CHARACTER (2),
                2 STAT_CODE CHARACTER (2),
                2 IN_PREFIX,
                  3 PRE_DATE FIXED DECIMAL (7),
                  3 PRE_TIME FIXED DECIMAL (7),
                  3 PRE_MSG_COUNT FIXED BINARY (31);

 /*******************************/
 /*  DECLARE SCRATCHPAD AREA  */
 /*******************************/

      DECLARE 1 SPA,
                2 DL FIXED BINARY (31),
                2 X CHARACTER (1),
                2 FLAG CHARACTER (1),
                2 RESERVED CHARACTER (2),
                2 TRAN CHARACTER (8),
                2 COUNT CHARACTER (1),
                2 IN_TEXT FIXED DECIMAL (15,2),
                2 PADDING CHARACTER (75);

 /*****************************************/
```

```
FILE: PLIPROG1 TEST     A   GPD COMMON CMS


 /*   DECLARE INPUT AND OUT MESSAGE AREAS  */
 /*****************************************/

      DECLARE 1 INPUT_MSG,
                2 LLIN FIXED BINARY (31),
                2 ZZIN FIXED BINARY (15),
                2 TXTIN CHARACTER (80),

              1 OUTPUT_MSG,
                2 LLOUT FIXED BINARY (31),
                2 ZZOUT FIXED BINARY (15) INITIAL (ERASE),
                2 TXTOUT CHARACTER (178);

1/*******************************/
 /*  DECLARE MESSAGE CONTENTS  */
 /*******************************/

      DECLARE
          (MSG9 CHARACTER (18) INITIAL
                ('START INPUT HERE.¢'),   /* LAST CHAR SMI */
           MSG10 CHARACTER (41) INITIAL
                ('  TWO QUESTIONS. DO YOU WISH TO CONTINUE?'),
           MSG11 CHARACTER (46) INITIAL
                ('  AND SHOULD THIS RESULT BE USED AS SUBTOTAL?'),
           MSG12 CHARACTER (35) INITIAL
                ('  ANS QUESTIONS BY YY OR YN OR NN. '),
           MSG14 CHARACTER (33) INITIAL
                ('RESULT WILL BE USED AS SUBTOTAL. '),
           MSG15 CHARACTER (55) INITIAL
                (' NOT CORRECT ANSWER. WILL ASSUME NN.  PROBLEM END.'),
           MSG16 CHARACTER (34) INITIAL
                ('  ANS WAS NN.  CONVERSATION ENDED.'),
           MSG17 CHARACTER (49) INITIAL
                ('YOU MUST ENTER 2 OPERANDS WITH OPERATOR BETWEEN. '),
           MSG19 CHARACTER (40) INITIAL
                (' YOU ARE NOT ALLOWED TO DIVIDE BY ZERO.'),
           MSG20 CHARACTER (9) INITIAL
                ('REENTER. '),
           MSG21 CHARACTER (44) INITIAL
                (' ONE OR BOTH OPERANDS EXCEEDS 7 CHARACTERS. '),
           MSG22 CHARACTER (38) INITIAL
                ('UNSPECIFIED ERROR. PGM ENDS. ONCODE = '),
           MSG23 CHARACTER (15) INITIAL
                ('YOUR ANSWER IS:'),
           MSG24 CHARACTER (10) INITIAL
                ('CONTINUE, '),
           MSG25 CHARACTER (23) INITIAL
                ('SPA RETURN STAT CODE = '),
           MSG26 CHARACTER (23) INITIAL
                ('GET UNIQUE STAT CODE = '),
           MSG27 CHARACTER (21) INITIAL
                ('GET NEXT STAT CODE = '),
           MSG28 CHARACTER (27) INITIAL
                ('NO VALID OPERATOR ENTERED.')
            ) STATIC;
```

```
FILE: PLIPROG1 TEST     A   GPD COMMON CMS


1/*********************************/
 /*  MISCELLANEOUS DECLARATIONS  */
 /*********************************/

      DECLARE
           RESULT FIXED DECIMAL (15,2),
           CRESULT PIC'S,SSS,SSS,SSS,SS9.V99',
           STRING CHARACTER (80) VARYING,
          (OPERAND1,OPERAND2) FIXED DECIMAL (9,2),
          (A,S,M,D,L,OPERATOR) FIXED BINARY (15),
           THREE FIXED BINARY (31) STATIC INITIAL (3),
           GU CHARACTER (4) STATIC INITIAL ('GU'),
           GN CHARACTER (4) STATIC INITIAL ('GN'),
           ISRT CHARACTER (4) STATIC INITIAL ('ISRT'),
           TXTANS CHARACTER (2),
           PLITDLI ENTRY,
           RETURN_POINT LABEL (TERMINATE,SAVE_INFO),
           ERASE FIXED BINARY (15) STATIC INITIAL (32),
                /* ERASE INITIALIZED TO X'0020' */
           NL CHARACTER (1) STATIC;
           UNSPEC (NL) = '00010101'B;  /* INITIALIZE NL TO X'15' */

 /***************/
 /*  ON UNITS  */
 /***************/

      ON CONVERSION BEGIN;
           DECLARE ONCHAR BUILTIN;
           ONCHAR = '0';
           END;

      ON ZERODIVIDE BEGIN;
           IF COUNT = '2' THEN COUNT = '1';
           RETURN_POINT = SAVE_INFO;
           LLOUT = LENGTH (MSG19) + LENGTH (MSG20) + LENGTH (MSG9) + 5;
           TXTOUT = MSG19 || MSG20 || NL || MSG9;
           GO TO OUTPUT_MESSAGE;
           END;

      ON ERROR BEGIN;
           DECLARE ONCODE BUILTIN,
                   CONCODE PIC'9999';
           CONCODE = ONCODE;
           RETURN_POINT = TERMINATE;
           LLOUT = LENGTH (MSG22) + LENGTH (CONCODE) + 4;
           TXTOUT = MSG22 || CONCODE;
           GO TO OUTPUT_MESSAGE;
           END;

1/*******************************/
 /*  BEGIN_EXECUTABLE_PROGRAM  */
 /*******************************/

 /************************/
```

```
FILE: PLIPROG1 TEST     A   GPD COMMON CMS


 /*  FIRST CALL TO SPA  */
 /***********************/

 BEGINNING:
      CALL PLITDLI (THREE,GU,TERMINAL,SPA);
      IF STAT_CODE = 'QC' THEN RETURN;
      IF STAT_CODE ¬= '  ' THEN GO TO BAD_GU;
      IF (COUNT < '1') | (COUNT > '4') THEN CCUNT = '1';

 /***********************/
 /*  GET TEXT SEGMENT  */
 /***********************/

      CALL PLITDLI (THREE,GN,TERMINAL,INPUT_MSG);
      IF STAT_CODE = 'QD' THEN GO TO BAD_NN;
      IF STAT_CODE ¬= '  ' THEN GO TO BAD_GN;
      IF COUNT = '1' | COUNT = '3' | COUNT = '4'

 /***************************/
 /*  PERFORM CALCULATIONS  */
 /***************************/

           THEN DO;
                L = LLIN - 4;
                IF L > 15 THEN GO TO LNG_ERROR;   /* (2*7) + 1 = 15 */
                STRING = SUBSTR (TXTIN,1,L);
                A = INDEX (STRING,'+');
                S = INDEX (STRING,'-');
                M = INDEX (STRING,'*');
                D = INDEX (STRING,'/');
                OPERATOR = MAX (A,S,M,D);
                IF OPERATOR > 8 THEN GO TO LNG_ERROR;
                IF L - OPERATOR > 7 THEN GO TO LNG_ERROR;
                IF OPERATOR = 0 THEN GO TO OP_ERROR;
                OPERAND1 = SUBSTR (STRING,1,OPERATOR-1);
                OPERAND2 = SUBSTR (STRING,OPERATOR+1,L-OPERATOR);
                IF A > 0 THEN RESULT = OPERAND1 + OPERAND2;
                     ELSE IF S > 0 THEN RESULT = OPERAND1 - OPERAND2;
                     ELSE IF M > 0 THEN RESULT = OPERAND1 * OPERAND2;
                     ELSE RESULT = OPERAND1 / OPERAND2;
                IF COUNT = '1' THEN COUNT = '2';
                IF COUNT = '3' THEN DO;
                     RESULT = RESULT + IN_TEXT;
                     COUNT = '2';
                     END;
                IF COUNT = '4' THEN DO;
                     IN_TEXT = 0;
                     COUNT = '2';
                     END;

1/*************************************/
 /*  OUTPUT ANSWER AND TWO QUESTIONS  */
 /*************************************/

                IN_TEXT, CRESULT = RESULT;
```

```
FILE: PLIPROG1 TEST     A   GPD COMMON CMS

                LLOUT = LENGTH (MSG23) + LENGTH (CRESULT) +
                        LENGTH (MSG10) + LENGTH (MSG11) +
                        LENGTH (MSG12) + LENGTH (MSG9) + 7;
                TXTOUT = MSG23 || CRESULT || NL ||
                         MSG10 || NL ||
                         MSG11 || NL ||
                         MSG12 || MSG9;
                RETURN_POINT = SAVE_INFO;

                END;

 /*****************************/
 /*  CONTINUING CONVERSATION  */
 /*****************************/

           ELSE DO;  /*  COUNT = '2'  */
                TXTANS = SUBSTR (TXTIN,1,2);

                IF TXTANS = 'YY' THEN DO;
                     RETURN_POINT = SAVE_INFO;
                     LLOUT = LENGTH (MSG14) + LENGTH (MSG9) + 4;
                     TXTOUT = MSG14 || MSG9;
                     COUNT = '3';
                     END;

                ELSE IF TXTANS = 'YN' THEN DO;
                     RETURN_POINT = SAVE_INFO;
                     LLOUT = LENGTH (MSG24) + LENGTH (MSG9) + 4;
                     TXTOUT = MSG24 || MSG9;
                     COUNT = '4';
                     END;

                ELSE IF TXTANS = 'NN' THEN DO;
                     RETURN_POINT = TERMINATE;
                     LLOUT = LENGTH (MSG16) + 4;
                     TXTOUT = MSG16 || NL;
                     END;

                ELSE GO TO BAD_NN;

                END;

1/*****************************/
 /*  INSERT OUTPUT MESSAGE  */
 /*****************************/

 OUTPUT_MESSAGE:
      CALL PLITDLI (THREE,ISRT,TERMINAL,OUTPUT_MSG);
      GO TO RETURN_POINT;

 /*****************************/
 /*  SAVE INFORMATION IN SPA  */
 /*****************************/

 SAVE_INFO:
```

```
FILE: PLIPROG1 TEST     A   GPD COMMON CMS


      CALL PLITDLI (THREE,ISRT,TERMINAL,SPA);
      IF STAT_CODE = '  ' THEN GO TO BEGINNING;
           ELSE GO TO SAVE_ERROR;

 /****************/
 /*   TERMINATE  */
 /****************/

 TERMINATE:
      TRAN = ' ';
      CALL PLITDLI (THREE,ISRT,TERMINAL,SPA);
      RETURN;

1/*********************/
 /*  ERROR ROUTINES  */
 /*********************/

 LNG_ERROR:
      RETURN_POINT = SAVE_INFO;
      LLOUT = LENGTH (MSG17) + LENGTH (MSG21) + LENGTH (MSG20) +
                      LENGTH (MSG9) + 7;
      TXTOUT = MSG17 || NL || MSG21 || NL || MSG20 || NL || MSG9;
      GO TO OUTPUT_MESSAGE;

 OP_ERROR:
      RETURN_POINT = SAVE_INFO;
      LLOUT = LENGTH (MSG28) + LENGTH (MSG20) + LENGTH (MSG9) + 6;
      TXTOUT = MSG28 || NL || MSG20 || NL || MSG9;
      GO TO OUTPUT_MESSAGE;

 SAVE_ERROR:
      RETURN_POINT = TERMINATE;
      LLOUT = LENGTH (MSG25) + LENGTH (STAT_CODE) + 4;
      TXTOUT = MSG25 || STAT_CODE;
      GO TO OUTPUT_MESSAGE;

 BAD_NN:
      RETURN_POINT = TERMINATE;
      LLOUT = LENGTH (MSG15) + 4;
      TXTOUT = MSG15;
      GO TO OUTPUT_MESSAGE;

 BAD_GU:
      RETURN_POINT = TERMINATE;
      LLOUT = LENGTH (MSG26) + 4;
      TXTOUT = MSG26 || STAT_CODE;
      GO TO OUTPUT_MESSAGE;

 BAD_GN:
      RETURN_POINT = TERMINATE;
      LLOUT = LENGTH (MSG27) + 4;
      TXTOUT = MSG27 || STAT_CODE;
      GO TO OUTPUT_MESSAGE;

      END DLITPLI;
```

MESSAGE FORMAT SERVICES

The following message format service statements show the message descriptions and device formats used in conjunction with the conversational PL/I programs illustrated elsewhere in this chapter. This format applies only to the 3270 Model 2.

MEMBER NAME TUBFMT

```
TUBFMT      FMT
******      FORMAT FOR TUBE PROGRAM
            DEV     TYPE=3270,FEAT=IGNORE
DPAGE1      DPAGE   CURSOR=((5,22))
FLDX        DFLD    POS=(03,02),LTH=10,ATTR=(NODISP,PROT)
FLDY        DFLD    POS=(04,02),LTH=9,ATTR=PROT
FLD1        DFLD    POS=(05,02),LTH=05,ATTR=PROT
FLD2        DFLD    POS=(05,08),LTH=13,ATTR=PROT
INPUT       DFLD    POS=(05,22),LTH=18,ATTR=HI
DPAGE2      DPAGE   CURSOR=((5,22))
FLD1        DFLD    POS=(01,02),LTH=04,ATTR=PROT
FLD2        DFLD    POS=(01,07),LTH=26,ATTR=PROT
FLD3        DFLD    POS=(02,02),LTH=41,ATTR=PROT
FLD4        DFLD    POS=(03,02),LTH=46,ATTR=PROT
FLD5        DFLD    POS=(04,02),LTH=35,ATTR=PROT
FLD6        DFLD    POS=(05,08),LTH=13,ATTR=PROT
INPUT       DFLD    POS=(05,22),LTH=02,ATTR=HI
FLDN        DFLD    POS=(18,02),LTH=4,ATTR=(NODISP,PROT)
DPAGE3      DPAGE   CURSOR=((5,22))
FLDA        DFLD    POS=(03,02),LTH=06,ATTR=(NODISP,PROT)
FLDB        DFLD    POS=(04,02),LTH=6,ATTR=PROT
FLDC        DFLD    POS=(04,09),LTH=27,ATTR=PROT
FLDD        DFLD    POS=(05,04),LTH=17,ATTR=PROT
INPUT       DFLD    POS=(05,22),LTH=18,ATTR=HI
DPAGE4      DPAGE
F1          DFLD    POS=(03,02),LTH=5,ATTR=(NODISP,PROT)
F2          DFLD    POS=(03,08),LTH=06,ATTR=PROT
F3          DFLD    POS=(03,15),LTH=10,ATTR=PROT
F4          DFLD    POS=(11,34),LTH=12,ATTR=(PROT,HI)
F5          DFLD    POS=(13,37),LTH=06,ATTR=(PROT,HI)
DPAGE5      DPAGE
FL1         DFLD    POS=(03,02),LTH=5,ATTR=(NODISP,PROT)
FL2         DFLD    POS=(03,08),LTH=03,ATTR=PROT
FL3         DFLD    POS=(03,12),LTH=17,ATTR=PROT
FL4         DFLD    POS=(04,06),LTH=21,ATTR=PROT
FL5         DFLD    POS=(11,37),LTH=07,ATTR=(PROT,HI)
FL6         DFLD    POS=(13,38),LTH=04,ATTR=(PROT,HI)
DPAGE6      DPAGE   CURSOR=((5,22))
A1          DFLD    POS=(02,02),LTH=52,ATTR=PROT
A2          DFLD    POS=(03,02),LTH=49,ATTR=PROT
A3          DFLD    POS=(05,02),LTH=05,ATTR=PROT
A4          DFLD    POS=(05,08),LTH=11,ATTR=PROT
INPUT       DFLD    POS=(05,22),LTH=18,ATTR=HI
A6          DFLD    POS=(04,02),LTH=08,ATTR=PROT
A7          DFLD    POS=(04,11),LTH=08,ATTR=(NODISP,PROT)
            FMTEND
TUBEMOD1    MSG     TYPE=OUTPUT,SOR=(TUBFMT,IGNORE),NXT=TUBEMID
            MFLD    FLD1,LTH=5
            MFLD    FLD2,LTH=13
            MFLD    (INPUT,'-----')
            MSGEND
TUBEMOD     MSG     TYPE=OUTPUT,SOR=(TUBFMT,IGNORE),NXT=TUBEMID
            LPAGE   SOR=DPAGE1,COND=(MSG1,=,'START')
MSG1        MFLD    FLDX,LTH=5
            MFLD    (FLD1,'START')
```

```
MEMBER NAME TUBFMT

            MFLD    FLD2,LTH=12
            LPAGE   SOR=DPAGE2,COND=(MSG2,=,'YOUR')
            SEG
MSG2        MFLD    FLDN,LTH=4
            MFLD    (FLD1,'YOUR')
            MFLD    FLD2,LTH=26
            SEG
            MFLD    FLD3,LTH=41
            SEG
            MFLD    FLD4,LTH=46
            SEG
            MFLD    FLD5,LTH=35
            MFLD    FLD6,LTH=13
            MFLD    (INPUT,'--')
            LPAGE   SOR=DPAGE3,COND=(MSG3,='RESULT')
MSG3        MFLD    FLDA,LTH=6
            MFLD    (FLDB,'RESULT')
            MFLD    FLDC,LTH=27
            MFLD    FLDD,LTH=17
            LPAGE   SOR=DPAGE4,COND=(MSG4,=,'  ANS')
MSG4        MFLD    F1,LTH=5
            MFLD    (F2,'ANSWER')
            MFLD    F3,LTH=10
            MFLD    F4,LTH=12
            MFLD    F5,LTH=06
            LPAGE   SOR=DPAGE5,COND=(MSG5,=,'  NOT')
MSG5        MFLD    FL1,LTH=5
            MFLD    (FL2,'NOT')
            MFLD    FL3,LTH=17
            MFLD    FL4,LTH=21
            MFLD    FL5,LTH=7
            MFLD    FL6,LTH=4
            LPAGE   SOR=DPAGE1,COND=(MSG6,=,'CONTINUE, ')
MSG6        MFLD    FLDX,LTH=10
            MFLD    (FLDY,'CONTINUE:')
            MFLD    FLD1,LTH=5
            MFLD    FLD2,LTH=12
            LPAGE   SOR=DPAGE6,COND=(MSG7,=,'REENTER.')
            MFLD    A1,LTH=51
MSG7        MFLD    A7,LTH=8
            MFLD    (A6,'REENTER.')
            MFLD    (A3,'START')
            MFLD    (A4,'INPUT HERE:')
            LPAGE   SOR=DPAGE6,COND=(MSG8,=,'REENTER.')
            MFLD    A2,LTH=40
MSG8        MFLD    A7,LTH=8
            MFLD    (A6,'REENTER.')
            MFLD    (A3,'START')
            MFLD    (A4,'INPUT HERE:')
            LPAGE   SOR=DPAGE6,COND=(MSG9,=,'REENTER.')
            MFLD    A1,LTH=52
            MFLD    A2,LTH=49
MSG9        MFLD    A7,LTH=8
            MFLD    (A6,'REENTER.')
            MFLD    (A3,'START')
            MFLD    (A4,'INPUT HERE:')
            MSGEND
TUBEMID     MSG     TYPE=INPUT,SOR=(TUBFMT),NXT=TUBEMOD
            MFLD    INPUT,LTH=18
            MSGEND
            END
```

CHAPTER 7. APPLICATION PROGRAMMING TESTING AIDS

DATA LANGUAGE/I TEST PROGRAM (DFSDDLT0)

The Data Language/I (DL/I) test program is an IMS/VS application program that issues calls to DL/I based upon control statement information. It compares, optionally, the results of those calls with expected results that are also provided in control statements. It is used to test DL/I.

To a limited extent, this program can be used as a general purpose data base utility program. However, the control statement language does not lend itself well to executing large volumes of calls. It is useful as a debugging aid because it can display DL/I control blocks. It provides an easy method of executing any call against any data base.

GENERAL DESCRIPTION

The DL/I test program is a control statement processor. There are four types of control statements used by the program:

- Status statements--establish print options and select processing PCB.
- Comments statements--conditionally or unconditionally print comments.
- Call statements--format the desired DL/I call.
- Compare statements--compare anticipated results with actual results.

The status statement is used to establish print options and to select which PCB within a PSB will be used. The call to be issued is provided in the CALL statement. A COMPARE statement is optional and is used to tell the program what the results of this call should be in the data base PCB and in the user input/output area. Various print and display options are available; these are based on whether the results of the call agree with the data in the COMPARE statement. COMMENTS statements are also optional. As the name implies, they are only comments and can be used by the programmer at his discretion. As will be seen later, there are two types of comments: conditional and unconditional.

The general sequence of operation is to read CALL statements until a noncontinued CALL statement is detected. The DL/I call is issued based on data in the CALL statements. The program then reads the next control statement. If a COMPARE statement is read, it compares the contents of the COMPARE statement with the corresponding field in the PCB, or, if a data COMPARE statement, with the data in the user input/output area. The comments, call, compare, PCB, input/output area, and compare data are printed if requested. If any control statement other than a COMPARE statement is read after a call was issued, the results of the prior call are printed first and the new control statement is then processed.

INTERFACES

Module DFSDDLS0 must be link-edited with DFSLI000 and placed in IMSVS.PESLIB under the name DFSDDLT0.

JOB	This statement initiates the job.
EXEC	This statement specifies the program name, or invokes a cataloged procedure. The required format is: PGM=DFSRRC00,PARM='AAA,DFSDDLT0,BBBBBBBB, CCCCCCCC,DDDDDDDD' where AAA is the region type and BBBBBBBB is the name of the PSB to be used. Parameters CCCCCCC and DDDDDDD are optional, and can be used to specify symbolic input terminal and output terminal names, respectively. Refer to the IMS/VS System Programming Reference Manual for other parameters that can be used.
STEPLIB DD	Defines the partitioned data set named IMSVS.RESLIB. If EXIT routine modules are used, they should be placed into this library or into another PDS concatenated to this library.
IMS DD	This statement defines two concatenated data sets. The first DD statement defines the library containing the PSB to be used by the test program. The second DD statement defines the library containing the DBD of the data base to be processed.
database DD	This statement references a specific data base. There should be one statement for each data base to be processed. In each statement the ddname must agree with the ddname specified in the DBD.
IEFRDER DD	This statement defines the log data set, if one is desired. A DD DUMMY statement may be used if a log is not desired. One form or the other of this statement is required.
PRINTDD DD	This statement defines the output data set for the test program, including displays of control blocks using the SNAP call. It must conform to the OS SNAP data set requirements.
SYSUDUMP DD	This statement is optional and is used by the test program only when normal termination is not possible.

SYSIN DD	This statement defines the control statement input data set.
SYSIN2 DD	This is an optional secondary input statement. See the description of "Special Control Statement Formats" for details.

CONTROL STATEMENTS

In the control statement formats below, the "$" indicates those fields which are usually filled in; the absence of the "$" indicates that the field can be left blank and the default used. If position 1 is left blank on any control statement, the statement type defaults to the prior statement type.

STATUS Statement

The STATUS statement establishes print options and determines the PCB that subsequent calls are to be issued against.

The format of the STATUS statement is as follows:

Position		Contents
$ 1	=	S identifies this as a STATUS statement.
2	=	Output device option. blank - use PRINTDD when in a DLI region; use I/O PCB in the MSG region. 1 - use PRINTDD in MSG region if the DD statement is provided; otherwise, use I/O PCB. A - same as if 1, and disregard all other fields in this STATUS statement.
3	=	Print comment option. blank - do not print. 1 - print always. 2 - print only if compare done and unequal.
4	=	Not used.
5	=	Print call option. blank - do not print. 1 - print always. 2 - print only if compare done and unequal.
6	=	Not used.
7	=	Print compare option. blank - do not print. 1 - print always. 2 - print only if compare done and unequal.
8	=	blank.

Position		Contents
9	=	Print PCB option. blank - do not print. 1 - print always. 2 - print only if compare done and unequal.
10	=	Not used.
11	=	Print segment option. blank - do not print. 1 - print always. 2 - print only if compare done and unequal.
12 - 15	=	Reserved.
16 - 23	=	DBD name. This determines the PCB against which subsequent calls will be issued; hence, it must be a DBD name given in one of the PCBs in the PSB. The default PCB is the first data-base-PCB in the PSB. If positions 16 through 23 are blank, the current PCB is used. If positions 16 through 18 are blank, and positions 19 through 23 are not blank, then the non-blank positions are interpreted as the relative number of the desired data-base-PCB in the PSB. The number must be right-justified to position 23, but need not contain leading zeroes. The user must insure that the relative data-base-PCB exists in the PSB because no checks are made to insure that a proper PCB is obtained in this manner.
24	=	Print status option. 1 - do not use print option in this statement. 2 - do not print this STATUS statement. 3 - do not print this STATUS statement or use print option. blank - use print option and print this statement.
25 - 28	=	PCB processing option -- This is optional and is only used when two PCBs have the same DBD name but different processing options. If non-blank, it is used in addition to the DBD name in positions 16 through 23 to select which PCB in the PSB to use. This must appear as it does in the processing option of the PCB desired.
29 - 80	=	Not used.

If no STATUS statement is read, the default PCB is first data base-PCB in the PSB, and the print status option is 2. New STATUS statements can be anywhere in the SYSIN stream, changing either the data base to be referenced or the options.

COMMENTS Statement

There are two types of COMMENTS statements. The first, the unconditional statement, allows for unlimited comments, all of which are printed. The second type, the conditional statement, allows only limited comments, which are printed or not depending on other factors as described below.

As the name implies, information on these statements is treated by the system as comments only. No action, other than printing, is taken when a COMMENTS statement is read.

Unconditional:

Position		Contents
$ 1	=	U specifies an unconditional COMMENTS statement.
2 - 80	=	Comments - any number of unconditional COMMENTS statements are allowed; they are printed when read. Time and date of printing are printed with each unconditional COMMMENTS statement.

Conditional:

Position		Contents
$ 1	=	T specifies a conditional COMMENTS statement.
2 - 80	=	Comments - up to 5 conditional COMMENTs statements per call are allowed; no continuation mark in position 72 is required. Printing is conditioned on the STATUS statement. Printing is deferred until after the following call and optional compare are executed, but prior to the printing of the following call.

CALL Statement

The CALL statement identifies the type of IMS/VS call to be made, and supplies information to be used by the call.

Position		Contents
$ 1	=	L identifies this as either a CALL or DATA statement.
3	=	SSA level (optional).
4	=	Format options-- U, if columns 16 onward are unformatted, with no blanks separating fields. Blank, for formatted calls with intervening blanks in positions 24, 34, and 37. V, for the first statement describing a variable length segment, when inserting or replacing only one variable length segment. It is also used for the first statement describing the first segment of multiple variable length segments. M, for the second through last statements that begin

Position		Contents
		data for a variable length segment, when inserting or replacing multiple variable length segments. P, when inserting or replacing via path calls. It is used only in the first statement of fixed length segment statements in path calls containing both variable and fixed length segments.
5 - 8	=	Number of times to repeat this call (optional) in the range of 0001 through 9999.
$ 10 - 13	=	DL/I, application program call function.
	=	DATA, indicates that this statement contains data to be used in an ISRT, REPL, SNAP, CHPT, or LOG call. See the following section on DATA statements for usage.
	=	CONT, if a continuation statement for field data that was too long for previous CALL statement.
$ 16 - 23	=	SSA segment name.
24	=	Not used.
$ 25	=	(, if segment is qualified.
26 - 33	=	SSA field name.
34	=	Not used.
$ 35 - 36	=	DL/I call operator or operators.
37	=	Not used.
$ 38 - XX	=	Field value (where the maximum value of XX=70).
$ XX + 1	=	), end character.
$ 72	=	Nonblank, if more SSAs. Blank, if this is the only or last SSA.

Position 3, the SSA level, is usually blank. If blank, the first CALL statement fills SSA 1, and each following CALL statement fills the next lower SSA. If the SSA level, position 3, is nonblank, the statement fills the SSA at that level, and the following CALL statement fills the next lower SSA.

Position 4 contains a U to indicate an alternative format for the CALL statement. In this case, from position 16 on is the exact SSA with no intervening blanks in positions 24, 34 and 37. If command calls (for example, *D) are to be used, then the U must specified.

Positions 5 through 8 are usually blank, but if used, must be right-justified. The identical call is repeated as specified in positions 5 through 8.

Positions 10 through 13 - the DL/I call function is required only for the first SSA of the call.

Positions 16 through 23 - the segment name is not specified for unqualified calls.

If there are multiple SSAs in the call, each SSA should be entered in positions 16 through 23 of a separate statement. A non-blank in position 72 of any statement indicates that another SSA follows. Positions 1 and 10 through 13 are blank for the second through last SSAs.

If the field value extends past 71, there is a nonblank in position 72 and CONT in positions 10 through 13 of the next statement, with the field value continued starting in position 16. Maximum field value is 256 bytes.

An alternate format for the CALL statement is available by putting a U in position 4. If you use this option, you must start the exact SSA in position 16, with no intervening blanks in positions 24, 34, and 37. To continue an unformatted SSA, put a nonblank character in position 72, a U in position 4, and CONT in positions 10-13 of the next statement. Include the data of the SSA that is continuing in positions 16 through 71. Maximum size for an SSA is 290 bytes. For additional information on SSAs, refer to the section "Segment Search Argument" in the "Data Base Batch Programming" chapter of this manual.

The maximum number of levels for this program is the same as the IMS/VS limit, which is 15.

DATA Statement

DATA statements provide IMS/VS with segment information required for ISRT, REPL, SNAP, LOG, and CHKP calls.

For an ISRT, REPL, SNAP, LOG, or CHKP call, statements containing segment data must follow immediately after the last (non-continued) CALL statement. The DATA statements must have an L in column 1, and DATA in positions 10 through 13. The segment data appears in positions 16 through 71. Data continuation is indicated with a non-blank in position 72. On the continuation statement, positions 1 through 15 are blank, and the data is resumed in position 16. The maximum length of a segment is set at 1500 bytes, but the user can change this by reassembling the program with the USERSEG field altered.

Note: On ISRT calls, the last SSA can have only the segment name, with no qualification or continuation.

When inserting or replacing variable length segments, as defined in a DBDGEN, or including variable length data for a CHKP or LOG call, position 4 of the CALL statement must contain either a V or M. V must be used if only one segment of variable length is being processed. Positions 5 through 8 must contain the length of the data, right-justified, with leading zeroes. This value is converted to binary, and becomes the first two bytes of segment data. Segment-data-statements can be continued, as described above, with the subsequent statements blank in positions 1 through 15, and the data starting in position 16.

If multiple variable-length segments are required (that is, concatenated logical child/logical parent segments both of which are variable-length) for the first segment, there must be a V in position 4 and the length of that segment in positions 5-8. If that segment is longer than 56 bytes, then the data is continued as above except that the last card to contain data for this segment must have a non-blank in position 72. The next statement applies to the next variable-length segment, and must contain an M in position 4 and the length of this segment in positions 5-8. Any number of variable-length segments can be concatenated in this manner, up to 1500 bytes of total length. The M or V and the length must appear only in statements that begin data for a variable-length segment.

When inserting or replacing via path calls, a P in position 4 causes the length field to be used as the length the segment will occupy in the user I/O area, without the length (LL) field of variable-length segments, as in the instructions for M, above. V, M, and P can be mixed in successive statements. The P appears in only the first statement of fixed-length segment DATA statements, in path calls which contain both variable- and fixed-length segments.

PARAMETER LENGTH, SNAP CALLS: On SNAP calls, the length of the SNAP parameters must be in positions 5-8. This number must be equal to the length of the SNAP parameters starting in position 16 plus an additional two bytes. The TEST program converts the length to binary and places it in the first half-word of the user I/O area passed to DL/I. The parameters from position 16 are placed in the I/O area immediately following this half-word. If positions 5-8 are blank, a default of 22 is used as the parameter length. For additional information on SNAP calls, see Sections 2 and 6 in Volumes 1 and 3, respectively, of the IMS/VS Program Logic Manual.

All parameters are passed without change, with the following exception: If the SNAP destination field specifies "DCB-addr" or ddname of PRINTDD, and if a PRINTDD statement is supplied to the test program, the test program replaces this parameter with the DCB address of the test program PRINTDD data set. If a PRINTDD DD statement is not supplied, the test program defaults to LOGØØØØ.

PARAMETER LENGTH, LOG CALL: The LOG call is normally used with the I/O PCB. It can be used in batch mode only if the CMPAT option of the PSBGEN statement (see the IMS/VS Utilities Reference Manual) is specified.

The LOG call can be specified in two ways:

1. A LOG call statement followed by a DATA statement with an L in column 1, a V in column 4, and the record length (in decimal) in columns 5-8, right-justified, and padded with zeroes. An example:

```
COL     COL     COL     COL
1       4       10      16

L               LOG
L       V0016   DATA    00ASEGMENT ONE
```

   When this method is used, the first halfword of the record is eliminated. However, the specified length must include the 2 bytes that are eliminated.

2. A LOG call statement followed by a DATA statement with an L in column 1 and the record length (in binary) as the first halfword of the record. The second halfword of the record is binary zeros. An example:

```
COL     COL     COL     COL
1       4       10      16

L               LOG
L               DATA    1000BSEGMENT TWO
```

When this method is used, columns 5-8 should be blank.

SEGMENT LENGTH AND CHECKING, ALL CALLS: Because this program does not know segment lengths, the length of the segment displayed on REPL or ISRT calls is the number of DATA statements that have been read, times 56. IMS/VS knows the segment length and uses the proper length.

This program does no checking for errors in the call; invalid functions, segments, fields, operators, or field lengths are not detected by this program.

COMPARE Statement Format for PCB Comparisons

This is the format of the COMPARE statement used for PCB comparisons.

Position		Contents
1	=	E identifies this as a COMPARE statement.
2	=	H indicates hold COMPARE statement (see below for details). Blank indicates a reset of the hold condition or a single COMPARE statement.
3	=	Option requested if results of the compare are unequal: Blank means "Use the default for the SNAP option." The normal default is 5. For an explanation of how to change the default, see the description of the "OPTION Statement Format." 1 request SNAP of the complete I/O buffer pool. 2 request SNAP of entire region. 4 request SNAP of DL/I blocks. 8 abort this step; go to end of job. S SNAP subpools 0-127. Note: Multiple functions of the first 4 options can be obtained by summing their respective hexadecimal values. For example, a value of 5 is a request for a print of the I/O buffers and the DL/I blocks; and a value of D snaps the I/O pool, snaps the DL/I blocks, and aborts the program run.

Position		Contents
4	=	Extended SNAP options, if results of a compare are unequal: Blank: this extended option is ignored; P the complete buffer pool is snapped; S subpools 0-127 are snapped. Note: In no case will an area be snapped twice; that is, a combination of 1P in positions 3 and 4 results in just one snap of the buffer pool. Similarly, a combination of SS results in just one snap of subpools 0-127.
5 - 6	=	Segment level.
7	=	Not used.
8 - 9	=	Status code, or one of the following: XX - do not check status code. OK - allow blank, GA, or GK.
10	=	Not used.
11 - 18	=	Segment name.
19	=	Not used.
20 - 22	=	Length of feedback key.
23	=	Not used.
24 - XX	=	Concatenated key feedback.
72	=	Nonblank to continue key feedback.

The COMPARE statement is optional. It can be used to do regression testing of known data bases, or to call for a print of blocks or buffer pool(s).

Any fields left blank are not compared to the corresponding field in the PCB. Since a blank is a valid status code, to not compare status codes, put XX in positions 8 and 9. To accept any valid status code, (that is, blank, GA, or GK), use OK in position 8 and 9.

To execute the same COMPARE after each call, put an H in position 2. This is useful when loading a data base to compare to a blank status code only. Since the compare was done, the current control statement type is F in position 1; the next control statement read must therefore have its type in position 1 or it will default to E. The HOLD-COMPARE statement stays in effect until another COMPARE statement is read. If a new COMPARE statement is read, two compares will be done for the preceding call, since the HOLD-COMPARE and optional printing are done prior to reading the new COMPARE statement.

The total number of unequal compares will be reflected in the condition code returned for that step.

COMPARE Statement Format for User I/O Area Comparisons

This is the format of the COMPARE statement used for user I/O area comparisons.

	Position		Contents
$	1	=	E identifies this as a COMPARE statement.
	3	=	Blank, the LL field of the segment is not included in the comparison, only data is compared.
		=	L, the length in positions 5-8 is converted to binary and compared against the LL field of the segment.
	4	=	V, if variable-length segment only, or if the first variable-length segment of multiple variable-length segments in a path call or concatenated logical child/logical parent segment. P, if fixed-length segment in a path call. M, if the second or subsequent variable-length of a path call, or concatenated logical child/logical parent segment.
		=	Blank, not variable-length or non-path call data compare.
	5 - 8	=	nnnn, length of a variable-length segment, right-justified with leading zeroes. If position 4 contains V, P, or M, then a value must appear in positions 5-8. If position 3 contains an L then this value is compared against the LL field of the returned segment. If position 3 is blank and the segment is not in a path call, then this value is used as the length of the comparison. The rules for continuations are the same as those described for the variable-length segment DATA statement in the description of the CALL statement. If this is a path call comparison, and position 4 contains P, then the value in positions 5-8 must be the exact length of the fixed segment used in the path call.
	10 - 13	=	DATA, this has to be specified in the first COMPARE DATA statement only.
	16 - 71	=	Data against which the segment is to be compared.
	72	=	Blank identifies the last COMPARE DATA statement for the current call, and causes the comparison to be made.
		=	Non-blank, if the comparison data exceeds 56 characters, data is continued in positions 16-71 of the subsequent statements for a maximum total of 1500 bytes.

This COMPARE statement is optional. Its purpose is to COMPARE the segment returned by IMS/VS to the data in this statement to verify that the correct segment was retrieved.

The length in positions 5-8 is optional except as already noted; if present, this length is used in the COMPARE and in the display. If no length is specified, the shorter of either the length of data moved to the I/O area by IMS/VS, or the number of DATA statements read times 56 is used for the length of the comparison and display.

If both a COMPARE DATA and a COMPARE PCB statement are present, the COMPARE DATA statement must precede the COMPARE PCB statement.

The conditions for printing the COMPARE DATA statement are the same as for printing a COMPARE PCB statement; position 7 of the STATUS statement is used. The same unequal switch is set for either the COMPARE DATA or COMPARE PCB. However, if control block displays are requested for unequal comparisons, a COMPARE PCB statement is required to request these options.

The total number of unequal comparisons will be reflected in the condition code returned for that step.

OPTION Statement Format

The purpose of the OPTION statement is to set the default SNAP option and/or the number of unequal comparisons before aborting the step. The default value for the number of unequal comparisons before aborting is 5.

The format of the statement is explained below.

Position		Contents
1	=	O identifies this as an OPTION statement.
2 - 80	=	Free-form coding. The first operand is SNAP=x, where "x" is the default SNAP option to be used. The second operand is ABORT=xxxx, where "xxxx" is a 4-digit numeric value that sets the number of unequal comparisons before aborting the step.

Use of the following example of the OPTION statement will cause the DL/I test program to operate as it did prior to the release of IMS/VS Version 1, Modification Level 1:

Col. 1

```
0␣SNAP=␣,ABORT=9999
```

PUNCH Statement

The PUNCH control statement provides the facility for this program to produce an output data set consisting of the PCB COMPARE statements, the user I/O area COMPARE statements, all other control statements read, or any combination of the above. An example of the use of this facility is to code the call, but not the COMPARE statements for a new test. Then, after verifying that the calls were executed as anticipated, another run is made where the PUNCH statement is used to cause the test program to merge the proper COMPARE statements, based on the results of the call, with the CALL statements read, producing a new output data set. This is then used as input for subsequent regression tests. If segments in an existing data base are changed, the use of this control statement causes a new test data set to be produced with the proper COMPARE statements. This eliminates the need to manually change the COMPARE statements because of a change in the segments of the test data base.

The PCB COMPARE statements are produced based on the information in the PCB after the call is completed. The COMPARE DATA statements are produced based on the data in the I/O area after the call is completed. All input control statements, other than COMPARE statements, can be produced to provide a new composite test with the new COMPARE statements properly merged. The data set produced can be sequenced.

Since the key feedback area of the PCB COMPARE statement can be long, two options are provided for producing these COMPARE statements. Either the complete key feedback can be provided, or the portion of the key feedback that does not fit on one statement can be dropped. Forty-eight bytes of key feedback fit on the first statement.

Getting the full data from the I/O area into the data COMPARE statement might also be excessive. An option is to put it all on the data COMPARE statements, or put only the first 56 bytes on the first statement and drop the rest. The test program only compares the first 56 bytes if it only receives one COMPARE DATA statement.

PUNCH STATEMENT FORMAT:

Position		Contents
$ 1 - 3	=	CTL identifies this statement type.
$ 10 - 13	=	PUNC further identifies this statement type as controlling the punch output data set, and tells the program to start punching.
		NPUN stop punching.
$ 16	=	Starts keyword parameters controlling the various options. These keywords are:
		PCBL, produce the full PCB COMPARE statement.
		PCBS, produce the PCB COMPARE, dropping the key feedback if it exceeds one statement.
		DATAL, produce the complete COMPARE DATA statements.
		DATAS, produce only one statement of compare data.

Position	Contents
	OTHER, reproduce all control statements except COMPARE control statements.
	START, starting sequence number to be punched in 73 through 80. Eight numeric characters must follow the START= parameter; leading and/or trailing zeroes are required.
	INCR, increment to be added to the sequence number of each statement. Four numeric characters must follow the INCR= parameter; leading and/or trailing zeroes are required.

Some examples of the PUNCH control statement are:

```
1         10

CTL       PUNC PCBL,DATAL,OTHER,START=00000010,INCR=0010
CTL       NPUN
```

PUNCH DD Statement

The DD statement for the output data set is labelled PUNCH; the data set characteristics are fixed, unblocked, with a logical record length of 80.

An example of the PUNCHDD statement is:

```
//PUNCHDD      DD  SYSOUT=B
```

SYSIN2 DD Statement

The data set specified by the SYSIN DD statement is the normal input data set for this program. It is sometimes desirable when processing an input data set that is on direct access or tape, to override or insert some control statements into this input stream. This is especially useful to obtain a SNAP after a particular call.

To provide this capability, a second input data set (SYSIN2) will be read if the DD statement is present in the JCL for the step. The records from the SYSIN2 data set are merged with records from the SYSIN data set, and the merged records become the input for this program.

The merging is done based on the sequence numbers in positions 73 through 80, and is a two-step process: first, positions 73 and 74 of SYSIN2 must be equal to the corresponding positions of SYSIN; then the merge is done based on positions 75 to 80.

This peculiarity of merging allows for multiple data sets (each with a different high-order sequence number in 73-74) that have been concatenated to form SYSIN, in other than positions 73-74 numeric sequence. The two-step merge logic permits SYSIN2 input to be merged appropriately into each of the concatenated data sets.

When the sequence numbers are equal, SYSIN2 overrides SYSIN.

Any statements or records in this data set must contain sequence numbers in columns 73-80. They will replace the same sequence number in the SYSIN data set, or be inserted in proper sequence if the number in SYSIN2 does not exist in SYSIN. Replacement or merging is done only for the run being made. The orginal SYSIN data is not changed.

Other Control Statement Formats

Position		Contents
1 - 4	=	DLCK - issues OS/VS checkpoint, followed by a DL/I checkpoint.
10 - 17	=	Contains a 1- to 8-character checkpoint ID (left-justified).
1 - 4	=	WTOR - puts message in remainder of statement on system console and waits for any reply, then continues.
1 - 3	=	WTO same as WTOR, but does not wait for reply.
1	=	. or N; disregard this statement.
1 - 5	=	ABEND - issues user ABEND 252 with the DUMP option.

Special CALL Statement Format

	Position		Contents
$	1	=	L identifies this as a CALL statement.
	5 - 8	=	Number of times to repeat a series of calls with a range from 0001 thru 9999 (default is 1).
$	10 - 13	=	STAK - Start stacking control cards for later execution.

END - Stop stacking control cards and begin execution.

The STAK function enables the user to repeat a series of calls which have been read from SYSIN and held in storage. All control statements between the STAK card and the END card are read and saved. When the END card is encountered, the series of calls is executed as many times as the number punched in positions 5 through 8 of the STAK card. This can be used to test exclusive control and scheduling by having two different regions executing stacks of calls concurrently.

STAT - Print the current buffer pool statistics.

Cols. 16-20 One of the following values is used to obtain the type and form of statistics required:

- VBASF provides the full VSAM data base subpool statistics in a formatted form.
- VBASU provides the full VSAM data base subpool statistics in an unformatted form.

Position	Contents
	VBASS provides a summary of the VSAM data base subpool statistics in a formatted form.
	DBASF provides the full ISAM/OSAM data base buffer pool statistics in a formatted form.
	DBASU provides the full ISAM/OSAM data base buffer pool statistics in an unformatted form.
	DBASS provides a summary of the ISAM/OSAM data base buffer pool statistics in a formatted form.

For more information on the STAT call, see the "System Service Calls" section in the "Data Base Batch Programming" chapter of this manual.

SNAP - Issue the DL/I Call. See sections 2 and 6, Volumes 1 and 3, respectively, of the IMS/VS Program Logic Manual.

DLCK - For any dependent region, DLCK gives an OS/VS checkpoint to a DD statement labelled CHKDD whose DSORG=PO. This is followed by a DL/I checkpoint call.

CHKP - Same as DLCK.

SKIP - Skip SYSIN statements until START statement encountered.

START- Start making DL/I calls again.

FORMAT OF DISPLAY OF DL/I BLOCKS

The IMS/VS SNAP call is used to display the DL/I blocks. For additional information on the SNAP call, see the "Process SNAP Call" diagram and the "SNAP Call Facility" discussion in Sections 2 and 6, Volumes 1 and 3, respectively, of the IMS/VS Program Logic Manual.

EXECUTION IN DIFFERENT TYPES OF REGIONS

This program is designed to operate in a DL/I or BMP region but can also be executed in a MSG region. The input and output devices are dynamically established based on the type of region in which the program is executing. In a BMP or DL/I region, the EXEC statement allows the program name to be different from the PSB name. There is no problem executing calls against any data base in a BMP or DL/I region. In a MSG region, the program name must be the same as the PSB name. In order to execute in a MSG region, the DFSDDLT0 program must be given the name or an alias of the PSB named in the IMS/VS definition.

When in a DL/I region, input is read from SYSIN and output is written to PRINTDD.

When in a BMP region, if a symbolic input terminal was specified as the fourth parameter of the EXEC statement, input is obtained from that SMB, and output is sent to the I/O PCB. The name of the I/O PCB can be specified as the fifth parameter of the EXEC statement. If SMB is not specified on the EXEC statement, SYSIN is used for input and PRINTDD is used for output, as in the DL/I region.

In the MSG region, the I/O PCB is used for both input and output unless position 2 of the STATUS statement is either a 1 or an A. In either of these cases, PRINTDD is used for output if the DD card is present in the JCL for that message region. A limit of 50 lines per schedule is sent to the I/O PCB and, after that, PRINTDD is used for output if present. If PRINTDD is not present, the program terminates.

Because the input is in fixed form, it is difficult to key it from a terminal. For ease of entry, however, Message Format Service (MFS) facilities can be used from a terminal to create the fixed-format input. One way to test DL/I in a message region, using this program, is to first execute another message program which, based on a message from the terminal, reads control statements stored as a member of a partitioned data set. Insert these control statements into an SMB. This program is then scheduled by IMS/VS to process those transactions. This allows the same control statements to be used to execute in any region type.

HINTS ON USAGE

1. To load a data base:

   This program is applicable for loading small data bases, because all calls and data must be provided to it rather than it generating data. It can be used to load large volume data bases if the control statements were generated as a sequential data set.

2. To display a data base:

   To display a data base, the following sequence of control statements can be used.

```
S 1 2 2 2 1          DBDNAME          Display comments and segment
L       GN                            DO 1 Get Next
EH8   OK                              Hold compare, GA, GK, OK, terminate
                                      on GB
L   9999 GN                          DO 9,999 Get Next calls
```

3. To do regression testing:

   This program can be used for regression testing. By using a known data base, calls can be issued and the results compared to expected results using COMPARE statements. The program then can determine if DL/I calls are being executed correctly. By making the print options of the STATUS statement all twos, only those calls not satisfied properly are displayed.

4. To use as a debugging aid:

   When doing debugging work, usually a print of the DL/I blocks is required. By use of COMPARE statements, the blocks can be displayed at appropriate times. Sometimes the blocks are needed even though the call is executed correctly, such as the call before the failing call. In those cases, a SNAP call can be inserted. This causes the blocks to be displayed even though the call was executed correctly.

5. To verify how a call is executed:

   Because it is easy to execute a particular call, this program can be used to verify how a particular call is handled. This is of value when DL/I is suspected of not operating correctly in a specific situation. The calls that are suspected can be issued using this program, and the results examined.

SAMPLE JCL

```
//JCLSAMP JOB ACCOUNTING,NAME,MSGLEVEL=(1,1),MSGCLASS=3,PRTY=8
//GET EXEC PGM=DFSRRC00,PARM='DLI,DFSDDLT0,PSBNAME'
//STEPLIB  DD   DSN=IMSVS.RESLIB,DISP=SHR
//IMS DD DSN=IMSVS.PSBLIB,DISP=(SHR,PASS)
//    DD DSN=IMSVS.DBDLIB,DISP=(SHR,PASS)
//DDCARD DD DSN=DATASET,DISP=(OLD,KEEP)
//IEFRDER DD DUMMY
//PRINTDD DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSIN DD *
S 1 1 1 1      DBDNAME
/*
```

SAMPLE CONTROL STATEMENT INPUT

Data Base Load

```
//SYSIN DD *
U START TEST LOAD
T ISRT ROOT SEGMENT A060000111
L          ISRT    A1111111
L          DATA    A0600011    1069999888  ROOT SEG1
EH
T ISRT ROOT SEGMENT A06000511
L          ISRT    A111111
L          DATA    A060000511  1069999488  ROOT SEG2
L          ISRT    A111111     (A1111111 = A060000511) X
                   AA222222
           DATA    XAA040511Z
/*
```

Data Base Retrieve and Update

```
//SYSIN DD *
S 1 1 1 1 1                1
L          GHU     JHNXXX      (J11NXXX = A10H102000)  *
                   JM2PABCX    (JM2PABCX = DI0HI02A10)
S 1 1 1 1 1                2
L          ISRT    J11NXXXX    (J11NXXXX = A10H02000)  C
                   JK2PADXX
L          DATA    A10HD02000D10HD02A1U
S 1 1 1 1 1                1
L          REPL
L          DATA    A10HD0200DB10HD02A10
/*
```

MESSAGE PROCESSING REGION SIMULATION

Message processing region simulation is not supplied as a part of the IBM IMS/VS program.

The checkout of any message processing program in the online terminal environment is often impractical. To enable a more practical and efficient checkout environment, a message processing region simulation can be used. The object of the simulator is to enable checkout of a message processing program, in a batch processing region, with a set of test data bases. Messages are read and written with unit record, tape, or disk data sets as opposed to input and output message queues. To be effective, the simulator should incur no, or minimal, change to the message processing program when it is moved from the simulated to the actual message processing region environment.

The user can accomplish simulation by appending the Simulator Interface A and Simulator Interface B modules to the message processing program in addition to the language interface. (See Fiugre 7-1.)

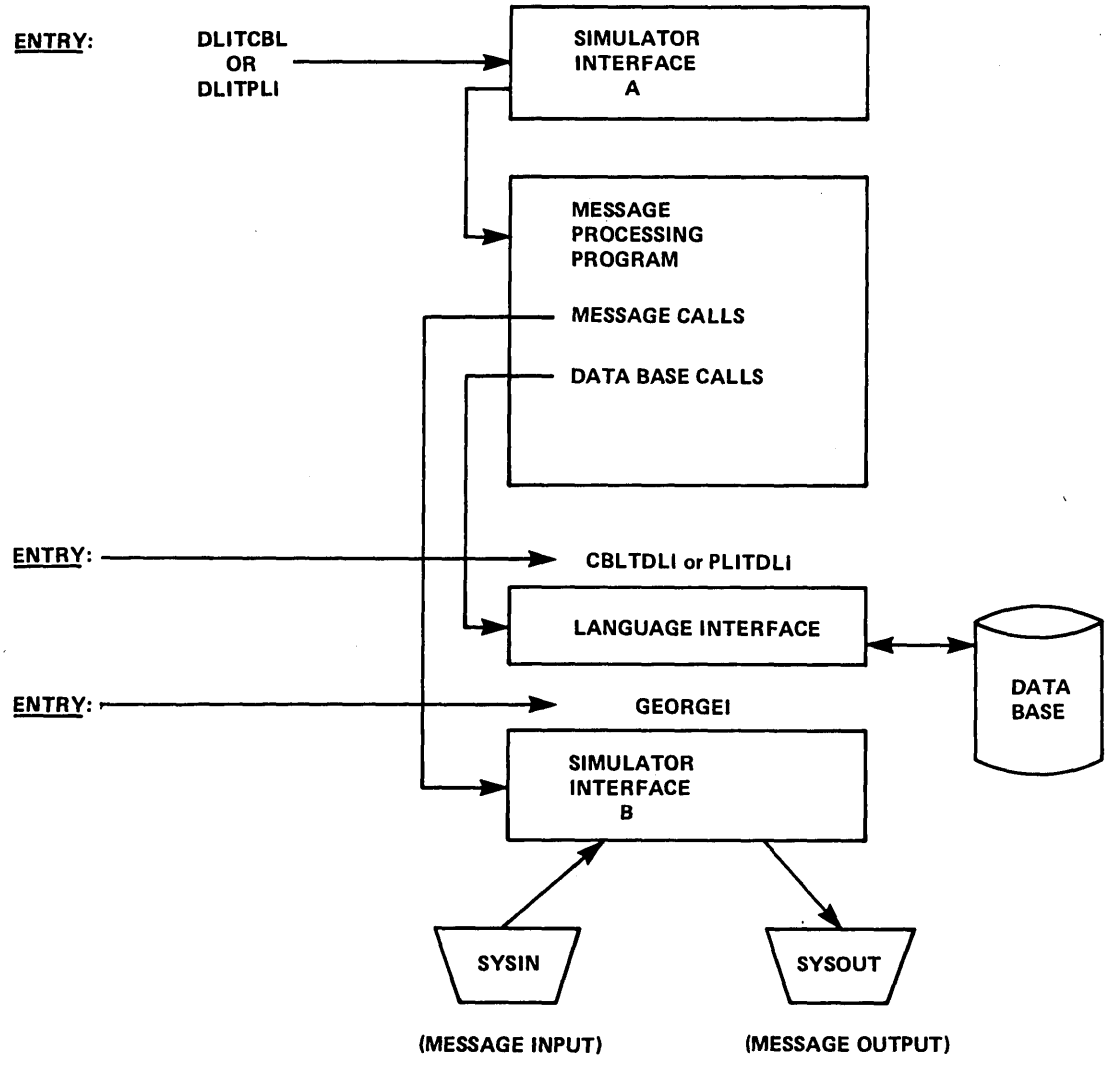


Figure 7-1. Message Processing Region Simulation

When the PSB is generated for the associated message program, the PCBs within the PSB are normally for Data Language/I data bases only. No PCB for an input and output terminal is provided. When the message program is loaded into a batch processing region, the PCB addresses are passed to the message program. No terminal PCB is provided.

When Simulator Interface A is link-edited with the message program with entry point DLITCBL or DLITPLI, the Simulator Interface A is entered. The interface prefixes the PCB address list with an input/output terminal PCB address. The PCB exists within Simulator Interface A, and its address is added as the first PCB address in the PCB address list passed to the message program. This PCB address is used by the message program as are the other PCB addresses in the list, except that this PCB address is used in calls from the message program to Simulator Interface B.

When a call is made from the message program to Simulator Interface B, the message program makes a Data Language/I call, with the terminal PCB address provided by Simulator Interface A. Simulator Interface B then utilizes OS/VS SYSIN and SYSOUT data sets as if messages were being read from and written to message queues. You may include alternate terminal PSBs within your PSB generation. The addresses for these PCBs are provided, upon entry to the user message program, in the order specified by PCB statements in PSB generation. If a Data Language/I call (CALL CBLTDLI) is issued with an alternate terminal PCB address in an IMS/VS batch region, an AL status code is returned in the PCB.

Data Language/I data base calls are executed with the appropriate PCBs to the link-edited language interface.

The following changes must be made when the message processing program is moved to a message processing region:

- Both Simulator Interface modules should be omitted.
- The entry point name of the message program must be renamed DLITCBL (COBOL or Assembler) or DLITPLI (PL/I).
- The CALL statement operand must be renamed from GEORGEI to the language interface entry point CBLTDLI or PLITDLI.

EXAMPLES

The following example shows a typical COBOL program that might be written to test a message program in a batch processing region. (Refer to Figure 7-1 in conjunction with this example.)

Simulator Interface A

```
IDENTIFICATION DIVISION.
PROGRAM-ID. 'CAB',
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01   INOUT-PCB
       02  IO-TERMINAL      PICTURE X(8).
       02  IO-RESERVE       PICTURE XX.
       02  IO-STATUS        PICTURE XX.
       02  IO-PREFIX        PICTURE X(12).
LINKAGE SECTION.
01   DB-PCB.
       02  DATA-BAS-DESC    PICTURE X(71).
PROCEDURE DIVISION.
           ENTRY 'DLITCBL' USING DB-PCB.
           CALL 'TEST' USING INOUT-PCB, DB-PCB.
           STOP RUN.
```

Message Processing Program

The following is an example of a section of the message processing program being tested. It shows the entry point and call to the Message Input and Output (Message Simulator Interface B). (Refer to Fiugre 7-1 in conjunction with this example.)

```
START-OUT.
     ENTRY 'TEST' USING TERMINAL INOUT-PCB,DB-PCB.
     CALL 'GEORGEI' USING GET-UNIQUE,INOUT-PCB,LINE-INPUT.
```

Simulator Interface B

The following example of message output should be reviewed in conjunction with the previous example and with Figure 7-1.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. 'IMSTEST'.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT MESSAGE-FILE ASSIGN TO 'TESTIN' UTILITY.
     SELECT TEST-OUTPUT-FILE ASSIGN TO 'TESTOUT' UTILITY.
DATA DIVISION.
FILE SECTION.
FD   MESSAGE-FILE
     RECORDING MODE IS V
     DATA RECORD IS INPUT-MESSAGE.
     01   INPUT-MESSAGE                          PICTURE IS X(143).
FD   TEST-OUTPUT-FILE
     BLOCK CONTAINS 10 RECORDS
     DATA RECORD IS PRINT-LINE.
     01   PRINT-LINE                             PICTURE IS X(133).
WORKING-STORAGE SECTION.
     77   OPEN-SWITCH PICTURE X                  VALUE ' '.
     77   END-SWITCH  PICTURE X                  VALUE ' '.
     77   MESSAGE-SIZE-WORK  PICTURE S9(4)       VALUE 0
                  USAGE COMPUTATIONS.
     77   BAD-FUNCTION-CODE PICTURE XX           VALUE 'QA'.
     77   NO-DATA-CODE       PICTURE XX          VALUE 'QC'.
     77   REC-SWT    PICTURE X VALUE ' '.
     77   MESS-OUT   PICTURE X VALUE ' '.
     77   C-329      PICTURE S9(6) VALUE 329
                  USAGE COMPUTATIONAL.
     01   MESSAGE-IN-WORK-AREA.
          02   HEADER-DATA-IN.
               03   MESSAGE-COUNT                 PICTURE 9(4).
               03   MESSAGE-TYPE                  PICTURE X.
               03   TERMINAL-NAME                 PICTURE X(8).
          02   MESSAGE-TEXT.
               03   FILLER PICTURE X OCCURS 130 TIMES
                    DEPENDING ON MESSAGE-SIZE-WORK.
     01   TEST-OUTPUT-HEADER.
          02   FILLER PICTURE X(18) VALUE
                 '  MESSAGE TYPE = '.
          02   FILLER.
               03   IN-OR-OUT-MESSAGE             PICTURE X.
               03   HEAD-OR-BODY                  PICTURE X.
          02   FILLER  PICTURE X(18)              VALUE
                 ',  MESSAGE COUNT = '.
          02   OUTPUT-COUNT                       PICTURE 9999.
          02   FILLER  PICTURE X(13)              VALUE
                 ',  TERMINAL = '.
          02   OUTPUT-TERMINAL                    PICTURE X(8).
```

```
        02  FILLER  PICTURE  XX  VALUE SPACES.
        02  OUT-RUN  PICTURE  XXXX.
    01  TEST-OUTPUT-TEXT.
        02  TEST-OUTPUT-CHAR  OCCURS 130 TIMES
                                          PICTURE X.
LINKAGE SECTION.
    01  INOUT-PCB.
        02  IO-TERMINAL      PICTURE X(8).
        02  IO-RESERVE       PICTURE XX.
        02  IO-STATUS        PICTURE XX.
        02  I-PREFIX  PICTURE X(12).
    01  FUNCTION  PICTURE XXXX.
    01  IO-AREAS-RECORD.
        02  RCC  PICTURE S9(4) USAGE COMPUTATIONAL.
        02  RCC-ZEROS  PICTURE  XX.
        02  TEXT.
            03  FILLER PICTURE X OCCURS 130 TIMES.
PROCEDURE DIVISION.
    ENTRY 'GEORGEI' USING FUNCTION, INOUT-PCB, IO-AREAS-RECORD.
OPEN-FILES.
    IF OPEN-SWITCH = '1' GO TO PROCESS-X.
        MOVE 0 TO TALLY.
    OPEN INPUT MESSAGE-FILE
        OUTPUT TEST-OUTPUT-FILE.
    MOVE '1' TO OPEN-SWITCH.
PROCESS-X.
    IF FUNCTION = 'GU  ' GO TO GET-HEADER.
    IF FUNCTION = 'GN  ' GO TO GET-BODY.
    IF FUNCTION = 'ISRT' GO TO WRITE-REPLY.
    MOVE BAD-FUNCTION-CODE TO IO-STATUS.
RETURN-TO-APPLICATION.
    RETURN.
FORMAT-INPUT-MESSAGE.
    MOVE 'I' TO IN-OR-OUT-MESSAGE.
    MOVE MESSAGE-TYPE TO HEAD-OR-BODY.
    MOVE MESSAGE-COUNT TO OUTPUT-COUNT.
    MOVE TERMINAL-NAME TO OUTPUT-TERMINAL.
    MOVE MESSAGE-TEXT TO TEST-OUTPUT-TEXT.
SET-UP-FOR-USER.
    MOVE MESSAGE-COUNT TO RCC.
    MOVE LOW-VALUES TO RCC-ZEROS.
    MOVE TERMINAL-NAME TO IO-TERMINAL.
    MOVE MESSAGE-TEXT TO     TEXT.
    MOVE '  ' TO IO-STATUS.
READ-MESSAGE-FILE.
    IF END-SWITCH = '1' GO TO FINISH-UP.
    READ MESSAGE-FILE INTO MESSAGE-IN-WORK-AREA
                  AT END MOVE '1' TO END-SWITCH
```

```
                    GO TO READ-MESSAGE-FILE.
    COMPUTE  MESSAGE-SIZE-WORK = MESSAGE-COUNT - 4.
    PERFORM  FORMAT-INPUT-MESSAGE.
    PERFORM  WRITE-TEST-OUTPUT-FILE.
WRITE-TEST-OUTPUT-FILE.
        MOVE  FUNCTION  TO  OUT-RUN.
    WRITE PRINT-LINE  FROM  TEST-OUTPUT-HEADER.
    WRITE PRINT-LINE  FROM  TEST-OUTPUT-TEXT.
GET-HEADER.
    IF  REC-SWT NOT = 'H'
            PERFORM  READ-MESSAGE-FILE
            GO TO REC-GOT.
    COMPUTE  MESSAGE-SIZE-WORK = MESSAGE-COUNT - 4.
    PERFORM  FORMAT-INPUT-MESSAGE.
    PERFORM  WRITE-TEST-OUTPUT-FILE.
REC-GOT.
    IF MESSAGE-TYPE NOT = TO 'H' GO TO GET-HEADER.
    PERFORM  SET-UP-FOR-USER. MOVE ' ' TO REC-SWT.
    GO  TO RETURN-TO-APPLICATION.
GET-BODY.
    PERFORM  READ-MESSAGE-FILE.
     IF MESSAGE-TYPE = 'B' NEXT SENTENCE ELSE
            MOVE 'H' TO  REC-SWT
            MOVE   '0D'         TO IO-STATUS
            GO TO RETURN-TO-APPLICATION.
    PERFORM  SET-UP-FOR-USER.
    GO TO RETURN-TO-APPLICATION.
WRITE-REPLY.
    MOVE IO-TERMINAL TO OUTPUT-TERMINAL.
        COMPUTE  MESSAGE-SIZE-WORK = RCC - 4.
    MOVE  RCC TO OUTPUT-COUNT.
    MOVE  'O' TO IN-OR-OUT-MESSAGE.
    MOVE  ' ' TO HEAD-OR-BODY.
    MOVE  TEXT TO   TEST-OUTPUT-TEXT.
    MOVE  MESS-OUT  TO  IO-STATUS.
    PERFORM  WRITE-TEST-OUTPUT-FILE.
FINISH-UP.
    IF FUNCTION = 'GU  ' MOVE '0C' TO IO-STATUS.
    IF FUNCTION = 'GN  ' MOVE '0D' TO IO-STATUS.
    GO TO RETURN-TO-APPLICATION.
```

APPENDIX A. DL/I STATUS CODES QUICK-REFERENCE TABLE

At the completion of a DL/I call, a status code that indicates the results of the call is returned to the application program in the PCB status code field. The user should follow each call in his program with statements which examine the returned status codes to determine if the requested action was completed properly.

Status codes fall into four different categories:

1. Exceptional but valid conditions encountered for the call (for example, GE, GB)

2. Warning or indicative status codes on successful calls (for example, GA, GK, II, QC, and QD)

3. Improper user specifications (the principal category)

4. Error conditions encountered during the actual execution of I/O requests

An IMS/VS installation should normally provide application programs with a standardized status code checking procedure to be applied after each call.

- Status codes from categories 1 and 2 can be handled by each application program according to its specific needs.

- Status codes from category 3 result from programming errors, and should be handled in a generalized way which supplies the application programmer with the information required to correct the error.

- Status codes from category 4 must be handled by procedures set up by the data base administrator; they should not be handled by each individual application programmer. Category 4 status codes often require recovery procedures which could affect other application programs and the integrity of the entire data base environment.

Figure A-1 provides a quick reference of DL/I status codes. These status codes are described in detail in Appendix B.

Figure A-1 (Part 1 of 2). DL/I Status Codes Quick Reference

	DATA BASE CALLS							MSG CALLS					SYSTEM SERVICE CALLS								
STATUS CODE	GU GHU	GN GHN	GNP GHNP	DLET REPL	ISRT (LOAD)	ISRT (ADD)	GU	GN	ISRT	CHNG	PURG	CHKP	ROLL	DEQ	LOG	SNAP	CALL COMPLETED	ERROR IN CALL	I/O OR SYST ERROR	CATE-GORY	DESCRIPTION
AA										X								X		3	CHNG CALL FOR RESPONSE ALTERNATE PCB CAN ONLY SPECIFY LOGICAL TERMINAL DESTINATION; TRANSACTION CODE DESTINATION SPECIFIED.
AB	X	X	X	X	X	X	X	X	X	X		X		X	X			X		3	SEGMENT I/O AREA REQUIRED, NONE SPECIFIED IN CALL
AC	X	X	X		X	X												X		3	HIERARCHICAL ERROR IN SSAs
AD																		X		3	INVALID FUNCTION PARAMETER
AH					X	X												X		3	CALL REQUIRES SSAs, NONE PROVIDED
AI	X	X	X	X	X	X													X	4	DATA MANAGEMENT OPEN ERROR
AJ	X	X	X		X	X												X		3	INVALID SSA QUALIFICATION FORMAT
AK	X	X	X		X	X												X		3	INVALID FIELD NAME IN CALL
AL	X	X	X	X	X	X												X		3	CALL USING LT PCB IN BATCH PGM
AM	X	X	X	X	X	X												X		3	CALL FUNCTION NOT COMPATIBLE W/PROCESSING OPTION OR SGMT SENSITIVITY
AO	X	X	X	X	X	X													X	4	I/O ERROR ISAM, OSAM, BSAM, OR VSAM
AQ							X	X				X							X	4	READ I/O ERROR, MESSAGE CHAIN CANNOT BE FOLLOWED, MINIMUM OF ONE MESSAGE LOST
AR							X	X				X							X	4	READ I/O ERROR, MESSAGE SEGMENT HAS BEEN LOST, MESSAGE CHAIN IS STILL INTACT
AT				X	X	X			X		X				X			X	X	3	USER I/O AREA TOO LONG
AU	X	X	X	X	X	X												X		3	SSAs TOO LONG
AY									X									X		3	RESPONSE ALTERNATE PCB REFERENCED BY ISRT CALL HAS MORE THAN ONE PHYSICAL TERMINAL ASSIGNED FOR INPUT PURPOSES. NOTIFY MASTER TERMINAL.
AZ											X							X		3	CONVERSATIONAL PROGRAMS WILL ISSUE PURG CALLS TO WRONG PCB
A1										X								X		3	CALL ATTEMPTED WITH 8-CHAR LOGICAL TERMINAL NAME NOT KNOWN TO SYSTEM
A2										X								X		3	CHANGE ATTEMPTED WITH INVALID PCB
A3									X		X							X		3	INSERT/PURGE ATTEMPTED TO A MOD TP PCB WITH NO DESTINATION SET
A4										X								X		3	SECURITY VIOLATION
A5									X		X							X		3	FORMAT NAME SPECIFIED ON 2ND OR SUBSEQUENT MSG ISRT OR PURG
A6									X									X		3	OUTPUT SEGMENT SIZE LIMIT EXCEEDED ON ISRT CALL.
A7									X									X		3	NUMBER OF OUTPUT SEGMENTS INSERTED EXCEEDED THE LIMIT BY ONE.
A8									X									X		3	ISRT TO RESPONSE ALTERNATE PCB FOLLOWED ISRT TO I/O PCB, OR VICE VERSA.
A9									X									X		3	RESPONSE ALTERNATE PCB REFERENCED BY ISRT CALL REQUIRES THAT SOURCE PHYSICAL TERMINAL RECEIVE THE OUTPUT RESPONSE.
DA				X														X		3	SEGMENT KEY FIELD HAS BEEN CHANGED
DJ				X														X		3	NO PRECEDING SUCCESSFUL GET HOLD CALL
DX				X														X		3	VIOLATED DELETE RULE
GA		X	X														X			2	CROSSED HIERARCHICAL BOUNDARY INTO HIGHER LEVEL (RETURNED ON UNQUALIFIED CALLS ONLY)
GB		X																		1	END OF DATA SET, LAST SEGMENT REACHED
GE	X	X	X			X														1	SEGMENT NOT FOUND
GK		X	X														X			2	DIFFERENT SEGMENT TYPE AT SAME LEVEL RETURNED (RETURNED ON UNQUALIFIED CALLS ONLY)
GL															X			X		3	INVALID USER LOG CODE.
GP			X															X		3	A GNP CALL AND NO PARENT ESTABLISHED OR REQUESTED SEGMENT LEVEL NOT LOWER THAN PARENT LEVEL
II						X														1	SEGMENT TO INSERT ALREADY EXISTS IN DATA BASE

STATUS CODE	DATA BASE CALLS: GU GHU	GN GHN	GNP GHNP	DLET REPL	ISRT (LOAD)	ISRT (ADD)	MSG CALLS: GU	GN	ISRT	CHNG	PURG	SYSTEM SERVICE CALLS: CHKP	ROLL	DEQ	LOG	SNAP	CALL COMPLETED	ERROR IN CALL	I/O OR SYST ERROR	CATE-GORY	DESCRIPTION
IX						X												X		3	VIOLATED INSERT RULE
LB					X															1	SEGMENT TO INSERT ALREADY EXISTS IN DATA BASE
LC					X															3	KEY FIELD OF SEGMENTS OUT OF SEQUENCE
LD					X															3	NO PARENT FOR THIS SEGMENT HAS BEEN LOADED
LE					X															3	SEQUENCE OF SIBLING SEGMENTS NOT THE SAME AS DBD SEQUENCE
NE				X															X	4	DL/I CALL ISSUED BY INDEX MAINTENANCE CANNOT FIND SEGMENT
NI				X	X	X													X	4	INDEX MAINTENANCE UNABLE TO OPEN AN INDEX DB, OR FOUND DUPLICATE SEGMENT IN INDEX
NO				X	X	X													X	4	I/O ERROR ISAM, OSAM, BSAM, OR VSAM
QC							X			X										1	NO MORE INPUT MESSAGES
QD								X												1	NO MORE SEGMENTS FOR THIS MESSAGE
QE								X										X		3	GET NEXT REQUEST BEFORE GET UNIQUE
QF									X	X	X							X		3	SEGMENT LESS THAN FIVE CHARACTERS (SEG LENGTH IS MSG TEXT LENGTH PLUS FOUR CONTROL CHARACTERS)
QH									X		X							X		3	TERMINAL SYMBOLIC ERROR – OUTPUT DESIGNATION UNKNOWN TO IMS/VS (LOGICAL TERMINALS OR TRAN CODE)
RX				X														X		3	VIOLATED REPLACE RULE
UC																				1	CHECKPOINT* TAKEN
UR																				1	RESTART*
US																				1	STOP*
UX																				1	CHECKPOINT AND STOP*
V1				X	X	X												X		3	INVALID LENGTH FOR VARIABLE LENGTH SEGMENT
X1									X		X								X	4	I/O ERROR WRITING SPA
X2									X		X							X		3	1ST INSERT TO TRAN CODE PCB THAT IS CONVERSATIONAL, IS NOT AN SPA
X3									X		X							X		3	INVALID SPA
X4									X		X							X		3	INSERT TO A TRAN CODE PCB THAT IS NOT CONVERSATIONAL AND THE SEGMENT IS AN SPA
X5									X		X							X		3	INSERT OF MULTIPLE SPAs TO TRAN CODE PCB
X6									X		X							X		3	INVALID TRAN CODE NAME INSERTED INTO SPA
X7									X		X							X		3	LENGTH OF SPA IS INCORRECT (USER MODIFIED FIRST SIX BYTES)
X8									X		X								X	4	ERROR ATTEMPTING TO QUEUE AN SPA ON A TRAN CODE PCB
X9									X		X							X		3	INCOMPATIBLE CONVERSATIONAL PROGRAM CALL PATH
XA									X		X							X		3	ATTEMPT TO CONT. PROC. CONV. BY PASSING SPA VIA PGM–TO–PGM SW. AFTER ANSWERING TERMINAL
XB									X		X							X		3	PGM PASSED SPA TO OTHER PGM BUT TRYING TO RESPOND
XC									X		X							X		3	PGM INSERTED MSG WITH Z1 FLD BITS SET RESERVED FOR SYSTEM USE
XD												X					X			1	IMS IS TERMINATING. FURTHER DL/I CALLS MUST NOT BE ISSUED. NO MESSAGE RETURNED.
XE									X		X							X		3	TRIED TO ISRT SPA TO EXPRESS PCB
XF									X									X		3	ALTERNATE PCB REFERENCED IN ISRT CALL FOR SPA HAD DESTINATION SET TO A LOGICAL TERMINAL, BUT WAS NOT DEFINED AS ALTRESP=YES.
XG									X									X		3	CURRENT CONVERSATION REQUIRES FIXED-LENGTH SPAS. ATTEMPT WAS MADE TO INSERT SPA TO TRANSACTION WITH A DIFFERENT OR NON-FIXED LENGTH SPA.
bb	X	X	X	X	X	X	X	X	X	X	X	X		X	X		X				GOOD. NO STATUS CODE RETURNED, PROCEED.

bb indicates blanks

*Utility Control Facility Status Codes

Figure A-1 (Part 2 of 2). DL/I Status Codes Quick Reference

APPENDIX B. DATA LANGUAGE/I STATUS CODES

The status codes that appear in tabular form in Appendix A are described in full detail in this section.

AA Error in call.

Explanation: The change call was ignored because the response alternate PCB specified a transaction code destination. Response alternate PCBs can only reference a logical terminal destination.

Action: Correct the application program.

AB Error in call.

Explanation: On a data base or message call, the segment I/O area is required but was not specified in the call.

Action: Correct program.

AC Error in call.

Explanation: SSA(s) contains an error in hierarchical sequence.

Possible causes:

1. No segment name equal to that specified in SSA was found within the scope of this PCB.
2. The level at which this SSA appears is out of sequence with that specified by the PCB.
3. Two segments of the same level are specified in the same call.
4. The statistics function that was specified or a STAT call was not a defined function.

Action: Correct the program.

AD Error in call.

Explanation: An invalid function parameter was supplied.

Possible causes:

1. A GU or GN was requested for a terminal PCB other than the I/O PCB.
2. An invalid function string exists.
3. An invalid request type was made for a TP PCB.
4. A call has been issued to the message queues with a DB PCB.

Action: Correct program.

AH Error in call.

Explanation: No SSA(s) was specified in the call. The call required at least one SSA (or RSA if GSAM being used), and none was specified.

Action: Correct the program by specifying SSA (or RSA) in call.

AI I/O, system, or user error

Explanation: Data management open error.

Possible causes:

1. An error exists in the DD statements.
2. The data set was opened for something other than load mode, but it is not loaded.
3. The buffer is too small to hold a record that was read at open time. See the IMS/VS System Programming Reference Manual for specification of the minimum buffer pool size.
4. DD statements for logically related data bases not supplied.
5. For an OSAM data set, the DSORG field of the OSAM DCB, DSCB, or JFCB does not specify PS or DA.
6. For an old OSAM data set, the BUFL or BLKSIZE field in the DSCB is zero.
7. The data set is being opened for load, and the processing option for one or more segments is other than L or LS.
8. The allocation of the OSAM data set is invalid; the allocation is probably (1,,1) rather than (1,1) and this causes the DSORG to be PO.
9. The processing option is L, the OSAM data set is old, and the DSCB LRECL and/or BLKSIZE does not match the DBD LRECL and/or BLKSIZE.
10. Incorrect or missing information prevented computation of block size or the determination of the logical record length.
11. A catalog was not available for accessing a VSAM data base that was requested.
12. OS could not perform on OPEN, but the I/O request is valid. Information is either missing, or data definition information is incorrect.

Action: Check the DD statements: ensure that the ddname is the same as the name specified on the DATASET statement of the DBD. The segment name area in the PCB has the ddname of the data set which could not be opened.

AJ Error in call.

Explanation: The SSA qualification format was invalid.

Possible causes:

1. Invalid command codes were used.
2. Invalid relational operators were used.
3. A right parenthesis or Boolean connector was missing.
4. More than eight Boolean members were specified.
5. The DLET call has multiple SSAs or qualified SSAs.
6. The REPL call has qualified SSAs.
7. The ISRT call has the last SSA qualified.
8. A path insert call into an existing data base involves a logical child segment.
9. The Record Search Argument (RSA) parameter is invalid.

Action: Correct the program.

AK Error in call.

Explanation: An invalid field name was supplied in the call.

Possible causes:

1. Unable to find the specified field name.
2. When accessing a logical child from the logical parent path, the field specified has been defined for the logical child segment and at least partially includes the portion of the logical child that contains the concatenated key of the logical parent.

Action: Correct program.

AL Error in call.

Explanation: The call is using a terminal PCB in a DL/I program.

Action: Correct program.

AM Error in call.

Explanation: The call function was not compatible with the processing option, segment sensitivity, or transaction-code definition.

Action: Correct program, PSB, or system definition.

Possible causes:

1. The D command code was used for a path retrieval call without path sensitivity.
2. The processing option of L and call function is not insert.
3. A DLET, REPL, or ISRT call was made without corresponding segment sensitivity.
4. A DLET, REPL, or ISRT call was issued by a program while a transaction defined as inquiry was being processed.

   A GET call was attempted for a segment with KEY sensitivity. Correct the error by specifying DATA sensitivity.
5. This status code occurs for a checkpoint (not restart) call if a GSAM/VSAM data set is opened for output.
6. An invalid request was included in a GSAM call.

AO I/O error

Explanation: There is a BSAM, GSAM, ISAM, VSAM, or an OSAM physical I/O error. When issued from GSAM, this status code means that the error occurred when: (1) a data set was accessed, or (2) the CLOSE SYNAD routine was entered. The error occurred when the last block of records was written prior to closing of the data set.

Action: Determine whether the error occurred during input or output, and correct the problem.

AQ Read I/O error

Explanation: The message chain cannot be followed; a minimum of one message is lost.

Action: If it is imperative to recover any messages that are lost, perform an emergency restart with the BLDQ option.

AR I/O error

Explanation: There is a read I/O error. A message segment has been lost, but the message chain is still intact.

AT Error in call in a VS system.

Explanation: The length of the user's I/O area data is greater than the area reserved for it in the control region. The length of the area reserved was determined by the ACB utility program, DFSUACB0, and printed as part of its output.

Action: Correct the PSB or the program in error.

AU Error in call in a VS system.

Explanation: The total length of the user's SSAs is greater than the area reserved for them in the control region. The length of the area reserved was determined by the ACB utility program, DFSUACB0, and printed as part of its output.

Action: Correct the PSB or the program in error.

AY Error in call.

Explanation: Insert call ignored because the logical terminal referenced by the response alternate PCB currently has more than one physical terminal assigned to it for input purposes.

Action: Ask the master terminal operator to determine (use /DISPLAY ASSIGNMENT LTERM X) which physical terminals (2 or more) refer to this logical terminal. Use the /ASSIGN command to correct the problem.

AZ Error in call.

Explanation: This status code is used to prevent asynchronous conditions involving the MPP, SPA content, and terminal. Possible causes for this status code are:

1. The conversational program inserted the SPA with a PURG call.
2. The TP-PCB destination is a conversational SMB, and there is no way to determine if the SPA was inserted to this PCB.
3. The TP-PCB destination is a logical terminal, and the TP-PCB is the I/O PCB or a response alternate PCB.
4. PURG is the only parameter (no PCB was specified), and ' ' status is returned; no action is taken if conditions 1, 2, or 3 (above) exist.

Action: Correct the application program and rerun.

A1 Error in call.

Explanation: The CHNG call was attempted with an eight-character logical terminal name which was unknown to the system.

Action: Correct program.

A2 Error in call.

Explanation: The CHNG call was attempted with an invalid PCB. It was either not an alternate PCB, was not defined as modifiable, or had a message in process but incomplete.

Action: Correct program.

A3 Error in call.

Explanation: An INSERT or PURGE call was attempted to a modifiable alternate PCB which had no destination set.

Action: Issue a CHNG call to set the PCB destination, and reissue the INSERT or PURGE call.

A4 Security violation

Explanation: The terminal entering the current transaction did not have the security to allow a message to the named SMB.

Action: User determined.

A5 Error in call.

Explanation: An invalid call list was supplied. A fourth parameter (MOD name) was supplied, but the function was not PURG or ISRT for the first segment of an output message.

Action: Correct the ISRT or PURG call and retry the application program.

A6 Error in call.

Explanation: Insert call ignored because output segment size exceeded specified limit.

Action: Correct the application program.

A7 Error in call.

Explanation: Insert call ignored because number of output message segments inserted exceeded specified limit by one. If another attempt is made to insert too many segments before the program issues another GU, the program is abended.

Action: Correct the application program.

A8 Error in call.

Explanation: Insert call ignored because an insert call to a response alternate PCS must not follow an insert call to the I/O PCB, or vice versa.

Action: Correct the application program.

A9 Error in call.

Explanation: Insert call ignored because it referenced a response alternate PCB that requires (SAMETRM=YES) the source physical terminal to receive the output response.

This status code can also occur if the input terminal is in response mode and the response alternate PCB is not associated with the input terminal.

Action: Determine whether the application program is in error, the output logical terminal has been erroneously reassigned (/ASSIGN command), or if SAMETRM=YES should not have been specified.

DA Error in call.

Explanation: Segment key field has been changed.

Action: Correct.

DJ Error in call.

Explanation: No previous successful GET HOLD call.

Action: Check and correct.

DX Error in call.

Explanation: Violated delete rule. Review the delete rule in the "Data Base Design Consideration" chapter of the IMS/VS System/Application Design Guide.

Action: Correct program.

GA Call is completed

Explanation: A hierarchical boundary into a higher level was crossed (see the discussion on hierarchical pointers in the "Data Base Design Considerations" chapter of the IMS/VS System/Application Design Guide), or the final call in a series of STAT calls was issued for VSAM buffer subpool statistics. This status code is returned on unqualified calls only.

Action: User determined.

GB Call is not completed.

Explanation: An attempt was made to satisfy a GN call and the end of the data base was encountered. (If this situation occurs on a GU or ISRT call, a GE status code is returned.) This status code is also returned when a GSAM data set has been closed.

Action: User determined.

GE Call is not completed.

Explanation: This status code is returned when: (1) an attempt is made to satisfy a GU or GN call but a segment cannot be found that satisfies the qualification, (2) an attempt is made to position for an ISRT call but one of the parents of the segment to be inserted cannot be found, (3) a STAT call is issued for ISAM/OSAM buffer pool statistics when the buffer pool does not exist, (4) a STAT call is issued for VSAM buffer subpool statistics when the subpools do not exist, and (5) a statistics function is specified on a STAT call for ISAM/OSAM buffer pool statistics.

Action: User determined.

GK Call is completed.

Explanation: Different segment type at same level returned. This status code is returned on unqualified calls only.

Action: User determined.

GL Call is not completed.

Explanation: Log code is not a valid user code. (Only codes X'A0' through X'E0' are reserved for users.)

Action: Check and correct.

GP Error in call.

Explanation: No parent for this GNP call, or the requested segment level is not lower than the parent level.

Action: User determined.

II Call is not completed.

Explanation: The segment that the user tried to insert already exists in the data base.

Possible Causes:

1. Segment with equal physical twin sequence field already exists for parent
2. Segment with equal logical twin sequence already exists for parent
3. Logical parent has logical child pointer, logical child does not have logical twin pointer, and segment being inserted is second logical child for logical parent
4. Segment type does not have physical twin forward pointer and segment being inserted is second segment of this type for parent or is second HDAM root for one anchor point
5. The segment being inserted is in an inverted structure; that is, the immediate parent of this segment in the logical structure is actually its physical child in the physical structure.

Action: User determined.

IX Error in call.

Explanation: Violated insert rule. Review the insert rule in the IMS/VS System/Application Design Guide.

Possible Causes:

1. Insert of logical child and logical parent (insert rule of logical parent is physical and the logical parent does not exist)

2. Insert of logical child and logical parent (insert rule is logical or virtual and the logical parent does not exist) and, in the user I/O area, the key of the logical parent does not match the corresponding key in the concatenated key in the logical child.

3. Insert of logical child (insert rule of logical parent is virtual and logical parent exists) and, in the user I/O area, the key in the logical parent does not match the corresponding key in the concatenated key in the logical child.

4. ISRT request after previous Open, Close or I/O error status code.

5. A GSAM ISRT call was issued after a previous AI or AO status code was returned.

Action: Correct program.

LB Call is not completed.

Explanation: The segment that the user tried to load already exists in the data base. Other possible causes are:

1. A segment with an equal physical-twin-sequence field already exists for the parent.

2. A segment type does not have a physical-twin-forward pointer, and the segment being inserted is either the second segment of this segment type for the parent or the second HDAM root for one anchor point.

3. An application program inserted a key of X'FF'..FF' into a HISAM or HIDAM data base.

Action: User determined.

LC Call is not completed.

Explanation: Key field of segments is out of sequence.

Action: Check and correct.

LD Call is not completed.

Explanation: No parent has been loaded for this segment.

Action: Check and correct.

LE Call is not completed.

Explanation: Sequence of sibling segments is not the same as the sequence in the DBD.

Action: Check and correct.

NF Call is not completed.

Explanation: Indexing maintenance issued a DL/I call, and the segment has not been found.

Action: User determined.

NI Data management open error or duplicate segment.

Explanation: Index maintenance was unable to open an index data base, or there was a duplicate segment in the index.

Possible causes for being unable to open the index data base:

1. Error in DD cards
2. The data set was opened for something other than load mode, but it is not loaded.
3. Buffer too small to hold record read at open time. See the IMS/VS System Programming Reference Manual for minimum buffers pool size.
4. DD cards for logically related data bases not supplied.
5. For an OSAM data set, the DSORG field of the OSAM DCB, DSCB, or JFCB does not specify PS or DA.
6. For an old OSAM data set, the BUFL or BLKSIZE field in the DSCB is zero.
7. The data set is being opened for load and the processing option for one or more segments is other than L or LS.
8. The allocation of the OSAM data set is invalid; allocation is probably (1,,1) rather than (1,1) and this causes the DSORG to be PO.
9. Processing option is L, the OSAM data set is old, and the DSCB LRECL and/or BLKSIZE does not match the DBD LRECL and/or BLKSIZE.

Action: Check DD cards; ensure ddname same as name specified on DATASET card of DBD. Segment name area in PCB has ddname of data set which could not be opened.

Possible causes for a duplicate segment in the index:

1. Index segment was incorrectly deleted earlier - Index should be rebuilt.
2. Index DBD incorrectly specifies unique key value - secondary index only.

NO I/O error

Explanation: There was a BSAM, ISAM, VSAM, or OSAM physical I/O error during a DL/I call issued by indexing maintenance.

Action: Check and correct.

QC CHKP was successful; GU was not successful (no more messages).

Explanation: There are no more input messages.

Action: As appropriate.

QD Call is not completed.

Explanation: There are no more segments for this message.

Action: As appropriate.

QE Error in call.

Explanation: A GET NEXT call was issued before a GET UNIQUE.

Action: Check and correct.

QF Error in call.

Explanation: Length of segment is less than five characters. (Allowable segment length is length of message text plus four control characters.)

Action: Check and correct.

QH Error in call.

Explanation: This is a terminal symbolic error -- the output designated is unknown to IMS/VS (logical terminal or transaction code).

Action: Check and correct.

RX Error in call.

Explanation: Violated replace rule. Review the replace rule in the "Data Base Design Considerations" chapter of the IMS/VS System/Application Design Guide

Action: Correct program.

UC Checkpoint record written to UCF Journal data set.

Explanation: During the processing of a HD Reorganization Reload or a user's Initial Load program under the supervision of the Utility Control Facility (UCF), a checkpoint record was written to the UCF Journal Data set. This status code is returned to indicate that the last ISRT call was correct and the User Initial Load program may continue or perform his checkpointing procedure before continuing.

UR The user's Initial Load program is being restarted under the UCF.

Explanation: During the processing of a user's Initial Load program under the UCF, a termination had occurred. The job was resubmitted with a Restart request.

Action: The user's Initial Load program must get itself back in step with Data Base Loading. Examination of the User I/O area or PCB key-feedback area can be used.

US The user's Initial Load program is preparing to stop processing.

Explanation: During the processing of a HD Reorganization Reload or a user's Initial Load program under the supervision of the Utility Control Facility (UCF), the operator replied to the WTOR from UCF and requested the current function to terminate. The last ISRT call was processed.

Action: The user's Initial Load program should checkpoint its data sets and return with a non-zero value in Register 15.

UX A checkpoint record was written and processing stopped.

Explanation: This is a combination of UC and US status codes; see the descriptions of those codes for further explanation.

Action: Refer to UC and US status codes.

V1 Error in call.

Explanation: An invalid length was supplied for a variable-length segment. The LL field of the variable-length segment is either too large or too small. The length of the segment must be equal to or less than the maximum length specified in the DBD. The length must be long enough to include the entire reference field; if the segment is a logical child, it must include the entire concatenated key of the logical parent and all sequence fields for the paired segment.

This status code is also returned when an invalid record length is specified in a GSAM call.

Action: Correct the program.

X1 System error.

Explanation: An I/O error occurred while IMS/VS was reading or writing the SPA.

Action: Terminate the conversation.

X2 Error in call.

Explanation: The first insert to a transaction code PCB that is conversational is not a SPA.

Action: Insert the SPA; then reinsert the data segment.

X3 Error in call.

Explanation: Invalid SPA (user modified the first six bytes).

Action: Correct the program, and restore the original bytes.

X4 Error in call.

Explanation: An insert was made to a transaction code PCB that is not conversational and the segment is a SPA.

Action: Do not pass the SPA to the transaction code. Send only data segments.

X5 Error in call.

Explanation: Multiple SPAs were inserted to a transaction code PCB.

Action: Only one SPA is allowed per message. Correct the program.

X6 Error in call.

Explanation: An invalid transaction code name was inserted into SPA.

Action: Correct the program to set the proper transaction code name.

X7 Error in call.

Explanation: The length of the SPA is incorrect (user-modified first six bytes).

Action: Correct the program.

X8 System error

Explanation: Error attempting to queue an SPA on a transaction code PCB.

Action: Terminate the conversation.

X9 Error in call.

Explanation: Incompatible conversational program call path.

Action: Design error. Report this to your system programmer.

XA Error in call.

Explanation: An attempt has been made to continue processing the conversation by passing the SPA to another program through a program-to-program message switch after already responding to the terminal.

Action: If a response has been generated, the SPA should be passed back to the I/O PCB. Review the rules for conversational programs in this manual and correct the program.

XB Error in call.

Explanation: The program has passed the SPA on to another program for processing but is trying to respond to the terminal.

Action: No response is allowed by a program which has passed control of the program through a program-to-program message switch. Review the rules for conversational programs in this manual.

XC Error in call.

Explanation: Program has inserted a message which has some Z1 field bits set which are reserved for IMS/VS use.

Action: Correct the program to prevent it from setting those bits.

XD IMS/VS is terminating by a CHECKPOINT FREEZE or DUMPQ.

Explanation: This code is returned only from a CHKP call issued by a batch-message application program. If the application accesses the message queues, no message is returned.

Action: Any subsequent DL/I call will result in an abend. The application should terminate.

XE Error in call.

Explanation: An attempt has been made to insert a SPA to an alternate PCB which was generated with the EXPRESS=YES option.

Action: Regenerate the PSB and remove the EXPRESS=YES option from the PCB or define another PCB (whose mode is not express) to be used in the insert call.

XF Error in call.

Explanation: Insert call for SPA ignored because the referenced alternate PCB had its destination set to a logical terminal but was not defined as ALTRESP=YES during PSB generation.

Action: Correct the application program or change the PSB generation for that alternate PCB to specify ALTRESP=YES.

XG Error in call.

Explanation: Insert call ignored because the current conversation requires fixed length SPAs and the insert was to a transaction with a different or non-fixed length SPA.

Action: Correct the program or IMS/VS System Definition.

bb Call completed.

Explanation: Your call was completed!

Action: Proceed!

INDEX

IMS/VS Version 1 Application Programming Reference Manual Printed in U.S.A. SH20-9026-4

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

IMS/VS Version 1
Application Programming Reference Manual
SH20-9026-4

Reader's Comment Form

Your comments about this publication will help us to improve it for you. Comment in the space below, giving specific page and paragraph references whenever possible. All comments become the property of IBM.

Please do not use this form to ask technical questions about IBM systems and programs or to request copies of publications. Rather, direct such questions or requests to your local IBM representative.

If you would like a reply, please provide your name, job title, and business address (including ZIP code).

Fold on two lines, staple, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

Fold and Staple

First Class Permit
Number 6090
San Jose, California

Business Reply Mail

No postage necessary if mailed in the U.S.A.

Postage will be paid by:

IBM Corporation
P. O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and Staple

IMS/VS Version 1 Application Programming Reference Manual Printed in U.S.A. SH20-9026-4

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)