SH20-9025-5

Program Product

IMS/VS Version 1 System/Application Design Guide

Program Number 5740-XX2

Release 1.4

IBM

Sixth Edition (July 1977)

This edition replaces the previous edition (numbered SH20-9025-4) and its technical newsletters (numbered SN20-9148 and SN20-9206) and makes them obsolete.

This edition applies to Version 1 Release 1.4 of IMS/VS, program number 5740-XX2, and to any subsequent releases unless otherwise indicated in new editions or technical newsletters.

Information concerning the Multiple Systems Coupling feature is provided for planning purposes until availability of the feature with Release 1.4. This feature is supported on installations running IMS/VS under OS/VS1 or OS/VS2 MVS system control programs.

Technical changes are summarized under "Summary of Amendments" following the list of figures. 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, P.O. Box 50020, Programming Publishing, San Jose, California 95150. All comments and suggestions become the property of IBM.

PREFACE

This publication presents the design considerations associated with installing and operating Information Management System/Virtual Storage (IMS/VS). It presents IMS/VS concepts, and the facilities available for designing IMS/VS Data Base (DB) and Data Base/Data Communication (DB/DC) systems.

This publication provides data base administrators, system designers, system programmers, and application programmers with the information they require to design an IMS/VS system and to design the applications which operate under IMS/VS.

Prerequisite to this publication is the IMS/VS General Information Manual, GH20-1260. In addition to information in the IMS/VS General Information Manual, the reader is expected to have a knowledge of Operating System/Virtual Storage (OS/VS) and OS/VS access methods. The chapters in this publication are:

1. "Design, Installation, and Maintenance of the IMS/VS Data Base System" that addresses the factors to be considered when installing a DB system.

2. "Design and Control of the Data Base/Data Communication System" that addresses the factors to be considered when installing a DB/DC system.

3. "Application Program Design" that includes considerations for design of batch and teleprocessing IMS/VS applications.

4. "Data Base Design Considerations" that describes data base concepts, structures, and the options available for designing IMS/VS data bases.

5. "Design Considerations for the Multiple Systems Coupling (MSC) Feature" that describes MSC and contains design considerations for its use.

6. "Design Considerations for the Fast Path Feature" that describes Fast Path and contains design considerations for its use.

RELATED PUBLICATIONS

IMS/VS Installation Guide, SH20-9081
This publication presents step-by-step details for the IMS/VS installation process.

IMS/VS System Programming Reference Manual, SH20-9027
This manual provides system programming personnel with installation considerations and details for generation (definition) of an IMS/VS system.

IMS/VS Application Programming Reference Manual, SH20-9026
This document is a reference manual for the application programmer. It provides information about the coding techniques necessary to implement a designed application under the IMS/VS system.

IMS/VS Utilities Reference Manual, SH20-9029
This manual provides a description of the IMS/VS system utility programs. It describes how to execute these utilities under the operating system.

IMS/VS Operator's Reference Manual, SH20-9028
This manual provides the master terminal, remote terminal, and system console operators with the information associated with operating IMS/VS once the system has been established in a user environment.

IMS/VS Message Format Service User's Guide, SH20-9053
This manual describes the use, definition, and implementation of the Message Format Service (MFS).

IMS/VS Advanced Function for Communications, SH20-9054
This manual explains the IMS/VS support for advanced function communications systems. It addresses the areas that programmers or analysts involved in communicating with IMS/VS must be familiar with.

IMS/VS Messages and Codes Reference Manual, SH20-9030
This manual lists, explains, and suggests appropriate responses to the completion codes and messages produced by all the IBM-supplied components of the IMS/VS system.

IMS/VS Program Logic Manual,
Volume 1 of 3, LY20-8004
Volume 2 of 3, LY20-8005
Volume 3 of 3, LY20-8041

REFERENCE PUBLICATIONS

- Introduction to the IBM 3850 Mass Storage System (MSS), GA32-0028
- OS/VS Mass Storage System (MSS) Planning Guide, GC35-0011
- IBM 3850 Mass Storage System (MSS) Principles of Operation, GA32-0029

CONTENTS

FIGURES

VERSION 1, RELEASE 1.4

NEW PROGRAMMING FEATURE

The Fast Path feature provides data base and data communication facilities for applications requiring high transaction rates but needing only simple data base structures. The Fast Path feature uses functions of the Data Communication feature and operates in existing telecommunication networks.

Fast Path provides two new types of data bases that are accessed with standard DL/I calls and, optionally, with Fast Path DL/I calls. The feature includes a message-handling facility to expedite the processing of Fast Path messages.

VERSION 1, RELEASE 1.2

This publication has been revised to reflect technical and editorial changes made for Release 1.2.

TECHNICAL CHANGES

Multiple Systems Coupling Feature

The Multiple Systems Coupling feature allows a user to define a configuration consisting of up to 255 interconnected IMS/VS systems running on any combination of OS/VS1 and OS/VS2. Information on channel-to-channel communication with the Multiple Systems Coupling feature is for planning purposes only until IMS/VS support for this facility becomes available.

3270 Information Display Station

With the addition of Synchronous Data Link Control, 3270s can now be attached on SDLC lines as well as BSC lines.

3350 Direct Access Storage Device

The 3350 may now be specified for data base and message queue data set residence.

3767 Communication Terminal
3770 Data Communication System

The 3767 and 3770 are supported on an SDLC link through VTAM. Full IMS/VS functional capabilities are included.

EDITORIAL CHANGES

- The IMS/VS planning information about MSS (mass storage system), previously contained in DB/DC MSS Planning Information: IMS/VS, CICS/VS, and GIS/VS, is now contained in Chapter 2 of this publication. The former MSS publication is now obsolete.
- The information previously contained in Chapter 1 of this publication has been moved to the IMS/VS General Information Manual.
- The information previously contained in Chapters 6 and 7 and Appendixes C and D of this publication has been moved to the IMS/VS System Programming Reference Manual.
- The information previously contained in Appendix A of this publication has been moved to the IMS/VS Installation Guide.
- The information previously contained in Appendix B of this publication has been moved to the IMS/VS Application Programming Reference Manual.

CHAPTER 1. DESIGN, INSTALLATION, AND MAINTENANCE OF THE IMS/VS DATA BASE SYSTEM

This chapter addresses the factors to be considered by the user data processing organization in planning, scheduling and controlling the installation of the IMS/VS Data Base (DB) System. Three major time phases should be considered:

- Pre-installation system design and configuration
- Installation
- System tuning and phased expansion

For each of these phases, this chapter suggests the steps to be taken, referencing the tools provided by the data base management services of IMS/VS to facilitate the effort, and identifying those elements of the user installation which are involved or affected.

DESCRIPTION OF FACILITIES

The data base management services of IMS/VS are packaged as basic material in an orderable component called the Data Base (DB) system. The DB system supports the implementation of multiple user-written batch processing applications in a common data base environment.

The DB system provides the user with full IMS/VS facilities to:

- Define, load, and reorganize data bases
- Access a data base from application programs via a high-level language interface called DL/I
- Support systems integrity via data base logging, checkpoint/restart, and data base recovery programs
- Use system-provided exits to incorporate user extensions to IMS/VS
- Migrate to a full IMS/VS DB/DC system in a shared terminal environment

The major execution-time elements of the IMS/VS DB system are the DL/I (Data Language/I) interface and the data base logging program. DL/I interfaces between the problem program and the data bases the program wishes to access. The use of DL/I and its functions are described in detail in the IMS/VS Application Programming Reference Manual. The data base logging capability is one of the principal IMS/VS recovery features. It provides a log of all activity against a data base. The log enables a user to analyze and tune his system, and is the basic support for recovery, restart, and backout activity. The log is discussed later in this chapter.

In addition, several utility programs which assist in creation and maintenance of the DB System are supplied. Included in this utility program set are:

IMS/VS System Definition

IMS/VS Data Base Description Generation

IMS/VS Program Specification Block Generation

IMS/VS Application Control Blocks Creation and Maintenance

IMS/VS Data Base Reorganization and Load

IMS/VS Data Base Recovery

IMS/VS Utility Control Facility

System definition is described in the IMS/VS Installation Guide; the other programs are described in the IMS/VS Utilities Reference Manual.

SYSTEMS

The DB system operates on an IBM System/370, using the services of OS/VS in its multiprogramming configurations OS/VS1 and OS/VS2.

In the IMS/VS DB system, applications are scheduled for execution through the OS/VS job stream in a process called batch scheduling. The basic unit of work is assumed to be the operating system job step. The application itself can be either transaction-oriented or batch-oriented. A transaction-oriented program facilitates migration to a DB/DC environment.

It is common system practice to implement the full DB/DC capabilities in a phased manner by installing a batch DB system first. Once an initial program/data base cluster has been designed and installed, users can see the step-wise expansion leading to a comprehensive on-line installation.

Figure 1-1 shows the relationship of OS/VS, the IMS/VS DB system, and an application program at execution time. The program and the DB system are contained in a single batch-processing problem program address space (region, memory). A second application program can occupy a second address space, with a replica of the DL/I and data base logging functions, accessing a separate data base and writing a second log tape. Two or more IMS/VS batch systems can run concurrently in separate address spaces, if they do not access the same data bases. Most of the IMS/VS DB system is composed of reenterable code.

The user's application program operates as an OS/VS problem program. As illustrated in Figure 1-1, the application program has two basic interfaces. These are:

1. Transaction Input and Response Output

2. Data Base Input/Output Operations

Although this is a batch processing environment, the concept of transaction processing is advocated, because it can be carried over to the IMS/VS message processing environment. Typically, transaction input and response output are performed with OS/VS data management. Within the application program, file descriptions and read/write statements are in COBOL, PL/I, or Assembler Language syntax. Alternatively, the user of IMS/VS can build an interface for transaction input and response output similar to the data base input/output interface described below.

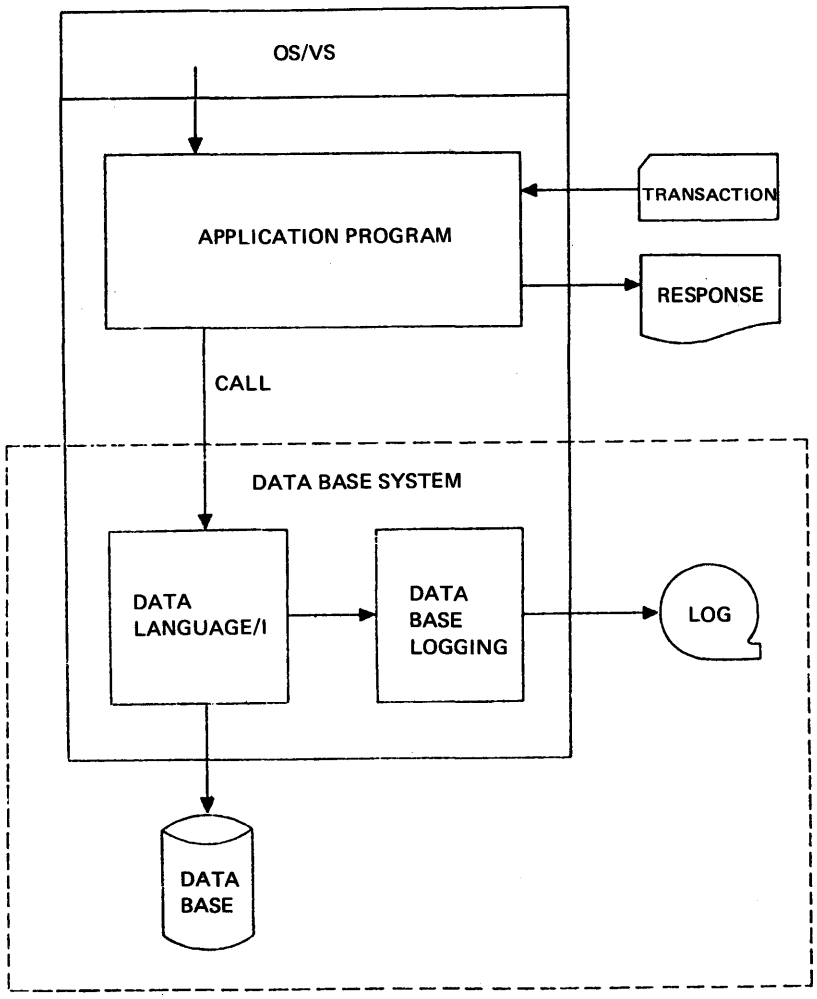


Figure 1-1. IMS/VS Data Base System Environment

All data base operations are initiated by the application program interface with DL/I. This interface consists of execution of a CALL statement from the application program. Parameters in this CALL statement provide the information necessary to perform a data base operation on a specific data element or segment in a specific data base. An application program can interface with one or more DL/I data bases. In addition, standard OS/VS data management can be used for any purpose in the application program.

The arguments in the CALL statement issued by an application program allow DL/I to determine which data base is to be used and which data segment in the data base is to be retrieved, inserted, replaced, or deleted. From this information DL/I performs a VSAM, SAM, ISAM, or OSAM input/output operation. If the desired data segment already exists within the data base main storage buffers, no input/output operation is required.

When the data base operation consists of data segment insert, replace, or delete, a record of the data base modification is written on an IMS/VS log for the batch processing address space. The content and format of logged information are described in a subsequent section of this chapter.

One significant concept of the data base input/output interface is that the format and content of all information used to establish the interface are symbolic. None of this information is dependent upon a specific data management access method or organization.

DB SYSTEM GENERATION DESIGN DECISIONS

Before the DB system can be used for batch data base processing, it must be tailored to the user's data processing environment. This process of system tailoring is called system definition. The details of IMS/VS system definition are provided in the IMS/VS Installation Guide. For IMS/VS, the system definition function is similar, in concept, to OS/VS system generation.

OS/VS OPTION CONSIDERATIONS

The DB system operates under OS/VS1 or OS/VS2. Very little difference is experienced by the IMS/VS user, whether VS1 or VS2 is used. The primary differences are attributable to the OS/VS option characteristics. Chapter 2 of this manual describes the considerations for operation under VS1 or VS2. These are primarily concerned with main storage management and reliability/serviceability. The effects of VS1 versus VS2 are considerably greater with the IMS/VS DB/DC system.

Only one OS/VS option defined during OS/VS system generation is a requirement for the DB system. This is user SVC inclusion.

Other OS/VS options or features which are desirable, but not required, for IMS/VS are VSAM access method, ISAM access method, ANS-COBOL, PL/I F, PL/I Optimizing Compiler, and a Sort/Merge function.

The ASSEMBLER, an IMS/VS requirement, is automatically incorporated in OS/VS. Alternatively, the Assembler H program product may be used.

IMS/VS SYSTEM DEFINITION

The DB system definition process includes the specification of the following parameters.

- OS/VS system under which IMS/VS operates -- VS1 or VS2
- OSAM access method's supervisor call number and channel end appendage name

During and after system definition and before DB system execution, several IMS/VS library data sets must be defined. These include control block libraries, load module libraries, and a procedure library. The details of data set definition and allocation are defined in the IMS/VS Installation Guide. Libraries which are of importance to the discussion in this chapter are:

IMSVS.RESLIB - the IMS/VS system load module library

IMSVS.PGMLIB - the user's application program library

IMSVS.DBDLIB - the IMS/VS control block library containing data base descriptions (DBD). Each member describes the logical structure of data and its physical storage in a data base.

IMSVS.PSBLIB - the IMS/VS control library containing application program specification blocks (PSB). Each member contains a description of how its asscciated application program uses one or more data bases.

IMSVS.ACBLIB - the IMS/VS library which contains control blocks required for a specific application program. This is a combination of the DBDs and PSBs in an internal format required by DL/I for data base system execution.

IMSVS.PROCLIB - the IMS/VS procedure library containing IBM-supplied procedures.

IMSVS.MACLIB - the IMS/VS macro instruction library containing at least DBD generation and PSB generation macro instructions.

Special Access Method -- OSAM

The Overflow Sequential Access Method (OSAM) is a special data management access method supplied with IMS/VS. It is used with some of the IMS/VS data base organizations. The functions which OSAM performs vary and depend upon the data base organization specified for a particular OSAM data set. These functions are described in a subsequent chapter of this manual. The other modules of IMS/VS interface with OSAM through OPEN, CLOSE, READ, and WRITE macro instructions similar to those provided for any OS/VS access method. OSAM modules interface with the OS/VS input/output supervisor through the EXCPVR macro instruction in VS1 and SVS and through the I/O driver interface in MVS. As far as OS/VS is concerned, an OSAM data set is described as data set organization equals physical sequential (DSORG=PS). In fact, an OSAM data set can be read using BSAM or QSAM. The advantages of OSAM to IMS/VS relative to either BSAM or BDAM are:

1. An OSAM data set can occupy as many extents and direct access volumes as a BDAM data set.

2. An OSAM data set can be opened for update in place and extension to the end through one data control block (DCB). The phrase, extension to the end, means that records may be added to the end of the data set and that new direct access extents may be obtained.

3. An OSAM data set need not be formatted prior to use.

4. An OSAM data set can have fixed length blocked or unblocked record format.

5. File mark definition is always used to define the current end of the data set. The addition of a new block causes the file mark to be placed after the new block. This concept is used as a reliability aid while the OSAM data set is open.

During data set open, OSAM requires a type 4 supervisor call and a channel end appendage. The SVC number and the channel end appendage name are specified during IMS/VS system definition. The corresponding SVC modules and channel end appendage must be placed in SYS1.SVCLIB by the user. See the IMS/VS Installation Guide for information on how to accomplish this task.

It should be remembered that other OS/VS access methods, VSAM, ISAM, and SAM are used for physical storage of data elements in addition to OSAM.

OSAM data sets are restricted to a 31 bit addressing limit.

Generalized Sequential Access Method (GSAM)

The Generalized Sequential Access Method (GSAM) provides accessing support for simple physical sequential data sets, such as tape files, SYSIN, SYSOUT, and others that are not hierarchical. These are data sets which, before GSAM, could not be used as IMS/VS data sets.

Support provided includes sequential or direct retrieval by a record identifier which defines the relative position of that record.

Support is provided for both OS/VS Sequential Access Method (SAM) and OS/VS Virtual Storage Access Method (VSAM) for entry sequenced data sets (ESDS). GSAM is fully described in IMS/VS Application Programming Reference Manual. The concepts of hierarchy and segment described in this manual do not apply to GSAM.

DATA BASE AND APPLICATION DESIGN DECISIONS

DATA BASE DESCRIPTION (DBD) GENERATION

Subsequent to IMS/VS system definition and its related functions (such as incorporation of the OSAM SVC modules and channel end appendage module into SYS1.SVCLIB (OS/VS1) or SYS1.IPALIB (CS/VS2)), DBD generations can be performed. A DBD must be provided for each data base to be used by an application program, prior to execution of the program. The IMS/VS Utilities Reference Manual describes the details of DBD generation.

DBD generation is the execution of IMS/VS macro instructions to create a description of a data base. This data base description includes a definition of:

- The data base organization and access method
- Segment formats, whether fixed or variable
- Whether the segments are subject to data compression routines
- Inter-segment relationships
- Field formats within segments
- The existence of index relationships for any field
- The relationships, if they exist, between segments in two or more data bases

Figure 1-2 illustrates the execution of a DBD generation. The IMS/VS user creates control card statements that are presented to DBD generation as a normal OS/VS problem program job. The IMS/VS macro instructions used for DBD generation exist in IMSVS.MACLIB. The result of a DBD generation execution is the placement of the compiled DBD into IMSVS.DBDLIB as a member of the partitioned data set. The members of the IMSVS.DBDLIB library can be used during the Data Base System execution.

A job control language procedure, named DBDGEN, is placed in the IMSVS.PROCLIB data set by IMS/VS system definition for subsequent DBD generation execution. This procedure is described in the IMS/VS System Programming Reference Manual.

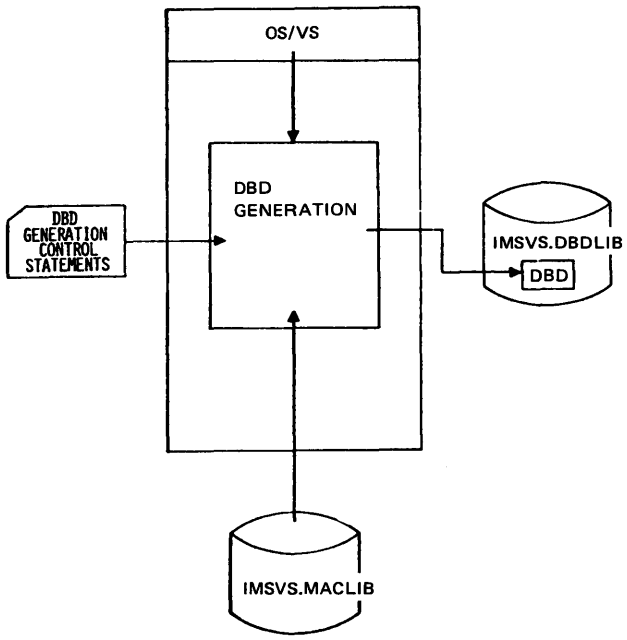


Figure 1-2. DBD Generation Execution

PROGRAM SPECIFICATION BLOCK (PSB) GENERATION

The third necessary function prior to execution in the DB system batch processing environment is PSB generation. Associated with any batch processing application program is a PSB control block. The PSB control block defines the data bases used by the application program. In addition, it defines the manner in which the data bases are used (that is, retrieval only, retrieval and update, or data base create) and the segments within each data base to which the application program is sensitive. It also defines which, if any, additional secondary indexes can be used to assist in segment selection.

PSB generation is the execution of IMS/VS macro instructions to define an application program's use of one or more data bases. The IMS/VS user creates control statements that are executed during PSB generation as a normal OS/VS job. The IMS/VS macro instructions used for PSB generation are in IMSVS.MACLIB. The result of PSB generation execution is the placement of the compiled PSB into IMSVS.PSBLIB as a member of the partitioned data set (Figure 1-3). The members of the IMSVS.PSBLIB data set are used during the Data Base's system execution. The IMS/VS Utilities Reference Manual describes the details of PSB generation.

A procedure, named PSBGEN, is placed in the IMSVS.PROCLIB data set by IMS/VS system definition for subsequent PSB generation execution. This procedure is described in the IMS/VS System Programming Reference Manual.

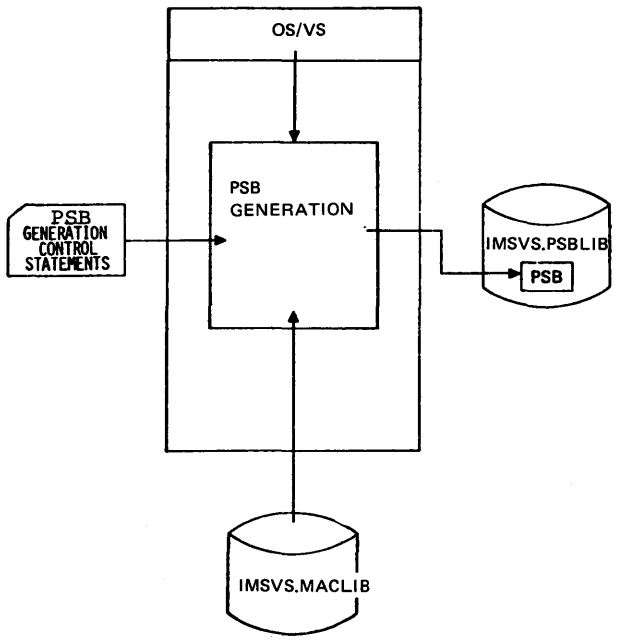


Figure 1-3. PSB Generation Execution

APPLICATION CONTROL BLOCKS (ACB) CREATION AND MAINTENANCE

The fourth necessary function prior to execution of the data base system, is ACB creation and/or maintenance. This function is optional in a DB system. It is required in a DB/DC system. Associated with all batch processing application programs are DL/I control blocks which define the data bases, structures, and methods to be used with a particular application.

The information in these blocks can be constructed in either of two ways: (1) at initialization time, the logical block builder module (DFSDIBI0) is called to construct the blocks from the PSBs and DBDs associated with the application program to be scheduled; or (2) the Application Control Blocks Creation and Maintenance utility program (DFSUACB0) is used to prebuild the control blocks for the application program. In this case, the necessary control blocks are loaded directly from the IMSVS.ACBLIB data set, saving processing overhead.

Application Control Blocks Creation and Maintenance requires no IMS/VS resources other than IMSVS.PSBLIB, IMSVS.DBDLIB, and IMSVS.ACBLIB. The user supplies control statements which specify the operations to be performed. (See Figure 1-4.) The IMS/VS Utilities Reference Manual describes the details of ACB creation and maintenance.

A procedure, named ACBGEN, is placed in the IMSVS.PROCLIB data set by IMS/VS system definition for subsequent ACB creation and/or maintenance. This procedure is described in the IMS/VS System Programming Reference Manual.

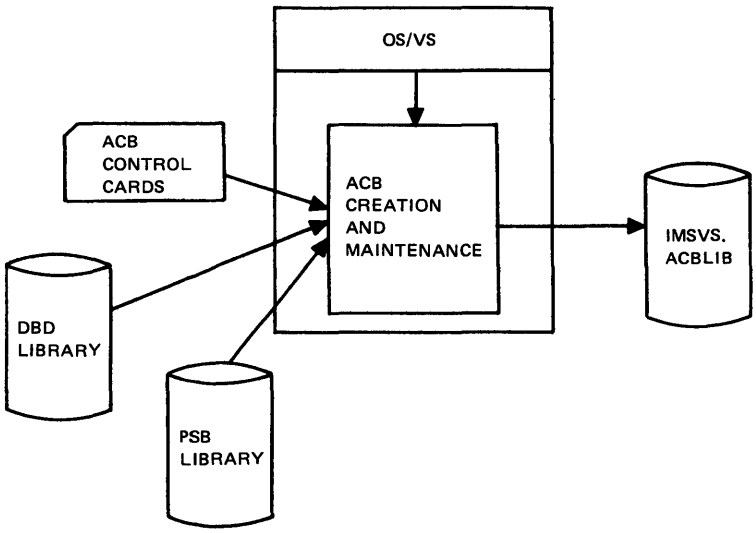


Figure 1-4. ACB Creation and/or Maintenance

APPLICATION PROGRAM DESIGN

The final function performed prior to DB system execution is the creation of application programs. Application programs are required to create and maintain all user-defined data bases. These programs are written in Assembler Language, COBOL, or PL/I, and must be placed in the IMSVS.PGMLIB data set after compilation and link edit. IMS/VS JCL procedures are available to the user for program compilation and link edit. These procedures are placed in the IMSVS.PROCLIB data set by IMS/VS system definition. Each compiled application program must be link-edited with modules that will be called by the application program during execution. JCL procedures cause this link-edit to be performed. For details see the IMS/VS System Programming Reference Manual.

EXECUTION AND CONTROL OF THE DATA BASE SYSTEM

This section of the chapter describes batch processing in the DB system. Prior to execution, the functions of IMS/VS system definition, DBD generation, PSB generation, and application program compilation, and optionally, ACB creation, are assumed to have been performed.

ESSENTIAL PROGRAM ELEMENTS FOR EXECUTION

Figure 1-5 illustrates the program elements necessary for batch execution. The IMS/VS control blocks are obtained from the IMSVS.ACBLIB (if prebuilt blocks are to be used) or are constructed dynamically at execution time from the PSBs and DBDs associated with the application program. The application program is obtained from the IMSVS.PGMLIB data set. The IMS/VS DB system modules are obtained from the IMSVS.RESLIB data set.

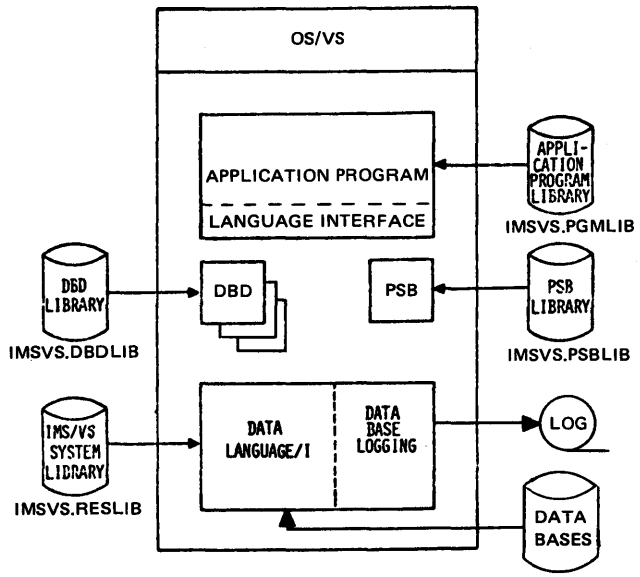


Figure 1-5. Essential Program Elements for Execution

Program Specification Block (PSB)

As previously described, there is a PSB associated with the batch processing application program. The PSB is composed of one or more subordinate control blocks called data base program communication blocks (PCB). Each data base PCB specifies a data base or logical structure of data segments used by the application program. The PCB specifies the name of the DBD associated with the desired data base and the names of segments within the data base to which the program is sensitive. Secondary indexes can be specified to aid in segment selection. Segments to which an application program is sensitive can be retrieved, updated, inserted, and deleted. Segments to which an application is not sensitive are never presented to the application program. The concept of segment sensitivity provides some degree of data independence. Additional constraints can be placed on the manner in which an application program is sensitive to a segment. Levels of sensitivity can be defined for each segment. The lowest level of sensitivity is segment retrieval only. The next level of sensitivity allows segment retrieval, update, insert, or delete.

Data Base Description Block (DBD)

Each data base PCB in the PSB associated with the application program to be executed defines a DBD by name. This means that one or more DBDs are required for any batch program execution. Each DBD defines the organization and segment structure in its associated data base. The concept of a logical data base and associated DBD is defined in a subsequent chapter of this manual. If the DBD named in a data base PCB is associated with a logical data base, one or more additional DBDs are required to define the data base and identify the segments within the data base to which the program is sensitive.

Application Control Block (ACB)

Together, the PSBs and DBDs are used to construct the IMS/VS application control blocks. This may be done dynamically or by a utility program which prebuilds the blocks. The IMS/VS Utilities Reference Manual describes this process.

Application Program

An application program to be executed in the batch processing DB system environment may be written in COBOL, PL/I, or Assembler Language. A subsequent chapter of this manual describes design considerations for an application program. The details of application program implementation are provided in the IMS/VS Application Programming Reference Manual.

IMS/VS System Modules

The IMS/VS modules utilized in the DB system environment are obtained from the IMSVS.RESLIB data set. These modules are placed in that data set by the execution of IMS/VS system definition. The majority of the modules are involved with handling data base requests from the application program. These modules in turn utilize the data management access method modules of VSAM, ISAM, OSAM, GSAM, and SAM.

The primary IMS/VS modules are:

Data Base Retrieval Module

Data Base Insert Module

Data Base Delete/Replace Module

Data Base VSAM Interface Module

Data Base ISAM Simulator Module

Data Base Hierarchical Direct Space Management Module

Data Base ISAM/OSAM Buffer Handler Module

Data Base Buffer Handler Router Module

Data Base Hierarchical Direct Index Maintenance Module

GSAM Access Method Modules

OSAM Access Method Modules

Data Base Logging Modules

A later chapter of this manual describes the IMS/VS data base access methods. Each of these data base access methods uses either standard OS/VS data management access methods or OSAM for the physical storage of segments. The following illustrates the relationships.

DATA BASE ACCESS METHOD	LOW LEVEL ACCESS METHOD USED FOR PHYSICAL STORAGE
HSAM	QSAM or BSAM
HISAM	BISAM-OSAM, QISAM-OSAM, or VSAM
HDAM	OSAM or VSAM
HIDAM	BISAM-OSAM, QISAM-OSAM, or VSAM
GSAM	ESAM or VSAM (ESDS only)

If sequential processing of an HSAM data base is defined, QSAM is used in support of HSAM. If nonsequential processing of an HSAM data base is requested, BSAM is used in support of HSAM. When a HISAM data base is created or reorganized, QISAM load mode and OSAM are used. When an existing HISAM data base is used for retrieval, insert, update, and/or delete, QISAM scan mode and OSAM are employed. If a PSB generation defines two or more data base PCBs which relate to the same HISAM data base, BISAM read and write are used for HISAM retrieval, update, delete, and/or insert. OSAM or VSAM is used for all data segment storage when the data base access method is HIDAM or HDAM. The use of ISAM (BISAM or QISAM) in an HIDAM data base is for index segment storage only. The use of BISAM or QISAM in support of the HIDAM data base access method is equivalent to that described for HISAM.

DATA BASE SYSTEM EXECUTION

Once the functions of IMS/VS system definition, DBD generation, PSB generation, and application program creation have been accomplished, execution of the Data Base system may be performed. The initial DB system execution presumably loads data into one or more of the data bases previously defined by DBD generation. (The load process is described in the IMS/VS Utilities Reference Manual.) Subsequent executions would perform retrieval, update, insert, and/or delete operations against existing data bases and/or create additional data bases.

When a batch processing execution of the DB system is initiated, the control blocks associated with the application program must be obtained and initialized. This control block initialization process is part of the batch processing job step execution but precedes the loading of the application program. The first step involves obtaining the DL/I control blocks. If PARM=DBB was specified, the required control blocks are obtained from IMSVS.ACBLIB by the block loader module. If PARM=DLI was specified, the block builder module is called to construct blocks dynamically. In this case, IMSVS.PSBLIB and IMSVS.DBDLIB are used to obtain the required PSBs and DBDs. Once the blocks have been obtained, the initialization routines load the required DL/I action modules, initialize STAE, and format the necessary storage areas in preparation for loading and giving the application program control. Figure 1-6 illustrates this initialization process.

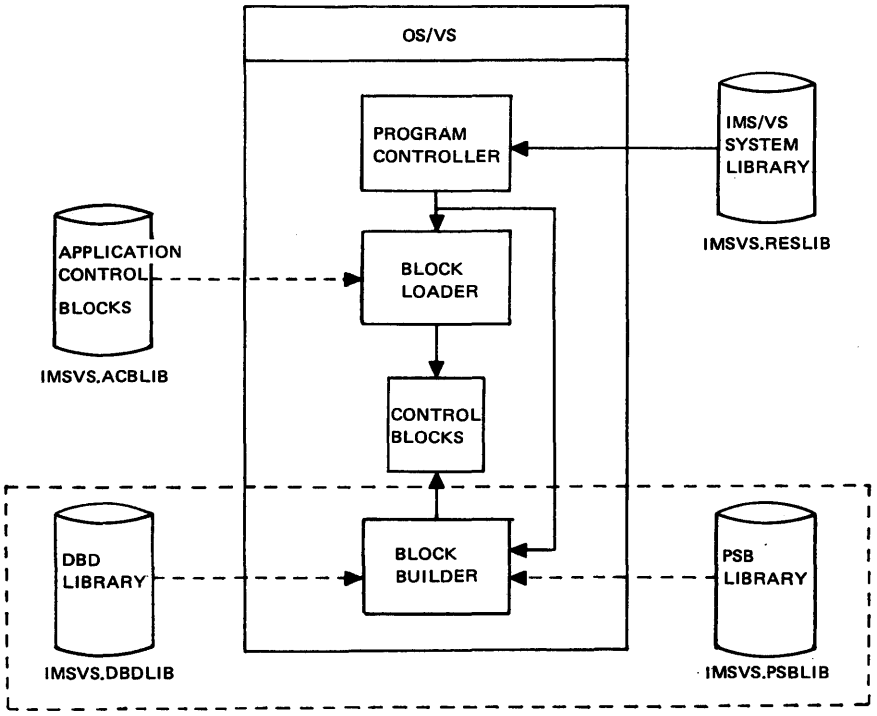


Figure 1-6. Initializing the Batch Data Base System, Step One

The IMS/VS module which controls the DB system environment is called the program controller. The primary functions of the program controller are:

- Initiate the IMS/VS DB system block building process by passing control to the IMS/VS control block building modules (Figure 1-6).
- Initiate the DB system DL/I and data base logging modules and pass control to the user's application program (Figure 1-6).
- Terminate the DB system execution by returning to OS/VS.

The EXEC statement provided as part of the OS/VS job control language for the DB system batch processing execution includes, within the values of its PARM= operand, the names of the PSB and the application program to be executed. Control is passed from the region controller (not shown in Figure 1-6) to the program controller, to the block loading and building modules. The name of the PSB is supplied. Using the PSB name, the required control blocks are obtained. If the first PARM= value is DBB, the required control blocks are loaded from the IMSVS.ACBLIB data set. If the first PARM= value is DLI, the named PSB is loaded from the IMSVS.PSBLIB data set and referenced DBDs are loaded from the IMSVS.DBDLIB data set. From the PSB and DBD control blocks, internal IMS/VS control blocks are built for subsequent input/output operations in the DB system environment.

After the control block construction is complete, control within the OS/VS address space is returned to the program controller. At this point the remainder of the DB system functions are initialized (Figure 1-7). This includes loading of the required DL/I data base modules, loading of the data base log modules, creation of a data base buffer pool, and loading of the required data management access method modules.

Depending upon the data base organizations and the manner in which each data base is used by the application program, only the necessary DL/I and access method modules are loaded.

Finally, the application program to be executed is obtained from the application program library, IMSVS.PGMLIB. Control is given to the application program.

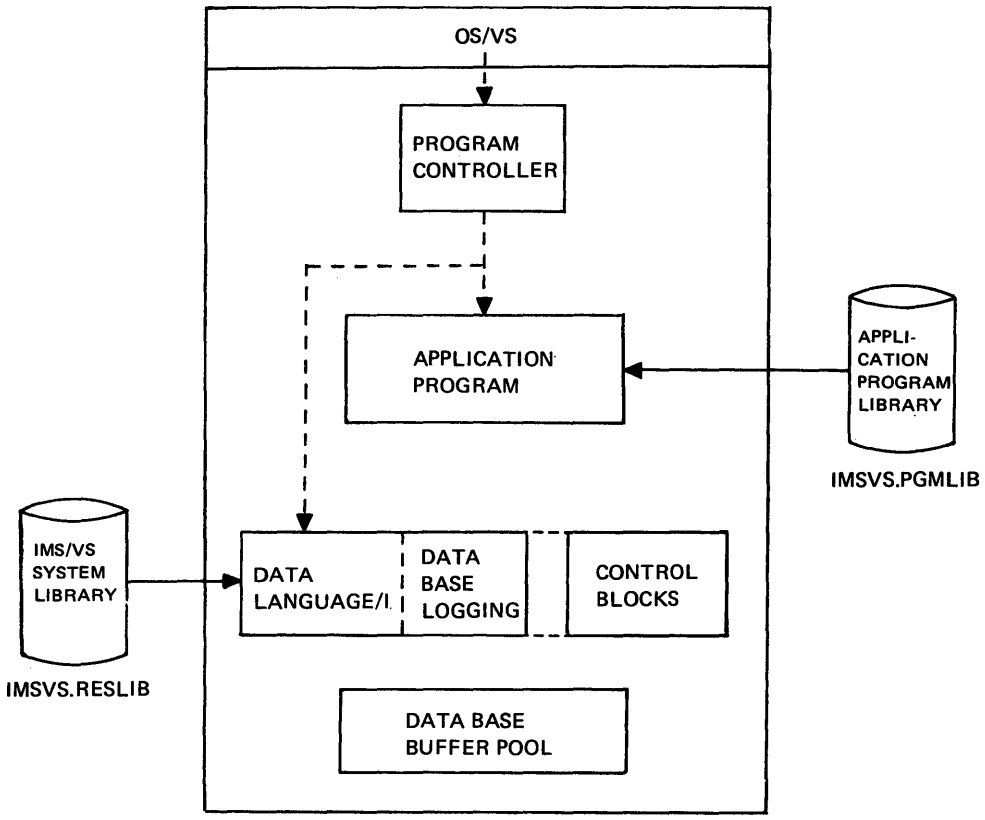


Figure 1-7. Initializing the Batch Data Base System, Step Two

Data Base System JCL Considerations

IMS/VS provides procedures for DB system batch processing execution. They are DLIBATCH, DBBBATCH, IMSPLIGO, and IMSCOBGC. They are placed in IMSVS.PROCLIB during IMS/VS system definition. These procedures are described in detail in the IMS/VS System Programming Reference Manual and include the basic JCL for execution. The user must add DD statements for all data bases to be used. The content and format of these DD statements are described in the IMS/VS Utilities Reference Manual.

Data Base System Control Sequence Flow

Figure 1-8 illustrates the DB system control sequence flow once the application program has been given control. Upon entry to the application program, a parameter address list is provided. The addresses in this list provide visibility to each data base PCB in the PSB for the application program (see arrow 1). These data base PCB addresses are subsequently used by the application program when issuing data base input/output call requests.

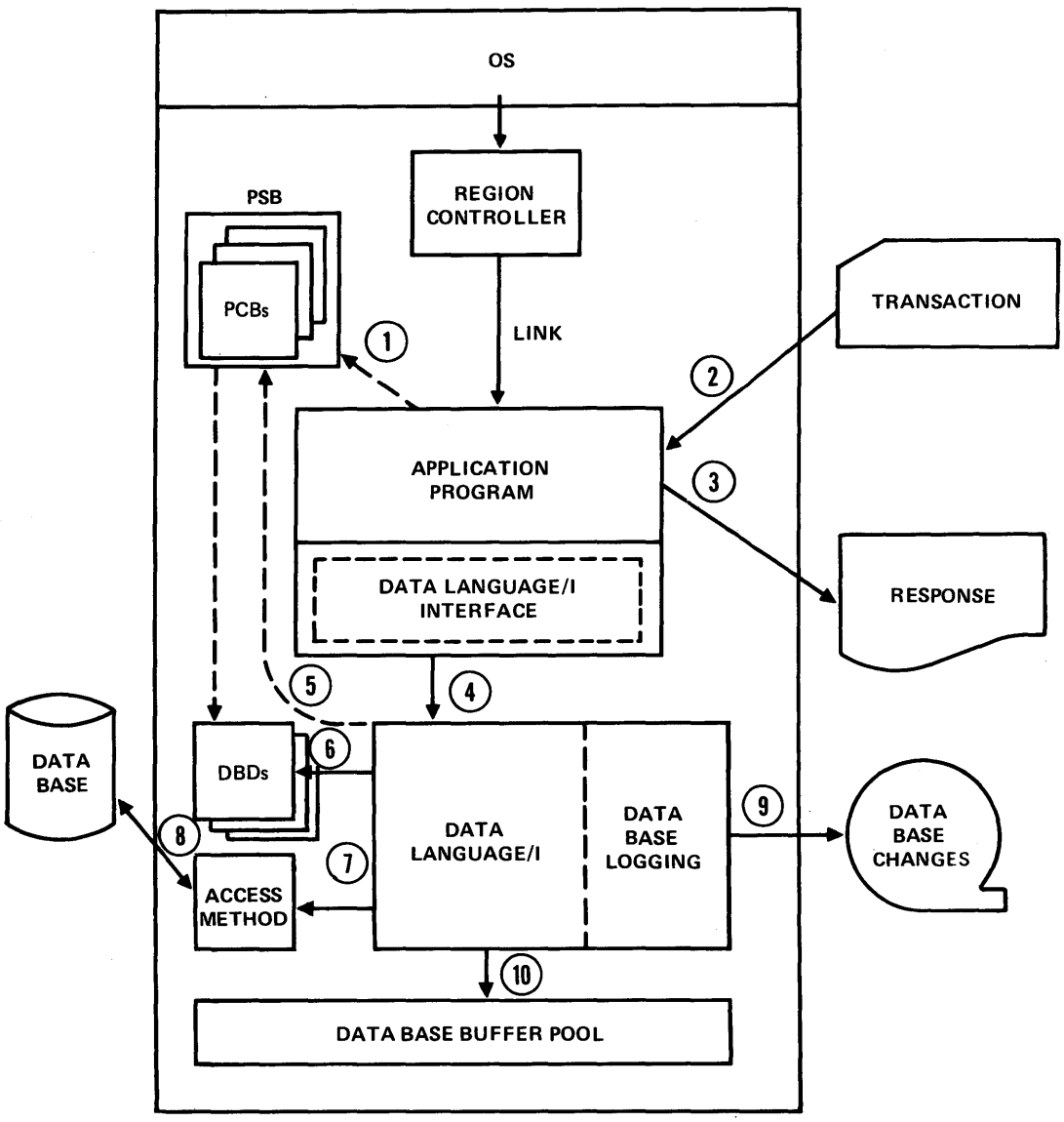


Figure 1-8. Data Base System Flow

Arrows 2 and 3 in Figure 1-8 indicate the transaction input and response output interface with the application program.

All application programs operating under IMS/VS have a language interface link-edited with the application program. The language interface accepts a data base call from the application program and passes control to the DL/I data base modules (arrow 4).

The purpose of the language interface is to provide a consistent format to DL/I for all data base call requests, independent of the programming language used to write the application program. The IMS/VS-supplied OS/VS procedures for compiling and link editing application programs are described in the IMS/VS System Programming Reference Manual. The link edit step for each of these procedures provides for the inclusion of the language interface with the correctly compiled PL/I, COBOL, or Assembler Language application program.

The language interface function of IMS/VS is reenterable and is upwardly compatible with that of IMS/360 Version 2. To take advantage

of the reenterable capability of the IMS/VS language interface, application modules from IMS/360 installations must be re-link-edited, replacing the IMS/360 language interface with the IMS/VS language interface.

After control is passed to the DL/I modules for execution of the data base call request, the following functions are performed.

1. The parameters in the call request are checked for valid content. This checking involves the use of data base PCBs (arrow 5) and DBDs (arrow 6).

2. If the data base call involves segment retrieval, the information contained in control blocks and data base buffers is used to attempt to satisfy the request. If the request can be satisfied, the desired data segment is placed in the input/output work area of the application program provided in the data base call.

3. If the retrieval request cannot be satisfied with information already contained in the data base buffer pool, the appropriate data management access method modules are invoked (arrow 7), and the data management access method modules perform the necessary input requests to place necessary data in the data base buffer pool (arrow 8).

4. If the data base call request involves data segment update or deletion, the segment must first be retrieved (from either the buffer pool or secondary storage). Subsequently, the segment is deleted or updated (arrow 10).

5. If the data base call request involves data segment insertion, the segment is placed in the data base buffer pool and subsequently written to direct access storage (arrow 10).

6. When the data base call request involves segment insertion, updating, or deleting, a record of the data base modification is placed on an IMS/VS log for the batch address space (region) execution. This logical information can subsequently be used for data base recovery or reconstruction (arrow 9).

Data Base Buffering

IMS/VS maintains two data base buffering functions: one for VSAM data bases, and one for ISAM/OSAM data bases. A separate pool of buffers is allocated for each type of data base (VSAM and ISAM/OSAM) and the data management access methods (VSAM, ISAM, and CSAM) are directed to read into and write from these buffers.

The concept of the buffer pool is to allow blocks of data to remain in main storage as long as possible to avoid secondary storage reads and writes. Data in the buffer pool can be accessed and updated without I/O as long as there is no need to reuse the buffer space the data occupies. A use chain determines the order in which the buffers are used. Empty buffers are placed at the bottom of the use chain and are always available for reuse. As buffers are accessed, they are placed at the top of the use chain. When a retrieve request occurs, the buffer pool is searched using the use chain (for ISAM/CSAM a hush table is used to direct the search), to determine if the requested data is already in main storage. If the data is not found, the least recently used buffer (bottom of the use chain) is selected, the old data is written out if it has been changed, and the requested data is read into the selected buffer.

The size of the data base buffer pools can have a significant effect on the performance of the IMS/VS system. The size of the buffer pool is defined during DL/I initialization, based on control statements provided by the user (see the section "Defining the IMS/VS Buffer Pool" in the IMS/VS Installation Guide). The size of the ISAM/OSAM buffer pool, for IMS batch jobs, can be defined by the BUF parameter on the EXEC statement for the job step or by buffer control statements.

At the beginning of each of the data base buffer pools, there exists a work area used by IMS/VS to record statistics on the activity in the pool. These statistics are of value to the IMS/VS user in determining the appropriate buffer pool size for a given application program. The DL/I statistics call (STAT) can be used to obtain these statistics in an application program (see the IMS/VS Application Programming Reference Manual for a description of the STAT call).

For additional information on buffering see the section "DL/I Data Base Buffering Facilities" in the IMS/VS System Programming Reference Manual.

SYSTEM INTEGRITY AND MAINTENANCE CONSIDERATIONS

DATA BASE LOGGING

All modifications to any data base used in the DB system environment are recorded on the IMS/VS log tape for the address space. If multiple data base system executions are performed concurrently under OS/VS1 or VS2 a separate IMS/VS log tape is associated with each address space. Unless a data base is being used for retrieval purposes only in all address spaces accessing the data base, no attempt should be made to access the same data base from more than one OS/VS address space at any one time.

Data base logging provides the IMS/VS system user with a recording of all modifications to all data bases used during a data base execution. The log can be written with BSAM or QSAM. See "Chapter 2. Log Facility" in the IMS/VS System Programming Reference Manual for performance improvement considerations using TOSAM. An IMS/VS option, log-tape write-ahead, insures that log records are written before the data in the data base is changed. See the section "DL/I Data Base Buffering Facilities" in the IMS/VS System Programming Reference Manual for additional information on log-tape write-ahead. The IMS/VS log tapes can be used in conjunction with the IMS/VS Data Base recovery utility to rebuild a data base. The IMS/VS Utilities Reference Manual provides details on the use of data base log information for recovery.

If no data base changes will be made or if no data base recovery utilities will be used, the logging function can be made inoperable by specifying DD DUMMY on the primary log DD statement (DD name IEFRDER). If dual system logs are used and the primary log is specified as DD DUMMY the secondary log is ignored and no logging is done.

If a data base is destroyed because of input/output errors, it can be restored with the following procedure.

- Restore the data base with a previously dumped copy. The Data Base Reorganization utilities can be used for this purpose. Refer to the IMS/VS Utilities Reference Manual for detailed information.

- Apply all data base modifications made to the data base since the dumped copy was created. The IMS/VS Data Base Recovery utility programs provide this function.

- Repeat the current DB system processing from the beginning.

The image for the data base log tape record format is contained in the IMS/VS Program Logic Manual.

Note: The above discussion of data base logging and recovery does not apply to the HSAM organization. Since the old data is not destroyed when updating HSAM, logging and backout are not required to maintain data base integrity.

BATCH CHECKPOINT/RESTART

The batch checkpoint facility provides for synchronizing checkpoints.

The CHKP function call to DL/I allows the coordination of program activity with data base activity. Lacking any means to identify significant events in an application program, DL/I treats data base calls as one continuous string of related actions. When a CHKP call is issued, the program is indicating to DL/I that a sync point has been reached and the data base buffers should be written to secondary storage. For batch programs, a checkpoint record is also recorded on the log data set to indicate the sync point and set the maximum point to which the data base backout occurs if it becomes necessary.

In the batch environment, the duration of the job may be long and the number of data base changes may be large. If the job lasts for many hours then the time for reloading direct access data sets and rerunning, if necessary, may be excessive. Batch checkpoint/restart allows the user to take one or more sync points during execution. The sync point then determines the amount of time required to backout the data base if restart occurs. Backout is effected only back to a specified checkpoint record.

The action taken by the DB system for batch programs when a CHKP call is issued is as follows:

1. Altered data base buffers are written.

2. The checkpoint ID supplied in the CHKP call is written to the log tape.

3. Message DFS681I, containing the checkpoint ID supplied, is sent by a WTO to the system console, and to the programmer.

4. Optionally, an OS/VS checkpoint of the user's region is taken. If the IMS/VS log access method is OSAM, the OS/VS checkpoint is not taken.

It is the user's responsibility to checkpoint any non-IMS/VS information or data sets (such as transaction/response data sets) with issuance of the CHKP call. This can be done with the OS/VS checkpoint/restart option in the DL/I CHKP call.

As an alternative to the OS/VS checkpoint/restart option, the user can specify the IMS/VS expanded checkpoint/restart facility. This consists of a restart call (function code XRST) and optional parameters on the CHKP call. If used, the XRST call is the first call to IMS/VS issued by the user program. If a restart is not in progress, the XRST call is effectively a NOP.

The issuance of an XRST call causes the following action to be taken for subsequent CHKP calls issued by the program:

1. Optionally, user specified areas, that is, application variables, control tables, and position information for non-IMS/VS data sets, are recorded on the IMS/VS log.

2. The fully qualified key of the last record processed by the program on each IMS/VS data base is recorded on the log.

3. The functions of the standard CHKP call are performed, except that the OS/VS checkpoint of the user's region is not taken. The user has the option of using OS/VS Checkpoint/Restart, or the IMS/VS restart (XRST call), or neither, but not both.

Batch Backout Utility Program

A checkpoint ID can be supplied to the IMS/VS Batch Backout utility program through a control statement. The backout of data segments from the data base is done from the end of the log tape until the matching checkpoint record is encountered.

In the case of a batch program, the checkpoint/restart facility used can then be invoked to restart the program from that point.

The batch checkpoint facility is implemented by the use of the CHKP system service call from the application program. This call is used to indicate a sync point at which any data base updates can be restarted. The actual checkpointing of the batch program environment and the routine used to restart it are at the option of the user. The DL/I checkpoint program cannot issue the CS CHKPT macro.

If the DL/I user chooses to write his own checkpoint/restart routines, he must:

- Record application variables and control tables.
- Record position information for non-IMS/VS data sets.
- Provide a restart entry point and reinitialization procedure.
- Initialize IMS/VS control blocks, for example, PXPARMS.

Use of the XRST call and user area parameters on the CHKP call simplifies the task for the user writing his own restart routines.

- A restart situation is indicated by specifying a checkpoint ID in the PARM field (on the EXEC statement in the JCL) or in the XRST call itself.
- Normal entry point and initialization procedures are used.
- User areas recorded at checkpoint time are restored.
- A GET UNIQUE is issued for each GSAM data base for the last used record, if the data base was open at the time the checkpoint was taken.
- No data is returned as the result of the GU, but key feedback and status codes are saved in the user PCBs.
- If the data base was opened for output, then a PNT function code, requesting PCINT, is used.

- GSAM data bases are automatically repositioned at restart if the XRST call is used.

- The checkpoint ID is returned to the user program to allow it to link to its own restart subroutine.

Batch programs that do not utilize the batch checkpoint facility should be reprogrammed to do so. The major advantage comes from significantly shorter backout runs after failure, and the ability to terminate a long running job and restart it at a current point with very small backout preparation and minimal rerun time.

IMS/VS USE OF STAE/ESTAE

In order to ensure that all IMS/VS data base log buffers are written, IMS/VS establishes a STAE/ESTAE routine in order to gain control during an abnormal termination.

If an application program uses the STAE/ESTAE facility, it should issue only one STAE/ESTAE. When IMS/VS STAE/ESTAE processing is completed the abnormal termination is reissued to ensure that the application STAE/ESTAE routine is given control. If the application program issues multiple STAE/ESTAEs, in order to be first on the list, the following precautions must be observed:

1. The application program must not delete the IMS/VS STAE/ESTAE control block, nor use the OVERLAY option in issuing its STAE/ESTAE.

2. If the application STAE/ESTAE routine gains control prior to the IMS/VS STAE/ESTAE routine, the abnormal termination must be reissued in order to give control to the IMS/VS STAE/ESTAE routine. Also, the IMS/VS log buffers and control blocks must not be modified and no DL/I call can be issued.

Note: 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.

IBM SYSTEM/370 POWER WARNING FEATURE SUPPORT

The IMS/VS Power Warning Log Terminator program supports the power warning feature on System/370 Models 158 and 168. This support enables the user to close the IMS/VS log from a dump data set without having to restore memory. The procedure used to accomplish this is described in the IMS/VS Operator's Reference Manual.

IMS/VS DB MONITOR

The IMS/VS DB monitor is a tool for collecting performance data to investigate specific application designs, data base designs, and resource allocations. It consists of a monitor module, and a Monitor Report Print program. When activated, it analyzes and records the internal activities of the IMS/VS DB system. The monitor report print program is processed offline to produce reports that summarize and categorize, at various levels of detail, the information recorded by the monitor module. The actions required to activate the monitor module are described in the IMS/VS Operator's Reference Manual. The monitor report print program is described in the IMS/VS Utilities Reference Manual.

The monitor module collects data from IMS/VS control blocks during operation of the batch system (with minimum interference to the system) and records the data either on an independent data set or on the IMS log. The monitor remains resident and is activated and deactivated through the system console.

The following are recommendations for use of the DB monitor:

- Collecting data -- The DB Monitor enables an IMS/VS user to collect performance data to assist in analyzing an existing IMS/VS batch system. The amount of data collected and the analysis time to understand the report output suggest short traces during various time periods. Reports produced from profiles of a batch execution considered as normal can be used as a profile and compared with reports produced during a batch execution with unusual performance characteristics.

- Tuning system -- The DB Monitor can be used to quantify the effect of actual changes to data base structures, program characteristics, data set placement, and pool sizes.

- Testing application -- In the final testing of new or revised applications, the DB Monitor can be useful in validating the internal operation of the programs and data bases. For example, the programmer thought a specific DL/I call could be satisfied with a single I/O retrieval, yet the DL/I call report indicated a large data base scan as shown by many IWAITs. Investigation of such items could assist in determining whether a new or revised application meets the performance objectives. Data contained in the reports may also assist in defining the resources required by an application program.

CHAPTER 2. DESIGN AND CONTROL OF A DATA BASE/DATA COMMUNICATION SYSTEM

This chapter concerns the decisions and planning that must precede the installation and use of the IMS/VS Data Base/Data Communication (DB/DC) system. Familiarity with chapter 1 of this manual is assumed.

RELATIONSHIP OF DB/DC TO DB SYSTEM

For the most part, the design and control considerations of a Data Base system, as discussed in chapter 1, are applicable to the combined DB/DC (Data Base/Data Communication) system discussed in this chapter. The fundamental differences in the data base oriented considerations, when viewed from within the combined DB/DC environment, have to do with multiplicity and interaction. In the Data Base system, for example, only one program and its associated program specification block are used at a time. In the DB/DC system, planning must consider that multiple programs and their associated control blocks may be in use at the same time. Furthermore, the interactive effects among those multiple programs and control blocks must be considered. This kind of relationship applies, as well, to other resources managed in the DB/DC system; such as data base buffers, data base control blocks, terminal buffers, and message processing regions.

The contents of this chapter provide guidance in:

- Selecting an OS/VS configuration
- Selecting an IMS/VS configuration
- Establishing a message scheduling algorithm
- Selecting and configuring a physical terminal network
- Establishing a logical terminal network

ORGANIZATION OF DB/DC PROCESSING

Regions are distinguished by the kind of processing performed within them. There are three kinds of regions. They are the IMS/VS control, message processing, and batch processing regions. A further distinction is made between batch processing using a private or local control program, and batch processing through the online control program. The private use of the control program is simply called batch processing. Batch processing is provided by the DB system of IMS/VS. The use of the online control program to support batch-oriented operations is called batch-message processing. There are three names that relate to mode of operation: control, message, and batch. A combination of batch and message processing is called batch-message processing.

When necessary, communication between regions is supplied through type 2 SVCs (supervisor call routines). The control within the IMS/VS control program region, where multiple input/output operations are occurring asynchronously, is provided by use of the OS/VS EVENT capability.

MESSAGE PROCESSING -- MPP REGION

The IMS/VS control program region is initiated through an OS/VS START command. The IMS/VS control program and one or more message processing regions are initiated by the job management facilities of OS/VS.

FAST PATH PROCESSING -- IFP REGION

The IFP Region is described in Chapter 6.

BATCH MESSAGE PROCESSING -- BMP REGION

A BMP region may contain an application program for processing against data bases in the batch manner. The application program in the batch region is scheduled by OS/VS job management, but may utilize the DL/I facility for data base reference. An application program executed in a BMP region can access only IMS/VS data bases that are defined in the IMS/VS control region.

A BMP region, in addition to being able to process data bases used for message processing, has access to input message queues and can provide output to the message queues. Access to the input message queues is provided by specifying, in the JCL for a BMP region, a transaction type to which access is wanted. Access to the output message queues is provided by specifying output terminal PCBs in the PSB for the application program that executes in the BMP region.

When the data bases normally used for message processing are not being used for that purpose, they can be processed in a batch processing region as described in chapter 1 of this publication. This can be done when the IMS/VS control program is not operative as an OS/VS job.

CONFIGURING THE SYSTEM THROUGH OPTIONS

OS/VS OPTIONS

Fixed or Variable Tasking

The selection of an OS/VS configuration has some effect on the potential performance and reliability and availability characteristics of IMS/VS. Certain options are required by IMS/VS in all of the applicable configurations. The functional characteristics of IMS/VS, based on the use of these options, are identical regardless of the OS/VS configuration selected.

The IMS/VS DB/DC system runs under OS/VS1 and OS/VS2.

IMS/VS Program Module Preload Function

OS/VS, IMS/VS, and application programs that will run in regions of IMS/VS can be made permanently resident in virtual storage. This can significantly improve throughput and response time for frequently referenced transactions, if sufficient virtual storage is available with high performance paging DASD.

Programs can be made permanently resident in two ways:

1. In LINKPACK/RAM

   a. These programs are shared among all regions, resulting in a saving of virtual storage space. Initial access can be slow because the region JOBPACK and STEPLIB/JOBLIB are searched before LINKPACK is searched. Subsequent access can be at CPU speeds, if the region JOBPACK has not been purged by OS/VS space management (this would be the case if sufficient virtual storage were not available to satisfy a user request for space).

   b. These programs are made resident by the same method used for OS/VS and IMS/VS modules.

   c. Application modules with names identical to PSBs should not be placed in OS/VS LINKPACK, since this causes a conflict during the ACB generation process.

2. In REGION/PARTITION

   a. These program modules are used only for transactions serviced by the region involved, and only for the duration of that region.

   b. OS/VS and IMS/VS modules that are used in a dependent region can reside in that dependent region. An example is the Interregion Copy routine.

   c. Because these modules are in the region JOBPACK, they are invoked without repeating the overhead of searching STEPLIB/JOBLIB, LINKPACK, and SYS1.LINKLIB. The overhead of fetching the module into virtual storage is encountered only at region initialization time.

   d. They are made resident by invoking the IMS/VS Program Module Preload function via the step execution JCL parameter.

   e. In addition to those modules automatically preloaded into the IMS/VS control region, other OS/VS and IMS/VS modules can be made resident in the IMS/VS control region by using Module Preload.

   f. Module Preload can also be used for modules resident in LINKPACK/RAM. Thus, although the modules are physically residing in LINKPACK (and are being shared among multiple regions), the overhead involved in searching program libraries and LINKPACK are only experienced at region initiation.

Serially reusable programs can be resident only in the region/partition. They are made resident by invoking the Module Preload function via the step execution JCL parameter.

The OS/VS task under which modules are preloaded varies based on the IMS/VS region type:

IMS/VS Region Type	OS/VS Task
Control (CTL)	Physical Log
Message (MSG)	Region/Program Control
Batch Message (BMP)	Region/Program Control
Batch (DLI)	Region/Program Control
Fast Path (IFP)	Region/Program Control

Performance Considerations for Modules Preloaded in MPPs/IFPs

If modules are preloaded into MPPs, the following performance considerations apply:

1. Preloaded applications are invoked via the BRANCH instruction; this avoids OS overhead.

2. Applications that are not preloaded and that have not been previously invoked are located by issuing the BLDL macro instruction; this reduces operating system overhead by avoiding a PDS directory search for the application modules in subsequent scheduling. The maximum number of BLDL entries in the BLDL list can be specified in the PARM field of the MSG region JCL. The entries are kept in the list on the basis of:

   a. most recently referenced

   b. most often referenced

3. All non-reentrant preloaded modules are reloaded after each abnormal termination.

4. If an abnormal termination with DUMP occurs, all preloaded modules will be printed.

IMS/VS IN AN OS/VS SYSTEM

Supported Configurations

	OS/VS1		OS/VS2		
REGION TYPE	V=R	V=V	V=R	V=V	SWAP
CONTROL		X		X	
MESSAGE		X		X	X
BATCH-MESSAGE		X		X	X
BATCH	X	X	X	X	X*
FAST PATH				X	

* IMS/VS determines whether a batch region is swapped regardless of whether the user specifies the region as being swappable. To IMS/VS, a batch region is swappable if it has no log, and it is not swappable if it has a log.

OS/VS OPTIONS REQUIRED OR RECOMMENDED FOR IMS/VS

- Options Required

  Many of the OS/VS system generation macro instructions must specify certain options and values to support an IMS/VS DB/DC system. The required operands of the macro instructions are discussed below.

- Options Recommended

  Certain OS/VS options are not required by IMS/VS, but are recommended for various reasons. A discussion of the reasons why they are recommended, by macro instruction, follows:

Either requirements or recommendations are made for the following OS/VS SYSGEN macro instructions:

CENPROCS	DATAMGT	PARTITNS	LPALIB
CTRLPROG	EDITOR	RESMODS	SVCTABLE
IODEVICE	MACLIB	SVCLIB	

CENPROCS

Values that can be assigned to the MODEL keyword are System/370 machines equal to or greater than 384K. The CPU instruction set must be specified as INSTSET=UNIV.

CTRLPROG

In VS1 RESIDNT=TRSVC is required to allow for making load 4 of the IMS/VS type 4 SVC routine resident. The value of MAXIO should be reviewed to ensure that sufficient capacity is available to support the DB/DC system. Use of the asynchronous overlay supervisor, in conjunction with PCI fetch, is recommended. At least the ADVANCED level of the overlay supervisor is required. If the configuration is OS/VS1, system queue area size must be increased over the default size. Although in OS/VS2, system queue space expands dynamically as required, it is recommended that the initial size be adjusted to account for increases in the DB/DC environment. See "IMS/VS Storage Estimates" in the IMS/VS System Programming Reference Manual for further information about system queue space. Additional transient areas are recommended. A discussion of the potential improvements in performance available through the use of additional transient areas is in the topic "Fixed or Variable Tasking."

IODEVICE

If the online system contains 7770-3 lines, the IODEVICE specification for the 7770-3 lines must indicate a teleprocessing device class in the device type field.

DATAMGT

Access methods ISAM and BTAM are required. If VSAM and/or VTAM are to be used, they must be included at system generation.

EDITOR

The F-level linkage editor is required.

MACLIB

Consult the IMS/VS System Programming Reference Manual to determine the block size of the DB/DC system macro definition library. The block size of the OS/VS and IMS/VS macro definition libraries should be the same.

PARTITNS

The PARTITNS macro instruction is used only on an OS/VS1 system. Although partitions can be reconfigured through use of the OS/VS DEFINE command, it is recommended that they be established at generation. The DB/DC control program must operate from a partition defined as a resident reader partition. Refer to "IMS/VS Storage Estimates" in the IMS/VS System Programming Reference Manual to determine the partition size. In OS/VS1 the partition number is related to execution dispatching priority. A lower partition number implies higher relative dispatching priority. P1 has a higher dispatching priority than P2 but lower than P0.

RESMODS

Three user SVCs are supplied with IMS/VS. Two are nucleus-resident. If the IMS/VS SVCs are available at OS/VS system generation, it is convenient to incorporate them by using this macro instruction.

SVCLIB

One of three user supervisor calls (SVCs) supplied with IMS/VS is a Type 4 SVC. If it is available at OS/VS system generation, this macro instruction provides a convenient way to incorporate it. This macro instruction is valid for OS/VS1 only.

LPALIB

Provides the same function for OS/VS2 as SVCLIB provides for OS/VS1.

SVCTABLE

Three user SVCs are supplied with IMS/VS. See the paragraph entitled "Supervisor Call Routines" for further planning information. The SVC numbers are assigned through your installation's normal procedures. The attributes of the user SVCs are:

	OS/VS1	OS/VS2
TYPE1		Not required
TYPE2	Entered disabled No APF required	No locks Entered enabled
TYPE4	Six doublewords SRVB extension, Load 4 resident	Six doublewords SRVB extension

OS/VS SUPERVISOR CALL ROUTINES

The Data Base/Data Communication System uses three SVCs. Provision for these SVCs must be made during the generation of your OS/VS configuration. The SVC numbers designated must also be specified during IMS/VS system definition. It is recommended that the three numbers designated be established as an installation or company standard to provide for interchangeability among CPUs, operating system configurations, and installations. In the Data Base System, only one SVC number, a type 4 SVC, is required. In the DB/DC System, that same number plus one type 1 and one type 2 number are required. The type 1 and 2 SVCs are resident in the operating system nucleus. The main storage requirements for the type 1 and type 2 SVCs are quite small. If possible, it is recommended that they be incorporated in the nucleus of each operating system that is used on a CPU licensed for IMS/VS. The routines are appendage tables that serve only as access routes to the actual SVC code. The extended SVC code is resident only while the DB/DC system is active. Because of the limited function performed by the resident appendage tables, the likelihood of change through future modification level releases is small.

In VS1, load 4 of the IMS/VS Type 4 SVC routine must be made resident for all IMS services. Accordingly, IEARSV00 must be modified to include the module name, or an alternate list must be specified as a parameter at IPL time.

SPECIAL ACCESS METHOD -- OSAM

The functions and operations of OSAM described for the Data Base system in chapter 1 of this publication also apply to the DB/DC system. The DB/DC system uses OSAM for message queue management. Further discussion of message queue management appears later in this chapter.

Allocation of OSAM Data Sets

The normal mode of OSAM data set allocation is through the use of the JCL at the time the data set is loaded. This can be for single or multiple volumes, and is done using the SPACE parameter.

If the installation control of direct access storage space and volumes is such that the OSAM data sets must be preallocated, or if it is decided that a message queue data set will require more than one volume, the OSAM data sets may be preallocated.

Preallocation is done by any of the accepted methods, with the following restrictions:

- DCB parameters should be specified only if they exactly match those that will be assigned to the DCB at load time.
- If the data set is to be expanded beyond the preallocated space, a secondary quantity must be specified in the DD statements during preallocation. Note that queue data sets will not be expanded beyond their initial or pre-allocated space quantity.

When a multiple-volume data set is preallocated, the method of allocation should either allocate extents on all of the volumes to be used, or guarantee that the end of the data set is correctly indicated in the DSCB for the last extents. Suggested methods are:

1. Issue the IEFBR14 utility once for each volume.
2. Write the first extent using BSAM or QSAM.
3. Execute a program that opens and closes a DCB to preallocate on the first volume only. A filemark should be written on the first track of the extent.
4. Use IEFBR14 on the first volume only but use AMASPZAP to correct the DSCB. Do not simply use IEFBR14 and specify a DD card for a multi-volume data set. This will put an extent on the first volume only and will not indicate that the volume is the last volume of the data set.

Do not preallocate more volumes for data base extents than the initial load of the data set will use. IMS/VS has no way to indicate the end of data before the last volume on which the data resides. If extra volumes are allocated, OSAM will be forced to close the data set at the end of the load step without indicating the end of the good data (by placing a valid TTR in the DSCB of the last volume). This restriction does not apply to message queue data sets.

If the data set is preallocated by a program that writes records in the extents, an invalid (to OSAM) TTR is placed in the last volume of the data set. Unless the CLOSE after loading the data base overrides the TTR, OSAM is unable to properly re-open the data set. The Unload utility program, and any other programs using the OSAM data set as a sequential data set, will operate sucessfully, since physical sequential processing starts at the beginning of a data set, and ends at the filemark placed by OSAM to define the logical end of the data set.

IMS/VS FEATURES

Configuring the IMS/VS system for a particular user environment is accomplished through IMS/VS system definition. The IMS/VS Installation Guide provides the information necessary to perform a system definition. IMS/VS system definition consists of macro statements, the operands of which tailor the IMS/VS system for the user. The next several sections of this chapter discuss the design considerations in selecting various system definition options. You may wish to subsequently review this chapter with the system definition details in the IMS/VS Installation Guide.

CONTROL PROGRAM

OSAM

For OSAM operations, IMS/VS allows you to include the EXCPVR option at system definition time. This increases performance and lowers CPU overhead because it provides translation of channel programs in place.

Processing Regions

The specification of maximum processing regions places a limit upon processing capabilities that can be changed only through redefinition. The value assigned to the MAXREGN keyword of the IMSCTRL macro statement includes both message and batch message processing regions. The maximum number of regions specified influences the calculation of maximum I/O requests. The largest value that should be specified is 15.

Active I/O Requests

The specification of active I/O requests is one of the system-related specifications that directly influences the performance potential of the DB/DC system. It governs the maximum number of I/O requests outstanding at any time. You must specify a value that exceeds, by at least two, the maximum number of regions that can be executing concurrently. It is recommended that the value be one-half the sum of the number of communication lines, plus the number of concurrently operating processing regions. This number should reflect a prediction of maximum to average number of active communication lines. If the autopoll feature is used, it should be possible to reduce the assigned value without significantly affecting performance. The value assigned to the MAXIO keyword of the IMSCTRL statement requires a significant amount of main storage. Each potentially active I/O event requires approximately 500 bytes of storage.

Checkpoint Frequency

The selection of a checkpoint frequency should be influenced by anticipated message and data base processing activity, and the need for rapid restart. The frequency value chosen determines the number of log records that are written between automatic environment checkpoints. Whatever the value chosen, it is somewhat self-adjusting to system processing rates. That is, as more messages and data base update activities are processed, more log records are written. Hence, automatic checkpoints occur more frequently.

Immediate Checkpoint

The Immediate Checkpoint feature reduces the impact of frequent IMS/VS system checkpoints on system performance by eliminating forced termination of message processing programs at simple checkpoint time. System activity is interrupted only during the time the system control blocks are actually being written to the log.

System Queue Space

System queue space must be sufficient to support the requirements of the OS/VS control blocks necessary for the operation of an IMS/VS system. See "IMS/VS Storage Estimates" in the IMS/VS System Programming Reference Manual for the information necessary to estimate the required system queue space.

IMS/VS Enqueue/Dequeue

Main storage is obtained dynamically within the control region by the IMS/VS enqueue/dequeue routines. The maximum amount of main storage that these routines obtain, and the maximum that these routines keep on an internal free chain, are specified by the CORE keyword in the IMSCTF macro statement.

Program Isolation

Under the program isolation concept, all activity (data base modifications and message creation) of an application program active in the DB/DC system is isolated from any other application program(s) active in the system until that application program commits, by reaching a synchronization point, that the data it has modified or created is valid.

This concept makes it possible to dynamically back out the activities of an application program that terminates abnormally, without affecting the integrity of the data bases controlled by IMS/VS. It does not affect the activity performed by the other application program(s) processing concurrently in the system.

With program isolation and the dynamic backout facility, it is possible to provide data base segment occurrence level control to application programs. A means is provided for resolving possible deadlock situations in a manner transparent to the application program. The deadlock situation is detected by an IMS/VS routine called Exclusive Control Enqueue/Dequeue. Upon detecting a deadlock situation, one of the application programs involved in the deadlock is abnormally terminated with a special abnormal termination code. The abnormal termination causes the activity of the terminated program to be dynamically backed out to a previous synchronization point. Its held resources are freed. This allows the other program(s) to process to completion. The special code causes the transaction that was being processed to be saved. The application program is rescheduled. This process is transparent to application programs.

Performance is enhanced by allowing control of data base updates to be maintained dynamically, as opposed to establishing the control at message scheduling time. This dynamic maintenance is controlled by the DL/I action modules through the use of the IMS/VS Enqueue/Dequeue routine. During the scheduling process, an analysis is made of the intent of an application program toward the data base it uses. If a conflict exists with the data base usage of a currently scheduled

transaction, the scheduling process must select another transaction code and try again.

MESSAGE SCHEDULING

Within IMS/VS each input message type is declared through system definition. Message types are called "transaction codes" or "transactions." At the time a transaction code is declared, many optional attributes can be selected. These attributes, either directly or indirectly, affect the schedulability of a transaction. They can also affect the manner in which a physical terminal reacts to entry of a transaction type.

Application programs are declared in separate but related macro instructions. However, the application program designated to process a particular transaction code is really just another transaction attribute. The process through which a completely received input transaction is united with its associated application program is called "message scheduling." The variable attributes associated with the transaction code, the number and relative importance of other transaction codes, the number of received but not processed messages, the intent of associated application programs toward the data to be processed, the amount of currently available space in control block storage pools and buffers -- these and other factors influence the process of message scheduling. The influencing factors are called the "message scheduling algorithm."

Through selection of options at system definition time, through the design and use of data bases, specification of buffer sizes, and, most directly, through the declaration and selection of transaction code-related options, the IMS/VS system designer can influence the scheduling algorithm. Depending upon the breadth of his understanding of the algorithm, he can enhance the performance of the system by manipulating the algorithm to meet his requirements.

The remainder of this section on message scheduling considers the scheduling algorithm in these topics:

- Message class and region class
- Load balancing
- Selection priorities
- Processing limits
- Application program output limits
- Multiple and single segment messages
- Multiple and single message mode
- Response mode
- Non-update transaction processing
- Conversational attribute
- Data base processing intent
- Processing intent propagation
- Application program abnormal termination

- Contention for resources
- Control block buffers -- PSB and DMB

Message Class and Region Class

Each message is assigned a class at system definition time. This class assignment determines into which message region an application program is loaded. When the IMS/VS message regions are started, they are assigned from one to four message classes. When a message region is assigned more than one class, the scheduling algorithm treats the first class specified as the highest priority class, and each succeeding class as a lower priority class.

If more than one class is specified, the message selection process is handled as follows. The first class specified is scanned, in transaction priority sequence, for waiting messages. If there are no messages waiting for the first class, the second and following classes are also scanned in priority sequence. If there are messages waiting in the first class, the highest priority message is selected for scheduling. If, for external reasons (for example, program or transaction stopped by master terminal operator), this message is not schedulable, the next message of equal or lower priority in that class, or the highest priority message in the next class, is selected for scheduling. If the highest priority message in the first class is not schedulable for internal reasons (data base intent or no more space in PSB pool or DMB pool to bring in needed blocks), the scheduling option of the transaction indicates the type of scheduling algorithm that is used. The scheduling option is specified at system definition by the TRANSACT macro. The options are:

1. Schedule only transactions of equal or higher priority in the selected class.
2. Schedule higher priority transactions in the selected class.
3. Schedule any transaction in the selected class.
4. Skip to the next class and attempt to schedule the highest priority transaction in that class.

Note that these scheduling options are specified for each transaction; therefore, each attempt to schedule a different transaction may change the algorithm, if the algorithms are different for transactions within the same class.

Message region class assignments and transaction class assignments can be modified at execution time to control message throughput.

If multiple message regions process the same message class and a data base processing intent conflict occurs, the highest priority transactions scheduled against a data base will not necessarily be processed before processing lower priority transactions scheduled against the same data base. If you desire to process all higher priority transactions scheduled against a data base before processing any lower priority transactions, no processing limit should be specified for the higher priority transactions, or only one message region should process that message class.

Load Balancing

Load balancing is the facility to schedule the same application program and the same transaction in multiple message regions. The application program and the transaction are designated for parallel scheduling at system definition time. The application must be designated as a parallel scheduled application before any transaction processed by that application will be scheduled in multiple regions.

When an SMB is available to be scheduled but is already scheduled in another region, it is checked to determine whether it can be parallel scheduled. The PARMLIM value of the TRANSACT macro specifies the number of messages that should be enqueued before another region is scheduled. This value is multiplied by the number of regions already scheduled for this transaction. If the result is less than the number of messages enqueued, another region is scheduled for the transaction. If the region is unschedulable for internal reasons (data base intent), the next transaction within the class is scheduled. No cutoff priority will be set as the transaction is already scheduled within IMS/VS.

Selection Priorities

When more than one transaction of a given type is waiting to be scheduled, the specified transaction scheduling priority determines which transaction code is selected next. It does not determine which is actually scheduled. Only the tests of the transaction's readiness for scheduling, which occur after selection, determine if the transaction queue is allocated to an application program. The selection priorities are useful for influencing the response time to input transactions and for load balancing. Two priorities can be specified. One is called the "normal priority"; the other, "limit priority." Related to the normal and limit priorities is a "limit count." When the number of input messages of a specific transaction type waiting to be scheduled is equal to or greater than the limit count, the normal priority is reset to the limit priority value.

The priority of a transaction code causes it to be selected either before or after other transaction codes. If there are multiple transaction codes at the same priority, they are selected on a first-in/first-out basis. However, if there are multiple transaction codes at the same priority and the same class, with many messages already enqueued for each transaction code, the individual transaction codes will be selected on a first-in/first-out basis, but the different messages may not be selected in the same sequence in which they were entered. For example, A, B, and C are transaction codes with processing limit counts of 1. These codes are entered in the sequence ABCBACCAC. The sequence in which they are selected is ABCABCACC. An example of the typical use of selection priorities can be found under the topic "Message Scheduling Facility" in the IMS/VS General Information Manual.

Another effective way to utilize the selection priorities is to declare a normal priority value of zero. Zero priority is a null or "not eligible for scheduling" level. Messages accumulate until the processing limit count is reached; at this point the limit priority is effected and scheduling occurs. This technique is called "batching messages."

The normal priority is not restored until all messages enqueued on the transaction code have been processed. It is possible that more messages will be added to the queue while the transaction is waiting or in process at the limit priority. Note that the priorities are selection priorities, not execution priorities. Once a transaction has been selected for scheduling, the selection priorities have no influence until it is again recognized to be waiting for scheduling.

The effectiveness of the selection priority assignments is related to how frequently the selection process occurs. The following section discusses a means of influencing this.

Processing Limits

Through the establishment of processing limits, the frequency with which scheduling selection occurs can be influenced. In the time between schedulings, processing is going on in the message regions. Meanwhile, messages are accumulating in the message queues. As they accumulate, the interactive effects introduced by new message types, and the changing of selection priorities, are rearranging the order of waiting transaction codes. Conceivably, while a large queue of messages is being processed, important activity assigned to a high priority transaction code is waiting.

When the program processes a large queue of messages and updates data base segments, other application programs wishing to access an updated segment are placed into a wait state. The length of time that the other application programs have to wait depends on whether the updating program is processing its queue in multiple or single message mode.

To allow controlled re-entry to the message scheduling selection process, a processing limit count can be specified for each transaction code. Each time a scheduled (processing) program requests a new message, the limit count is checked. When the number of requests exceeds the limit count, the application program is told by the control program that there are no more messages. In fact, there may be more. When the application program is told there are no more messages, it completes its processing and returns the transaction queue to its proper place among others waiting to be scheduled. If it is returned to a priority level where other transaction codes are waiting, it assumes an eligibility for selection below them, even though all have the same numeric priority.

Application Program Output Limits

By establishing program output limits during system definition, the IMS/VS user can influence the number and the size of the output segments from the application program to the message queues. When an application program exceeds the previously-specified limits, a status code is returned indicating an error. Any further attempt by the application program to exceed the limits results in abnormal termination.

Abnormal terminations can be prevented by checking the number and the size of application program segments. This process of checking eliminates IMS/VS system abnormal terminations that occur when application programs loop while inserting messages or segments into the message queues, or when they inadvertently insert segments of invalid lengths.

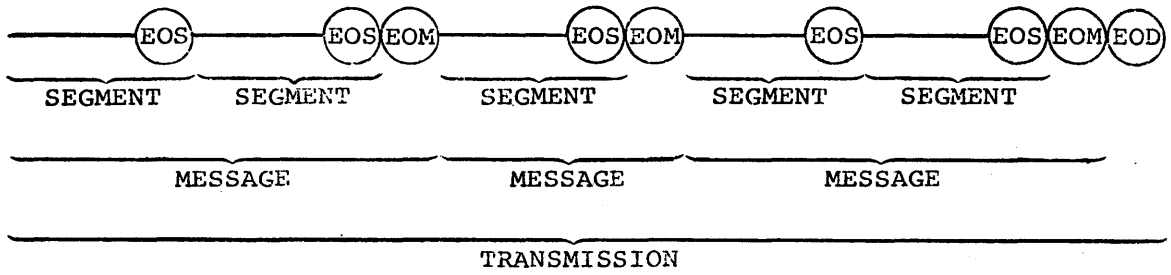
Multiple and Single Segment Messages

A message, in the most general sense, is a finite sequence of transmitted symbols. In the context of IMS/VS, this is called a transmission. A transmission is terminated by a logical condition called end-of-data (EOD). The transmission is partitioned into sequences of symbols, called messages, by an end-of-message (EOM) symbol.

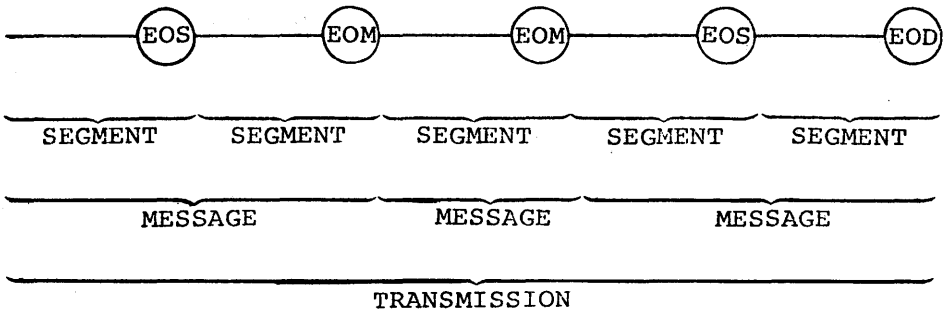
A message is partitioned into smaller sequences of symbols, called segments, by an end-of-segment (EOS) symbol. There are only three valid combinations of the conditions represented by EOS, EOM, and EOD. They are:

Condition	Represents
EOS	EOS
EOM	EOS/EOM
EOD	EOS/EOM/EOD

In the most complex case, a transmission containing several multisegment and single segment messages would look like this:



Using the simple symbols, the same transmission would be represented like this:



The assignment of values to the symbols that represent the conditions end-of-transmission, message, and segment is not significant to this discussion. However, it is significant that the conditions can be represented by more than one transmitted data character. Most key-driven terminals generate only one character per keystroke. Thus, it may be necessary for the terminal operator to perform more than one manual operation to signify EOS or EOM.

For input transmission, detection of the EOM condition by IMS/VS indicates that a complete message has been received. A complete message is eligible for scheduling, and ultimately processing, by the application program to which it is destined.

At the time the first EOS condition, or the first EOS following an EOM, is detected, IMS/VS examines the text of the preceding segment. Within the extent of the segment there must appear a valid transaction code, predefined through use of the TRANSACT macro instruction. One of the attributes that can be assigned to a transaction code specifies whether a message is multiple or single segment. The effect of this specification is null if multiple segment specification is selected.

If single segment specification is selected, the system equates the EOS condition to an EOM condition. Thus, each segment is treated as a complete message.

The primary concerns when selecting the multiple or single segment attribute are human factors, application requirements, and physical terminal characteristics. For example, let us assume the following:

- An application requires only single segment entry.
- Most users enter data from a key-driven terminal.
- The terminal has an automatic character generation feature (EOS after pressing carriage return).

Then, the selection of multiple segment as an attribute of a transaction code would require an additional keystroke to signify EOM or EOD.

Another example: assume all of the preceding conditions are true except that the line length of the data to be entered exceeds the single-line capacity of the terminal. The appropriate and more natural selection is multiple line. However, if the application were one with very high usage, the overhead of processing multiple line messages might be sufficient to justify adjustment of the short message buffer length. The operational characteristics of transaction entry would be altered. Using the same terminal with the special EOS generation feature disabled, the operator enters the first line, presses the carriage return, enters the remaining data on the second line, then presses the EOS key. The result is a single segment message.

When the Message Format Service (MFS) is used to format input, the relationship between the segments described above and the actual message segment created by MFS is user-defined.

When MFS is used to edit input, the end of input for a given message is signalled by:

- EOM or EOD
- Completion of processing for all defined DFLDs

At the end of input for a message, MFS presents the completed message segments to the DC component of IMS/VS; this component looks for a destination name. If the destination is a transaction defined as a single segment and more than one segment has been created by MFS, an error message is sent to the input terminal.

For more information on MFS, see the section in this chapter called "Message Format Service."

Consult the IMS/VS Operator's Reference Manual for more information about the various terminals supported by IMS/VS.

MULTIPLE AND SINGLE MESSAGE MODE

The message mode attribute of a transaction code is used to notify the system of the manner in which an application program views the transactions it processes. Single mode indicates that each message is processed independently of all other messages that are read. Multiple mode indicates that all the messages of this transaction code, read during a given scheduling of the application program, are to be considered as related to one another. For example, the application program accumulates control totals that are written out only at program termination. It does not affect the message selection criteria of the

scheduling algorithm. It does, however, affect the amount of main storage required by program isolation, message processing throughput, and, potentially, the integrity of the data bases used by user programs. If multiple mode is selected, there is potential for greater throughput. Multiple mode results in fewer system-generated I/O operations, and less system time per message, when more than one message is processed per application program scheduling. However, all data base resources modified in any way by the user remain enqueued until user program termination.

If more than one message is processed per scheduling of a user program, large storage requirements for IMS/VS program isolation could result. If program isolation enqueue chains become very long, throughput is adversely affected. Also, if a program must be terminated and rolled back by IMS/VS to break a data base deadlock situation, the backout and reprocess time is increased in proportion to the number of messages processed. In addition, very long backout chains in the Dynamic Log may require extra I/O operations and increase the possibility of exceeding the capacity of the data set. If this happens, application activity is quiesced.

Internally, the difference in single and multiple mode transaction processing is related to the frequency at which pending data base buffers are written. In single mode, all pending data base buffers are written each time a new message is requested by the application program. These operations are performed regardless of the value assigned as a processing limit count. Multiple mode defers buffer write until the application program terminates, unless a CHKP call was issued by the application program. A CHKP call causes all buffers modified by the user to be written at the time the call is issued.

An additional consideration is imposed for program isolation. When the transaction code causes data base updates, the enqueue of the updated segments is held until the point at which the program can be restarted without having to reprocess those updates. In single mode, this point is reached each time a new message is requested by the application program. Multiple mode defers reaching this point until the application program terminates. This causes more segments to be enqueued, and the enqueued segments to be held longer. Other programs needing access to the enqueued segments are delayed, and the chance of deadlock is increased. Since message response is also held, and not sent to its destination until the same point is reached, the choice of multiple mode processing can significantly increase terminal response time. For information on the use of the checkpoint call in conjunction with multiple mode processing, see the IMS/VS Application Programming Reference Manual.

Response Mode

Response mode describes a connection between IMS/VS and a communication line or terminal that can occur only for certain terminal types under condition specified during IMS/VS system definition. When response mode is in effect, IMS/VS will not accept any input from the communication line or terminal until it has sent the output response to the previous input.

Response mode is in effect from the time the last segment of a transaction has been received by IMS/VS until the application program inserts a response to the response PCB, which is usually the I/O PCB. When more than one message is inserted using the response PCB, response mode is reset when the first message using the response PCB is transmitted. Any remaining messages issued by the application program are treated as non-response application program output. If the application program does not produce a response, the terminal remains

in response mode and master terminal intervention is required to restore proper terminal operation.

The terminal types that can be defined (TERMINAL or TYPE macro) to operate in response mode are the: 1050, 2740, 2741, CPT-TWX, 3270, 3600, 3767, 3770, and 3790. The 3790 is forced to operate in response mode. The others may be defined as: "forced" -- always operating in response mode, "negated" -- never operating in response mode or "transaction dependent" -- operating in response mode only when a transaction defined by the TRANSACT macro as a response mode transaction is entered.

Response mode for the 1050, 2740, 2741, and CPT-TWX terminals stops all operations on the communication line and is referred to specifically as "line response mode." Response mode for the 3270, 3600, 3767, and 3790 stops all operations on the terminal and is referred to specifically as "terminal response mode."

Design Considerations: Before response mode definitions are specified for terminals and transaction codes, consider the following:

- On a switched line, response mode enforces synchronization terminal operation.
- In response mode, terminal operators can only enter one transaction at a time and must wait for a reply before entering another transaction.
- For a terminal defined as "transaction dependent," transaction that are not defined as response mode transactions permit entry of additional input without waiting for a reply from the previous transaction.
- Master terminal intervention is required when an application program fails to respond to a transaction from a terminal in reponse mode.
- In some environments, specifying "forced" response mode for some terminal types may result in fewer line operations and improved performance.
- For some terminal types (2740 without Station Control Feature, 2741, and CPT-TWX), a specification of response mode prevents the operator from having to enter a null message to receive the response to the last input.

For BTAM or VTAM terminals don't use the following specifications:

- The combined specification of FORCRESP and NPGDEL.
- The combined specification of TRANRESP and NPGDEL if MSGTYPE=RESPONSE is specified for the TRANSACT macro.

The first specification is not recommended because NPGDEL prevents the current output message from being dequeued at the time of input; the second specification prevents the terminal response from being reset until the current output message is dequeued.

Further information on terminal operations using response mode is contained in IMS/VS Operator's Reference Manual, IMS/VS Installation Guide, and IMS/VS Message Format Service User's Guide. IMS/VS Advanced Function for Communications contains specific information about 3600 and 3790 operations.

Non-Update Transaction Processing

A transaction code which does not cause an update to a data base can be so defined to the system. This allows a program that handles multiple transaction codes, and only updates the data base for a subset of these transactions, to be scheduled concurrently with other update programs when it is to process a transaction that does not cause an update.

Transactions must be defined as non-update transactions when entered from non-error-checked terminals supported by IMS/VS. They are also non-update for entry by switched terminals that are signed on for INQUIRY purposes.

When a transaction is defined as non-update, the associated application program is prevented from updating the data base. This is the case even though the processing option in the PSB specifies update capability.

Conversational Attribute

Scratch pad areas (SPAs) are work areas through which an application program and a terminal establish a quasi-interactive relationship called a conversation. That is, continuity is established with the terminal operator, by the application, across multiple message entry and response sequences. The conversation can be suspended, reinstated, or terminated, by the terminal operator, through the command language. A conversation is normally terminated by the application, not the terminal operator.

The system maintains scratch pad areas on a direct access data set or in main storage. Residency is specifiable by transaction code. The choice of main storage or direct access residency influences the response time for transactions that have the conversational attribute.

Although the system can operate with one maximum size main storage SPA, or one direct access SPA, then only one conversation can be in process at a time. If a high percentage of transaction processing is conversational, a similar number of SPAs should be specified in the system definition SPAREA macro statement. If a conversational transaction code is entered, and all SPAs are in use, that transaction is rejected by the system. An insufficient number of available SPAs could result in terminal user dissatisfaction.

If a transaction code has the conversational attribute, it can have effects on overall system performance. The choice of main storage versus direct access residency affects, not only system performance, but also the response characteristics of the conversational transaction. The following discussion elaborates on the potential effects of buffer fragmentation and the relative throughput and response characteristics of conversational transactions.

Main storage resident SPAs, and read/write space for direct access resident SPAs, are acquired from the buffer pool when a conversation is initiated by entry of a conversational transaction code. Main storage is retained throughout subsequent exchanges between the terminal and destination applications. It is released upon termination of the conversation. Since the main storage SPA buffer space is retained over a relatively long period of time, its potential ability to fragment the buffer pool is relatively high. Fragmentation of the buffer pool can cause processing delays in terminal I/O service and in the initiation of other conversations. However, since the SPA is both main storage resident and dedicated for the life of the conversation, response and throughput are significantly improved.

If the SPA is on direct access, space is initially acquired from the buffer pool at the same time as for a main storage resident SPA. However, it is retained only long enough to write to direct access. SPA buffer space is freed. Space is re-acquired when the application program returns the SPA, and is freed as soon as the SPA is rewritten to a direct access device. The use of direct access SPAs decreases the possibility of extended delays introduced by buffer fragmentation. However, because buffer requests are made during the time an application is active in the message processing region, any delay due to lack of buffer space directly affects throughput. In addition, since the application must wait for the SPA to be written out, overall processing time for each transaction is increased and response time extended.

To enhance performance for conversational processing, conversational transactions can be defined with fixed-length SPAs. For such transactions, the main storage SPA uses only the fixed length that was defined. For direct access resident SPAs, the defined maximum length is always used, however, performance is increased on program-to-program switches because the direct access SPA is not updated.

Fixed-length SPAs defined during conversation initialization must remain in effect for the duration of that conversation. For conversations whose first transaction code defined fixed-length SPAs, all successive transactions used as destination applications in the same conversation must also be defined with fixed-length SPAs of the same length. If not, a status code indicating an error is sent to the calling application. If the Multiple Systems Coupling feature is used and conversations will run in a system that is not the input terminal system, fixed length SPAs must be used. For more information about the Multiple Systems Coupling feature, see Chapter 5 of this publication.

An additional performance enhancement for conversational processing is the automatic compaction of all SPAs for queuing and logging. All blanks and X'00's are eliminated for queuing and logging, and the application program receives the unpacked SPA.

Data Base Processing Intent

A factor that can significantly increase the overhead of the scheduling process is the intent of an application toward the data bases it uses. Intent is determined by examining the intent list associated with the PSB to be scheduled. At initial selection, this process involves bringing the intent list into the control region. The location of the intent list is maintained in the PSB directory. If the analysis of the intent list indicates a conflict in data base usage with a currently scheduled transaction, the scheduling process must select another transaction code and try again.

There are several intent levels that can be stated for a given segment type. The list below shows the level of intent, how it may be stated for the PSB Generation utility program, and the code that is used in the decision tables that follow:

N = No sensitivity -- segment type not referenced.

R = Express read-only -- segment type referenced -- PROCOPT=GO.

Note: See "Processing Intent Specifications" in this chapter for an explanation of the various processing options (PROCOPTs).

G = Retrieve segment type referenced -- PROCOPT=G or K.

U = Update -- segment type referenced -- PROCOPT=A, I, R, or D or segment type has propagated intent.

E = Exclusive use -- segment type referenced -- PROCOPT=E.

If exclusive use is specified for a program, that program will not be scheduled concurrently with any other program that is sensitive to the same segment types.

The following decision table shows which programs can be scheduled concurrently.

	Intent of Currently Executing Program				
Intent of Program Being Scheduled	E	U	G	R	N
E	X	X	X	X	
U	X				
G	X				
R	X				
N					

X Indicates Conflicting Actions When Transaction Scheduling Is Attempted

Since exclusive intent does not allow a program to be scheduled while programs sensitive to those segments are operating, no dynamic serialization is done via the enqueue/dequeue facility.

Conflicting actions occur only if the same segment type is declared "Exclusive Use" by at least one of two programs intending to reference the segment type.

A PSB that contains a PCB for a SHISAM segment that has delete sensitivity will be scheduled exclusively. This is because the method used by IMS to ensure program isolation cannot be used for SHISAM deletes. Since there is no delete flag, a VSAM erase must be done to delete the segment, and since IMS/VS uses relative byte addresses as the identification of a segment, there is no way to prevent another user from inserting a segment with the same key prior to the time the program which did the delete reaches a sync point.

Unless the PSB for the program being scheduled is currently resident in the PSB buffer pool, determining schedulability involves a direct access I/O operation.

Exclusive intent may be required for long running BMP programs that do not issue checkpoint call because of the excessively large size of the enqueue/dequeue table.

An exception to the use of the enqueue/dequeue facility to provide program isolation is accomplished by the use of the GO option; this allows programs to access segments without an enqueue being done for those segments. When this occurs, a program can retrieve segments which have been altered or modified by programs which are still active and while the changes are subject to being backed out. See the IMS/VS Utilities Reference Manual for a detailed explanation of the GO option.

Processing Intent Specifications

The only processing intent that affects the schedulability of IMS/VS programs is exclusive intent. However, if the data base portion of IMS/VS is utilized by CICS, program isolation is not operative and the following information on scheduling intent applies. If you do not use CICS, skip this section.

The processing option parameter (PROCOPT) of the PCB and SENSEG statements of a PSB generation determines the processing intent an application program has on data. The scheduling options are:

G = Get function.

I = Insert function.

R = Replace function.

D = Delete function.

A = All, includes the previous four functions.

E = Used in conjunction with the previous five functions, and specifies that an application program has exclusive use of the data base or segment specified.

K = Indicates key sensitivity only.

Scheduling Intent Types: There are three scheduling intents used to determine the schedulability of an application program. Exclusive intent prohibits the concurrent scheduling of any programs that reference the same segment types as the program that has specified exclusive use. Update intent allows any number of programs that reference the same segment types for read only to be scheduled with the updating program. All programs that reference the same segment type for update intent must be scheduled serially. Read only intent allows the program to be scheduled with any number of other read only users and one update intent program. If a segment has more than one intent type as the result of multiple references or intent propagation, the most restrictive use is set. These intent types are associated with the aforementioned processing intent specifications in the following manner:

- Read Only Intent

  This intent is set for any segment that specified PROCOPT=G or PROCOPT=K on the associated SENSEG statement. In addition, this intent is propagated to all segments that are required to obtain the information necessary to satisfy a DL/I call. For example, a logical child segment is requested in a call, and the logical parent's key was specified as "VIRTUAL." All segments that must be retrieved to construct the logical parent's concatenated key have read only intent set. The extent of propagation is discussed below.

- Update Intent

  This intent is set if the associated SENSEG statement specified PROCOPT=I, R, or D. Update intent is set for the associated segment regardless of any key sensitivity specification, either explicitly or implicitly specified. This intent can be propagated to other segments in this data base or related data bases. The amount of propagation is determined by the processing options specified, the data base organization, the pointer combinations used, and the SEGM statement RULES options chosen at physical DBDGEN time. The implications and extent of intent propagation are discussed below.

- Exclusive Intent

  This intent is set if the associated SENSEG statement specified PROCOPT=E and the segment does not have key sensitivity. Key sensitivity can be specified on the associated SENSEG statement, using the KEY/DATA option of the SOURCE operand in a logical DBD, or by omitting the specification of the complete concatenated segment in a logical DBD. This occurs when you specify the logical child segment and not the logical or physical parent in the concatenated segment definition. There is no propagation of the E option. Note that the specification of both PROCOPT=E and K on a SENSEG statement causes the exclusive (E) option to be ignored.

Implications and Extent of Intent Propagation: As discussed earlier in this section, the implications of intent propagation depend on many factors. Some of these factors are physical organization, pointer combinations, processing options, segment rules, and logical relationships. The following paragraphs explain their effect on scheduling concurrency as they relate to typical data base structures. Each processing option is discussed in a separate section. Keep in mind the fact that if a segment has more than one processing intent type (as the result of explicit or implicit processing options) the most restrictive intent is used.

- Get Processing Option

  A segment using PROCOPT=G or K causes read only intent to be set for that segment. In addition, read only intent is propagated to all segments that are used to complete a GET type call. Sensitivity to a logical child segment implies sensitivity to its associated logical or physical parent. In either case, read only intent is propagated to the associated parent segment, and all its parent segments, in a direct line upward to the root segment.

- Replace Processing Option

  A segment using PROCOPT=R causes update intent to be set for that segment. If the segment is part of a concatenated segment definition, and the logical parent/physical parent part of the concatenation can be replaced, it has update intent propagated to it. No other propagation of intent occurs.

- Insert Processing Option

Insert intent propagation is based on two basic rules. These rules do not apply if Program Isolation is operative.

1. Programs that separately insert a physical parent segment and its physical child are not scheduled concurrently. If the program inserting the physical child terminates first, and if IMS/VS abnormally terminates before the program inserting the physical parent terminates, the physical parent segment is backed

out of the data base by /ERESTART processing, leaving a dangling physical child segment.

2. Programs that insert child/logical parent concatenated segments involving the same logical parent are not scheduled concurrently. If the insert rule of the logical parent is either virtual or logical. The physical insert rule prohibits inserting the logical parent by means of a concatenated segment. Only the logical child need be inserted.

Update intent is set for the segment type designated by PROCOPT=I in the SENSEG statement of the PCB, or for all the segments designated by SENSEG statements when the PROCOPT=I is coded in the PCB statement (rule 1). Update intent is propagated to all the immediate children (down one level) from the designated segment because of rule 2. If the designated segment is a logical child, the update intent is propagated to the logical parent segment as specified by rule 3, and to the immediate children of the logical parent as specified by rule 2. If the insert rule of the logical parent is physical, then one program per logical child segment type can be concurrently scheduled.

The first variable that affects insert intent is the data base organization. Since segments in a HISAM data base are hierarchically related by physical juxtaposition, a segment insert can cause other segments in the data base record to shift physical location. However, since a data base record can reside in several separate data set groups, only the data set group containing the inserted segment type is affected. The rule is: all segments residing in the same HISAM data set group as the segment type to be inserted have update intent propagated to them.

The second variable that affects insert intent is the printer combinations specified for segments residing in HD type data base organizations. When physical child pointers are selected to address the designated segment, the physical parent has a different pointer for each of its children that concurrent programs maintain separately. However, if the choice is hierarchical pointers to address the designated segment, the physical parent addresses all of its children by a single hierarchical pointer chain. Concurrent update programs for the different physical children, therefore, violate rule 1. When the immediate physical parent segment has hierarchical pointers, the data structure is scanned in an upward direction until a parent segment is found that uses physical child pointers, or until a root segment is encountered. The immediately previous physical child segment of the parent segment so located, and all dependent segment types of that immediate physical child segment, have update intent propagated to them.

- Delete Processing Option

  The propagation of update intent from segments designated with PROCOPT=D is based on the physical child's dependence on the physical parent. If the physical parent is deleted, its physical children must also be deleted. Therefore, beginning at the designated segment type, update intent is propagated to all its physical dependent segment types and to their physical dependents, down to the lowest level of the data structure. When a segment that is a logical child is encountered in the downward scan, its logical parent's delete rule is determined. If the rule is virtual, update intent is propagated to the logical parent and its physical dependents. When a segment type that is a logical parent is encountered in the downward scan, the delete rules of its logical children and their physical parents are determined.

If the delete rule is virtual and/or bi-directional virtual, then update intent is propagated to the logical child and to its physical dependents, and/or to the physical parent and its physical dependents. Since the propagation is downward, all segments in the downward scan are inspected for logical relationships. As they are encountered, the logical child/logical parent/physical parent segment types are processed in the same manner as the original segment type. Deletion of the parent requires deletion of all physical dependents.

When the immediate physical parent of the designated segment has hierarchical pointers, the data structure is scanned in an upward direction until a parent segment is found that is a root segment, or a parent segment is found that is pointed to by physical child pointers. That segment type found, along with all its dependent segment types, have update intent propagated to them.

Application Program Abnormal Termination

Upon abnormal termination of a message or batch-message processing application program, internal commands are issued to prevent rescheduling. These commands are the equivalent of /STOP. They prevent continued use of the program and the transaction code in process at the time of abnormal termination. The master terminal operator can restart either or both stopped resources. At the time abnormal termination occurs, a message is issued to the master terminal and to the input terminal that identifies the application program, transaction code, and input terminal. It also contains the system and user completion codes. In addition, the first segment of the input transaction, in process by the application at abnormal termination, is displayed on the master terminal.

The stop action is performed automatically. Even though a message is issued, its occurrence could go unnoticed by the master terminal operator. Such a failure, involving a major application that serves many transaction codes, could have adverse effects on system performance.

The potential effects of commands entered from any terminal, that cause unavailability of scheduling resources, are severe. Operators should be instructed to display system status frequently. If a program that terminated abnormally inserted any message segments, they are transmitted, although the message may not be logically complete.

Program isolation dynamically backs out data base updates, and cancels message output made by application programs that terminate abnormally. To avoid the adverse effect that this backout can have on programs concurrently processing in the system, data base segments that have been changed are enqueued using the IMS/VS enqueue/dequeue routine. This routine ensures that no other programs can access the changed data base segments until either the application program that requested the change completes successfully, or terminates abnormally; and until all changed segments are restored to their original states.

Program isolation ensures that a dynamic log (IMSVS.DBLLOG) is maintained. The dynamic log is a sequential data set, on direct access storage, written with OSAM to facilitate following chains through it. All the log records created because of a given user program are back-chained, with the chain anchor in the PST to which the program is attached. The chain pointer is the block number and the offset within the block. When a synchronization point is reached, or if the program terminates successfully, the anchor in the PST is reset to zero. If the program terminates with an abnormal termination, the data base changes are backed out to the last synchronization point specified by the MODE parameter of the scheduled transaction code. If it is a

batch-message processing program that does not reference a transaction code, or whose transaction specified MODE=MULT, it is backed out to its schedule point, or to the last checkpoint, whichever is most recent. The backout is accomplished by passing the data base log records, that were dynamically logged and chained, from the PST to the data base backout module.

A synchronization point is defined as the point at which an application program can be restarted.

		SYNC POINT IS LATEST ACTION		
		Msg GU	CHKP	SCHEDULING
If MPP	Transaction MODE=Single	X	X	X
	Transaction MODE=Multiple		X	X
If BMP	Transaction MODE=Single	X	X	X
	Transaction MODE=Multiple or No Transaction		X	X

All output messages inserted by an application program, with the exception of messages inserted to alternate PCBs that have been designated to have the Express Message feature, are enqueued to a temporary destination associated with the PST. The Express Message feature is a PSB Generation option for an alternate PCB. It specifies that messages sent to this PCB are not to be backed out if the application program terminates abnormally. When the application program successfully reaches a synchronization point, the program's output messages are transferred from their temporary destinations to their final destinations. If the application program abnormally terminates, all messages enqueued to temporary destinations are deleted and cancelled. Those messages inserted to the alternate PCBs that have the Express Message feature were never enqueued to the temporary destination and cannot be cancelled.

Program Isolation provides a call function (ROLL) through which an application program can remove the effect of its processing. Issuance of this call function abnormally terminates the application program task with an indicative completion code. Voluntary abnormal termination using this call function does not cause the program and transaction to be stopped, nor does it produce a storage dump.

One other application program abnormal termination situation is possible. Since data base updates are isolated by the program isolation

enqueue/dequeue facility, the possibility of deadlock situations can arise. These situations are avoided by selecting one of the deadlocked programs for abnormal termination, with a special code that causes the program's data base updates and unsent message output to be backed out. The transaction input that was being processed by the program is retained, and the program is rescheduled.

Since deadlock is usually an exception, it can be treated in this manner to make deadlock detection and correction transparent to the application program and the terminal operator. The program to be abnormally terminateded and rescheduled in a deadlock situation is determined based on the decision table that follows:

		And if the waiting program which completes the deadlock circuit				
If calling program whose request will cause a deadlock		Is a MSG program with		Is a BMP program with		Is multiple programs
		Single Mode Tran	Mult Mode Tran	Single Mode Tran	Mult Mode or no Tran	
Is BMP program with	Mult Mode or no Tran	DO B	DO B	DO B		
	Sngl Mode	DO B	DO B			
Is MSG program with	Mult Mode Tran	DO B		DO A		
	Sngl Mode Tran					

A = ABEND the calling program

B = ABEND the program with which the calling program would deadlock

In IMS/VS installations running under OS/VS1, after any abnormal termination in an MPP (for example, application program abnormal termination, program isolation deadlock, or ROLL call), the MPP may not be able to reclaim all the storage used by the application for future scheduling. A system abend may eventually occur because of insufficient storage and the MPP will be terminated by IMS/VS, after two consecutive GETMAIN failures are detected, to release unusable storage in the region.

Contention for Resources

Contention for the use of resources affects scheduling in more ways than have been discussed. The material that appears here is based on

the specifications for IMS/VS. If more detailed information about how the IMS/VS system works is needed, consult the IMS/VS Program Logic Manual.

Control Block Buffer Pools -- PSB and DMB

Control block pools are maintained in the IMS/VS control region for program specification blocks (PSB) and data management blocks (DMB). Each buffer pool must be at least as large as the largest control block it will contain, plus the next successively larger block, for each additional processing region concurrently active.

The IMSVS.ACBLIB data set must contain control blocks for all application programs (PSBs) and all data bases (DMBs) referenced by the application programs. When an application program is to be scheduled, the PSB and DMB pools are examined to determine which control blocks must be brought into main storage. If all required blocks are resident, the program is scheduled. If required control blocks are not resident, the applicable pool is searched for space to hold the block. If space is found, the block is loaded and the program is scheduled. If the pool does not contain the required free space, the blocks currently resident are examined to determine which unused blocks can be removed. When the selection process is complete, any open data bases referenced by the unused blocks are closed, and the space is released for use by the new control blocks. The new control blocks are then loaded, and the application program is scheduled.

Excessive loading of control blocks can have a severe impact on performance. If possible, the DMB pool should contain enough space to hold all DMBs used with online data bases. This reduces the number of OS/VS opens and closes, and their impact on system performance.

DATA BASES

System definition requires that data bases to be used in the DB/DC system configuration be identified. The positive declaration of data base names enables the system to limit the domain of the online control program to only some specific subset of all installation data bases.

BATCH CHECKPOINT/RESTART

The batch checkpoint facility provides batch-message programs with the means of synchronizing checkpoints taken of their environment with the IMS/VS log tape. It also enhances the integrity of data bases updated by batch-message programs, by allowing the restart facility to back out data base changes being made by such programs at the time of a system failure. If a batch-message processing program abnormally terminatess, program isolation ensures that backout procedures occur automatically. The data to be backed out is the data base change records logged since the last synchronization point created by a CHKP call. The time lag is significantly less using program isolation with batch checkpoint/restart, than if the data base had to be stopped and taken offline for batch backout.

The batch checkpoint facility is implemented by the use of the IMS/VS checkpoint (CHKP) system service call from the application program. This call is used to indicate a synchronization point at which data base updates can be restarted. The actual checkpointing of the batch program environment, and the routine used to restart it, are at the option of the user. If OS/VS checkpoint is to be used, the user must request, as part of the DL/I CHKP call, that the system take the checkpoint.

Note: The checkpoint ID table, as referenced below, is used to coordinate the checkpoints on the IMS/VS system log with the activity of any batch-message regions, for the purpose of emergency restart. This table also contains the ID and serial number of the last startup or shutdown checkpoint.

For batch-message programs (not message-driven):

A non-message driven BMP program functions like a batch program, but receives data base service like an MPP. No identifiable (explicit) synchronization point exists until the program issues the CHKP call.

1. Optionally, an OS/VS checkpoint of the user's region is taken.
2. Altered data base buffers are written.
3. The checkpoint ID, supplied in the CHKP call, is written to the log tape.
4. The checkpoint-ID table is updated, for use in subsequent emergency restarts.
5. The dynamic log is updated by releasing all change records prior to the current synchronization point.

For batch-message programs (message-driven):

BMP programs that access the message queue via the I/O PCB, have a defined (implicit) synchronization point established by the MODE= parameter in the TRANSACT macro. To IMS/VS, the BMP program looks like an MPP. If MODE=MULT is selected, end-of-job is the natural synchronization point. BMP programs can issue the CHKP call to cause an explicit synchronization point, and define a point from which restart can be performed. Care must be taken to ensure that the dynamic log buffers do not become full because the CHKP calls are too infrequent. All output messages, that are not destined to express alternate PCBs, are held until a synchronization point occurs. All input messages since the last CHKP are reprocessable. The following general events occur in this type of BMP:

1. Optionally, an OS/VS checkpoint of the user's region is taken.
2. Altered data base buffers are written.
3. The checkpoint ID, supplied in the CHKP call, is written to the log tape.
4. The checkpoint-ID table is updated, for use in subsequent emergency restarts.
5. The dynamic log change records for the calling BMP are released.
6. Output messages to all TP PCBs are sent, and input messages are dequeued.
7. A GU to the I/O PCB is internally generated for the application program.

If MODE=SNGL is specified on the TRANSACT macro instruction, a natural synchronizatcn point exists at each GU on the I/O PCB. Functions similar to those above are performed by IMS/VS; however, the user does not have to execute a CHKP call because the GU causes the necessary synchronization points.

Instead of the OS/VS Checkpoint/Restart option, the user can specify the IMS/VS Expanded Checkpoint/Restart facility. This consists of a restart call (function code XRST) and optional parameters on the CHKP call. If used, the XRST call is the first call to IMS/VS issued by the user program. If a restart is not in progress, the XRST call is effectively a NOP.

The issuance of an XRST call causes the following action to be taken for subsequent CHKP calls issued by the program:

1. Optionally, user specified areas, that is, application variables, control tables, and position information for non-IMS/VS data sets, are recorded on the IMS/VS log.

2. The fully qualified key of the last record processed by the program on each IMS/VS data base is recorded on the log.

3. The functions of the standard CHKP call are performed, except that the OS/VS checkpoint of the user's region is not taken. The user has the option of using OS/VS Checkpoint/Restart, the IMS/VS restart (XRST call), or neither, but not both.

For message processing programs:

The CHKP call functions exactly as a message GU for a single mode program, allowing a program operating in multiple mode to control the spacing of its synchronization points.

In the case of a checkpoint FREEZE or DUMPQ shutdown, IMS/VS waits for any batch-message programs that are processing to issue a CHKP call, before proceeding with the shutdown. This action makes it possible to identify the point at which the batch-message program should be restarted.

In the case of a PURGE shutdown, IMS/VS waits for batch-message programs to terminate before proceeding with the shutdown.

The log record containing the checkpoint ID is used by emergency restart as follows:

Using the checkpoint-ID table, emergency restart determines, and identifies to the operator, the point on the log where restart processing is to begin in order to back out incomplete updates made by the message and batch-message programs processing at the time of the system failure. It initiates restart processing from that point. If backout is successful, the CHKP ID from which each BMP can be restarted is identified to the operator.

Transactions, partially processed by message processing programs at the time of system failure, that caused data base modifications, have their associated data base modifications backed out by emergency restart.

The IMS/VS user must determine the means of checkpointing and restarting his batch and batch-message processing programs. He may use the OS/VS checkpoint/restart facility, or create one of his own.

If the DL/I user chooses to write his own checkpoint/restart routines, he must, as a minimum:

- Record application variables and control tables.
- Record position information for non-IMS/VS data sets.
- Provide a restart entry point and reinitialization procedure.
- Properly initialize IMS/VS control blocks; for example, PXPARMS.

Use of the XRST call and user area parameters on the CHKP call simplifies the task for the user writing his own restart routines.

- A restart situation is indicated by specifying a checkpoint ID in the parm field on the execute card in the JCL or in the XRST call itself.
- Normal entry point and initialization procedures are used.
- User areas recorded at checkpoint time are restored.
- A GET UNIQUE is issued for each GSAM data base for the last used record if the data base was open at the time the checkpoint was taken.
- No data is returned as the result of the GU, but status codes are saved in the user PCBs.
- If the data base was opened for output, then a PNT function code, requesting POINT, is used.
- GSAM data bases are automatically repositioned at restart if the XRST call is used.
- The checkpoint ID is returned to the user program to allow it to link to its own restart subroutine.

In the case of batch-message programs, an actual checkpoint/restart routine may not be required. If the program is truly driven by the message queues, IMS/VS repositions the queues to the point where a CHKP call was issued. The user need only start the batch-message program normally.

Even though most batch-message programs require some re-programming to accommodate the CHKP function, the increased data base integrity and availability should justify the effort.

Since the IMS/VS control region waits until all batch-message regions issue CHKP calls before proceeding with a shutdown checkpoint, a batch-message program with few or no CHKP calls can delay or prevent system shutdown. The /STOP REGION command with ABDUMP can be used to force the abnormal termination of such a region. However, it is recommended that the user add CHKP calls to batch-message programs, particularly if a FREEZE or DUMPQ checkpoint is to be requested. If a PURGE shutdown is used, re-programming is suggested for batch-message programs that run for a long time, because IMS/VS waits for these programs to terminate before proceeding with the shutdown.

MESSAGE QUEUES

The IMS/VS control program provides the ability to queue messages received on direct access storage and in main storage. Messages can be received from communication terminals or application programs and can be destined for communication terminals or application programs. A message destined for an application program is called a transaction and begins with a transaction code. All transactions of the same type (same code) are queued in a serial chain based upon time of receipt by IMS/VS. A serial queue exists for each defined transaction code. All messages destined for a particular communications logical terminal are queued serially like transactions. A serial queue exists for each defined logical terminal (Figure 2-1).

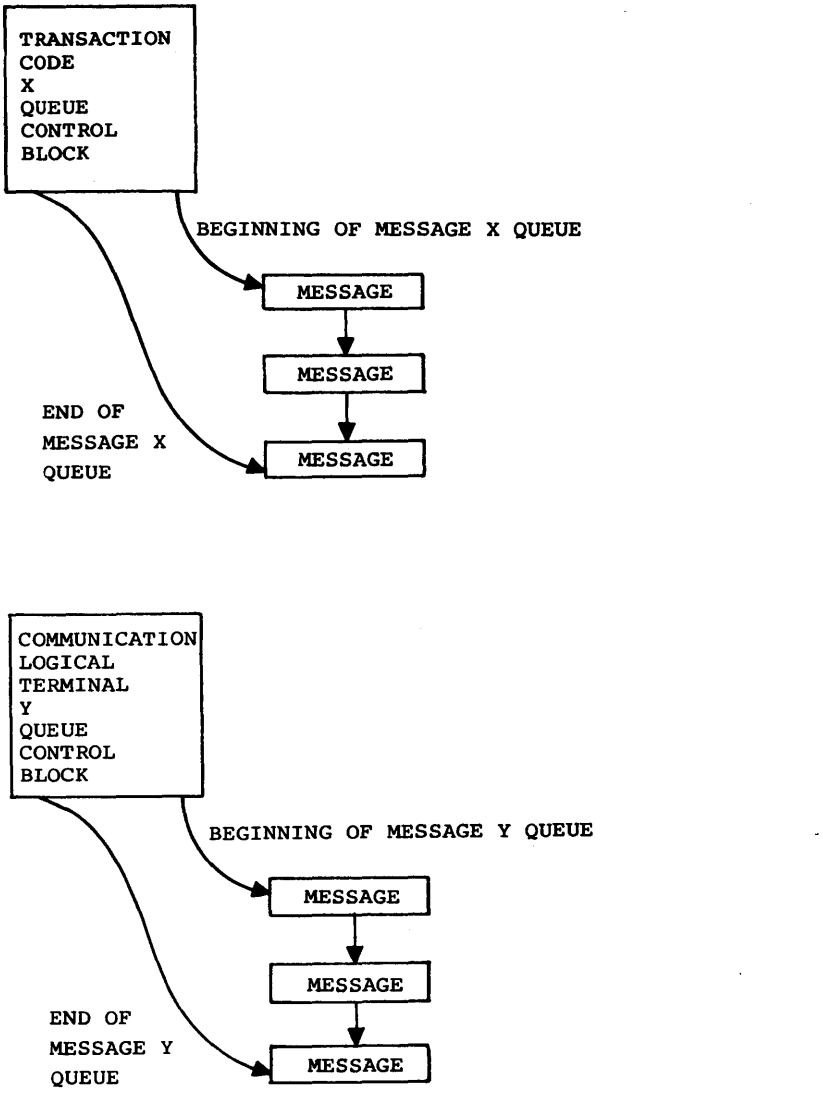


Figure 2-1. General Message Queue Structure

QUEUE DATA SETS

The IMS/VS control program utilizes three OSAM data sets for direct access queue storage. All queue data sets have the same block size, which is specified by the IMS/VS user at system definition time.

Figure 2-2 illustrates the relationship between the three queue data sets.

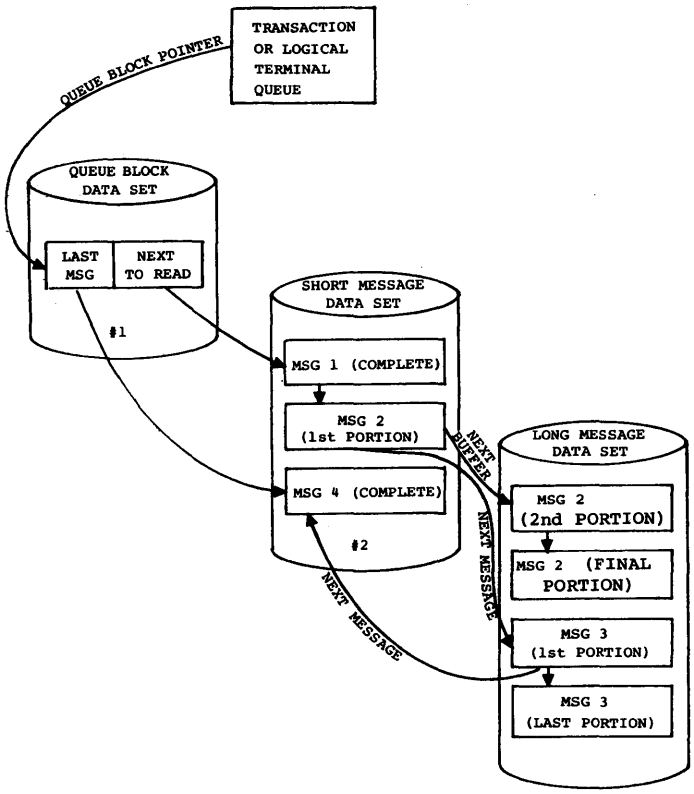


Figure 2-2. Queue Data Set Relationships

OPERATION OF QUEUES

All messages received are assigned OSAM relative record numbers. However, they are not written immediately to the queue data sets. If no space is available in the main storage buffers, the buffer which has been referenced the least is written to its queue data set, and the space in main storage is assigned to the new message. If a message still exists in main storage when it is dispatched to its destination (input to a program or output to a terminal or another program), no reference to the direct access data sets is required. All messages are logged by the IMS/VS control program to provide message queue recoverability in case of failure of either the IMS/VS or host operating system control programs.

Messages received are represented by either single or multiple segments. The amount of space required to contain a message determines the size of the records to which it is allocated. When the transaction or logical terminal queue is known, the average message size is also used to determine the record to be allocated. The lines of text are placed in a variable-blocked format within a record.

The IMS/VS message queue data sets must be preformatted before initial usage. The use of preformatted queues provides increased reliability. Reliability is increased with the preformatted data sets becaus the count field of the direct access device record X is not relied upon to write record X+1. Preformatting is performed upon request during restart procedures. The need to reformat the message queues arises only if an input/output error occurs within a queue data set. A write error does not result in the inability to write subsequent records in the data set as is the case with unformatted queue data sets. Approximately 1.5 seconds is required to format each 2314 cylinder in an IMS/VS message queue data set and .8 seconds for each 3330 cylinder.

In order to provide for message queue recoverability if the queue data sets are destroyed, the IMS/VS control program logs:

- all input and output message text
- the queue pointers to each message queue chain, whenever a message is enqueued onto or dequeued from the chain

If a system failure occurs and the message queue data sets are retained intact, the restart facilities of IMS/VS can reposition the queues by use of the enqueue/dequeue pointers which were logged. If the queue data sets are destroyed, the restart facilities of IMS/VS can be employed to rebuild the queues from the log entries of message text.

EMERGENCY RESTART QUEUE REPOSITIONING

In an emergency restart situation, the message queues are repositioned as follows:

- SNGL mode processing

  The message being processed at the time of the failure is the first message processed after the restart.

- MULT mode processing

  All messages read by the program are processed at the time of the failure are returned to the queue. The first message processed after the restart is the first message read after the program's most recent CHKP call or scheduling.

MESSAGE QUEUE REUSE

Message queue records reside in fixed-length blocks with a block size common to all three data sets. The first record in each data set is a bit map which controls the assignment of the next n records (n = 8 * LRECL-1). Records in each data set are assigned from low to high by testing the bit map for the first bit which is on. When a bit is found on, it is turned off to indicate that the corresponding has been assigned. When a record contains a message that has been completely processed at its destination (has been dequeued and will not be required in restarting the system), the bit corresponding to the record is turned on. This makes the record available for reuse.

For details on message queue data set space allocation, refer to the IMS/VS System Programming Reference Manual.

PHYSICAL TERMINALS

A physical terminal is the actual hardware device attached to the computer. The types of terminals supported include typewriters, CRTs (cathode ray tubes), paper tape readers, card readers, high-speed printers, and remote computers. The IMS/VS terminal configuration is defined to IMS/VS during system definition.

DEVICES SUPPORTED

IMS/VS supports:

- IBM 1050 Data Communication System
- IBM 2260 Display Station, Models 1 and 2
- IBM 2265 Display Station, Model 1
- IBM 2740 Communication Terminal Models 1 and 2
- IBM 2741 Communication Terminal
- IBM 2770 Data Communication System
- IBM 2780 Data Transmission Terminal
- IBM 2980 General Banking Terminals, Models 1, 2, and 4
- IBM 3270 Information Display System
- IBM 3600 Finance Communication System
- IBM 3767 Communication Terminal
- IBM 3770 Data Communication System
- IBM 3740 Data Entry System, Models 2 and 4
- IBM 3790 Communication System
- IBM 7770 Audio Response Unit, Model 3 with a Touch-Tone* (or equivalent) telephone or IBM 2721 Portable Audio Terminal
- IBM System/3 Model 10
- IBM System/7
- IBM Communicating Magnetic Card/Selectric Typewriter (CMC/ST)
- Card reader/printer devices
- 33/35 Teletypewriter (ASR)

IMS/VS supports various communication/attachment modes for the above terminals. The major distinction is whether the attachment is local (through a channel) or remote (over telephone lines). Remote attachments are further broken down into two categories: switched and nonswitched (or leased). Switched communication lines permit the attachment of only one remote station or terminal at a time to a line, and require that the terminal operator use a data set, which is attached to the remote terminal, to dial up the main computer to establish connection. Nonswitched communication lines are leased; that is, they are dedicated to use by the terminals physically attached to them. A nonswitched line may be either a contention or polled line. Contention or polled refers to the line discipline used to communicate with the terminal. Only one contention-type terminal may exist on a line, while one or more can share a polled line concurrently. A polled line with more than one terminal is called a multipoint line.

* Registered Trademark of the American Telephone & Telegraph Co.

See the IMS/VS General Information Manual for a description of the communications modes supported by IMS/VS for each physical terminal and for lists of the required and optional features for each supported terminal, control unit, and CPU.

BTAM DATA SET LINE GROUPS

The LINEGRP macro is used to describe each BTAM data set line group. The terminal(s) defined for any one LINEGRP must be of the same type (communication mode, polling techniques, transmission code). This means that a separate line group must be used for each of the following terminal configurations (when used):

- 1050 switched
- 1050 nonswitched with poll
- 1050 nonswitched with autopoll
- 2260/2265 remote and 2260 local mode, nonswitched
- 2740 switched with transmit control
- 2740 nonswitched contention
- 2740 nonswitched polled
- 2740 polled with autopoll
- 2741 switched*
- 2741 nonswitched EBCDIC and nonswitched correspondence
- 2770 nonswitched
- 2780 nonswitched polled
- 2780 nonswitched polled ASCII
- 2780 nonswitched polled 6-bit transcode
- 2780 nonswitched contention EBCDIC
- 2780 nonswitched contention ASCII
- 2780 nonswitched contention 6-bit transcode
- 2980 nonswitched
- 3270 local
- 3270 local printer
- 3270 polled remote
- 3270 polled remote ASCII
- 3270 switched
- 3270 switched (ASCII)
- 3740 switched
- 7770 switched
- System/3
- System/7 nonswitched contention
- System/7 nonswitched polled
- System/7 nonswitched polled with autopoll
- Local card reader
- Local output device (printer, punch, tape, DASD)
- Spool SYSOUT
- 33/35 switched

*For 2741 switched, transmission codes for any one line group do not have to be of the same type.

For further definition of a BTAM data set line group, refer to OS/VS BTAM, GC27-6980. At least one communication line must exist within each line group. At least one physical terminal must exist for each communication line.

TERMINALS ATTACHED THROUGH VTAM

Terminals attached through VTAM are defined according to terminal type by using the TYPE macro. Valid terminal types are: 3270 local, 3270 remote, 3601, 3614, 3767, 3770, and 3790.

PHYSICAL TERMINAL NETWORK DESIGN

Selection of terminal types should be based on what function is expected, the location and personnel using the equipment, and the speed or volume of data which the terminals are expected to handle.

Inquiry and conversational capabilities are best suited to typewriter or graphic type devices, the graphic devices being faster, while the typewriter gives a hard copy of the transaction.

Batches of input can best be handled by cards or paper tape, with the 2770, 2780, or 3770 being used for high volume and the 1050 for low.

The printer-type terminals are best suited for applications where the shop floor requires information from the computer but has no need to supply any in return. Again, the 2770 or 2780 is best suited for high volume, with the others handling less volume.

Once the types of terminals required for the job are determined, the method of connecting them to the computer must be considered.

If many terminal locations are required with minimal volume, a switched network should be considered. This allows the use of standard telephone lines. The terminal operator dials the computer when he wishes to make an entry. One drawback to this approach is the possibility of busy lines, which may cause the operator to place a call several times. Another disadvantage is that voice-grade lines are more susceptible to malfunction than leased lines. This might require the operator to request entry more than one time to allow the computer to read it error-free. Unchecked terminals (2741, 33/35, and 7770) can cause input and output to be lost due to line errors which are transparent to the IMS/VS system.

When high volume is required or the terminal must be connected to the computer for long periods of time, a leased line may be more practical. This type of line is generally more error-free, can handle higher volume of data, and requires no operator action to connect to the computer. If the leased line is chosen, the next step is to determine how many terminals are to be connected to this line. If several unbuffered terminals are connected to the line, significant delay may occur in the response to a terminal. It is therefore recommended that unbuffered terminals be attached alone to a line. Another consideration may be the need to cluster several terminals in one location. The expense of running several telephone lines to the same location may be prohibitive. If so, buffered terminals should be considered. Their slightly higher cost may be more than offset by the need to run only one line, thus reducing the contention for line time, as the data is transferred to a buffer at operator speed and then sent across the line at machine speed.

Most terminals supported by IMS/VS are polled. For some terminals, consideration should also be given to the type of polling to be used: programmed or autopoll. For small networks, programmed polling may prove more economical, since autopoll, except for binary synchronous lines, is an extra cost feature. However, programmed polling requires more CPU interruptions and, for a larger network, may use enough CPU time to make the cost of autopoll worthwhile. For each terminal in the system, programmed polling causes a hardware interrupt approximately every second. Autopoll causes this interruption only when the operator has initiated some action on the terminal, which will generally be several minutes.

Lines can be collected by terminal type into line groups. Each new line group requires main storage for control blocks used by IMS/VS and

the operating system. All lines for a particular type of terminal can be collected into one line group, minimizing this storage requirement. However, this means that all these lines must be allocated to the system at all times. When one is removed (possibly for use by a different job or system), IMS/VS does not function properly. Therefore, if one or more lines are to be used by IMS/VS on a part-time basis, and it is desired to allocate them to other functions at times, they should be organized into separate line groups. Lines may be removed from the system by line group.

When binary synchronous terminals (except 3270) are used in the online IMS/VS system, timeout conditions can occur when the system is so loaded that it cannot process an input line buffer and respond to the terminal. If the terminal operator re-enters the data before verifying the application program response, to determine the proper restart point in the data stream, this could lead to duplicate data.

LOGICAL TERMINALS

DEFINITION OF THE LOGICAL TERMINAL CONCEPT

The characteristics of terminal devices vary widely. There are differences in the control mechanics, transmission code, display media, entry keyboards, switches, timing, and optional features. Communication line and network characteristics further complicate and multiply the possible combinations of characteristics that must be managed in the data communication environment. It is readily apparent that the application program should not become directly involved with or dependent upon the characteristics of the terminal network with which it deals.

By isolating the application program from its terminal network, economies in development cost, development time, and maintenance are achieved. In addition, a certain degree of, if not complete, device independence is available. Applications written to a device-independent interface may be expanded without modification for the use of new terminal types or classes.

At the same time, use of device class dependent functions may be highly desirable in certain application areas. Control of device class dependent functions for an application system which serves only CRT-type devices could enhance the usability of that application.

Another requirement directly related to device independence is application independence. An application-supported function must be available from different terminal types. It is not feasible or practical to expect that a unique terminal be assigned to each function to be performed.

For reasons of security or resource management, it may be desirable to associate the use of a physical terminal with its user. Whereas users may exist in greater numbers than physical terminals, they must be represented by abstractions. The primary characteristic of the abstract terminal is its identity. The identity is known within IMS/VS as the "logical terminal name" or simply as "logical terminal."

THE IMS/VS LOGICAL TERMINAL

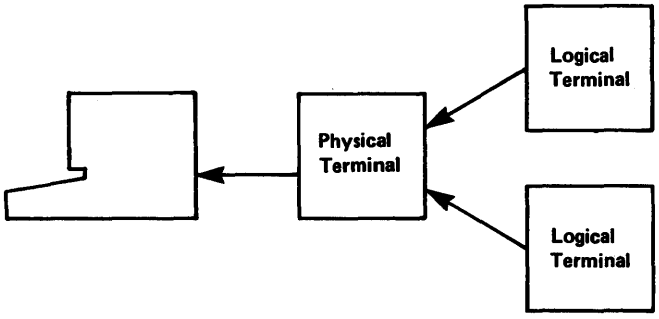
Each logical terminal within IMS/VS has a unique set of attributes. A description of the attributes constitutes a partial description of the features available through use of the logical terminal concept.

- Current physical terminal assignment -- this characteristic may be dynamically altered for reasons of terminal resource management. Once a sign on has been accomplished by connecting a logical terminal to a physical terminal, the functions and services available are the same as those for a nonswitched terminal.

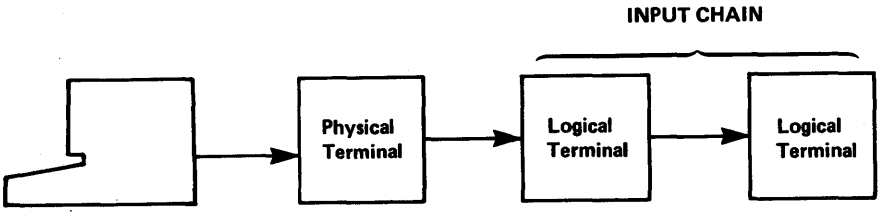
- Security authorization -- can be unique for each logical terminal in the system or can represent a security level or group.

- Next logical terminal assignment -- multiple logical terminals can be associated with a single physical terminal. This provides, in conjunction with security, the ability to uniquely identify multiple users of a single physical terminal.

Logical terminals can be assigned to physical terminals for output and input purposes. When a logical terminal is assigned to a physical terminal for output purposes, all messages sent to that logical terminal are transmitted to its associated physical terminal. More than one logical terminal can be assigned to a given physical terminal for output purposes. Only one physical terminal can receive the output for a given logical terminal. The diagram below shows the relationship between physical and logical terminals for output purposes:



When a physical terminal is assigned to a logical terminal for input purposes, any message entered from the physical terminal is considered to have originated at the logical terminal. When more than one logical terminal is assigned to a physical terminal for input purposes, a chain of input logical terminals is formed. Any input from the physical terminal is considered to have originated at the first logical terminal on the chain. If, for some reason (such as security or a stopped logical terminal), the first logical terminal is not allowed to enter a message, all logical terminals on the input chain are interrogated in chain sequence for their ability to enter the message. If the physical terminal is a 3770 or a 3767, only the logical terminals associated with the input component are scanned. The first appropriate logical terminal found is considered the originator of the message. If no appropriate logical terminal is found, the message is rejected with an error message. The diagram below shows the relationship between physical and logical terminals for input purposes:



Use of a queue for input messages received or pending output messages enables the application to be independent of time of arrival or transmission of messages. Association of the queue with the logical rather than the physical terminal permits it to be moved, independent of the application, from device to device. Within restrictions, it permits a queue of messages to be moved even among device classes.

The logical terminal provides a stable platform or reference for the application program. Regardless of how the physical terminal network changes, the application remains insensitive. To the application program, 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 the same call interface mechanics described for the DB system.

When 2980 terminals are defined, IMS/VS uses a logical terminal to define the 2972 common buffer. This is an exception to the physical terminal/logical terminal relationship, in that the 2972 common buffer is not a physical terminal in the conventional sense.

LOGICAL TERMINAL NETWORK DESIGN

Design of a logical terminal network can be as important as design of a physical terminal network. It has potential impact upon system security, maintainability, and usability. Careful consideration should be applied from each viewpoint.

System security administration can be hampered by not providing an appropriate number of logical terminals through which proper terminal security authorization may be applied. Too few logical terminals limits the number of unique security authorizations. Too many may prove cumbersome or ineffective in achieving security objectives. A judicious combination of password and logical terminal security can reduce the number of logical terminals required to administer security policy.

Where a community of users deals with multiple applications through a common set of physical terminals, output volumes, schedules, priorities, human factors, and terminal availability are some of the more important usability factors to consider. If priorities require that management or supervision have ready access to terminals ordinarily used for operational purposes, then provision must be made for interrupting operational work. A physical terminal might have two logical terminals ordinarily assigned -- one for operations, one for priority work. Authorization of the /LOCK command to the priority logical terminal would enable it to stop input and output from the operations terminal. Further discussion of the security planning for this particular case may be found under the topic "Security and Privacy" in this chapter. It is mentioned here to illustrate several of the aspects of logical terminal network planning. The same solution to the security or priority aspect, that is, multiple logical terminals, can be applied if the control of output volume is a problem.

Where particular applications make use of device class dependent functions, such as cursor control, it might be useful to specify a separate set of logical terminals which have a relationship to that group of applications. Calling the application group an application class and the logical terminal group a logical terminal class, it is possible through logical terminal security to associate all input and output relationships with a known set of logical terminals. At the same time, non-device-class sensitive transactions may be used through non-specific logical terminals from the same physical terminals. Processing applications are insensitive to the separation. The following example (Figure 2-3) illustrates this use of logical terminals:

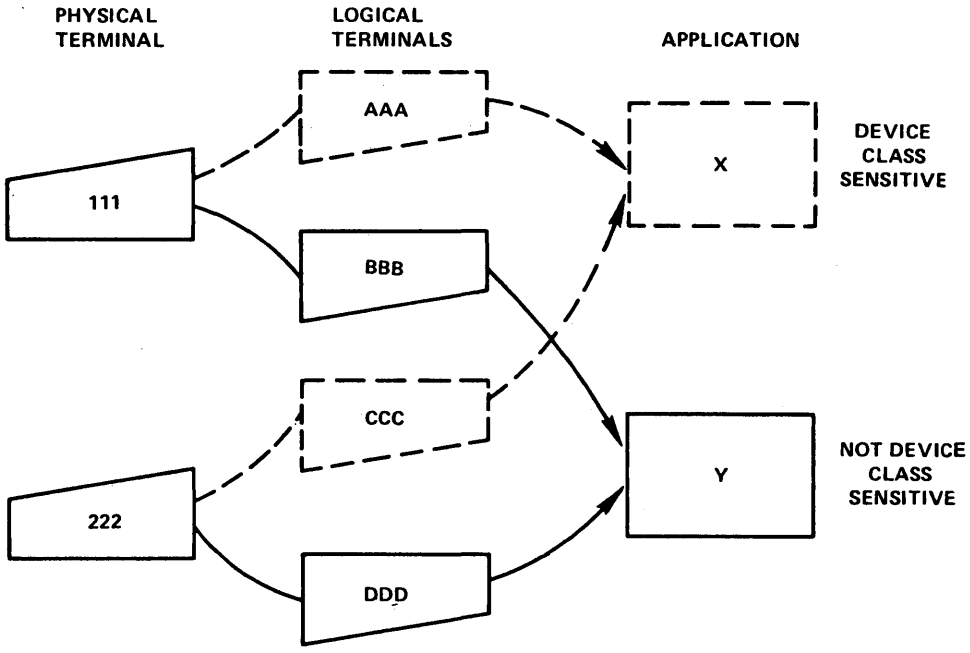


Figure 2-3. Separating Device Class Sensitive Terminal I/O

To establish such a relationship requires defining two logical terminals for each physical terminal, then securing the transactions destined for application X through logical terminals AAA and CCC. The common entry security for AAA and CCC could be referred to as a device class sensitive security group. All logical terminals defined for that purpose would then be secured in the same group.

In certain applications it may be necessary to associate a different physical device for output than the one ordinarily used for input. Conversely, certain physical terminal types are input-only devices. If output is required, a different device must be associated with this type for output. IMS/VS system definition and commands support assignment of output devices different from the input device. The allowable physical/logical relationships which can be expressed are shown in Figure 2-4.

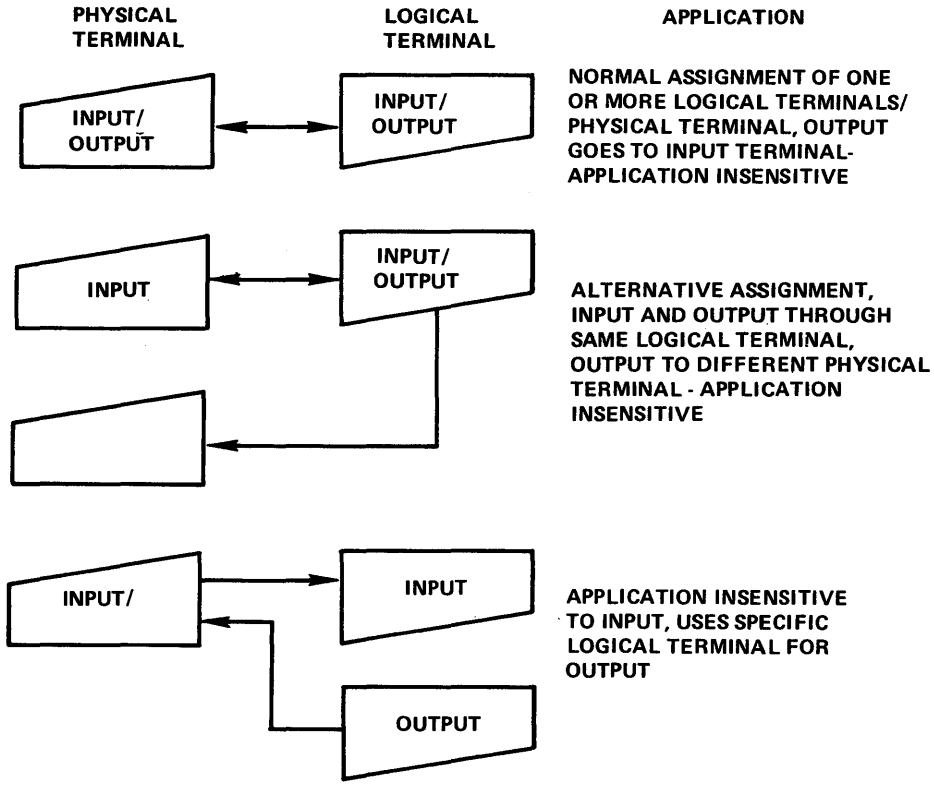
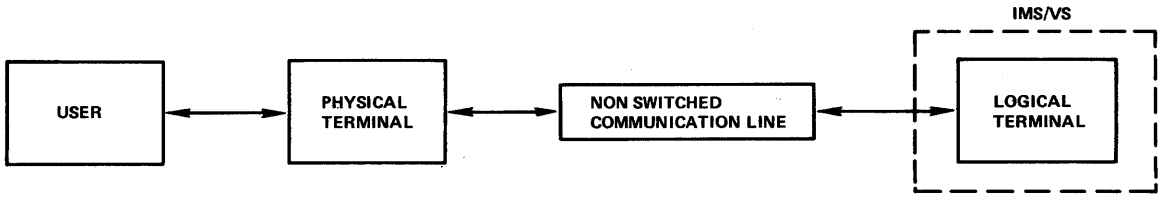


Figure 2-4. Possible Physical/Logical Terminal Relationships

Logical Terminal/Physical Terminal Relationship

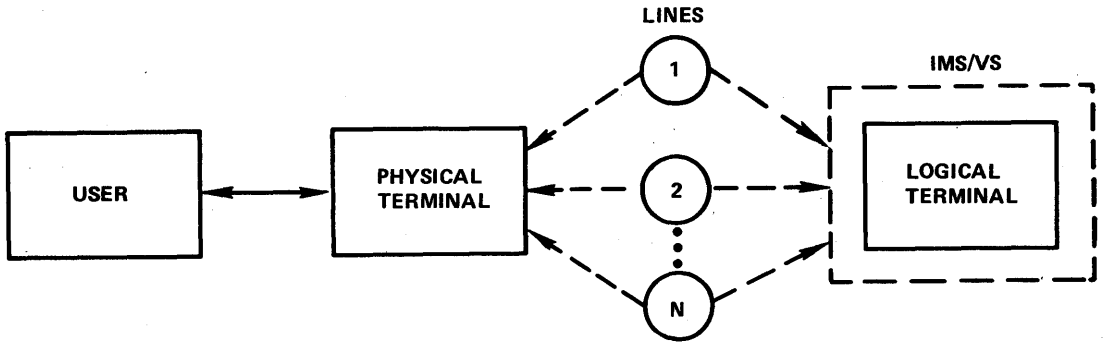
Nonswitched Communications Network: The best way to describe the relationship between a terminal user, a physical terminal, a communication line, and a logical terminal is a diagram:



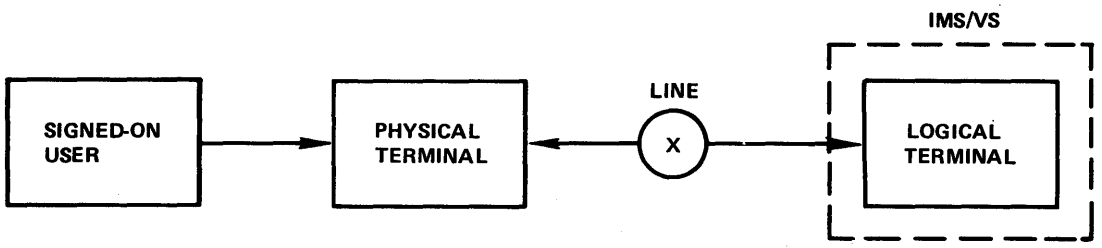
IMS/VS system definition describes the characteristics and relationship of physical terminals, communication lines, and logical terminals. On a nonswitched communication line, the relationship between a physical terminal at one end and a logical terminal within IMS/VS at the other is a stable relationship defined at system definition time. If there is only one user of a particular physical terminal, typically there would be a one-to-one relationship between physical terminal and logical terminal. However, if a physical terminal is operated by multiple users, it can have many logical terminals associated with it. IMS/VS system definition might include a separate logical terminal for each user of a particular physical terminal.

The relationship established between a physical terminal and one or more logical terminals at system definition can be changed through the command language or by a new system definition. The /ASSIGN command changes logical/physical relationships dynamically. It is normally executable only from the master terminal.

Switched Communications Network: The logical/physical terminal relationship on a switched communications network is considerably more complex than in the nonswitched communication line environment. IMS/VS system definition defines the characteristics of a physical terminal, communication lines, and logical terminals. However, the relationship between a particular physical terminal and a logical terminal is not established until the remote terminal user dials the System/370 computer to communicate with IMS/VS. The relationship between a terminal user, a physical terminal, a communication network, and logical terminals at system definition time is depicted in the following diagram:

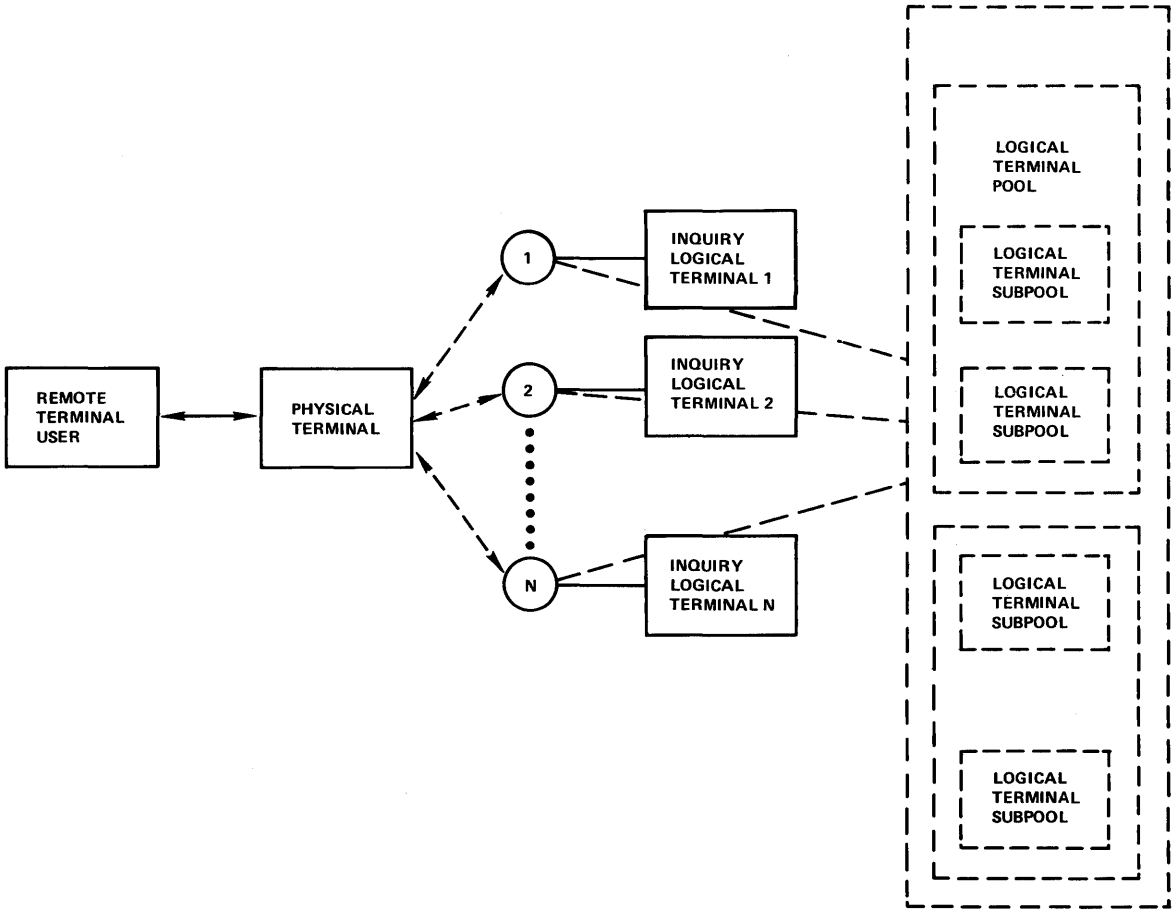


Once the remote terminal user dials in to the computer and issues the /IAM command to sign himself on to IMS/VS, a stable relationship between the physical terminal and one or more logical terminals is established.

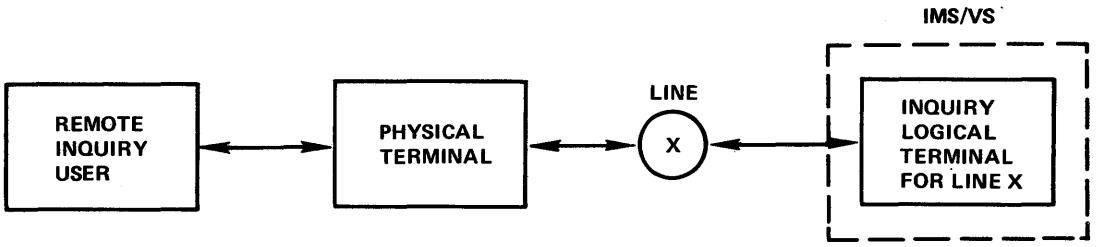


In a switched communications network environment, one logical terminal per line is created automatically as the inquiry logical terminal. In addition to the physical line/terminal definition, and the automatic creation of the inquiry logical terminal, a pool of logical terminals can be defined at system definition time. When a remote terminal user dials into IMS/VS, an /IAM command can be issued which associates logical terminals from a pool with the physical line and physical terminal issuing the /IAM command.

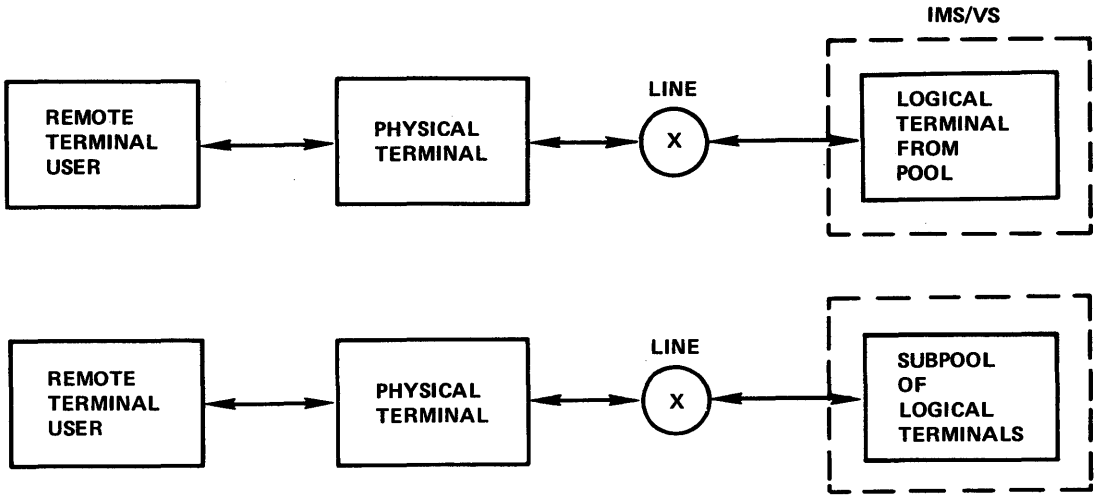
Within any logical terminal pool for a switched communications network, the IMS/VS user must define one or more logical terminal subpools. A logical terminal subpool is composed of one or more logical terminals within a given logical terminal pool. A particular logical terminal can exist in only one pool and subpool. A remote terminal user can dial the IMS/VS system and sign on for a single logical terminal or all logical terminals within a logical terminal subpool. At system definition, the environment appears as indicated in the following diagram:



After a remote terminal user has dialed the System/370 computer operating under IMS/VS, several situations can exist. If the /IAM command is used to sign on and the LTERM parameter specifies the inquiry logical terminal, the following diagram applies:



If the /IAM command is used to sign on and the LTERM parameter specifies a logical terminal from the logical terminal subpool, the following diagram applies:



If the /IAM command is used to sign on and the LTERM and PTERM parameters are specified, all logical terminals within a subpool are associated with the physical terminal.

The use of the logical terminal subpool concept allows for efficient use of communication facilities. All output queued on each of the logical terminals in the subpool for which the /IAM command was issued is sent to the physical terminal.

A subpool can be defined to contain the logical terminals for all of the users of a single physical terminal. While a user is signed on to a logical terminal within the subpool, the subpool is unavailable to users signing on from other physical terminals.

All inquiry logical terminal names must begin with INQU. When signing on for an inquiry logical terminal, only these first four characters are considered significant by IMS/VS. This lets a user call any autoanswer line and sign on for, and use, the inquiry logical terminal (for inquiry transactions only), if he is aware of the INQU prefix. The inquiry logical terminal can only be used for non-update transactions, and queued output is preserved only while the user is signed on. So that IMS/VS can distinguish inquiry logical terminal names from subpool logical terminal names at the time a user signs on, no subpool logical terminal name can begin with INQU.

MASTER TERMINAL

The master terminal is the IMS/VS control center. It must be either a 1050, a station-controlled 2740, a 3270, a 3767, or a 3770. If a 1050 or 2740 is used, it must be attached through a non-switched polled communications line. A 3270 master terminal can be attached locally or through a non-switched polled line. The IMS/VS provision for a 3770 master terminal is intended for the 3770 console component. The non-console components will not operate correctly if they are used as the master terminal.

The master terminal operator should know all the operating aspects of the system. The physical location of the master terminal in relation to the computer console is important. If, for security reasons, they are not close, telephone communications should be provided.

The details of starting the system, checkpoint, restart, and all commands available to the master terminal are in the IMS/VS Operator's Reference Manual.

SYSTEM CONSOLE SUPPORT

IMS/VS provides support for the OS/VS system console using the OS/VS write-to-operator (WTO) and write-to-operator-with-reply (WTOR) facilities. All functions available to the IMS/VS master terminal are available to the system console. The system console and master terminal can be used concurrently, to control the system. Usually, however, the system console's primary purpose is as a backup to the master terminal. The system console is arbitrarily defined as IMS/VS line number one.

SYSTEMS WITH INOPERABLE MASTER TERMINAL

IMS/VS requires a master terminal be defined for its use during IMS/VS system definition. Under certain conditions, however, it may be impractical to provide a master terminal facility; for example when the 270X line is inoperable. In these instances, the OS/VS system console can be utilized to replace the IMS/VS master terminal. If desired, the master terminal DD statement can be omitted. If the master terminal is inoperable, messages will continue to be routed to it until they are routed to the system console or another terminal with the /ASSIGN command. In addition, all of the functions ordinarily performed at remote operational terminals can also be performed through the System/370 console.

MESSAGE FORMAT SERVICE

Through the Message Format Service (MFS), a comprehensive facility is provided for IMS/VS users of 2740, 2741, 3270, 3600, 3767, and 3770 devices. MFS allows application programmers to deal with simple logical messages instead of device dependent data; this simplifies application development. The same application program may deal with different device types using a single set of editing logic while device input and output are varied to suit a specific device. The presentation of data on the device or operator input may be changed without changing the application program. Full paging capability is provided for display devices. Input messages may be created from multiple screens of data.

A program using MFS need not be concerned with the physical characteristics of the device used for input and output messages unless it wants to use certain very specific device features. Even when these features are utilized, the program can request functions in a logical manner; no device control characters or orders may be sent directly from the program or may be received by the program. The presentation of data on the device may be changed without application program changes. Both logical and physical paging facilities are provided for the 3270 and 3604 display stations; this allows the application program to write a large amount of data that will be divided into multiple screens for display on the terminal. The capability to page forward and backward to different screens within the message is provided for the terminal operator. The conceptual view of the formatting operations for messages originating from or going to an MFS-supported device is shown in Figure 2-5.

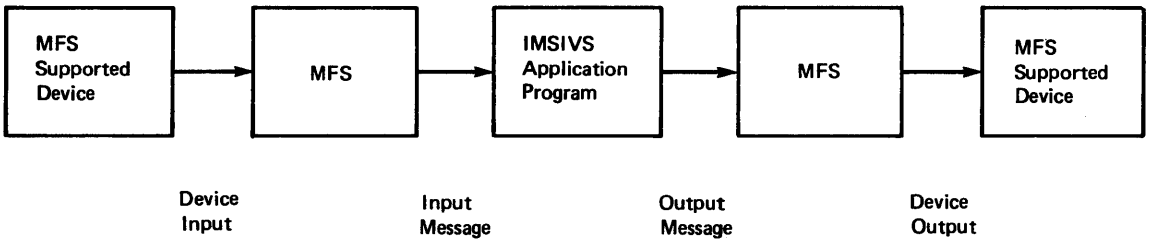


Figure 2-5. Message Formatting Using MFS

MFS has three major components:

- MFS language utility
- MFS pool manager
- Message editor

The MFS language utility is executed offline to generate control blocks and place them in a format control block data set named IMSVS.FORMAT. The control blocks describe the message formatting that is to take place during message input or output operations. They are generated according to a set of utility control statements specified by the IMS/VS system designer. There are four types of format control blocks:

- Message input descriptor (MID)
- Message output descriptor (MOD)
- Device input format (DIF)
- Device output format (DOF)

The MID and MOD blocks relate to application program input and output message segment formats, and the DIF and DOF blocks relate to terminal I/O formats. The MID and DIF blocks control the formatting of input messages, while the MOD and DOF blocks control output message formatting.

The message editor and MFS pool manager operate online during the normal production mode of operation. The message editor performs the actual message formatting operations using the control block specifications. The MFS pool manager controls residence in the main storage MFS buffer pool of the format control blocks required by the message editor. Efficient use of available pool space is provided by look-ahead fetching of required control blocks from direct access storage, and by maintenance of last-referenced format control block chains for reuse of pool space.

Two other MFS components, a MFS service utility and a MFTEST pool manager are available to support optional MFS operations.

The MFS service utility provides a method for additional control of the format control block data sets. It executes offline and can be used to create and maintain an index of control blocks for online use by the MFS pool manager.

The MFSTEST pool manager replaces the MFS pool manager to support the optional MFSTEST mode of operation. The IMS/VS /TEST MFS command can be used to place online MFS terminals into MFSTEST mode during which new applications and modifications to existing applications can be exercised without disrupting production activity.

Figure 2-6 provides an overview of major MFS operations. The circled numbers reference notes that indicate major distinctions in MFS processing when the MFSTEST facility is used. The IMS/VS Message Format Service User's Guide provides a complete description of MFS.

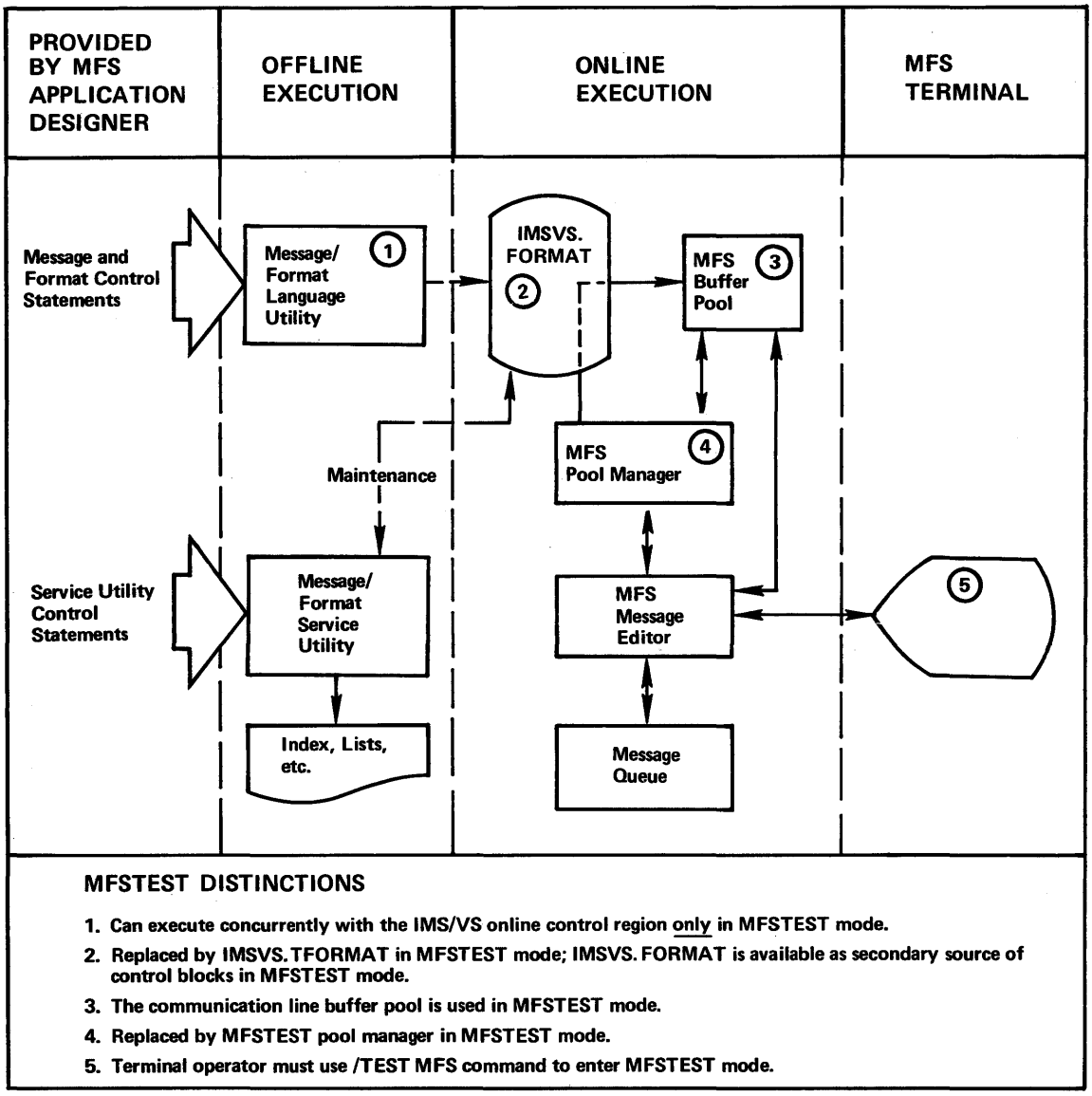


Figure 2-6. Overview of Message Format Service

OVERVIEW OF IMS/VS 3270 SUPPORT

The IMS/VS Message Format Service (MFS), described in the previous section, is used exclusively to format data transmitted between IMS/VS and the devices of the 3270 Information Display System. MFS provides a high level of device independence for the application programmers and a means for the application system designer to make full use of the 3270 device capabilities in terminal operations. The IMS/VS Message Format Service User's Guide contains a complete description of MFS.

3270 COPY FUNCTION

When an IMS/VS system is defined to include printer components of the 3270 Information Display System attached through a polled BSC or SDLC line, it is possible to allow an automatic or operator-controlled hard copy of the video output (or input) to be sent to a 3284/3286 printer. This hard copy can be requested through the use of the SCA field in the application program's output data, the definition of the message (see IMS/VS Message Format Service User's Guide), or by operator action. The hard copy listing is produced on an appropriate printer, which must be attached to the same control unit (3271 or 3275) as the display station containing the information to be copied. If a request is sent to a terminal that is not defined as allowing the copy function, or that does not support the copy function (3270 local attachment), the request for the copy function is ignored.

The format of the printed output can vary from that on the display station as a result of blank lines (or null lines), which are ignored by some models of the 3284/3286 printers. In all cases, the buffer size of the printer must be equal to or larger than the buffer size of the display station to be copied (3275/3284 Model 3 has no printer buffer and this consideration does not apply).

When printers are attached to a 3271, the IMS/VS system definition process determines which printers are eligible to receive the hard-copy output of a copy operation. These printers are called candidate printers. When a copy operation is requested by the operator or an application program, the candidate printers are searched in a predetermined order to find a printer that can be used. The first printer that is not stopped, is not currently printing a message, is not in exclusive status, and is ready, is used. If the operator made the copy request and all printers are busy, the keyboard on the display station is left inoperable until a printer is available and the message is successfully copied to the printer. If the copy request is from an application program and all printers are busy, the message is not displayed until a printer becomes available. This prevents the operator from altering the data to be printed before the message is successfully copied to the printer. If no candidate printers are currently available, an appropriate error message is sent to the display station requesting the copy operation. If the copy operation was requested by the application program or the format description (DEV statement, DSCA operand), an attempt to send the message will be retried when the error message is cleared from the screen through the Message Advance Function (see the IMS/VS Operator's Reference Manual). If the copy function was requested by the operator, the operator can ready the candidate printer(s) and retry the copy operation.

Candidate printers for a particular display station result from the way the physical terminals are defined during IMS/VS system definition. Candidate printers for a display station must be defined after that display station but before any other display station-printer groups. Other display stations can intervene between a display station and its candidate printers, but other display station-printer sequences can not intervene. For example, in Figure 2-7, PTERM 1 might be a 3275

with its own dedicated printer. If PTERM 2 and 3 allow the copy function, then PTERMs 4 and 5 will be the candidate printers for these PTERMs. If PTERM 6 is allowed to use the copy function, then PTERM 7 will be the candidate printer for PTERM 6. Note that the candidate printer PTERM 7 will not be used for copy functions from PTERMs 2 and 3, nor will candidate printers PTERMs 4 and 5, be used for copy functions from PTERM 6. And, in no instance, is a copy function permitted across line, linegroup, or 3270-control-unit boundaries.

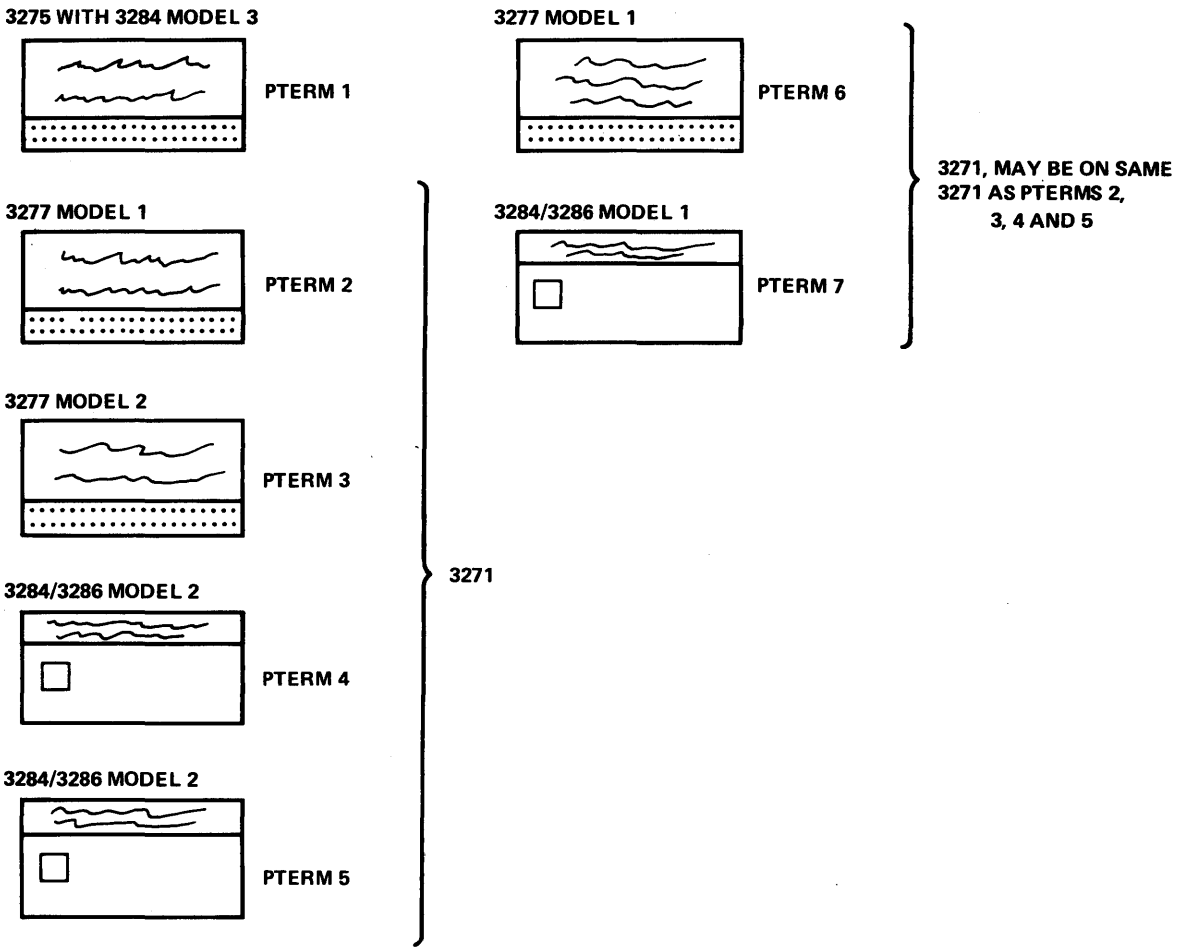


Figure 2-7. 3270 Copy Function Example

3284 MODEL 3 PRINTER SUPPORT

The 3284-3 printer, when attached to a 3275, is supported by IMS/VS as a component of the 3275 terminal. Messages are sent to the two components on a rotating basis, as with any component-type terminal. If no messages can be sent to the printer component, messages are sent continuously to the display component, just as if no printer component existed. If no messages can be sent to the display component, messages are sent to the printer as though the display component did not exist. As long as messages can be sent to the printer, no operator intervention is required. When a message is sent to the display component while messages are enqueued for the printer, the operator must intervene to allow either further display output or printer output. Any situation (such as a stopped LTERM or an inoperable printer) that prevents the

sending of messages for the LTERM(s) assigned to a particular component causes message transmission to cease to that component.

3270 MASTER TERMINAL SUPPORT

IMS/VS supports a 3270 terminal as a master terminal. A 3270 master terminal consists of two 3270 components: a 3277 display and a 3284/3286 printer. The 3275 with an attached 3284-3 is not supported as a 3270 master terminal.

When IMS/VS uses a 3270 master terminal, all messages are routed to the display component. Certain selected system-generated messages, critical to IMS/VS operation, are also sent to the 3284/3286 component.

INTELLIGENT REMOTE STATION SUPPORT

IMS/VS provides for attachment of a System/3 Model 10 and System/7 using the IRSS (intelligent remote station support) interface. The interface provides a remote station with powerful tools to control the flow of data between a System/370 and terminals attached to the intelligent remote station. This interface provides the definition of transmission block formats. A primary purpose for these formats is to define message transmission associated with one or more terminals attached to the intelligent remote station. These formats are described in detail in the IMS/VS System Programming Reference Manual.

Conversational processing as well as presetting of destinations are available to terminals attached to the remote station. IMS/VS provides the facility of routing transaction responses to the originating source as well as to alternate destinations without application program involvement. IMS/VS provides a restart facility for the remote station by logging and retransmission of appropriate block and message identifiers.

TRANSMISSION BLOCKS

Two types of transmission blocks are defined in the IRSS interface. The data block type is used to carry message segments. The synchronization block type is used to carry all other required information such as shutdown, restart, status change, ask for output, and dequeue output.

Each data block contains a block identifier containing, in four bytes, information that can be used by the remote station to restart its transmission of data to IMS/VS, if it has a restart facility. The content of this identifier is up to the remote station, but if the same identifier appears in the first data block received by IMS/VS as was contained in the restart message, after IMS/VS has transmitted a restart message, IMS/VS considers the block retransmitted and will scan for a restart point as described below.

Each data segment in a data block contains a message identifier. This one byte message identifier contains information that enables the remote station to identify a message or segment within a block. In addition, IMS/VS appends the message identifier from a segment in error, if an error message must be transmitted by IMS/VS to the remote station due to an error discovered while processing a segment. The message identifier is also contained in restart messages and can be used by the remote station to restart its transmission of data because it indicates the last complete message processed by IMS/VS within the identified block.

The message identifier is used by IMS/VS to scan for a restart point if a block was retransmitted after restart. IMS/VS scans the received block until a segment with the same message identifier as in the restart message, and which is flagged as the last segment, is found. IMS/VS then starts processing with the segments following the one found, if any. The entire block is discarded if no segment that meets the above specifications is found. Cold start messages do not contain block and message identifiers since none are available, but they imply binary zero identifiers. Therefore, the remote station should not use a block identifier of binary zeros in the first block transmitted to IMS/VS following a cold start message from IMS/VS, or the block will be ignored.

A two-byte terminal identifier is used by the IRSS interface for destination control. The terminal identifier used in communication with IMS/VS must be defined when performing the IMS/VS system definition. The TERMINAL macro is used for this purpose. IMS/VS treats each defined terminal identifier as a physical terminal. Since IMS/VS has no knowledge about the actual physical terminals attached to a remote station, there is no requirement that the terminal identifier correspond to a physical terminal address. The number of physical terminals attached is also independent of the number of terminal identifiers specified. The terminal identifiers employed by IMS/VS IRSS provide a means of extending all IMS/VS facilities characteristics of a physical terminal to any logical destination within a station supported by IRSS. Since IMS/VS has no knowledge of the terminal itself, this designator can be used to accomplish a variety of application-dependent functions; for example:

- Routing to specific terminals or devices in the remote station
- Scheduling of specific application programs within the remote station
- Batch-type terminal support similar to 2770 or 2780 terminals by proper definition of the remote station I/O components
- Data collection from a variety of I/O devices into a single stream identified to IMS/VS as a unique terminal for specific IMS/VS application program processing

Prior to the enqueue of a message received from a remote station, IMS/VS logs the identifiers pertaining to the last block and segment of the message. This information is also kept in the communications restart block (CRB) and is restored by restart. The identifiers, pertaining to the last message enqueued, are transmitted to the remote station in all types of restart messages except the cold start message.

SYSTEM/3 AND SYSTEM/7 PROGRAM FUNCTION REQUIREMENTS

The IMS/VS support for System/3 and System/7 does not include a program resident in either computer. The IMS/VS user must supply this program. The user's program residing in the System/3 or the System/7 must be able to handle at least the following parts of the IRSS interface:

- Transmission control
- Data blocks
- Immediate shutdown request from IMS/VS
- Send output complete message to IMS/VS

It is recommended that the program be capable of recognizing error messages. All other information provided by IMS/VS can be used or ignored at the discretion of the user.

IMS/VS System Messages

All IMS/VS system messages contain a message identification whose first three characters are DFS. The IRSS support extracts the number from the message, in case of an error message, and builds a synchronization block. All user initiated messages should be set up so they cannot be confused with an IMS/VS system message.

TRANSMISSION CONTROL

IMS/VS receives transmission blocks from a remote station in input mode and transmits blocks to a remote station in output mode.

IMS/VS may request the line to do the following while in input mode:

- Transmit error messages pertaining to received data.
- Transmit command completed messages pertaining to received commands.
- Return a test message if a terminal has been placed in test mode through the /TEST command.
- Transmit an immediate shutdown request message.

IMS/VS causes a reverse interrupt sequence to be transmitted if any of the preceding conditions occur when in input mode. IMS/VS then accepts one additional input block after transmission of the reverse interrupt. An attempt to transmit more than one block results in a transmission error and the station is logically deactivated.

Error messages and shutdown request messages are transmitted using the appropriate synchronization block. Command completed messages and test messages are transmitted using data blocks.

A message transmitted by IMS/VS in output mode must be removed from the queue through a request from the remote station. This is done to ensure that no message is removed from the IMS/VS queue until it has reached its final destination at the remote station. The request to remove a message is made using the appropriate synchronization block. This can be performed at any time after the last segment of the message has been received by the remote station but before any message is transmitted to IMS/VS using the same terminal identifier. IMS/VS retains an output message in progress on the queue if an input message is received for the same terminal identifier, even if the last segment has been transmitted but the remove request is not received.

The remote station can transmit an error message to IMS/VS at any time after the first segment of the message has been received, but before it is removed from the queue or retained on the queue because of an input message. An error message causes the logical terminal, on which the message is queued, to be stopped and a message sent to the master terminal. The message is retained on to the queue. Error messages are transmittted using a synchronization block. Messages transmitted by IMS/VS while in input mode are not queued and, therefore, cannot be removed from a queue. Consequently, the remove from queue message should not be sent.

Any error detected in the interface between IMS/VS and the remote station results in logical deactivation of the remote station by an EOT.

SYSTEM DEFINITION

The System/3 and System/7 are defined using the STATION macro. Included in this macro are the station's polling address (if applicable) and the station's operating modes.

Three operating modes may be defined in any combination:

- Postpone type -- non-postpone type
- Ask type -- non-ask type
- Transmission limit -- no-transmission limit

A System/7 station on a start/stop line has the added definition of output transmission code modes. The station can be defined to require all data blocks to be transmitted in PTTC/EBCD code, pseudo-binary PTTC/EBCD code, or to allow IMS/VS to determine the code.

Postpone Type Station

A station defined as postpone type is started with the postpone output flag set in all defined terminals. The remote CPU must send the resume output I/O synchronization block to IMS/VS to receive output.

A postpone type station has the advantage of specific terminal output requests by the user program in the remote CPU. This function can conserve resources within that system.

Ask Type Station

To allow the user's program to control when to receive blocks from IMS/VS, the station can be defined as ask type. After the restart message has been transmitted by IMS/VS, IMS/VS waits to receive an ASK message before transmitting anything else. The ASK message is sent by a remote station to inform IMS/VS that the station is ready to receive. This message is required:

- After IMS/VS has transmitted the NO-OUT message (I/O synchronization message flag value X'08') to the remote station.
- After IMS/VS has transmitted a user specified number of blocks to the remote station. This count is reset each time an ASK message is received. Messages sent following a LINE TURN AROUND requested by IMS/VS are not counted.

IMS/VS transmits blocks according to normal rules after an ASK message has been received. When all available output that can be sent has been sent, IMS/VS transmits the NO-OUT I/O synchronization message. IMS/VS then waits to receive an ASK message before transmitting any further output. The transmission of the NO-OUT message can be preempted by reaching transmission limit. The ASK message is an I/O synchronization message with flag value X'10'. The NO-OUT message is an I/O synchronization message with flag value X'08'. The format of these messages is described in the IMS/VS System Programming Reference Manual.

Transmission Limit

IMS/VS system definition allows for the specification of a transmission limit fcr each remote station defined. The transmission limit is the maximum number of transmission blocks, excluding the block transmitted following a reverse interrupt sequence and the shutdown synchronization block, that IMS/VS will send in output mode between remote station initiated resets. The remote station uses the ASK message to perform this function. The ASK message is sent by a remote station to inform IMS/VS that the station is ready to receive. The transmission limit defined to IMS/VS should be the number of buffers in the remote station minus one, because IMS/VS may be required either to transmit blocks to the remote station while in input mode (see the description under "Transmission Control" in this chapter) or send a shutdown synchronization block while in output mode.
This message is required:

- After IMS/VS has transmitted the NO-OUT message (I/O synchronization message flag value X'08') to the remote station.

- After IMS/VS has transmitted a user specified number of blocks to the remote station. This count is reset each time an ASK message is received. Messages sent following a LINE TURN AROUND requested by IMS/VS are not counted.

The transmission limit can range from 1 to 15, or be defined as zero, indicating unlimited transmission.

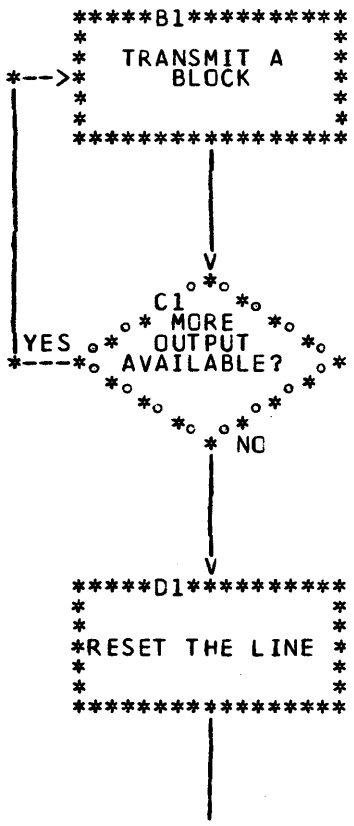
Combining Modes

The three remote CPU operating modes can be defined in any combination. The presence (or absence) of postpone type does not impact IMS/VS function. IMS/VS function does vary, however, when ask type and/or transmission limit are specified or are not specified.

The flowcharts below show IMS/VS function for the possible combinations of operating modes:

- Basic (non-ask type, no-transmission limit)
- Ask-type, no-transmission limit
- Non-ask type, transmission limit
- Ask-type, transmission limit

Basic (non-ask type, no transmission limit)

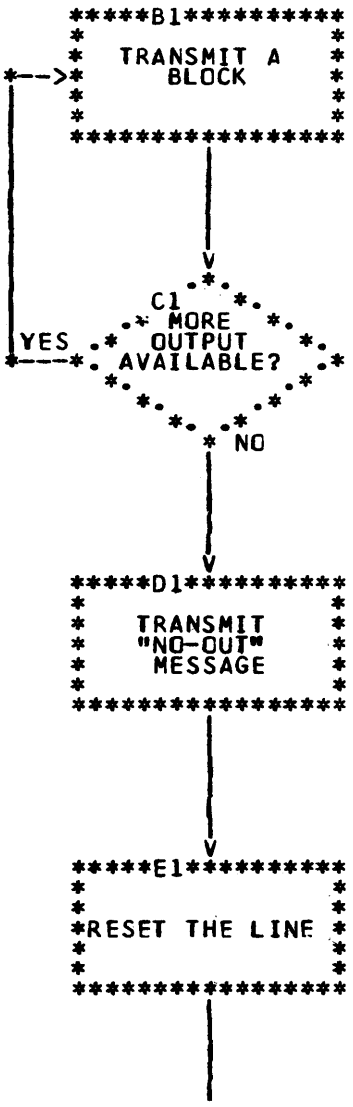


IMS/VS WILL START TRANSMITTING OUTPUT AS SOON AS THE LINE IS AVAILABLE AFTER AN OUTPUT MESSAGE HAS BEEN ENQUEUED.

IMS/VS CONTINUES TRANSMITTING AS LONG AS THERE IS OUTPUT AVAILABLE THAT MAY BE SENT.
IMS/VS TRANSMITS ONLY ONE MESSAGE FOR A GIVEN TERMINAL IDENTIFIER REGARDLESS OF THE NUMBER OF MESSAGES ENQUEUED.

IMS/VS TRANSMITS EOT TO TERMINATE TRANSMISSION.

Ask-type, no-transmission limit



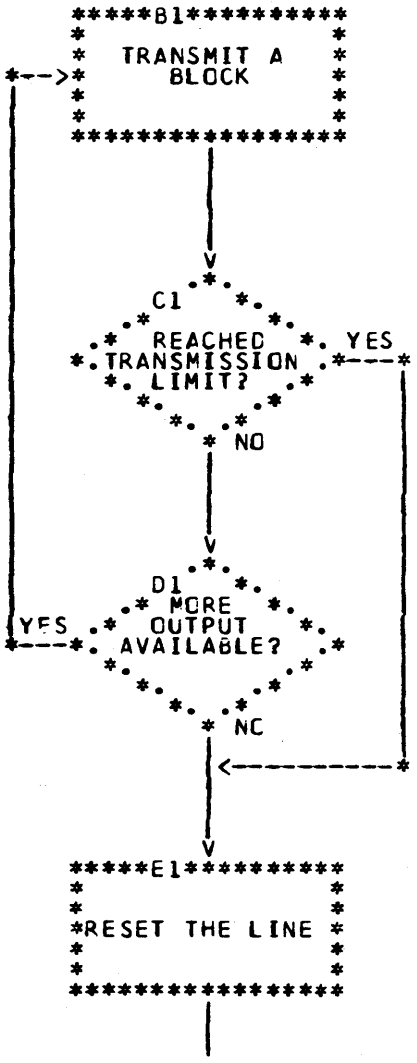
IMS/VS WILL START TRANSMITTING OUTPUT AS SOON AS THE LINE IS AVAILABLE AFTER AN "ASK" FOR OUTPUT MESSAGE HAS BEEN RECEIVED.

IMS/VS CONTINUES TRANSMITTING AS LONG AS THERE IS OUTPUT AVAILABLE THAT MAY BE SENT.
IMS/VS TRANSMITS ONLY ONE MESSAGE FOR A GIVEN TERMINAL IDENTIFIER REGARDLESS OF THE NUMBER OF MESSAGES ENQUEUED.

IMS/VS TRANSMITS A SYNCHRONIZATION BLOCK INDICATING TO THE REMOTE CPU THAT NO OUTPUT MESSAGES, THAT MAY BE SENT, ARE CURRENTLY AVAILABLE.

IMS/VS TRANSMITS EOT TO TERMINATE TRANSMISSION.

Non-ask type, transmission limit

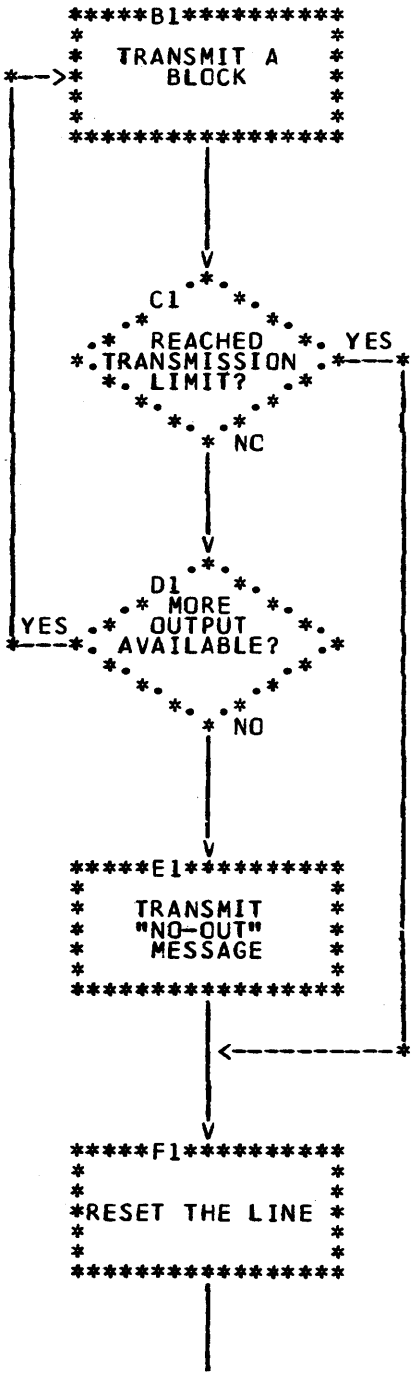


IMS/VS WILL START TRANSMITTING OUTPUT AS SOON AS THE LINE IS AVAILABLE AFTER AN OUTPUT MESSAGE HAS BEEN ENQUEUED.

IMS/VS CONTINUES TRANSMITTING AS LONG AS THERE IS OUTPUT AVAILABLE, THAT MAY BE SENT, UP TO A USER SPECIFIED NUMBER OF TRANSMISSION BLOCKS.
IMS/VS TRANSMITS ONLY ONE MESSAGE FOR A GIVEN TERMINAL IDENTIFIER REGARDLESS OF THE NUMBER OF MESSAGES ENQUEUED.

IMS/VS TRANSMITS EOT TO TERMINATE TRANSMISSION.

Ask type, transmission limit



IMS/VS WILL START TRANSMITTING OUTPUT AS SCON AS THE LINE IS AVAILABLE AFTER AN "ASK" FOR OUTPUT MESSAGE HAS BEEN RECEIVED.

IMS/VS CONTINUES TRANSMITTING AS LONG AS THERE IS OUTPUT AVAILABLE, THAT MAY BE SENT, UP TO A USER SPECIFIED NUMBER OF TRANSMISSION BLOCKS. IMS/VS TRANSMITS ONLY ONE MESSAGE FOR A GIVEN TERMINAL IDENTIFIER REGARDLESS OF THE NUMBER OF MESSAGES ENQUEUED.

IMS/VS TRANSMITS A SYNCHRONIZATION BLCCK INDICATING TO THE REMOTE CPU THAT NO OUTPUT MESSAGES, THAT MAY BE SENT, ARE CURRENTLY AVAILABLE.

IMS/VS TRANSMITS EOT TO TERMINATE TRANSMISSION.

System/7 Start/Stop Transmission Code Modes

IMS/VS requires synchronization blocks to be transmitted using the pseudo-binary PTTC/EBCD transmission code. This code is described in the System/7 Functional Characteristics Manual, GA34-0003.

Data blocks are transmitted using either the standard PTTC/EBCD transmission code or the pseudo-binary PTTC/EBCD transmission code. IMS/VS accepts either code on input and scans the output data to determine if the block contains any characters that cannot be transmitted using the standard PTTC/EBCD transmission code. If such characters are found, the block is converted to pseudo-binary PTTC/EBCD. Otherwise, the message is translated as standard PTTC/EBCD transmission code.

IMS/VS allows the user to specify at IMS/VS system definition, on a per station basis, that all data blocks should be transmitted in one of the above transmission codes. If all data blocks are to be transmitted in the standard PTTC/EBCD code, all characters that cannot be transmitted in that code are replaced by a colon.

The output buffer size specified by the user at IMS/VS system definition is doubled to allow for conversion to pseudo-binary PTTC/EBCD, unless the user specifies that all data blocks are to be transmitted using the standard PTTC/EBCD transmission code.

Supported System/7 Start/Stop Line Types

IMS/VS allows a System/7 to be attached on a nonswitched contention line or a nonswitched polled line. A polled line may be polled using programmed polling or autopoll.

IMS/VS can control a polled line and therefore initiate output, if allowed to, at any time data transfer is not taking place without a potential loss of data and without System/7 intervention. To try to avoid errors caused by loss of data on a contention line, some of the responsibility for keeping communication open is dependent upon the System/7 program. IMS/VS issues a read when output is not available to send and this read must be terminated by transmission from the System/7. Since there is no indication of whether, after receiving a block, IMS/VS intends to transmit or return to read, unless the System/7 is defined as ask type, it is recommended that a System/7, attached on a contention line, be defined as ask type. The receipt of the output not available (NO-OUT) message informs the System/7 program that IMS/VS, immediately following the completion of this message, is issuing a read.

Supported System/7 BSC Line Types

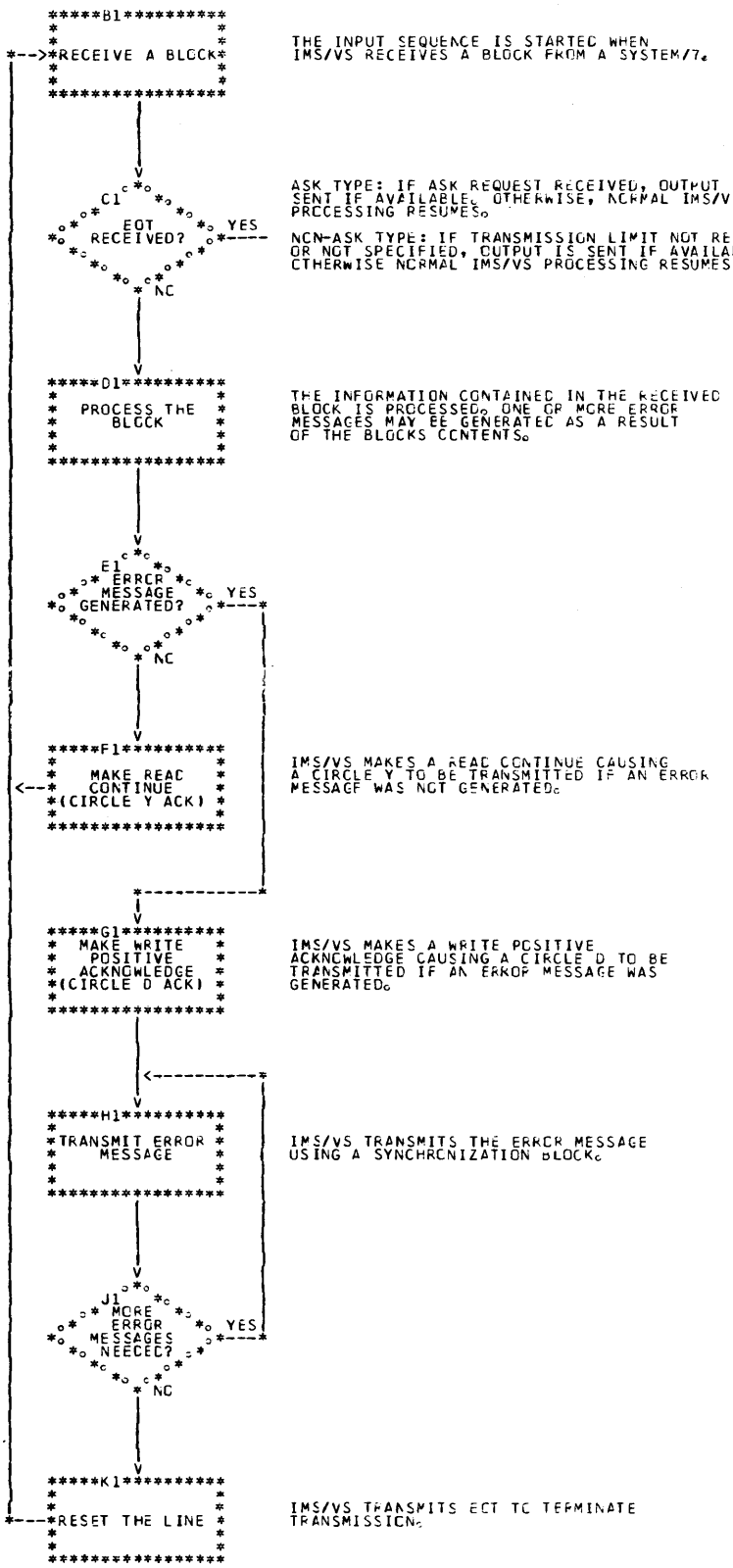
IMS/VS allows a System/7 to be attached on a nonswitched contention or polled line. IMS/VS is defined as the controlling station. All transmissions must be in BSC EBCDIC transparent mode.

Process Controlling System/7

Since there is no facility to prevent an IMS/VS shutdown checkpoint while a process controlling System/7 is active, the System/7 should transmit a message to the IMS/VS master terminal operator, when communication is started, informing the operator that a process controlling machine is attached and that the operator should not issue a shutdown checkpoint until informed that the process controlling machine is either stopped or stoppable.

IMS/VS Processing of a Block Transmitted Start/Stop from a System/7

The flowchart below shows how IMS/VS processes a transmission block received from a System/7.



CONSIDERATIONS UNIQUE TO SYSTEM/3

IMS/VS support of the System/3 is designed to provide a high degree of flexibility in function but is consistent with the main storage constraint inherent in smaller computers.

While IMS/VS IRSS does not require it, it is anticipated that System/3 programs designed to interface with IMS/VS will take advantage of the ask-type station facility described under "System Definition." This facility allows the System/3 programmer to allocate his main storage resource only when he is ready to accept data from IMS/VS; thus alleviating the requirement for a larger, permanently-dedicated buffer area.

Transmission of data in the EBCDIC transparency mode allows all types of data to be transmitted from an IMS/VS application program. This could save additional storage or programming in the System/3.

If the System/3 is used as a subhost for locally attached terminals, using either the MLTA (for start-stop) or MLMP (for BSC) features of the System/3 Disk System Control Program, the IRSS provides each of these terminals direct access to an IMS/VS system with the additional advantage of a common I/O interface.

Though IMS/VS IRSS supplies a large amount of status type information to the System/3, the System/3 programmer does not need to design his application to process all types. Consequently he can realize a savings in main storage or programming within the System/3.

To fully utilize the features provided by IRSS to the System/3, the System/3 programmer should design his application to use the Disk System Control Program with the BSCA Multiline Multipoint feature. This support allows the user to directly control the line discipline and to recognize many types of responses from IMS/VS.

Design of the System/3 Application Using MLMP

To utilize MLMP support in the System/3 to communicate with IMS/VS through the IRSS, the System/3 user should:

1. Define two BSCA files, one for transmit and one for receive.

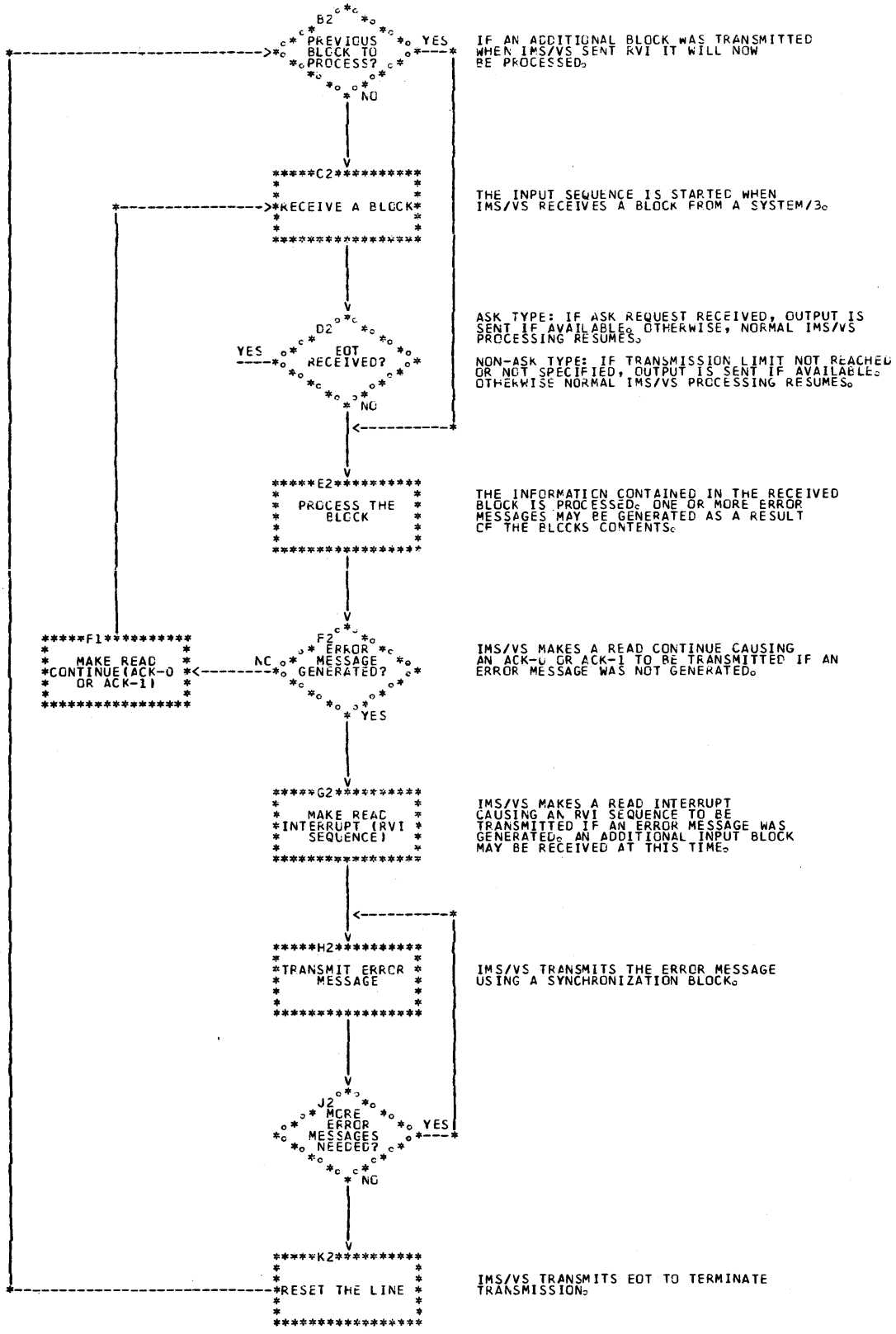
2. The transmit file should be single buffered to prevent more than one block being transmitted after a reverse interrupt (RVI) indicator has been sent by IMS/VS. The reverse interrupt indicator must be defined and recognized for this file.

3. It is recommended that the System/3 user utilize the get-block and put-block modes of the GET and PUT macros. This is recommended because IMS/VS IRSS data structures do not normally lend themselves to the record separator mode of deblocking (unless of course the data to be transmitted can be guaranteed not to contain a particular character).

4. If multiple System/3s are to be multidropped on a single communication line, it is important for the System/3 application program to take the necessary steps to assure a negative response to polling when communication is inactive between the System/3 and IMS/VS. This usually requires the issuance of a GET type operation in the System/3 and the use of the cancel function when the next direction of transmission is to be a PUT type operation for the System/3.

IMS/VS Processing of a Block Transmitted from a System/3 or a BSC System/7

The flowchart below shows how IMS/VS processes a transmission block received from a System/3.



CONTROL OF THE DB/DC SYSTEM

SECURITY AND PRIVACY

In government, education, and private industry, there has been a long-term trend toward centralization of data. Data base/data communication systems like IMS/VS are accelerating this trend. More and more information is being consolidated about individuals, businesses, governments, and institutions. It is predictable that this data will grow into vast, interconnected national or even international data networks. As consolidation increases, so do the fears of individuals and businesses that their rights to privacy and to protection of privileged and confidential information will be compromised. Equally important are the concerns for protecting data from malicious or accidental modification.

It is the objective of IMS/VS to provide safeguards through which access to data may be limited. The mechanics of the safeguard system can be used to administer security and privacy policies. Administration is accomplished by careful interpretation of policy in system and application design, and into parameters and control statements used for:

- System definition
- Program specification block generation
- Data base description generation
- Security maintenance program
- Statistical analysis program

Authorizing Use of Terminal Commands

In the newly generated DB/DC System, only a basic subset of the terminal commands is protected against unauthorized entry. The terminal command language, if properly used, provides flexible control of the DB/DC system while it is in operation. The commands permit alteration of the status of many system resources. Among the functions that can be performed, several are vital to the integrity of the security safeguards themselves. Other functions, although not threatening the security safeguards, are capable of crippling operations or shutting down the entire DB/DC system. Freedom from concern over capricious or malicious use of the commands can be achieved by designing and implementing a command authorization plan. The tool provided by IMS/VS to implement this plan is the Security Maintenance Program. Refer to the IMS/VS Installation Guide for information about the use of this program.

Commands are protected individually. Each command verb performs a separate function. However, it can perform that same function for many kinds of system resources. For example, the /START command function can be performed for communication lines and terminals, making them available for use. The /START command can also specify as its object programs, data bases, logical terminals, and transactions. The command function, as represented by its associated verb, is what the security safeguards protect.

The objects of command verb action are represented by keywords in the command language. These objects are, in general, a class of resources within the DB/DC system such as data bases, terminals, and

programs. A specified member or all of a class of resources can be the object of a command verb.

Command functions can be protected against unauthorized use in three ways: by permitting the command verb to be entered only from certain logical terminals, by requiring that a password be entered with the command verb, or both. Some objects of commands can be protected against unauthorized action by requiring a password to be entered with the parameter. The protection of the command object is controlled by assigning a protected attribute to each member of the class of objects to be protected.

For example, to require a password be entered to alter the status of logical terminals (LTERM) 111, 222, and 333, one must specify to the Security Maintenance Program a password for each terminal. If LTERMs 111, 222, and 333 are the only LTERMs in the system and all are protected by the same password, then the object keyword is secured throughout the system. If, however, it is only necessary to protect LTERM 222, then the LTERM keyword can be used without a password on LTERMs 111 and 333.

Another way to look at using safeguards to protect the command language is by individual user profile. Equate passwords to user sign-on or identification codes. An authorization plan can be developed that authorizes each user to use, individually, a set of command functions. That authorization could be localized geographically through restricting entry of the command verb to a group of logical terminals.

A class profile system could be used. For example, password x validates the use of four command verbs, YYYY validates three different command verbs. ZZZZ however is valid not only for the commands protected by x and YYYY, but also authorizes the holder to enter an immediate DB/DC system shutdown command from any terminal.

Although all documentation emphasizes the identity and importance of the master terminal, there are only a few characteristics that make it unique in the DB/DC system. It is the only logical terminal to which messages about the operational status of the system are automatically routed. It and the System/370 console are the only terminals through which the DB/DC system can restarted. The control that the master terminal exercises over the system is made possible through the IMS/VS command language.

A thorough examination of the commands, the system to be protected, the requirements of users, and the objectives of your security and privacy policy will provide guidance in the distribution of authority to use the command language. Refer to the IMS/VS Operator's Reference Manual for a complete description of the commands and the command language.

Restricting Entry of Transaction Codes

Through the entry of transaction codes, the terminal operator identifies the destination of the text or data that follows. When one examines the syntax of input messages, as defined by IMS/VS, it can be seen that all entries from terminals are classified by means of an identity code. The precise rules which govern the recognition of identity codes by IMS/VS vary from device to device. In general, there are two levels of recognition. The first level establishes whether the entered data is a command by reserving the initial character /. The first character of every input segment is examined for a /. If one is present, the segment is treated as a command segment. Input destined to a program or logical terminal must not contain a / as the first character of any segment. The second, or operational, level

verifies that the identity code is known to IMS/VS. If it is known, then it and the text that follows are classified based upon the attributes of the identity code. If the code was defined during system definition as a transaction code, the message is routed to the application program which is to process it. If the code was defined as a logical terminal name, then the message is routed to the physical terminal to which that logical terminal is attached. It becomes a message switch transaction.

The possible contents of a message destined for an application processing program, the actual functions which are performed by that program, and the content of any output subsequently generated by that program are unknown to IMS/VS. Because applications may deal with critical or private matters, safeguarding tools are provided by the system to prevent unauthorized entry of transaction codes, and hence unauthorized use of application program functions.

The entry of each transaction code can be limited to any one or any group of logical terminals in the system. Depending on the ratio of secured to unsecured transaction codes, the authorization plan that is developed can be inclusive or exclusive. To use the Security Maintenance Program effectively, the operational plan must be inclusive. That is, you must specify the transaction codes which are to be secured. There is no provision for specifying those which are not to be secured. There are, however, alternative views of the plan that can be helpful. You can look at the transaction codes as being authorized for entry from a list of logical terminals. Or, think of each logical terminal as being authorized to enter a list of transaction codes. Either viewpoint may be translated as easily into the operational input statements that describe what you want to do with the Security Maintenance Program. However, the number of input statements can vary substantially between the two viewpoints. If, for example, you have one transaction code you want to authorize from five logical terminals, six input statements are required. Conversely, if you specify five logical terminals and authorize the same transaction code from each, ten input statements are required.

Passwords can be used instead of, or in addition to, logical terminals to limit transaction entry. The security provided by passwords can be specified and viewed in the same manner as that provided through logical terminals. Without appropriate hardware features on physical terminals however, there is exposure to possible compromise of the passwords.

Display Bypass Feature

IMS/VS does not provide a software feature to blank out or obliterate passwords from the terminal device display media after they are accepted. It does remove passwords from messages prior to recording them on the log.

Most hard copy key-driven terminals have a feature which permits characters to be entered without displaying them. This feature is the bypass feature. Ordinarily, a terminal with this feature is operated continuously in display or bypass mode. If passwords are to be used to support security requirements, this feature is a necessity.

The bypass feature can be used operationally; that is, by establishing standards for protection not only of passwords, but also of command verbs, commands, transaction codes, and text.

Limiting Access to Data

Through centralized control over the content of Data Base Definitions, Program Specification Blocks, and the libraries which they reside, an effective scheme of protection attributes can be assigned to data. This assignment is made relative to each application program which has access to the data base. The smallest unit of data which may be so protected is the segment. The basic actions that can be authorized are:

- None -- no access to segment type.
- Read -- segment type may only be retrieved.

One or more of the following additional actions combined with read can be authorized:

- Add -- new occurrences of segment type can be inserted.
- Update -- an existing occurrence of a segment type can be replaced.
- Delete -- an existing occurrence of a segment type can be deleted.

Although access authorization is declared at the program level, if necessary, enforcement of the authorization can be made to appear at the transaction code or individual hierarchical level of a data base. If only one transaction code is associated with a particular program, then the access authorization has been promoted to the transaction level. Through use of passwords or through use of the transaction code and terminal bypass feature, access authorization can be promoted to the individual level.

For information about specifying access authorization, refer to "PSB Generation" in the IMS/VS Utilities Reference Manual. The control statements through which data access is authorized are PCB and SENSEG.

Users of the Interactive Query Facility (IQF) feature must name each PCB associated with a data base, and use this name when entering a query at the terminal. By having multiple IQF PSBs, and assigning a unique transaction code or set of transaction codes to each PSB, the IQF user can restrict the domain of IQF processing to specific subsets of all the installation's data bases. Each IQF PSB is thus treated as though it were a different application, and the IMS/VS security capabilities are easily extended to users of IQF.

VIOLATION CONTROL

As stated above, IMS/VS provides a terminal security arrangement or the user can create his own with the Security Maintenance Program. The amount of control the user wants exercised over the security arrangement can be defined during IMS/VS system definition.

- Password and/or terminal security can be defined as a system default; each time IMS/VS is initialized, the security tables are loaded automatically.
- The master terminal operator can be allowed to negate, or be prohibited from negating, the automatic loading of security tables. System initialization prevents an operator from starting the system without security if he is not authorized to do so.

IMS/VS records all security violation attempts on the IMS/VS system log tape. The violations recorded are:

- Input message from an unauthorized terminal
- Password omitted when one is required
- Password incorrect for authorization
- Misspelled password

IMS/VS rejects invalid input messages by sending a message to the terminal sending the message, and logging the violation.

The log tape provides an audit trail to look into possible security problems. If more immediate action is desired, the user can request notification to the master terminal at the time of the violation. Since the number of violations for a large network may be high due to misspelled passwords, transaction codes, commands, etc., the user can specify a threshold for notification such that the master terminal is not notified until the specified number of violations occur without a valid input from a given terminal. This eliminates or reduces the number of notifications due simply to operator error, while still providing evidence of real attempts to avoid security safeguards.

3270 Switched Terminal Security

3270s on a switched line can have their hardware IDs verified for authorization to access IMS/VS by use of the IDLIST macro. For additional terminal security information concerning 3270 switched terminals, refer to the IMS/VS Installation Guide.

IMS/VS DC MONITOR

The IMS/VS DC monitor is a tool for collecting performance data to investigate specific application designs, data base designs, and resource allocations. It consists of a monitor module, and a Monitor Report Print program. When activated, it analyzes and records the internal activities of the IMS/VS DB/DC system. The monitor report print program is processed offline to produce reports that summarize and categorize, at various levels of detail, the information recorded by the monitor module. The actions required to activate the monitor module are described in the IMS/VS Operator's Reference Manual. The monitor report print program is described in the IMS/VS Utilities Reference Manual.

The monitor module collects data from IMS/VS DC control blocks during operation of the online system, with minimum interference to the system, and records the data on an independent data set. The monitor remains resident and is activated and deactivated through master terminal control.

Following are recommendations for use of the IMS/VS DC monitor:

- Collecting data -- The IMS/VS DC monitor enables an IMS/VS DC user to collect performance data to assist in analyzing an existing IMS/VS online system. The amount of data collected and the analysis time to understand the report output suggest short traces during various time periods. Reports produced from profiles of a time period considered as normal can be used as a profile and compared with reports produced during a time period characterized by unusual responses.

- Tuning system -- The IMS/VS DC monitor can be used to quantify the effect of actual changes to data base structures, program characteristics, data set placement, pool sizes, number of message processing regions, transactions, and message region class scheduling.

- Testing application -- In the final testing of new or revised applications, the IMS/VS DC monitor can be useful in validating the internal operation of the programs and data bases. For example, the programmer thought a specific DL/I call could be satisfied with a single I/O retrieval, yet the DL/I call report indicated a large data base scan as shown by many IWAITs. Investigation of such items could assist in determining whether a new or revised application meets the performance objectives. Data contained in the reports may also assist in defining the resources required by an application program.

- Integrating applications -- The IMS/VS DC monitor can be used to determine the effects on the IMS/VS production system as new applications are merged from a test system to the production system. One of the basic problems in integration of new applications into an existing system is the requirement of re-tuning options in the production system, such as data set placement and buffer pool sizes, as discussed above in the tuning of the system.

- Communicating criteria -- If the above recommendations are implemented, then data is collected to establish a performance base, profiles are available for the problem periods, the system is tuned for the production and test systems, and applications are tested and merged into the IMS/VS production system with an understanding of their effects and interactions. Thus, the IMS/VS DC monitor reports can be used as a basis to communicate and define performance.

USING THE 3850 MASS STORAGE SYSTEM (MSS) FOR DB/DC PROCESSING

IMS/VS supports the IBM 3850 Mass Storage System (MSS) through its normal OS/VS1 and OS/VS2 interface. MSS extends the capacity of 3330 Disk Storage. The uses of MSS with IMS/VS are:

- As a residence device for batch and online data bases

- For the development and testing of new applications

- As the storage media for historical or cutoff versions of data base

- As a centrally controlled location for data (DB/DC and other types) in a data processing system

Each of the preceding uses requires an understanding of the characteristics of IMS/VS and MSS that could affect an installation. This section presents the information needed to understand and take advantage of these characteristics. Design considerations and guidelines related to the following topics are described in detail.

- Using MSS in a batch IMS/VS environment.

- Using MSS in an online IMS/VS environment. Three different online environments, ranging from simple to complex, are described.

- Sharing staging space.

- Data base organization and access methods.
- Using the additional capacity of the MSS with IMS/VS.

In this section, MSS is described only as it relates to IMS/VS, even though the considerations and guidelines generally apply to any DB/DC system. Since no attempt is made to explain the facilities of MSS and its operating system support, you should be familiar with the following publications:

- Introduction to the IBM 3850 Mass Storage System (MSS)
- OS/VS Mass Storage System (MSS) Planning Guide
- IBM 3850 Mass Storage System (MSS) Principles of Operation

TERMINOLOGY

The terms "stage," "bind," and "cylinderfault" are used in this section. Stage, bind, and cylinderfault specify how data that is stored on an MSS volume is to be staged.

Stage: Stage specifies that the data is to be staged from mass storage to a direct-access staging device when the cluster or component is opened. If it can't be staged at the time the cluster or component is opened, because of other staging activity, data is staged as a processing program needs it through page fault.

Bind: Bind specifies that the data is not only to be staged but also to be kept on the direct-access staging device until it is closed. If it can't be staged at open time because of other staging activity and if there is staging pack space available for the entire data set, data is staged as a processing program needs it through page fault.

Cylinderfault: Cylinderfault specifies that the data is not to be staged when the data set is opened. It will be staged as the processing program needs it.

As a general rule, MSS can be used in a batch IMS/VS environment in stage, bind, or cylinderfault mode. In an online environment, it is recommended that the data base reside on real DASD or that it be staged with the bind option if transaction throughput and response time are critical.

IMS/VS BATCH ENVIRONMENT

Figure 2-8 shows the use of MSS in an IMS/VS batch environment for data base residence. A batch environment that includes MSS allows:

- Operational control of an entire system of data bases and files through MSS.
- An extra dimension of flexibility because processing of large data bases can be done with fewer staging drives compared to the number that would be required if real drives were used. In this environment, it might be more efficient to do some types of processing at a reduced throughput rate and save the investment in additional disk drives.
- The testing of large data base applications can be done using fewer staging drives than would normally be required in a production environment; this can be tailored to meet the needs of an installation.

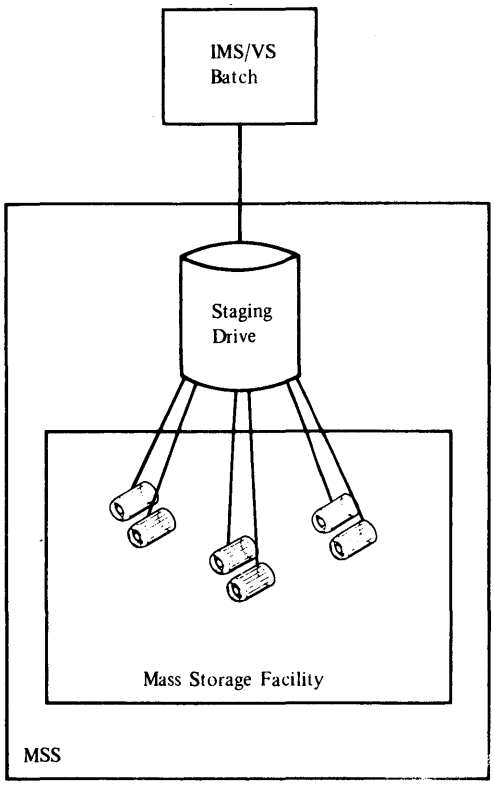


Figure 2-8. MSS in an IMS/VS Batch Environment

The paragraphs that follow describe how MSS can be used to advantage in an IMS/VS batch environment with certain disk drive saving opportunities. You may know from experience that you will process only a fraction of a very large data base. For example, only 10% of an insurance data base might have policy updates, claims, or billing activity in the course of a day. If the full data base occupied 10 disk drives, then some number of staging drives equal to or less than 10 might be sufficient to handle the day's activity in cylinderfault mode. The exact number of staging drives required would depend on data storage patterns, reuse of staging drive space, the distribution of data across cylinder pages, and MSS staging algorithms. The MSS publications referenced earlier contain detailed information on these factors.

If you know the reference patterns that your applications require, then possibly only a DL/I data set group has to be staged for processing.

Month-end cutoff processing of a data base also lends itself to the use of MSS with IMS/VS batch. In a non-MSS environment, a month-end cutoff copy of a data base is normally gotten by copying the data base to tape. The tape is later restored to disk so month-end reports, etc., can be written from the month-end copy of the data base. The MSS allows you to destage a month-end cutoff to MSS cartridges; later stage the data from the cartridges, possibly to a subset of data base staging drives; and process the month-end cutoff data without having to go through disk to tape and tape to disk dumps and restores.

IMS/VS ONLINE (DB/DC) ENVIRONMENT

Just as MSS offers an added dimension of flexibility in an IMS/VS batch environment, there are added opportunities in an online environment, but planning is far more critical. An online DB/DC environment usually includes certain transactions that require fast response and throughput as well as fast recovery. These transactions will be referred to as critical transactions.

In the online environment the following assumptions are made:

- Response time and transaction throughput for critical transactions should be the same whether or not MSS is part of the operational environment.
- Recovery time in the event of an IMS/VS, OS/VS, or hardware failure should be the same in an MSS environment as in a non-MSS environment.

To maintain the response and recovery time criterion required by your installation and still use MSS effectively in an online IMS/VS environment requires that you consider the following factors during planning:

- Logging and restart processing
- Sharing of staging drives
- Sharing of data bases
- Update activity
- Initialization and prestaging

The following contains considerations and guidelines for these factors in each of three different online environments: (1) IMS/VS online using bound data or real DASD and no batch applications, (2) IMS/VS online bound data using bound data and/or real DASD with IMS/VS batch, and (3) IMS/VS online using some bound and some unbound data.

Additional factors, data base organization and access methods, apply equally to all environments and will be described as a separate topic later in this section.

IMS/VS Online Using Bound Data and/or DASD without Batch

This is the simplest online environment to plan for. IMS/VS logs the system activity and runs recovery as necessary using the log tape much the same as in a non-MSS environment. Because all data is either mounted and bound on staging volumes or residing on real DASD, there is minimal planning necessary for sharing staging DASD. The sharing of data bases by multiple Message Processing Programs (MPPs) requires the same planning as in a non-MSS environment. Figure 2-9 shows this kind of environment.

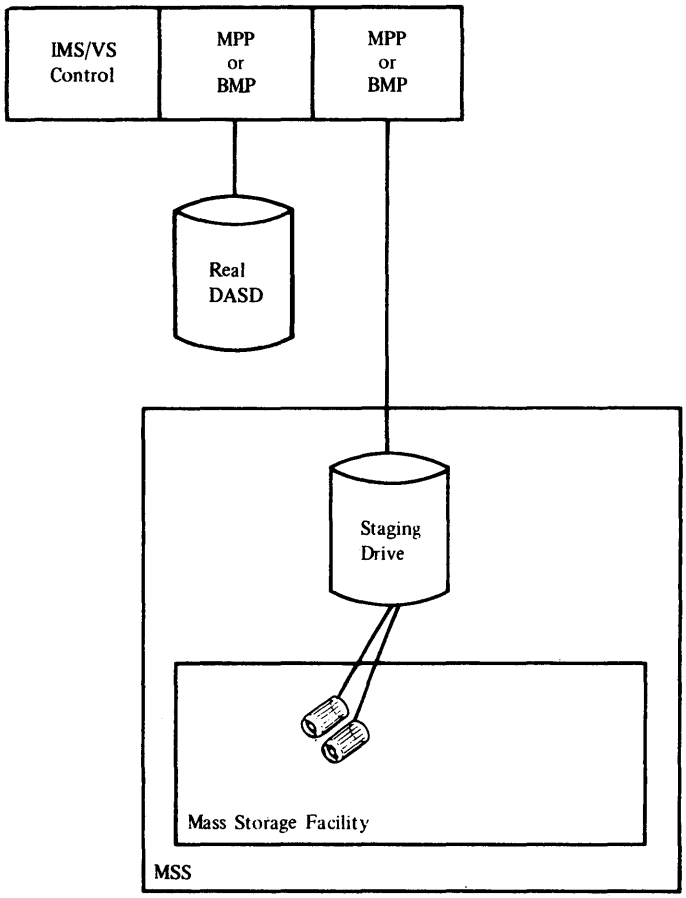


Figure 2-9. MSS in an IMS/VS Online Environment with Bound Data

Initialization and prestaging must be planned if real DASD is not used. If the data is to be bound, it is important to ensure that the data be staged and bound on the staging packs before starting IMS/VS online operations. If the staging packs are used to hold only the bound data, and never used to hold other data, then the data is staged and bound once, and it does not have to be restaged and bound each time IMS/VS is started. However, if the staging packs are used for other work (for example, during off-shift operations), the staging and binding of IMS/VS online data must be scheduled before starting IMS/VS at the beginning of each work day. The process of staging and binding data can be a lengthy process requiring careful scheduling to ensure that it completes prior to starting online IMS/VS operations. Also, there should be sufficient staging pack space available to hold all the data to be staged and bound.

Methods of staging bound data prior to data set OPEN processing are contained in "How to Use the Additional Capacity of MSS with IMS/VS" later in this section.

IMS/VS Online Using Bound Data and/or Real DASD with IMS/VS Batch

All planning considerations are the same as in the previous environment if the IMS/VS batch is also using bound data and/or real DASD, which is unlikely. If IMS/VS batch is cylinderfaulting to the staging volumes, there could be a delay as IMS/VS batch and online contend for staging pack space. As a general rule, if IMS/VS batch

will cause contention for staging pack space, stage and bind the critical online data before the batch operations begin.

Although this example uses IMS/VS online and batch, the same considerations apply to any activity (OS/VS batch, TSO, etc.) running with an online IMS/VS system. See "Sharing of Staging Space" in this section for additional information on this topic. Figure 2-10 shows this kind of environment.

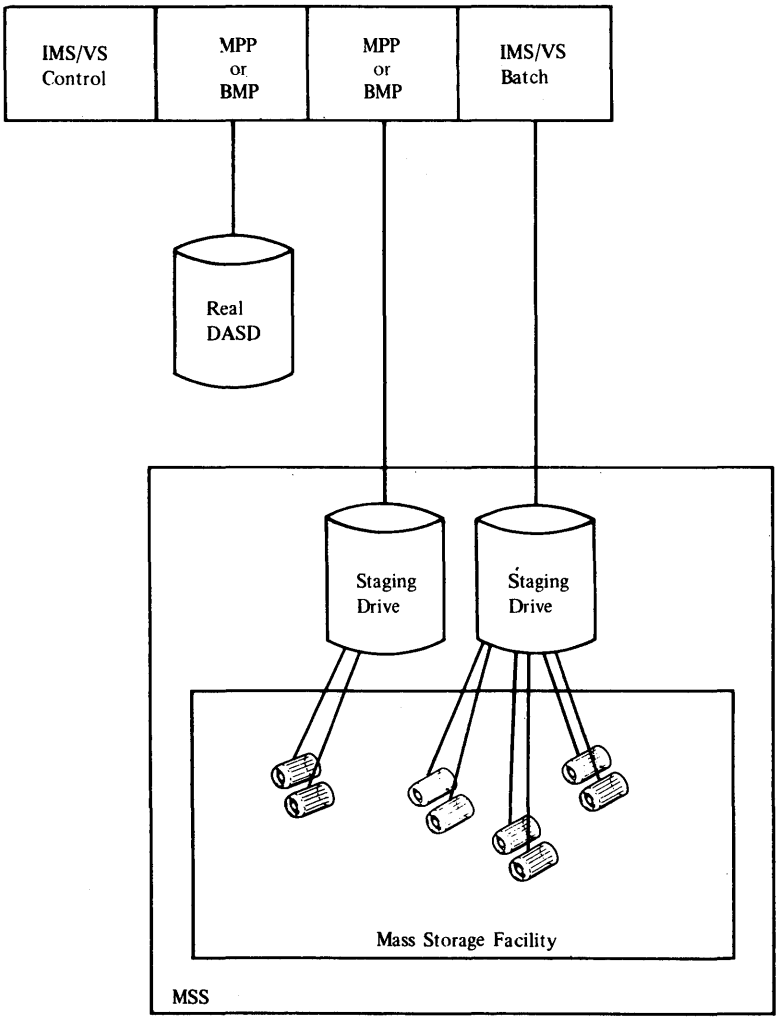


Figure 2-10. MSS in an IMS/VS Online and Batch Environment

Recovery should be the same in this environment as in a non-MSS environment because IMS/VS online has its own separate logging facility and data bases are not shared between online and batch operations. Operational procedures may differ, however. For example, where batch backout in a non-MSS environment involves quiescing activity to a data base, closing the log tape, and moving the data base and log tape to a different address space or CPU, the MSS environment involves quiescing the activity to the data base, closing the log tape, demounting the virtual volume, and remounting the virtual volume for batch backout. Again, this procedure varies depending on where batch backout is to be run: in the same host or in another host of a multihost system. Destage and restage may or may not be necessary at batch backout time

depending on the configuration and the virtual unit address (VUA) specified in the mount order for the mass storage volume at initial IMS/VS load.

Even though staging packs with their virtual volume data cannot be physically moved from one staging disk drive to another, as is often done for batch backout with real DASD, the data can be moved (destaged and staged to another disk drive) to accomplish the same purpose.

IMS/VS Online and Batch Using Some Bound and Some Nonbound Data

This environment requires considerable planning. Again, the objective is to work in this environment and for certain critical transactions to maintain the same response time and throughput as well as recovery time, in the event of failure as would be experienced in a non-MSS environment. Figure 2-11 shows this kind of environment.

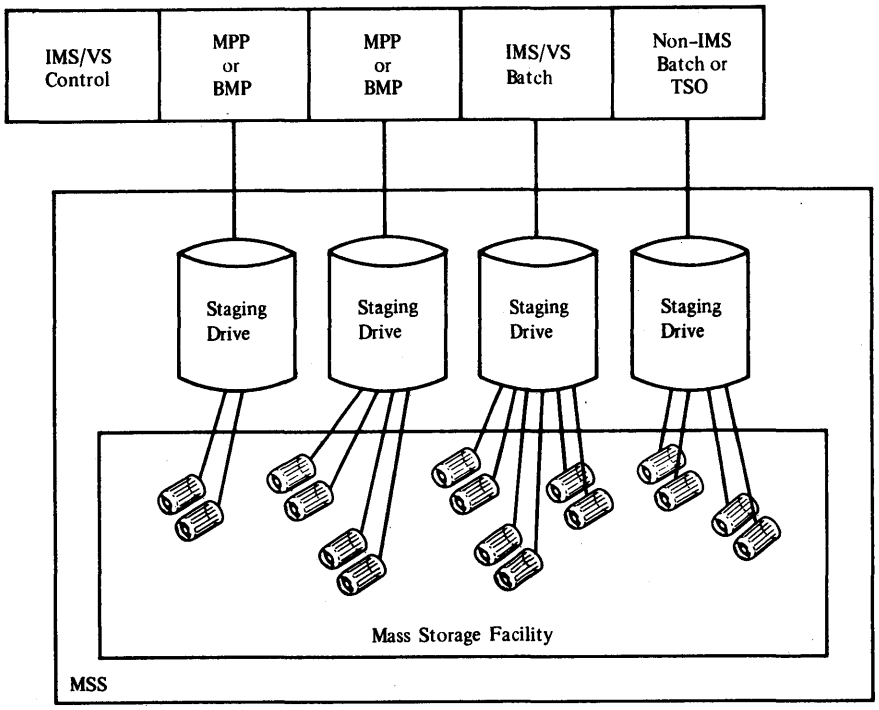


Figure 2-11. MSS with IMS/VS Online and Batch and Non-IMS/VS Data

The data to be bound should be staged onto the staging packs before IMS/VS transaction processing begins assuming there will be contention for the staging packs. Next, data sets or data set groups that will be referenced should be staged onto staging packs. Also, selected portions of a data base that will definitely be referenced, if that can be determined, can be staged onto the staging packs. This might be accomplished by starting a batch message processing (BMP) program that issues DL/I calls causing selected data to be staged in cylinderfault mode. This essentially is prestaging. Prestaging data to staging packs is somewhat analogous to putting and keeping data in the data base buffer pool for future reference.

Prestaging through an IMS/VS application program can be very effective in a DL/I environment because it allows the user to stage

selected portions of the DL/I data base using the standard DL/I facilities.

Frequently the data base reference patterns cannot be determined. For example, it is almost impossible to determine which customer of an insurance company will phone to report a theft. Prestaging, in this case, could be accomplished by entering a simple transaction that does no more than read the data from the data base using DL/I for the customer phoning the report. This would cause cylinderfault staging of policy information about that customer while information for a more complex transaction is being gathered based on the conversation between the insurance agent and his customer.

This environment assumes a mix of transactions in the system, some critical transactions requiring fast response and throughput as well as fast recovery in the event of failure, and transactions that are not so critical.

Since both the critical and the noncritical transactions share the same log tape used for emergency restart, and since critical transactions require a fast restart, it is important that cylinderfaults do not occur during emergency restart. This can only be guaranteed in this environment if all data required at restart time is available on real DASD or staging packs. This means that either there must be enough staging pack space available so data is never destaged to make room for other data, or that there is no update activity to data bases running in cylinderfault mode. Data base updates could cause cylinderfaulting during emergency restart if the changed data base records had been destaged to MSS cartridges to make room for other data on the staging packs and some of the data that had been destaged was required during restart. Emergency restart of critical transaction activity would be delayed by cylinderfaulting of noncritical transaction activity because backout is done serially.

If only BMPs are using online data bases in cylinderfault mode then specifying NOBMP at emergency restart would eliminate the backout of BMP updates and the delay to emergency restart caused by cylinderfaulting.

In addition, batch backout and program isolation dynamic backout will take longer if cylinderfaulting must occur during backout. Similar guidelines apply to batch backout and program isolation dynamic backout as apply to emergency restart with the exception of NOBMP, which applies only to emergency restart.

The preceding point regarding no update activity for data bases running in cylinderfault mode may appear restrictive. It is only a recommended guideline to ensure fast emergency restart. It is also tunable where the number of emergency restarts or batch backouts, the number of updates, and the degree to which staging pack space is overcommitted are the tunable considerations. For example, it may be satisfactory to allow updates to data bases running in cylinderfault mode depending on the number of updates per syncpoint or checkpoint, or if emergency restarts are infrequent.

Data bases that are read to gather data to generate reports are likely candidates for use of shared, overcommitted staging pack space. An example here might be month-end accounting reports that are generated from month-end cutoff versions of a data base.

In this environment, transactions requiring fast response and throughput should not be scheduled to run in the same message processing region as transactions that could require cylinderfault staging. The critical transactions could end up being queued until transactions requiring a call for data on MSS have completed.

Message class scheduling affects the queuing of transactions in IMS/VS. Transactions requiring fast response and throughput should be assigned to a separate class from transactions that could require cylinderfaulting. The classes should then be assigned to separate regions when IMS/VS is started.

The MSS staging drive group concept can be used for added tuning in this environment. Not all data bases have to use the same level of overcommitment. Staging drives can be divided into staging drive groups so that there may be more contention for staging drive space for very low priority work and less contention, or even no contention, for higher priority work. Activity from work outside the IMS/VS online system, such as batch work, could adversely affect the IMS/VS online system if all work shared the same staging drive group and the scheduling of work was not otherwise controlled.

The sharing of data bases has to be well planned in this environment. Avoid having a transaction that requires fast response use data from both a data base on a bound volume or real DASD and also from a data base residing on overcommitted staging packs. Also avoid a logical relationship between these same two data bases where processing, especially insert or delete processing, could slow down processing of the bound or real DASD data base.

Also to be avoided, but less obvious, is a situation with the following or equivalent characteristics:

- Program A scheduled by transaction A requires fast response.
- Program B scheduled by transaction B does not require fast response.
- Program A uses data base A which is on bound staging packs.
- Program B uses data base B which is on overcommitted staging packs and also uses data base A.

Figure 2-12 shows this situation. It is possible that a program B could impact program A's processing and it would depend on the extent to which program B was holding data from data base A while staging data base B, and program A required the same data being held by program B.

This may be an unusual situation but it points out that application scheduling and use of data bases in this environment should be well planned to avoid less obvious throughput problems. Again, the same caution regarding doing updates to a data base on overcommitted staging packs, as described earlier, applies in the above example.

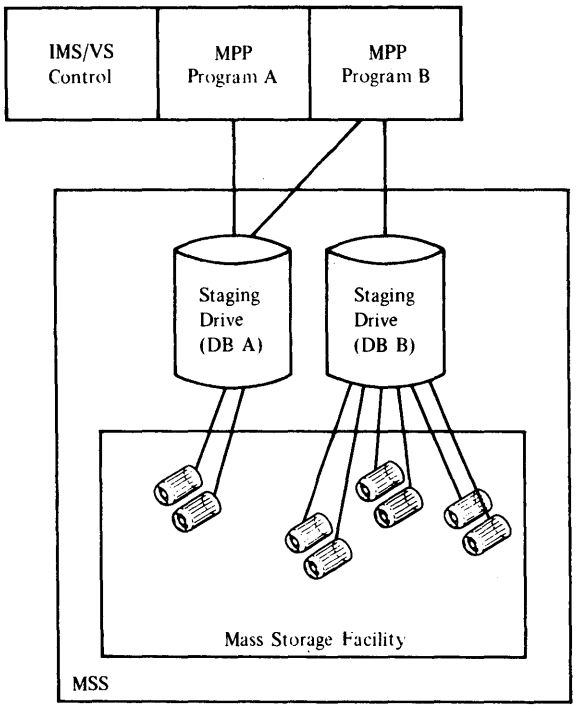


Figure 2-12. MSS in an IMS/VS Environment Using Shared Data Bases

SHARING OF STAGING SPACE

Closely related to the sharing of a data base is the sharing of staging space. Staging space sharing considerations were partially described under the second and third environments earlier in this section. Well planned use of the MSS staging drive groups is a valuable way to control the amount of staging pack space allotted to various applications or data bases.

When allocating staging space for IMS/VS, view the entire system as known to MSS. This could include not only IMS/VS online and batch systems, but non-IMS batch or TSO, for example. Staging pack space may be known to MSS across multiple CPUs. It is necessary, then, to consider possible interference to IMS/VS processing from outside of IMS/VS itself. MSS staging drive groups can be used to help control unwanted staging from shared staging space. This should be well planned, because there could be interactions between the IMS/VS and non-IMS/VS environments. The use of staging drive groups is further described under the topic "How to Use the Additional Capacity of MSS with IMS/VS."

DATA BASE ORGANIZATION AND ACCESS METHOD

ISAM data sets can only be accessed in the cylinderfault mode. This means that the ISAM portion of an ISAM/OSAM data base cannot be staged or bound at OPEN time; the time of the first DL/I call. Staging of a cylinder of data takes place only as the result of a DL/I call for data in that cylinder. It then follows that if ISAM/OSAM is used, and cylinderfaulting presents an unwanted delay, some form of prestaging or, better yet, VSAM should be used.

OSAM data sets can only be defined with the stage attribute. Because OSAM uses EXCPVR, the operation of MSS with EXCPVR applies to OSAM.

The OSAM portion of an ISAM/OSAM data base will be staged at OPEN time because that is when the data sets using EXCPVR are staged.

The entire extent of an OSAM data set will be staged, even at initial load time. An exception to this occurs when IMS/VS uses QSAM to write an OSAM data set as in recovery when the DFSRRC00 PARM is UDR. Under such a condition there is no staging of the output data set at OPEN time.

OSAM data sets cannot be bound. Therefore, there are some implications that should be considered. Even though the data requested in the first DL/I call will be returned to the application as soon as the requested page is staged, the entire data set will be scheduled for staging at the time of the first DL/I call. If you can determine ahead of time that data from the ISAM/OSAM data set will be required, it might be advisable to do one of the following: cause staging to begin shortly after IMS/VS is started by scheduling a simple application that issues a DL/I call to the ISAM/OSAM data base, or cause staging of the OSAM data by running a short IMS/VS batch job ahead of the IMS/VS job that requires the OSAM data to be staged. The IMS/VS batch job would issue a DL/I call, which would cause OPEN and staging. At the end of the batch job the data would remain staged. This assumes there is enough staging pack space available for the OSAM data. It also assumes other activity in the system does not cause destaging of the OSAM data before it is needed. Refer to "Data Reuse" in the Introduction to the IBM 3850 Mass Storage System (MSS) manual for details on the reuse of virtual volume data.

VSAM data bases can have the stage, bind, or cylinderfault staging attribute. Also, VSAM data can be destaged synchronously or asynchronously, where ISAM/OSAM data bases can only be destaged asynchronously.

The following points should be considered in determining whether a VSAM data base should be destaged at CLOSE time with the delayed response request. Destaging at normal CLOSE time with the delayed response request causes synchronous destaging and insures successful destaging of the updated cylinders before CLOSE is complete. However, there are conditions under which IMS/VS closes a data base during on-line processing, for example, if the DMB pool runs out of space, or if the /DBR command stops the data base. Under such a condition the IMS/VS control region waits for CLOSE processing to complete before allowing any other on-line processing to proceed. This temporarily stops all on-line processing until the destage of updated cylinders completes if the data base was closed with delayed response. Each installation has to determine how it wants to close its data bases depending on the frequency of situations that can close a data base during on-line operations, the importance of never interrupting on-line operations, and the importance of insuring a successful destage of update cylinders at CLOSE time.

In general, data bases should be VSAM based to provide maximum flexibility in both staging and destaging. For a very large noncritical HISAM or HIDAM data base, it might be desirable to define the HIDAM index cluster or the HISAM index component of the KSDS as bound and the data portion as cylinderfault. This could be somewhat better than accessing the entire data base in cylinderfault mode, and it would require bound disk drive space for just the index portion of a large data base.

The same guideline applies to secondary indexing. It might be helpful to define the secondary index as a bound data set.

HOW TO USE THE ADDITIONAL CAPACITY OF MSS WITH IMS/VS

Since MSS offers a vast amount of storage beyond what has, in the past, been used in an IMS/VS system, it is meaningful to ask how that additional storage should be used. What kind of work can be put on IMS/VS given the added capacity of the MSS?

As a general rule, applications requiring a small number of large, infrequently referenced, preferably read-only data bases, where response time or batch turnaround time is not critical, can be added to an existing IMS/VS system to take advantage of the additional capacity of the MSS.

The applications could be new or they could be old ones that previously required data to be restored to disk from a save tape before they could be run.

It is likely that the added applications would run in cylinderfault mode to avoid an investment in additional disk drive space. It then follows that the new applications would use new data bases that are separate from the existing critical IMS/VS online data bases.

The added data bases should be infrequently referenced. Added work in cylinderfault mode could eventually impact existing work in the MSS system as the staging facilities of the MSS are absorbed. This cannot be quantified because it depends on the existing load on the IMS/VS MSS system, but it should be considered.

It is recommended that the new applications be read-only because changes to a data base require a destage of updated cylinders. Therefore the impact of additional applications can be minimized by adding mostly read-only applications to the IMS/VS MSS system.

An example of a new application is a report writing application using a small amount of data from a large data base.

The staging and binding of data in an IMS/VS environment can be handled in several ways depending on when the user wishes to experience a possible delay for staging. As stated earlier, if staging packs are not used for other work during off-shift operations, then there is no staging required each day because the data still exists on the staging packs. Staging at a data set level is determined at data set OPEN time based on the DEFINE attributes for the data set. Since IMS/VS OPENs a data set only when required at the first DL/I call, any necessary data set staging could take place at the first DL/I call. Again, if the staging disk drives were not used for other work, this would not cause any delay in processing at the first DL/I call.

If your IMS/VS installation requires that bound data be available for critical processing during a relatively short period of time, for example 8 to 12 hours a day, it might be better or necessary to use the staging disk drives for other off-shift work. After the off-shift work has completed, and before IMS/VS critical processing is again started, it might be advisable to run a short IMS/VS batch job that OPENs the critical data sets and causes restaging and binding of the critical data. Then when IMS/VS critical processing is started, there will not be a delay for staging at the first DL/I call.

Because most installations require IMS/VS to be up for more than one shift, it is not necessarily restrictive to dedicate bound staging space for certain critical data bases. This can greatly reduce the staging time that might otherwise be required if staging space was used for other work during off-shift operations.

If certain staging drives are to be dedicated to bound staging space that will not be used by other work during off-shift, they should be put in a separate staging drive group to ensure that the disk drives are not inadvertently used when IMS/VS is stopped.

The staging drive groups are set up at MSS Table Create time. IMS/VS data sets are allocated to the staging drive groups as part of normal IMS/VS initialization via the UNIT assignment in the DD statements for the data bases.

This section has described some of the points that should be considered to effectively use MSS with IMS/VS. With proper planning, the MSS can provide both added storage capacity and flexibility to the IMS/VS system.

CHAPTER 3. APPLICATION PROGRAM DESIGN

This chapter includes considerations for design of both IMS/VS batch and teleprocessing applications. Information concerning the data base interface applies to batch and online applications. The designer of a teleprocessing application should cover all material in this chapter prior to designing his application. The designer of the batch application need only cover the material relating to batch applications, but is encouraged to cover the entire chapter prior to design of an application.

BATCH APPLICATION PROGRAM DESIGN

GENERAL CONSIDERATIONS

Design of IMS/VS batch application programs deals with the environment shown in Figure 3-1. This environment is established through the IMS/VS system definition utility. Considerations for establishing this environment can be found in Chapter 2 of this manual.

The application program, in conjunction with IMS/VS, runs as an operating system job in VS1 or VS2. For the individual application program design, DL/I can be looked at as an additional access method. The logging facility is a system function and does not involve the application program directly. Changes to the data base are automatically logged. Instead of the standard OS/VS terminology of SYSIN (input) and SYSOUT (output), TRANSACTION input and RESPONSE output respectively are used. This choice of nomenclature is used to encourage the design of a transaction-oriented system.

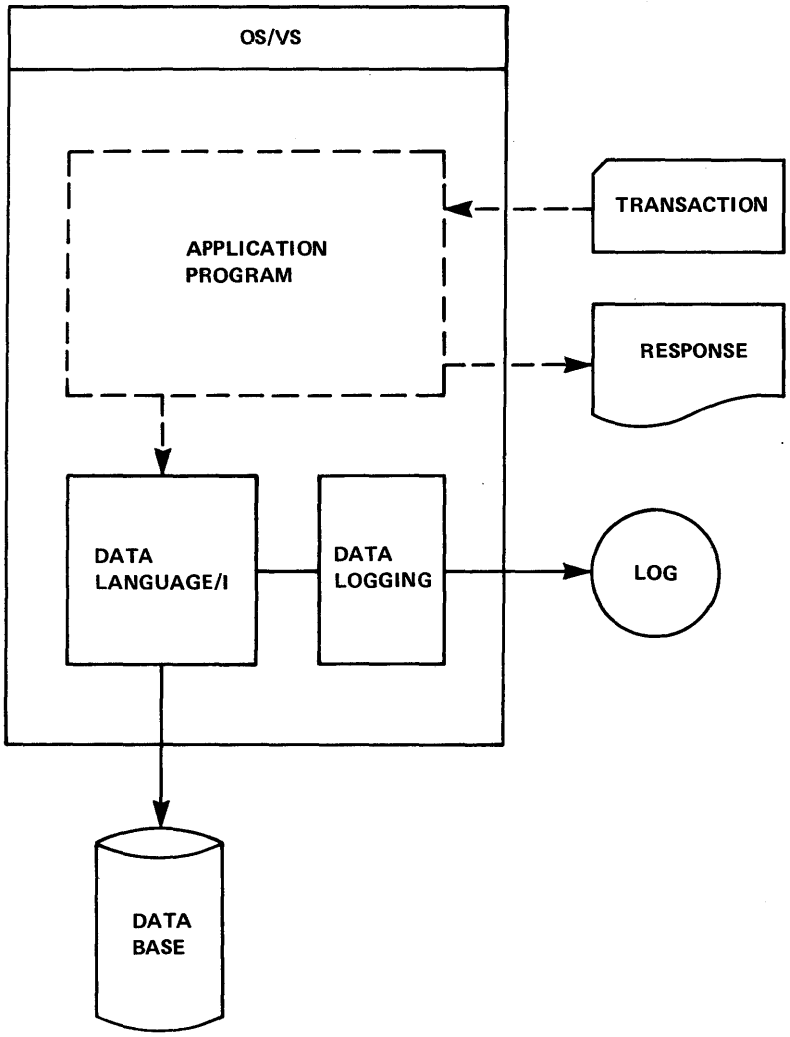


Figure 3-1. Batch Application Program Design

A transaction-oriented system can reduce recovery problems for program abnormal terminations and system failures. A transaction-oriented system is one in which there is a definite point at which each transaction (input) is considered complete. This point must be prior to receiving the next transaction. This isolates recovery problems to a particular transaction.

Design of the application program must be done in concert with data base design. Each influences the other. With good communications between application design and data base design, a more viable system will be developed. A viable system is one which accommodates change with minimum modification.

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.

Programming Language to be Used

The use of IMS/VS should have little influence on the choice of a programming language for the application. The standard operating system CALL interface is used for COBOL, PL/I, and Assembler. IMS/VS offers no special advantage to these languages. However, the basic benefits attained by using a high level language do apply.

The Program Module Preload function of IMS/VS does offer a potential performance improvement, if application programs are written to be serially reusable or reenterable.

Future Conversion to Teleprocessing

When designing an application program it is important to determine if there is a possibility of converting the program to a message processing program to be used in a teleprocessing system. Making this determination prior to design of the application can save conversion time and cost.

Figure 3-2 shows the essential difference between the batch and teleprocessing applications with IMS/VS. In the batch environment, the application program deals with DL/I for data base input/output and with OS/VS data management for external input/output such as SYSIN and SYSOUT. The same application program is shown below after being converted to a teleprocessing application. The basic change is the replacing of READ/WRITE or GET/PUT logic with calls to DL/I for external input (TRANSACTION) and output (RESPONSE).

By centralizing the statements in the batch application program which deal with external input/output, the future conversion to teleprocessing can be made with a great deal more ease.

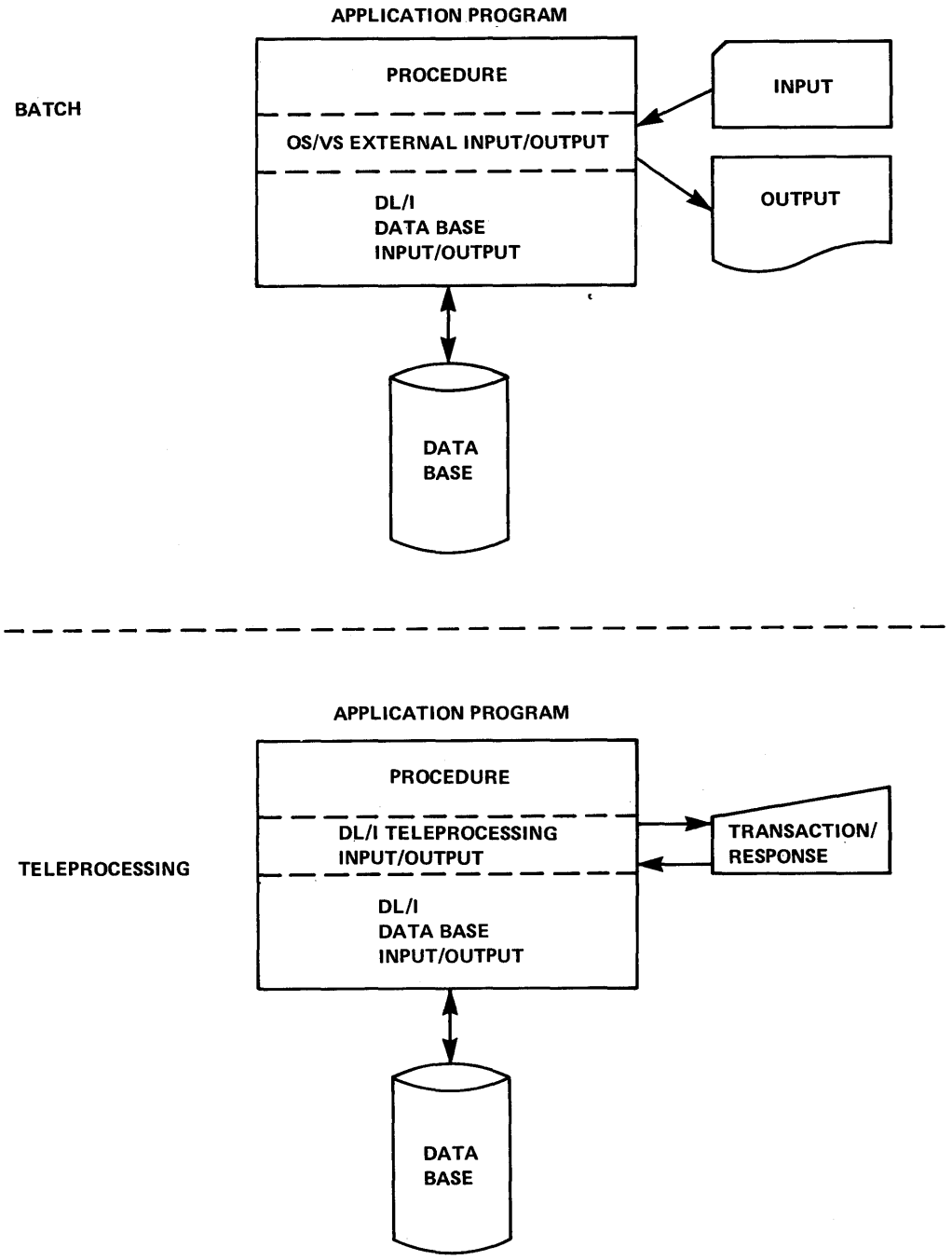


Figure 3-2. Planning Future Conversion to Teleprocessing

BATCH CHECKPOINT/RESTART CONSIDERATIONS

A general facility for batch checkpoint/restart is provided. It consists of a DL/I call function for checkpoint (CHKP) and the batch backout utility program. Batch checkpoint/restart can be complemented either by OS/VS checkpoint/restart or by installation-supplied checkpoint/restart routines. Installation-supplied routines can be simplified by using the IMS/VS restart call (XRST) and user-area parameters on the CHKP call.

To use batch checkpoint/restart, the application first invokes its complementary checkpoint routines. If the user wants to issue an OS/VS checkpoint within a batch only region, he must first close all open data bases, generate a unique checkpoint ID, issue an OS/VS checkpoint macro, and issue the DL/I call with a matching checkpoint ID (this is required because OS/VS restart does not restore data management storage in total). If the user wants IMS/VS to issue an OS/VS checkpoint for him, a fourth parameter, which points to an OS/VS checkpoint DCB, must be specified for the DL/I checkpoint call. Upon completion of those routines, and before issuing any other DL/I call, a DL/I checkpoint call is submitted. Along with the checkpoint call, the application passes the identification of its previously completed complementary checkpoint. DL/I ensures that all pending data base activity is physically recorded. Then the supplied checkpoint identification is recorded on the log tape and supplied to the operator in a WTO message. Control is returned to the application program, which can then proceed to execute, submitting other DL/I calls as necessary.

To restart at a selected checkpoint, the batch backout utility is used to restore the data bases to their condition at the time of the checkpoint. Then the batch job is restarted at the same checkpoint.

ESTABLISHING USEFUL CONVENTIONS

Designing applications for use with IMS/VS affords an opportunity for establishing useful conventions and procedures. Adopting conventions which prove useful in application design and implementation can reduce costs and development cycle time.

Testing

Each program that is designed and implemented must be tested. Testing the application requires a test data base. A test data base requires a data base description generation, a program specification block generation, and a data base load program. Since a number of application programs will be dealing with the same data structure, a central agency for generating and maintaining test data bases should exist.

Naming Conventions

It is important to establish naming conventions for data bases, data segments, fields, PSBs, and programs. A requirement in the IMS/VS DB/DC system is that the name of the PSB and the application program must be the same. Adopting this convention in the batch system can reduce conversion time.

As the system increases in scope through time there will be multiple data bases, each with a number of different segment types. One naming convention which can be helpful is to adopt a two-character code as the first two characters for a data base name. This two-character code can then be used as a prefix for all segment names within the data base. This ensures that no two segments will have the same name, eliminating communications problems.

As the system becomes more complex with the relationship of programs, PSBs, data bases, segment types, and fields, a dictionary will be necessary. Questions such as, "What data segments and fields does this program update?" or "Which programs update this segment?" could be included in a data dictionary. Maintenance and control of such a dictionary should be the responsibility of the Systems Operation personnel responsible for all system control.

Use of COPY or INCLUDE

Extensive use of COPY or INCLUDE can be made for segment I/O area definition, PCB masks, and Segment Search Arguments (SSAs) within an application program. The use of COPY or INCLUDE in conjunction with a data dictionary can reduce maintenance disruptions to a minimum.

Figure 3-3 shows an application program that is making use of standard operating system data sets and DL/I data bases. DL/I makes use of standard OS/VS data management facilities and provides a special access method called OSAM (Overflow Sequential Access Method).

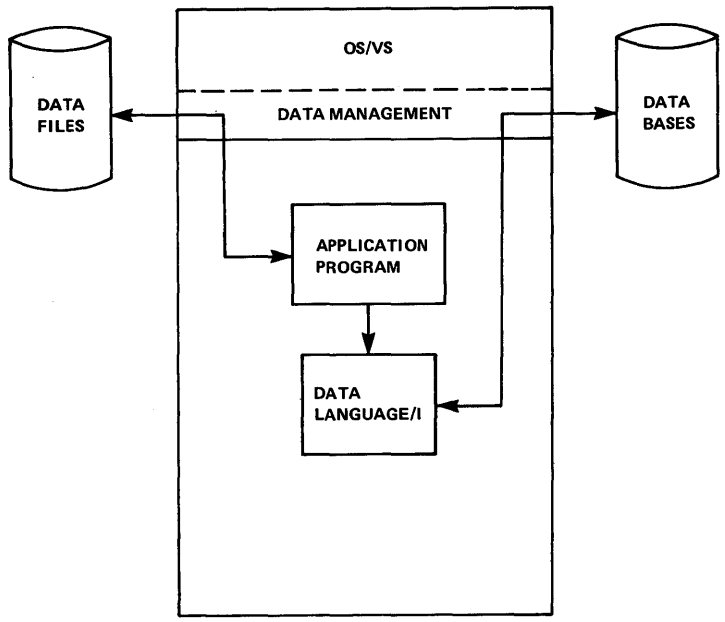


Figure 3-3. Application Program Using OS/VS Data Files and DL/I Data Base

A parallel can be drawn between operating system data management input/output and DL/I input/output. Figure 3-4 shows an application program making use of the READ/WRITE logic of COBOL, which in turn makes use of a file description block. The same program is also reading and writing a data base through DL/I. The DL/I interface makes use of the standard OS/VS CALL facility. The control block for DL/I that replaces the file description of the operating system is called a program communication block (PCB). Just as a file description block is used for each file that is accessed by a program, each logical data structure that is accessed requires a program communication block. The IMS/VS Application Programming Reference Manual provides a discussion of logical data structures and their relation to data bases.

A unique characteristic of the application program interface with DL/I is that all information passed across the interface is described symbolically. There is nothing in the interface definition which relates to a specific access method or physical storage organization.

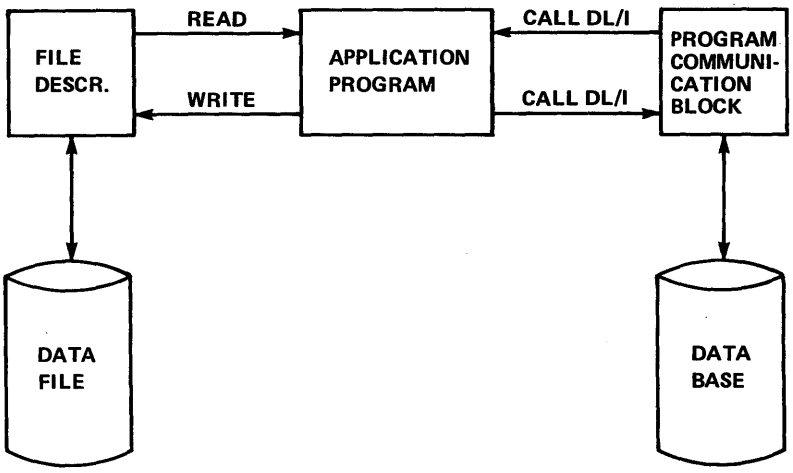


Figure 3-4. Application Program Using COBOL READ/WRITE Logic and File Description

Details of CALLs to DL/I can be found in the IMS/VS Application Programming Reference Manual.

Traditionally, application programs have been designed to obtain an entire record from a file and then deal with only selected portions of the record for reference or update. The application program was record oriented. With DL/I, the application program is designed to obtain only those portions of the record necessary to perform the required procedure. I/O is on a segment basis. The segment can contain one or more fields of information.

The fact that I/O is on a segment basis should have some influence on the design of the application, as well as on the design of the data base. Once a segment is retrieved with a Get Hold type call, the next call using the same data base PCB can be a replace or delete call. If an intervening call is made, it would be necessary to do another Get Hold type call to update the segment. One way to avoid this is to use multiple data base PCBs for the same data base. This allows multiple positions, as well as multiple segments, to be in HOLD status at one time.

Using the Right DL/I Call

Application programmers are sometimes faced with a decision concerning the use of DL/I function codes and segment search arguments. A function code is a four-character code which is supplied to DL/I by the calling application program to specify the input/output function to be performed. SSAs (segment search arguments) are used to give specific information about the path to be followed in satisfying a call. SSAs are qualified or unqualified. An unqualified SSA specifies only the name of the desired segment. A qualified SSA specifies, in addition to the segment name, a field name within the segment, a relation operator, and a comparative value.

For segment insert, delete, and replace, there is only one code for each specific function to be performed. For the segment retrieval function, however, there is a family of function codes: GET UNIQUE (GU), GET HOLD UNIQUE (GHU), GET NEXT (GN), GET HOLD NEXT (GHN), GET NEXT WITHIN PARENT (GNP), and GET HOLD NEXT WITHIN PARENT (GHNP). Each of these call functions provides for a variation in the method of retrieving a segment, depending on the existing position in the data

base and the segment qualification. There are times when more than one of these calls will accomplish the same thing.

When faced with a choice of GU, GN, or GNP with or without the HOLD option, there are a number of considerations. In addition to choosing a function code, the question of whether or not segment search arguments (SSAs) should be provided must be answered. If the SSAs are provided, the question of qualified or unqualified must be answered.

Generally speaking, the GU call is used to retrieve a specific segment or to obtain a specific position within a data base. The GN or GNP only moves forward in the data base, except when the F command code is used. Once a logical position is established within a data base, the GU or the GN and GNP, used in conjunction with the F command code, are the only calls which can establish a position at some earlier logical point in the data base.

There is no measurable difference between a GU, GN, or GNP call, if each has fully qualified SSAs and no logical position exists within the data base. If a logical position exists and movement is forward, a GN or GNP function call may be more efficient. An additional difficulty in making a choice of GU function calls comes when there is insufficient knowledge to provide complete qualification.

Normally, a GU call requires more time to execute than a GN or GNP call statement.

The implications of providing unqualified segment search arguments can be seen in Figure 3-5. The call on the left has an unqualified segment search argument at level two for B. As a result, DL/I searches through all Segment Bs under Segment A with key of 6. All Segment Cs are searched before finding that the call cannot be satisfied. The call on the right is the same, except that the segment search argument for B is qualified at level two. When DL/I encounters the B with a key of 4, the search ends. At this point, DL/I realizes that the call cannot be satisfied.

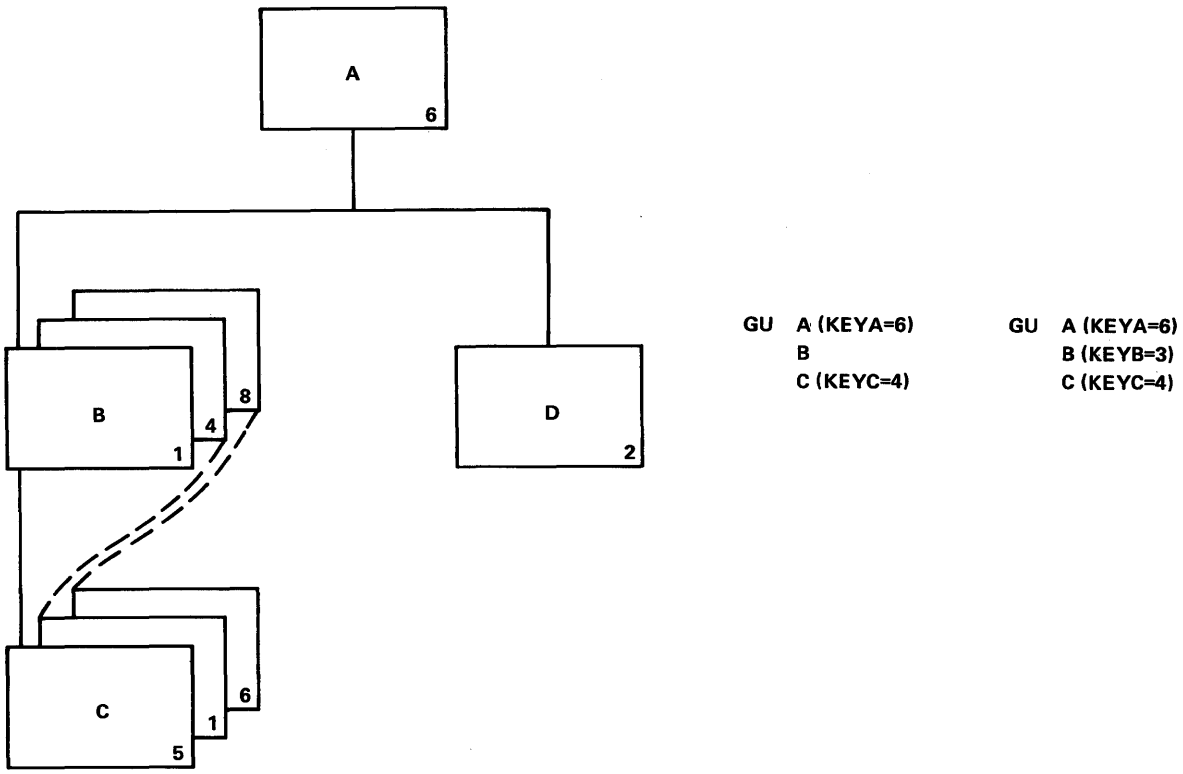


Figure 3-5. Qualified Segment Search Arguments

RELATIONSHIP BETWEEN DL/I CALLS AND PHYSICAL I/O OPERATIONS

Although DL/I calls issued by application programs are independent of the physical storage techniques used to store and access data, it is important for the reader to understand the physical I/O operations performed by IMS/VS. The use of any DL/I call may or may not require physical I/O operations. If the DL/I call can be satisfied from information in the data base buffer pool, no physical I/O operations are required. When this is not the case, the actual physical I/O operations performed depend upon the following:

- The DL/I call issued
- The physical data base access method and organization
- The current position in the data base known by the DL/I control blocks for this application's use of the data base
- Information in the data base buffer pool

The following tables should be of assistance in understanding the physical I/O operation which may be performed in satisfying GET UNIQUE (GU) or GET NEXT (GN) calls.

IMS/VS DB/DC CONTROL PROGRAM

DL/I CALL FUNCTION	DATA BASE	DATA BASE	ACCESS METHOD	
	HISAM	HDAM	HIDAM	
			INDEX Data Base	HIDAM Data Base
GU	VSAM-Get Direct or BISAM or OSAM Read	VSAM-Get Direct or OSAM Read	VSAM-Get Direct or BISAM Read or OSAM Read	VSAM-Get Direct or OSAM Read
GN	VSAM-Get Direct or BISAM or OSAM Read	VSAM-Get Direct or OSAM Read	VSAM-Get Direct or BISAM Read or OSAM Read	VSAM-Get Direct or OSAM Read

IMS/VS DB BATCH PROCESSING

DL/I CALL FUNCTION	DATA BASE ACCESS METHOD			
	HISAM	HDAM	HIDAM	
			INDEX Data Base	HIDAM Data Base
GU	VSAM-Get- Skip Seq or QISAM SetL & Get - - - - - - VSAM-Get- Direct or BISAM Read *	VSAM-Get Direct or OSAM Read	VSAM-Get Skip Seq or QISAM SetL & Get - - - - - - VSAM-Get- Direct or BISAM Read *	VSAM-Get Direct or OSAM Read
GN	VSAM-Get- Skip Seq or QISAM Get, OSAM Read - - - - - - VSAM-GET Direct or BISAM Read *	VSAM-Get Direct or OSAM Read	VSAM-Get- Skip Seq or QISAM SetL & Get - - - - - - VSAM-GET Direct or BISAM Read *	VSAM-GET Direct or OSAM Read

* BISAM read or VSAM Get Direct is used if two or more data base PCBs are defined in an application program's PSB for one physical data base. Random or direct access operation is assumed. QISAM SetL/Get or VSAM Skip Sequential is used if one data base PCB per physical data base is used.

It is suggested that, where possible, the GET NEXT call, with or without SSAs, be used in preference to the GET UNIQUE call function for segment retrieval. This results in more efficient operation. Remember, however, that the GET NEXT call function can only be used to progress forward in a data structure, and across data structures in a data base.

PERFORMANCE CONSIDERATIONS

Using Accumulated DL/I Statistics

During execution of the batch application program, statistics are accumulated by DL/I concerning reading, writing, and buffering activity. This information can be utilized to tune the application for higher performance. Details for obtaining these statistics are in the IMS/VS System Programming Reference Manual.

STATIC Declaration for PL/I

The programming language PL/I has the ability to allocate storage for variables, statically at compile time, or dynamically at execution time. It is suggested that if a batch or message program is written in PL/I, the STATIC attribute be used in the declaration of the variables (except the PCBs) so that they do not default to the AUTOMATIC attribute. The AUTOMATIC attribute allocates the storage dynamically. STATIC declaration speeds up the loading and execution of a PL/I program in an IMS/VS address space. Increased performance is obtained because OS GETMAIN and FREEMAIN operations required for storage of variables are eliminated at execution time.

TELEPROCESSING APPLICATION PROGRAM DESIGN

TELEPROCESSING INPUT/OUTPUT INTERFACE

Design of a TP (teleprocessing) application encompasses the batch application program design as well. There is little difference between the batch program and the teleprocessing program when using IMS/VS.

Figure 3-6 shows the environment in which the TP application functions. This shows DL/I as the interface between teleprocessing terminals as well as data bases. The application program in a TP environment deals exclusively with DL/I for input and output for terminals as well as data bases.

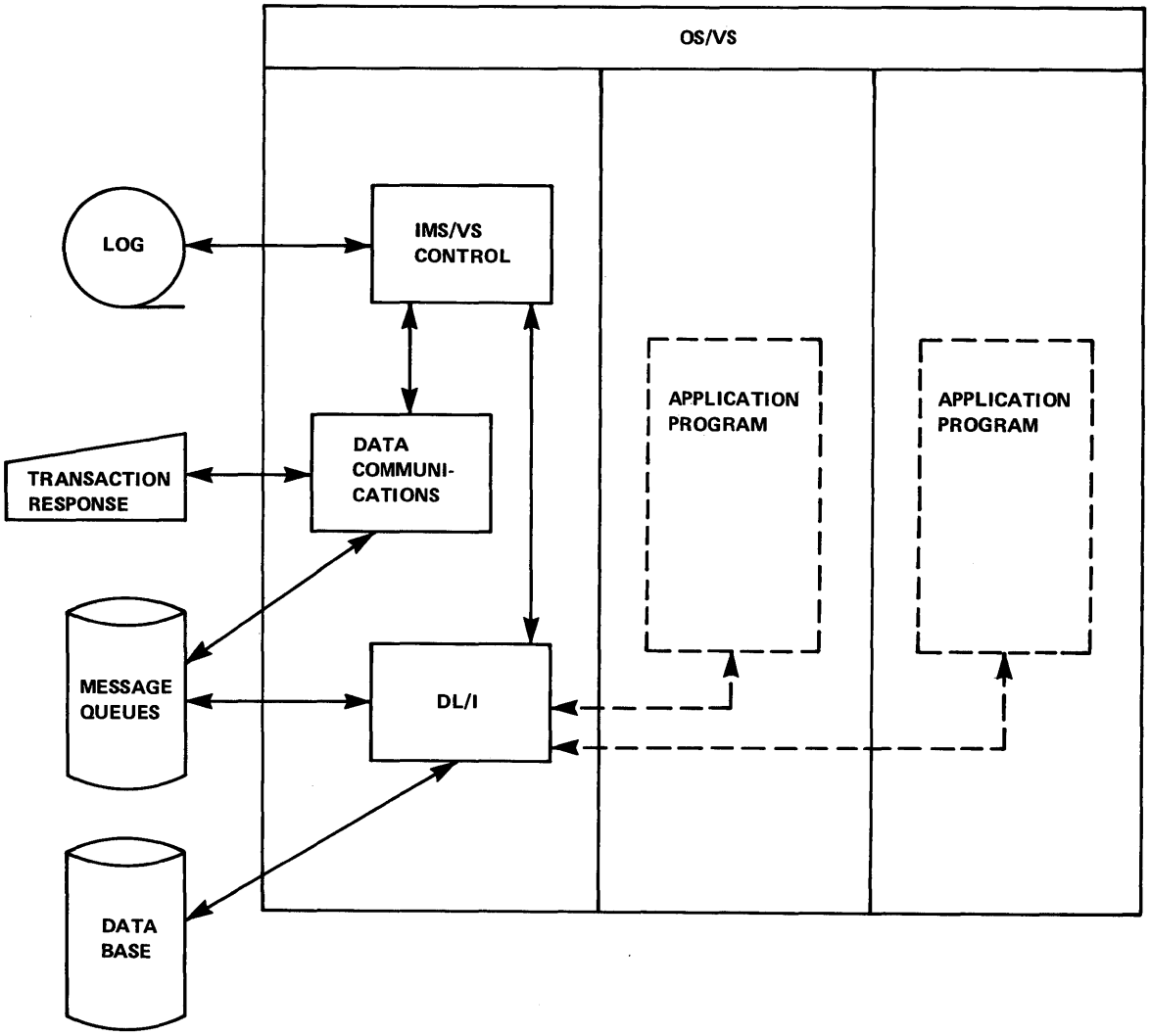


Figure 3-6. Teleprocessing Application Program Design

The three areas shown under the operating system each represent operating system jobs. Each is under a different storage protection key. The job on the left consists of the IMS/VS control program, which is responsible for all physical input/output for IMS/VS applications. The control program is also responsible for maintaining logical information for restart and recovery purposes. The two application programs shown are each contained in a message processing region. Each message processing region is an operating system job. This IMS/VS control region is responsible for causing the appropriate application program to be loaded for processing.

With IMS/VS, the interface to data bases is unchanged when going from a batch application to a TP application. In addition, the same interface used for data bases is used for input and output to terminals.

The application program deals with logical terminals. These are control blocks that IMS/VS associates with physical terminals. Thus the application programmer generally does not concern himself with the physical attributes of the terminal with which he is dealing. Figure 3-7 shows an application program's view of the terminal.

The control block with which the application program deals is the TP PCB (teleprocessing program communication block). There are two types of TP PCBs -- the I/O PCB and alternate PCBs. An I/O PCB is always provided by IMS/VS to an application program that executes in a TP environment. Alternate PCBs are optional, and are created as part of the PSB (Program Specification Block). To obtain an input message and reply to it, the application program must reference the I/O PCB. To send a reply to a terminal other than the terminal that orginated the input message, the program references an alternate PCB. The section named "Output to Alternate Destinations" contains a further description of alternate PCBs. Figure 3-8 shows a DB PCB (data base program communication block) in addition to the TP PCB. The data base is viewed as a logical structure and the terminal is viewed as a logical terminal.

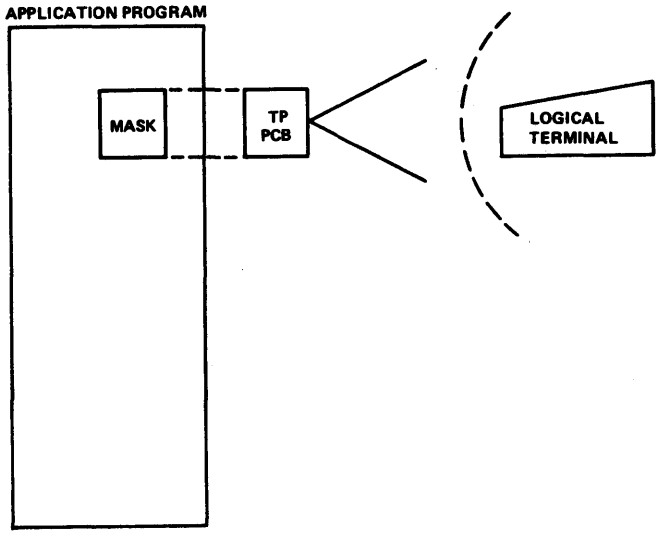


Figure 3-7. Application Program's View of the Terminal

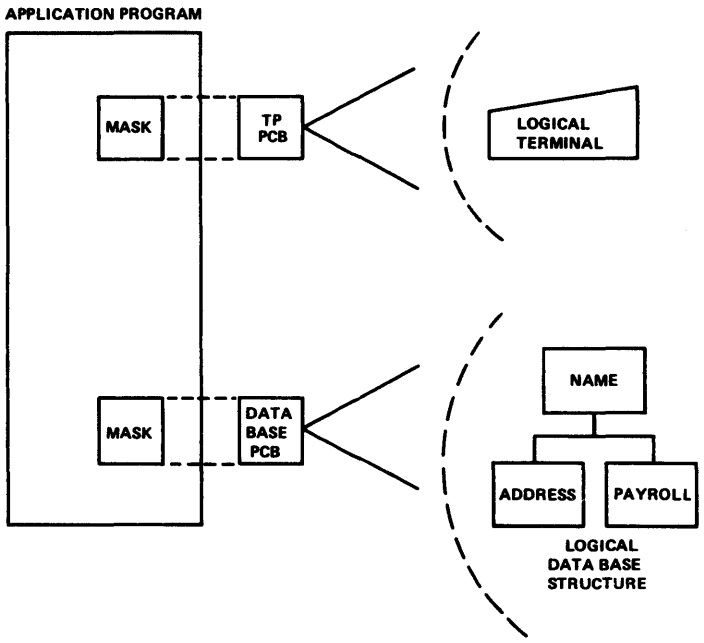


Figure 3-8. DB and TP PCBs

Information received from or sent to a terminal is called a message. A message is comprised of one or more segments. Figure 3-9 shows the format of a message segment. The I field specifies the length of the segment. If line addressing is being used, field Z is used for screen control when sending output to a 2260 or 2265 Display Station. The Z field is followed by the message text. This is the information input at the terminal. Below the segment format are shown two examples of input -- one with a password and the other without. Notice that the password has been eliminated from the text prior to the application program receiving it.

```
+-------------------------------------+
|  I   |   Z   |        TEXT          |
+-------------------------------------+
    |        |             |
    |        |             |--> Message segment up to 130 characters
    |        |
    |        |--> Reserved for DI/I (halfword binary)
    |
    |--> Segment length in bytes including I, Z, and TEXT
         (halfword binary)
```

Terminal input segment with password:	TRANS(PASSWORD) THIS IS THE SEGMENT TEXT
Received by application:	TRANS THIS IS THE SEGMENT TEXT
Terminal output segment without password:	TRANS THIS IS THE SEGMENT TEXT
Received by application:	TRANS THIS IS THE SEGMENT TEXT

Figure 3-9. Message Segment Format

Input Calls

Calls for input message segments are like calls for data base record segments, except that no segment search arguments are required. The get unique call is used to obtain the first segment of each message and the get next call is used to obtain subsequent segments. Figure 3-10 shows the format of the input call. The three parameters shown being passed to DL/I are the function code, the I/O PCB address, and the address of an input area. Message A, as shown, consists of three segments, while Message B consists of two segments.

```
ENTER LINKAGE.  COMMENT ONLY
CALL 'CBLTDLI' USING IFUN, LTPCB, IMSG-IO-AREA.
ENTER COBOL.  COMMENT ONLY
```

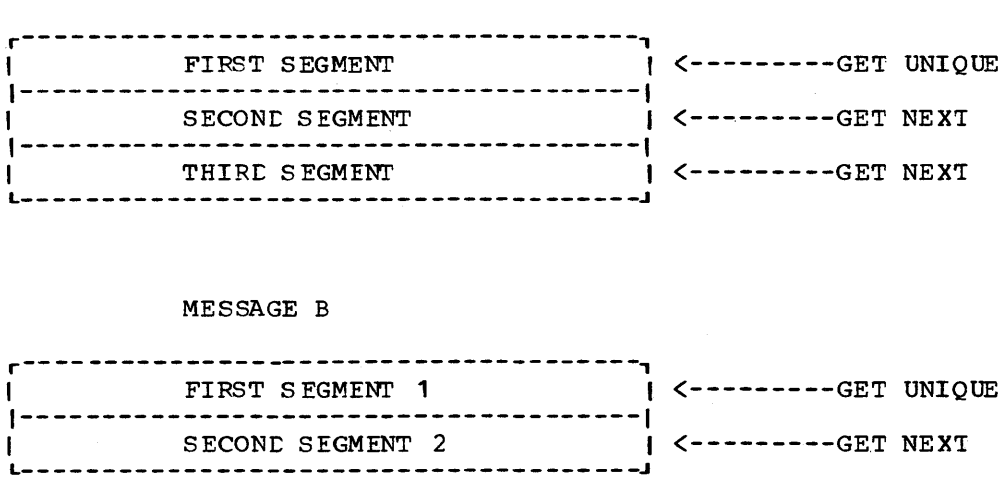


Figure 3-10. Input Call Format

Output Calls

Sending output to a logical terminal is like inserting new segments to a data base. As with the input call, no segment search arguments are required. Figure 3-11 shows a three-segment message being built. The parameters passed in the call to DL/I represent the function code, TP PCB, output area, and message format name. The message format name is ignored on systems without the Message Format Service. Format of the output message is the same as that of the input message. The application programmer must supply the character count.

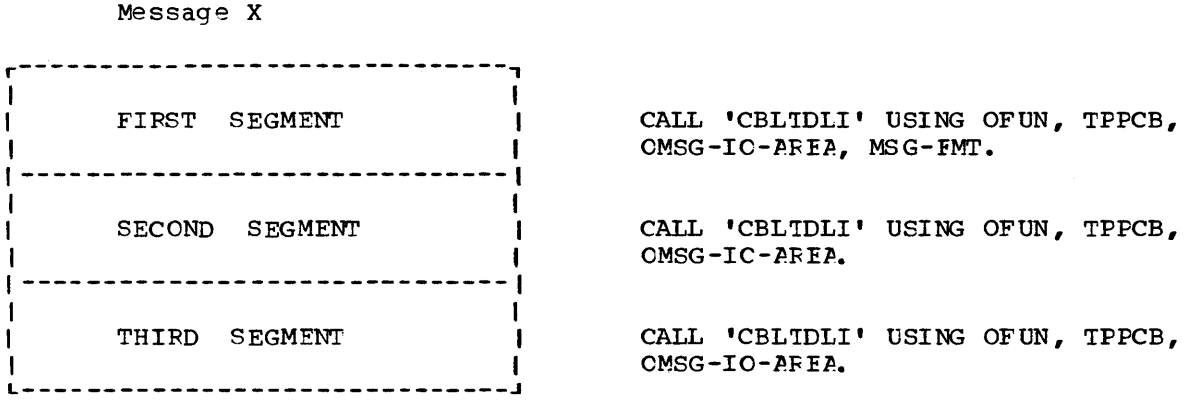


Figure 3-11. Three-Segment Message

OUTPUT TO ALTERNATE DESTINATIONS

In addition to sending output back to the terminal that generated the input, the application program can send output to additional destinations. Output can be sent to other logical terminals or to other programs. The mechanism for sending to these alternate destinations is the alternate PCB, as shown in Figure 3-12. When sending output to another program, the receiving program can be a message processing program or a batch message processing program. The batch message processing program, in addition to making use of online data bases and message queues, can utilize operating system data management facilities. Use of batch message processing programs is discussed later in this chapter.

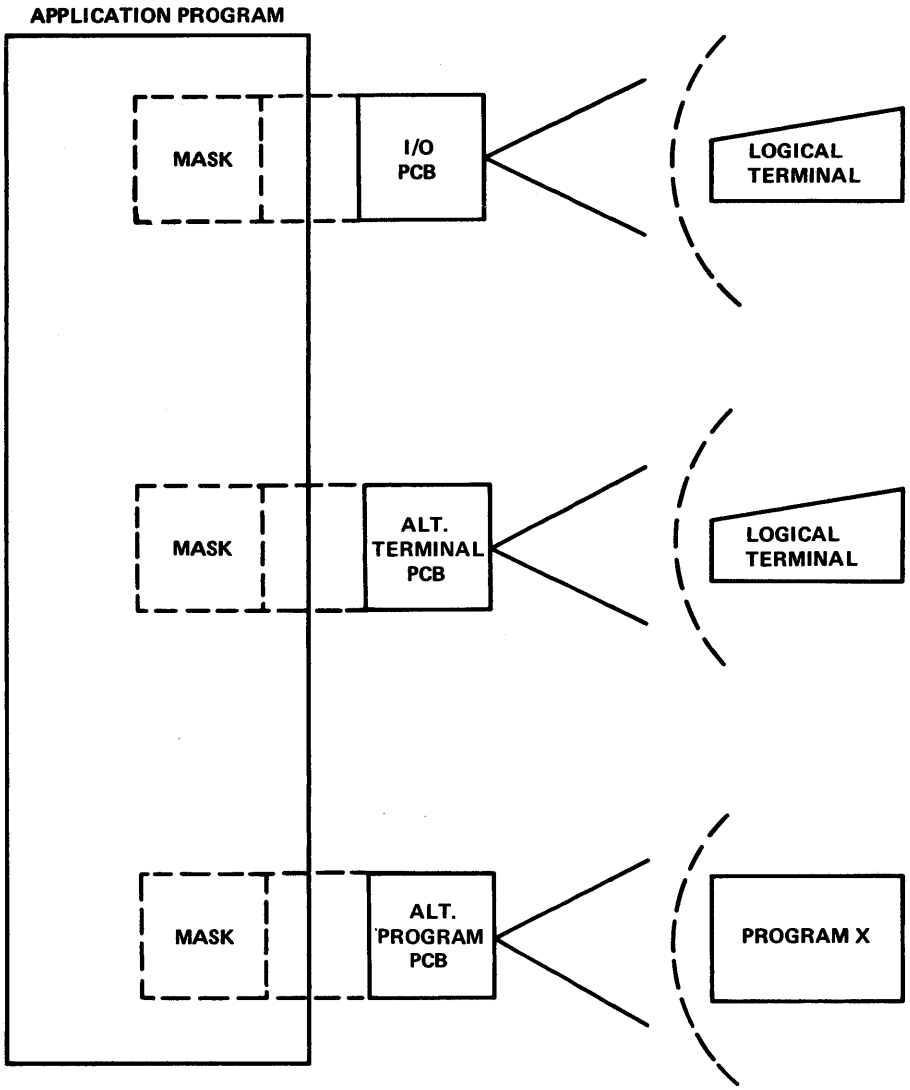


Figure 3-12. Output to Alternate Destinations

Modifiable Alternate PCBs

The modifiable PCB and change call have been prcvided for those users who would ctherwise require a prohibitively high number of alternate PCBs to allow for all possible destinaticns. This would, for example, be those 1050 or 2780 users with a requirement for an alternate PCB for each component assigned to the terminal represented by the I/O PCB. Without this function these users would require as many as fcur alternate PCBs per terminal defined in the system. By providing a naming convention within the IMS/VS system to allow the applicatiCn programmer tc identify a group of logical terminals by I/O PCB name, this requirement could be reduced tc fcur modifiable alternate PCBs or less.

For example: If NAME is found to be the I/O PCE logical terminal name, NAMECP is the logical terminal assigned tc the card punch, NAMEPRT is the printer, etc. With this ccnvention the user could add the suffix CP to the I/O PCE name to cause output to gc tc the card punch associated with the physical terminal that entered the message, PRT would allow the output to go to the printer, etc. This requires that

the naming convention be established by system definition and maintained by instructing the master terminal operator to reassign component LTERMs by groups, so that all the components are always associated with the same physical terminal.

This function could also be used by any application that, depending on the processing involved, requires one of many possible output destinations.

The user should define one modifiable PCB per possible destination per transaction, as the destination can be set only once per message without use of the purge call, which is not recommended. This means simply that the destination cannot be changed once a message segment has been inserted to the PCB until a get unique to the I/O PCB is issued.

Normal use of the function would therefore be:

```
GU      I/O PCB
CHNG    Modifiable alternate PCB
ISRT    Modifiable alternate PCB
GU      I/O PCB
CHNG    Modifiable alternate PCB
etc.
```

Response Alternate PCBs

An alternate PCB can be used to respond to terminals in response mode, conversational mode, or exclusive mode, if the PCB is so defined on the alternate PCB statement. Use of this response alternate PCB allows the application program to send output to a logical terminal that did not originate the input message, while still satisfying the requirements of these operating modes. This response alternate PCB is only valid for naming a logical terminal.

IMS/VS can also be directed to verify that the logical terminal named in the response alternate PCB is assigned to the same physical terminal as the logical terminal that originated the input message. This verification is required for alternate response PCBs used by conversational programs and response mode transactions. Verification is not needed if alternate response PCBs are used to send messages to output-only devices that are in exclusive mode. Additional information on response alternate PCBs is found in IMS/VS Application Programming Reference Manual.

CONVERTING FROM BATCH TO TELEPROCESSING

Conversion from batch to online with IMS/VS can be a simple process. Figure 3-13 shows a batch application program which deals with DL/I data bases. The procedural portion of the application program differs little between batch and TP. The DL/I data base I/O calls need not be altered at all. The area of conversion will be that portion which deals with external input (SYSIN) and output (SYSOUT). The TRANSACTION and RESPONSE in the application program shown represent the primary input and output. The READ/WRITE or GET/PUT in the batch system are replaced by DL/I calls for input and output for teleprocessing. Instead of the input coming in from SYSIN, it comes from a logical terminal. Output, instead of being written to SYSOUT, is written to a logical terminal. It can be seen that use of DL/I for transactions and responses as well as data base I/O, makes DL/I the single I/O interface with which the application program deals.

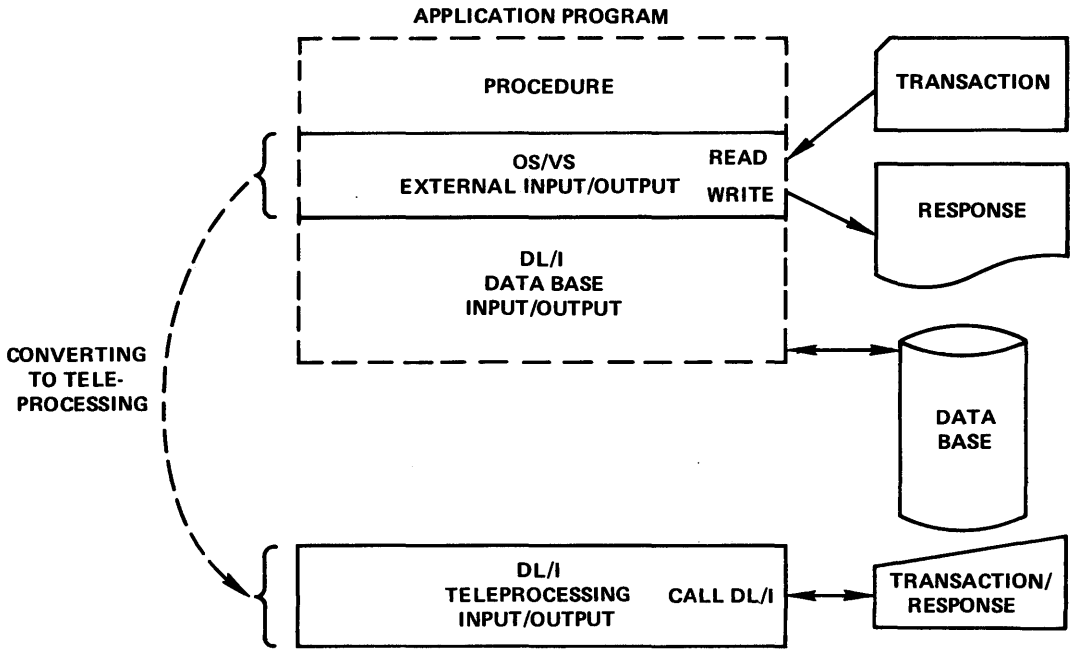


Figure 3-13. Converting from Batch to Teleprocessing

TELEPROCESSING DEVICE INDEPENDENT PROGRAMMING

If a variety of devices are to be used on an IMS/VS teleprocessing system, some consideration should be given to designing application programs in such a way that input is processed properly regardless of the physical terminal type from which it is received. For example, input might be received from either a 2740 or a 3270. The maximum physical line length for a 2740 is 130 characters, and one line of input is handled as one message segment. For a 3270, on the other hand, the user defines the structure and length of a message segment.

If I/O formats are to be consistent between devices with different length I/O characteristics, design must be aimed toward the limiting device. For example, a 3270 can only accommodate 80 characters on each line, while the 2740 can handle 130. A design for a 130-character line would not operate identically with the 2740 and the 3270. Another approach is to have the application program written so that it can determine the class of device with which it is dealing. This can be accomplished through the use of naming conventions. For example, the first character of each logical terminal associated with an 80 character device could begin with the letter V.

The application program has access to this name at the time the message is acquired.

DEVICE CLASS CONTROL CONSIDERATIONS

Control characters for control of output devices are the responsibility of the application programmer. The 2260, 2265, and the 2265 component of the 2770 system makes use of the Z field in the message format shown earlier, in conjunction with the line addressing feature of the 2260/2265 and the paging feature of IMS/VS. On a 2980 General Banking Terminal Model 1 or Model 4, the Z field of the message format is used to direct output message segments to a passbook; on a 2980 Model 2 terminal, this field is used to require the presence of

the auditor's key, in order to receive an output message segment. Switched devices (except 3270) also make use of the Z field in the message format shown earlier. This is used by the application program to request that the line be disconnected after the present message is sent. This field is ignored by communications control if the output is physically sent to a device without this capability.

Carriage return characters, or new line symbols, are embedded in the text portion of the message by the application programmer. If output is going to a 2770 printer component, 2780, or local printer (SYSOUT) device, the first two characters of the message can be carriage control characters. These are also the responsibility of the application programmer.

Vocabulary drum address characters may be the text portion (or part thereof) of the message going to a 7770-3 line. These are also the responsibility of the application programmer.

Under special conditions, it may be desirable to terminate an output message at a specific point. The DL/I purge call with TP PCB address can be used to accomplish this function. Figure 3-14 shows two messages to the destination being built.

The purge call releases the output message segments for processing without waiting for the application program to signify normal completion (by a get unique of the next transaction or normal termination) of the current transaction.

```
ENTER LINKAGE. COMMENT ONLY
CALL 'CBLTDLI' USING PURG, TPPCB.
ENTER COBOL. COMMENT ONLY
```

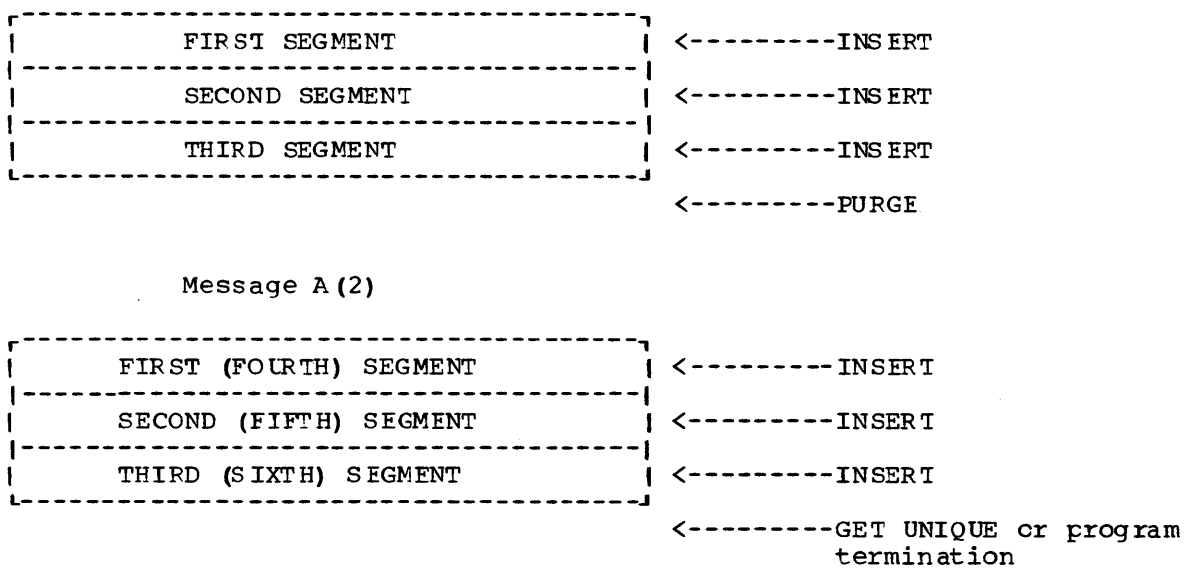


Figure 3-14. Six-Segment Message Separated into Two Three-Segment Messages by Use of the Purge Call

UTILIZATION OF SYSOUT DEVICES

The use of support provided for SYSOUT devices (printers, tape, or DASD) allows a wide range of applications, including:

- Local terminal simulation using a card reader and printer.
- High volume output, such as reports using either a printer or tape volume.
- Intermediate output to be used by a non-IMS/VS application program using either tape or disk volumes.

Since record formats, logical record lengths, and block sizes are user-defined, a SYSOUT data set can have a variety of different attributes.

By using the spool SYSOUT option, a local printer can be simulated without dedicating the device to the IMS/VS system.

Program Testing Using SYSIN/SYSOUT

SYSIN data streams can be assigned to a local card reader line, providing a means by which nonconversational teleprocessing application programs can be tested. When such an assignment is made, a program can be tested with data entered through SYSIN and output produced using any of the optional types of SYSOUT support available. Only one file of data can be entered per line. Any logical errors detected in processing the data stream (for example, invalid transaction codes) are ignored by IMS/VS. Care must be taken to avoid undesirable results when this type of error occurs in the first segment of a multisegment transaction, since all following segments are processed as new messages.

When SYSIN data streams are used by IMS/VS, no logging of position occurs while messages are being processed. Consequently, only a cold start of IMS/VS operation should be performed after using SYSIN input streams, or duplicate message processing may occur.

CONVERSATIONAL PROCESSING

Conversational processing is an optional IMS/VS feature available to users of the data communications facility. It allows a user's application program to retain information acquired through multiple interchanges with a terminal, even though the program leaves the message region between interchanges.

If conversational processing is to be used, it must be considered during system definition. Transactions that will invoke a conversational program must be identified at this time. The user must also describe the number and size of the SPAs (scratchpad areas) to be allocated, either in main storage or on a direct access device. An SPA is used to contain the information to be retained during conversational interchanges.

Figure 3-15 shows a simple conversational process. When IMS/VS receives a conversational transaction it assigns an SPA to the input terminal and schedules the associated application program.

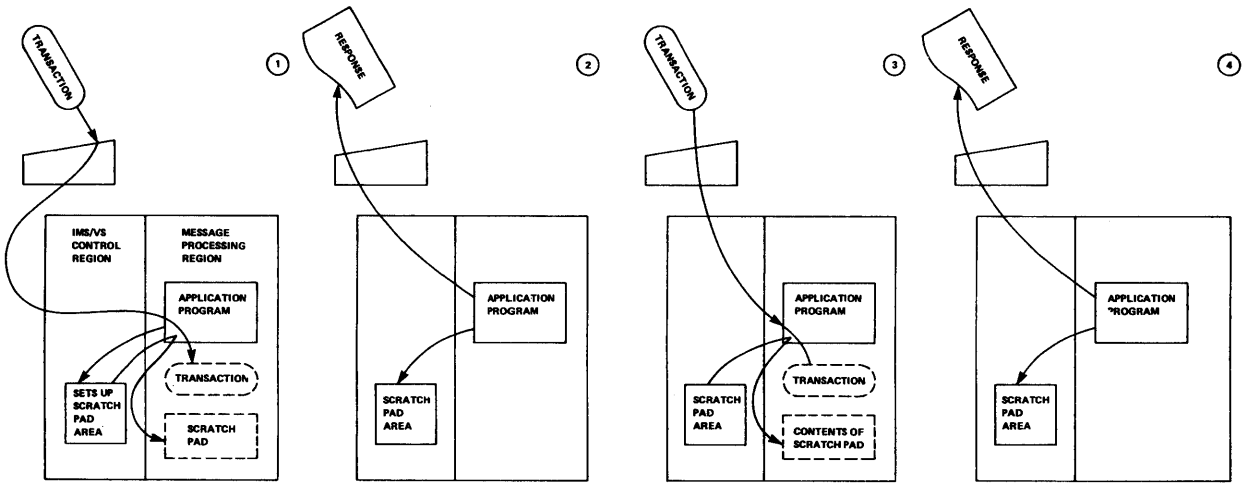


Figure 3-15. Conversational Program

When the program executes and issues its first GU, it receives the SPA. The first segment of the message input from the terminal is obtained by a GN call. After processing the segment, the program must issue an ISRT call to return the SPA to IMS/VS. IMS/VS retains the scratchpad either in main storage or on disk until needed. The program then must use ISRT to send an output message to the terminal in conversation.

A response to the terminal in conversation is required to allow the conversation to continue. The conversational transaction need only be entered to initiate the conversation; during subsequent interchanges IMS/VS considers all input from that terminal to be a part of the conversation.

IMS/VS allows more than one program to participate in a conversation. One conversational program passes control to another, either by changing the transaction code in the SPA to another conversational transaction, or by inserting the SPA to an alternate PCB identifying the program to take control of the conversation.

When a conversation is processed to its normal completion, it is terminated by the application program. The program places blanks in the transaction code in the SPA before returning it through the ISRT call. The program can also put the code of a nonconversational transaction in the SPA; the conversation then remains active until the next input is received from the terminal. IMS/VS routes this input to the nonconversational transaction, thus terminating the conversation.

IMS/VS terminal commands are valid during a conversation. Commands are provided, in fact, to allow the operator to temporarily suspend a conversation in progress (/HOLD command) and to resume it at a later time (/RELEASE command). The /EXIT command is available for the operator to terminate the conversation.

Some applications require that a conversational process not be interrupted once it has started. This is because non-program initiated termination could result in partially-updated data bases. This type of termination can occur if the operator prematurely uses the /EXIT command, or if a program involved in the conversation abnormally terminates. When this condition occurs, a user exit routine can be entered to analyze the termination and, if desired, to cause another program to be scheduled to complete or backout any data base updates. The user exit routine cannot cause the conversation to continue. The IMS/VS System Programming Reference Manual contains specifications for a conversation abnormal termination exit routine.

When the Interactive Query Facility (IQF) feature is incorporated into the user's system, the user must specify conversational mode as well as an SPA at system definition time. The IMS/VS system definition must contain one or more APPLCTN macro instructions for IQF. Each APPLCTN macro must have a TRANSACT macro with an SPA specified to define the transaction code used by the terminal user.

The SPA should be large enough to hold the maximum IQF query that could require a full file search; an additional 40 bytes are required for system usage. If an installation wants to process full file searches under a different transaction code (in order to control when the processing is to be done), another TRANSACT macro must be added. This additional TRANSACT macro defines a non-conversational transaction code and must not have an SPA defined for it. The non-conversational code is used by IQF only during internal processing; it should not be entered from a terminal.

PAGING FEATURE -- 2260 AND 2265

The paging feature allows an application program to insert a multiple screen message to the 2260 or 2265 which can be viewed by the operator in any sequence he wishes. If, after viewing the first screen, the operator chooses to skip all remaining screens and go to the next message, he can. Alternatively, as an example, he can look at the first screen, page forward to the 17th screen, page back to the fifth screen, view several screens in sequence, etc. He can go to the next message or series of screens at any time, whether or not he has looked at all the screens in this series. Once this option is taken however, he cannot return to look at any image from a previous series. IMS/VS prevents the operator from inadvertently paging past the end of one series into another, thus losing the current series.

The operator is supplied with a page-request indicator (=) to specify which page is to be viewed next. If Auto Delete was specified in system definition, any other input message, that is, one that does not begin with a page-request indicator, causes the series of pages being viewed to be considered complete and the series to be dequeued. Therefore, when an operator has completed viewing a series of pages he has merely to enter a new transaction code to signify this to the system. If a multiple-page message is routed to a non-paged terminal, such as a 2740, the paging is ignored, and the message is transmitted as any other message. If Auto Delete was not specified, the operator can enter a message while viewing a page. This causes the first page of the series to be redisplayed, and the operator must specifically enter a next-output indicator (?) to cause the series of pages to be dequeued. While this mode of operation may have merit in specific applications, it may prove cumbersome to the operator in a generalized system application. It is recommended, therefore, that the user be aware of the operational procedures required for non-Auto Delete operation before specifying this mode of operation.

BATCH MESSAGE PROCESSING PROGRAMS

While the IMS/VS teleprocessing system is in operation, it may be desirable to let a batch program have access to online data bases or input/output message queues. This can be accomplished by a batch message processing program (BMP). This program is loaded in the conventional operating system manner. It has access to online data bases and message queues, and can also make use of operating system data management facilities.

When starting a BMP, several parameters may be specified on the EXEC statement. These include the PSB and program name. For message processing programs, the PSB and program name must be the same; however, they can be different for BMPs. This allows a utility program to be run using different PSBs.

BMP can implement a checkpoint to purge data base and message buffers. It writes a checkpoint ID to the system log. This checkpoint is independent of CTL (control region) and other BMP region checkpoints. IMS/VS maintains a checkpoint table to correlate BMP checkpoints with control region checkpoints for purposes of emergency restart. Design considerations for using this checkpoint table are contained in chapter 2 under the topic "Batch Checkpoint Restart." The IMS/VS Application Programming Reference Manual contains a description of the checkpoint (CHKP) call.

Emergency restart after a system failure backs out all resources for each BMP region to the last checkpoint for that BMP region. The master terminal operator has the option of specifying that BMP data base changes NOT be backed out at emergency restart.

If there is no backout for a BMP, the operator also has the option of releasing the resources that were reserved for the BMP (that is, starting stopped data bases). If backout has been done, the resources are not reserved since data base integrity has been maintained.

USE OF BMP

The BMP is useful for several types of processing. If data is being collected for batch processing, the message processing program can retrieve the collected data from the queue. Upon a single loading, a BMP can deal with only one input queue (transaction code). The transaction code is also specified on the EXEC statement when the BMP is started. The BMP sends output to logical terminals or other programs through the queues. The BMP is useful for doing summary reports while the data bases are being utilized in a TP environment. Use of the BMP to update data bases, while the online system is in operation, can cause the scheduling of some message processing programs to be temporarily suspended. This occurs for message processing programs which are sensitive to the same segments for update as is the BMP.

BUFFERING

Heavy utilization cf data base buffers by a BMP can impact response time at a terminal, if a relatively small data base buffer pool is allocated. Since the pool is utilized for all data bases, and the oldest buffer is always freed for current I/C requests, additional I/O requests may be required for those TP programs performing data base updates. It is possible that a message processing program may obtain a segment for update, and pricr to the REPLACE call have the buffer containing the segment may be freed. DL/I must then reread the segment for replacement.

USEFUL TECHNIQUES

INTERMEDIATE DATA BASES

If the user wants to save information between lcads of an application program, without making use of the conversational capability, intermediate data bases can be utilized. Figure 3-16 shcws an intermediate data base being utilized for purchase order writing. Each logical terminal is represented by a root key in the data base. Since all logical terminal names are unique, there is nc possibility of conflicts between terminals. The application program has access to the source of each input through the input/cutput logical terminal PCB.

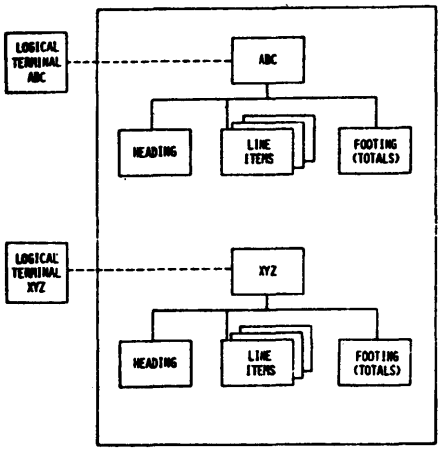


Figure 3-16. Intermediate Data Base

MESSAGE EDITING

If free form input is allowed from input terminals, a single message editing routine is an alternative to redundant code in the COBOL or PL/I program. The message editing routine can ccnvert the message from a free format intc a fixed format.

The edit routine can be located in the IMS/VS control region or in the message processing region with the applicatior program. If an edit routine is to be used a great deal, it should be included in the IMS/VS control region. In some instances this helps reduce the regicn size required for the message prccessing region.

If the edit routine is to be included in the message processing region as a part of the application program, the possibility of placing the module in link pack should be explored.

Use of a single message edit routine for all programs could have value for some user environments. The message edit routine could be the only user module making calls on DL/I for message input/output. This could reduce the amount of error checking required in each application program.

OUTPUTTING A MASK TO THE 2260

When a 2260 is being used in an interactive manner, terminal operator time can be saved by having the application program send out a mask or form to the 2260. The terminal operator then fills in the appropriate information and transmits the entire screen as input.

PASSING INFORMATION FROM ONE PROGRAM TO ANOTHER

When a program is to be designed in a modular fashion, there are several ways in which information can be passed from one program to another. The first way is by sending output to an alternate destination through the queues. Another is to store information in an intermediate data base, as discussed earlier. A third way is to use a scratchpad area for passing the information from one program to another. If the scratchpad area is resident in storage, the input/output overhead is less than by the other approaches.

INTERACTIVE QUERY FACILITY (IQF) FULL FILE SEARCHES

When it is not possible for IQF to eliminate portions of a file from consideration without reading them (by means of IQF indexes or IMS/VS root keys), IQF requests confirmation ('YES') from the terminal user before commencing a full file search. When the confirmation is received, IQF causes a transaction switch to a new transaction code defined in the IQF System Data Base. This allows an installation to control when queries involving a full file search are to be executed. The master terminal operator, for instance, can issue a /PSTOP for the new transaction code, and any future full file search processing can be queued for execution at a later time (when a /START command is issued).

This capability is desirable because in the normal course of operations, IMS/VS takes checkpoints at periodic intervals. If a checkpoint is to be taken while IQF is doing a full file scan (which could require a long processing time), the checkpoint cannot be performed until IQF terminates. The resulting delay could cause unacceptable response times for other transacitons in the system.

Depending upon the installation procedure, the IQF terminal user may know when a full file search transaction code has been queued (by a master terminal operator issuing a /PSTOP command); it might, however, be necessary for him to communicate with the master terminal operator for this information.

It is possible for IQF indexes to be somewhat out-of-date. In a normal IQF query, this situation results in the omission of fields that have been added since the last index update.

When it is imperative that information to be displayed by IQF be up-to-date, and the user data base might have been changed since the last update of the relevant IQF index(es), a full file search can be forced by including some criterion which does not affect the result, but which causes IQF to reject the use of indexes. For example, if the original criterion is:

```
WHEN SALARY EQ 20000
```

the user could force a full file search by changing the query to read:

```
WHEN SALARY NE SALARY OR SALARY EQ 20000
```

IQF does not recognize that this is a trivial special case; it only knows that indexes are useless in resolving certain "Field-Operator-Field" expressions and that a full file search is required.

SECURITY CONTROL IN IQF

Since the Interactive Query Facility (IQF) feature is a generalized transaction, its PSB could include the majority of the PCBs used by the transactions in an installation. This would make normal IMS/VS security control useless for IQF. To avoid this, an installation could establish several different transaction codes for IQF, each transaction code having its own password or terminal security, and a PSB which permits access to only those PCBs needed by a particular group of users.

CHOOSING IQF INDEXING PARAMETERS

The proper use of IQF's static indexing capability can significantly reduce query response time. In order to achieve the optimum indexing structure, an installation should consider the following factors:

- Choice of fields to be indexed
- Frequency of index updates
- Number of index data bases (0, 1 or 2) and size of each index field

As an installation becomes more familiar with the different possibilities for indexing, the system programmer will be able to optimize the choice of fields to be indexed. From time to time, a simple rerun of the Index utility program can be used to restructure the indexes to suit the requirements of the installation.

Choice of Fields to be Indexed

The choice of fields to be indexed can often be the most significant factor. If frequently used fields are not indexed, many queries can result in full file searches. If seldom used fields are indexed, the Index Data Bases can become unwieldy and the time required to update them can become excessive. Fields which occur only a few times, or which have only two or three possible values, probably should not be indexed.

In this connection, it should be noted that fields unknown to IMS/VS, or subfields of other fields (notably the month, day and year portions of a date field), can be indexed by IQF if they are specified to IQF by *FIELD cards.

Frequency of Index Updates

The user should be aware that between index updates the data bases described by the index(es) might have been updated by non-IQF applications. As an index becomes out-of-date, queries using the index will miss a growing number of valid occurences. Whether or not this creates a problem for the terminal operator is largely determined by the degree of accuracy desired. In establishing schedules for index updates, the system programmer should consider the frequency of updates to a given field, as well as the expected requirement for accuracy in queries involving that field.

Since it usually is not practical to update indexes as often as the data bases, the IQF user who requires absolute accuracy, at the expense of time, can bypass the indexes and force a full file search (as described under "IQF Full File Searches" in this chapter). This ensures selection of all data base records that are valid for a particular query.

Number of Index Data Bases and Index Field Size

The number of IQF Index Data Bases required is closely related to the size of the index field in each.

If most of the user data base fields being indexed are nearly the same size (and if none is very much larger), then one Index Data Base may suffice. Its index field must be as large as the largest field being indexed.

If, on the other hand, a few fields to be indexed are considerably larger than the rest, storage space and query time can be saved by designing one Index Data Base to index the relatively small fields and a second Index Data Base to index those fields larger than is allowable in the first Index Data Base.

USE OF PREDEFINED PHRASES IN IQF

IQF predefined phrases can be used in different ways.

- As abbreviations to save typing
- As null words to permit flexible typing style
- As general-purpose synonyms for clarity or easy learning
- As foreign-language equivalents

Performance can be adversely affected when predefined phrases are overused, and when the IQF Phrase data base has been updated considerably without being reorganized. The usual HIDAM reorganization techniques should be used frequently on the Phrase data base to help maintain IQF performance.

The tendency is to use IQF's predefined phrase capability heavily at first. After an installation has gained more experience with IQF, fewer predefined phrases are used (and these primarily as abbreviations). At that time, it is recommended that you delete unused phrases, both to eliminate confusion and to reduce the size of the Phrase data base, which is used at least once for each word in each query.

CHAPTER 4. DATA BASE DESIGN CONSIDERATIONS

The data management portion of IMS/VS is designed to simplify the task of assembling and maintaining large amounts of data while still offering flexibility in how the data is organized and used. To accomplish this, IMS/VS uses an organization for data called a data base.

Under IMS/VS, different types of data bases can be created by the user. Each requires two definitions before the data base can be used by application programs. The user defines the data base structure and content through Data Base Description Generation (DBDGEN). He also defines what data within the data base each application program will use, and what processing options each application program is allowed to use on that data through Program Specification Block Generation (PSBGEN). Each of the data base types is supported by and named after its own access method. The access methods and the data base type each is used for are:

Access Method	Data Base Type
• Hierarchic Sequential Access Method (HSAM)	HSAM
• Hierarchic Indexed Sequential Access Method (HISAM)	HISAM
• Hierarchic Direct Access Method (HDAM)	HDAM
• Hierarchic Indexed Direct Access Method (HIDAM)	HIDAM
• Logical	Logical
• Generalized Sequential Access Method (GSAM)	GSAM

All of the data base types, except the logical data base, are called physical data bases since each physically exists in auxiliary storage as defined through DBDGEN. A logical data base is comprised of data stored in one or more physical data bases that is structured logically to satisfy the requirements of an application program.

GSAM data bases are limited to a single, unstructured data set.

Prior to discussing the advantages and disadvantages of each type of data base, the concepts and terms used for IMS/VS data bases must be understood.

CONCEPTS OF PHYSICAL DATA BASES

In general, an IMS/VS data base is a hierarchic organization of the different types of data required by one or more applications. We'll examine first its content and structure, and we'll then describe the DL/I calls that are used to process a data base. Included in content are segments and fields. Structure includes the definition of the hierarchy of a physical data base. Note that GSAM data bases are not hierarchic and are based on physical records rather than segments.

SEGMENTS

A data base is a storage organization that enables the user to manage multiple sets of data, and multiple elements of each set. The content of a data base is defined by segment type through the DBDGEN utility. For each set of data the user wants to store in a data base, he defines

a segment type and the physical characteristics tc use when storing segments of that type. In turn, when multiple elements of data of a set are stored in a data base, they are stored as segments of the type defined and use the physical characteristics defined for that segment type. A segment in a data base is also called an occurrence. Both terms are used interchangeably to refer to a segment. A data base can contain a maximum of 255 segment types, and the number of segments of any type defined is limited only by the space allocated for the data base.

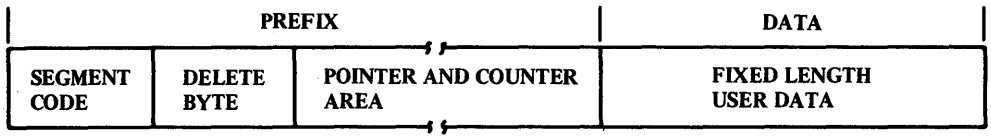
The input to DBDGEN that defines a segment type and the physical characteristics of that segment type is the SEGM statement. Among the physical characteristics defined for each segment type are the name, length, and hierarchic position to be used when storing segments of that type. The name specified is used to identify segments of that type in storage, and in turn, the application program uses the name of a segment type to address the type of segment to be processed. The length specified for a segment type tells IMS/VS the number of bytes of auxiliary storage tc use for the data portion of each segment of that type. For the segments in storage that contain a prefix and data portion, the prefix is used by IMS/VS in managing the segment, and the data portion of the segment contains the user data. The length specified for the data portion of a segment type must be fixed, except when VSAM is used as the operating system access method. When VSAM is used, the length specified for the data portion of a segment type can be either fixed or variable. When variable length is specified, the amount of auxiliary storage space used for the data portion of each segment of that type can vary between a user specified minimum and maximum number of bytes. In the case of fixed, the data portion of each segment of the same type occupies the same amount of auxiliary storage space. The length specified for a segment type cannot exceed the physical record length of the storage device used. The hierarchic position defined for a segment type determines how segments of that type are stored in a data base in relation to segments of other types. For an explanation of the hierarchic position of a segment type in a physical data base, refer tc "structure" in this chapter.

SEGMENT FORMATS

When defining a segment type, the segment length specified by the user can be either fixed or variable. In either case, segments in IMS/VS data bases contain a prefix and a data portion except for three cases where only the data portion is present. For an HSAM, simple HSAM or simple HISAM data base that contains only one segment type, no prefix is stored with occurrences of the segment type in the data base.

The fixed and variable length format for segments in HSAM, HISAM, HDAM and HIDAM data bases are shown in Figure 4-1.

Fixed length segment format



Variable length segment format

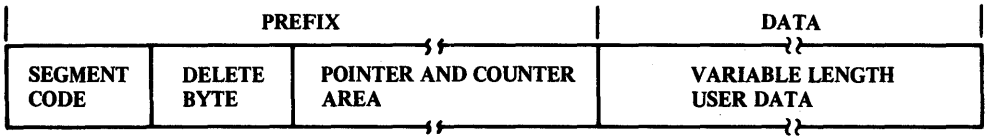


Figure 4-1. Segment Formats

Segments in all data base types contain a prefix and data portion except in the 3 cases stated previously. The prefix of a segment contains data used by IMS/VS that is transparent to application programs. The data portion of a segment contains user data. As a minimum, the prefix of a segment contains a segment code and a delete byte. The content of the remaining portion of the prefix varies by data base type. Segments are related through direct address pointers in HDAM and HIDAM data bases, so all segments in these data bases will contain one or more pointers in their prefix. Segments in HSAM and HISAM data bases are related through physical adjacency so they will have no direct address pointers in their prefix. The one exception to this is a segment in a HISAM data base that is involved in a logical relationship with a segment in an HDAM or HIDAM data base. When the segment in the HISAM data base points to the segment in the HDAM or HIDAM data base, it can have a direct address pointer specified to point to the HDAM or HIDAM segment directly. A counter is only present in the prefix of a segment under certain conditions when it is involved in a logical relationship. For the conditions under which the counter will be present in a prefix and the use of the counter, see "Pointers and the Counter Used in Logical Relationships" in this chapter.

Segment Code

To identify each segment stored in an IMS/VS data base, a one byte segment code is placed in the first byte of the prefix of the segment. The segment code is a number from 1 to 255 that identifies a segment type to IMS/VS in place of its name. Segment code values are assigned to the segment types in a data base in ascending sequence starting with the root segment type and then continuing to all dependent segment types following the hierarchic sequence defined for the segment types in a data base by the user.

Delete Byte

The delete byte is used by IMS/VS to maintain the delete status of segments within a data base. The meaning of each bit within the delete byte is shown in Figure 4-2.

DELETE BYTE

BIT	MEANING
0	SEGMENT HAS BEEN DELETED (HISAM OR INDEX ONLY)
1	DATA BASE RECORD HAS BEEN DELETED (HISAM OR INDEX ONLY)
2	SEGMENT PROCESSED BY DELETE
3	RESERVED
4	DATA AND PREFIX ARE SEPARATED IN STORAGE
5	SEGMENT DELETED FROM PHYSICAL PATH (PHYSICAL DELETE BIT SET)
6	SEGMENT DELETED FROM LOGICAL PATH (LOGICAL DELETE BIT SET)
7	SEGMENT HAS BEEN REMOVED FROM ITS LT CHAIN (BIT 7 IS ASSUMED SET IF BITS 5 AND 6 ARE SET)

Figure 4-2. Delete Byte

FIELDS

An application program addresses segments in a data base through the name specified for their segment type, and through the names of fields defined within a segment type. The segment name alone allows an application program to address a specific segment type within a data base. To address a specific occurrence of a segment type, fields must be defined within that segment type.

Within the data portion of each segment type, the user can define fields through the DBDGEN utility. Each is defined through a FIELD statement which is input to DBDGEN. The maximum number of fields that can be defined within a segment type is 255 and the maximum within a data base is 1000. The two types of fields that can be defined are sequence fields and data fields. Both fields can be used by an application program to address specific segments in a data base. A sequence field, often referred to as a key field, has two purposes in addition to that of the data field. It is used to store occurrences of a segment type in a sequential order. The order is determined by the value placed in the sequence field of each occurrence. The value in the sequence field of a segment is called the key of that segment. Occurrences of a segment type are stored in ascending order in the data base starting with the occurrence that contains the lowest sequence field key. The sequence field is also used as all or part of a symbolic pointer to a segment in a data base. The symbolic pointer is actually the concatenation of the keys in the sequence fields of all segments that must be retrieved to reach the desired segment including the sequence field key of the desired segment as shown in Figure 4-3.

Only one sequence field can be defined in each segment type. Sequence fields can be defined as unique or non-unique by the user. When defined as unique, occurrences of a segment type must contain sequence field values that uniquely identify them within a data base, and segments are stored in the data base in the manner described previously. When defined as non-unique, sequence field values do not have to uniquely identify a segment within a data base. In this case, segment occurrences are still sequenced according to their sequence

field value which controls all segments except those with the same value. If placement of those with the same value is of concern, the user must control their placement either through his data base load program, or through the combination of his load program and by stating insert rules for segment placement through DBDGEN.

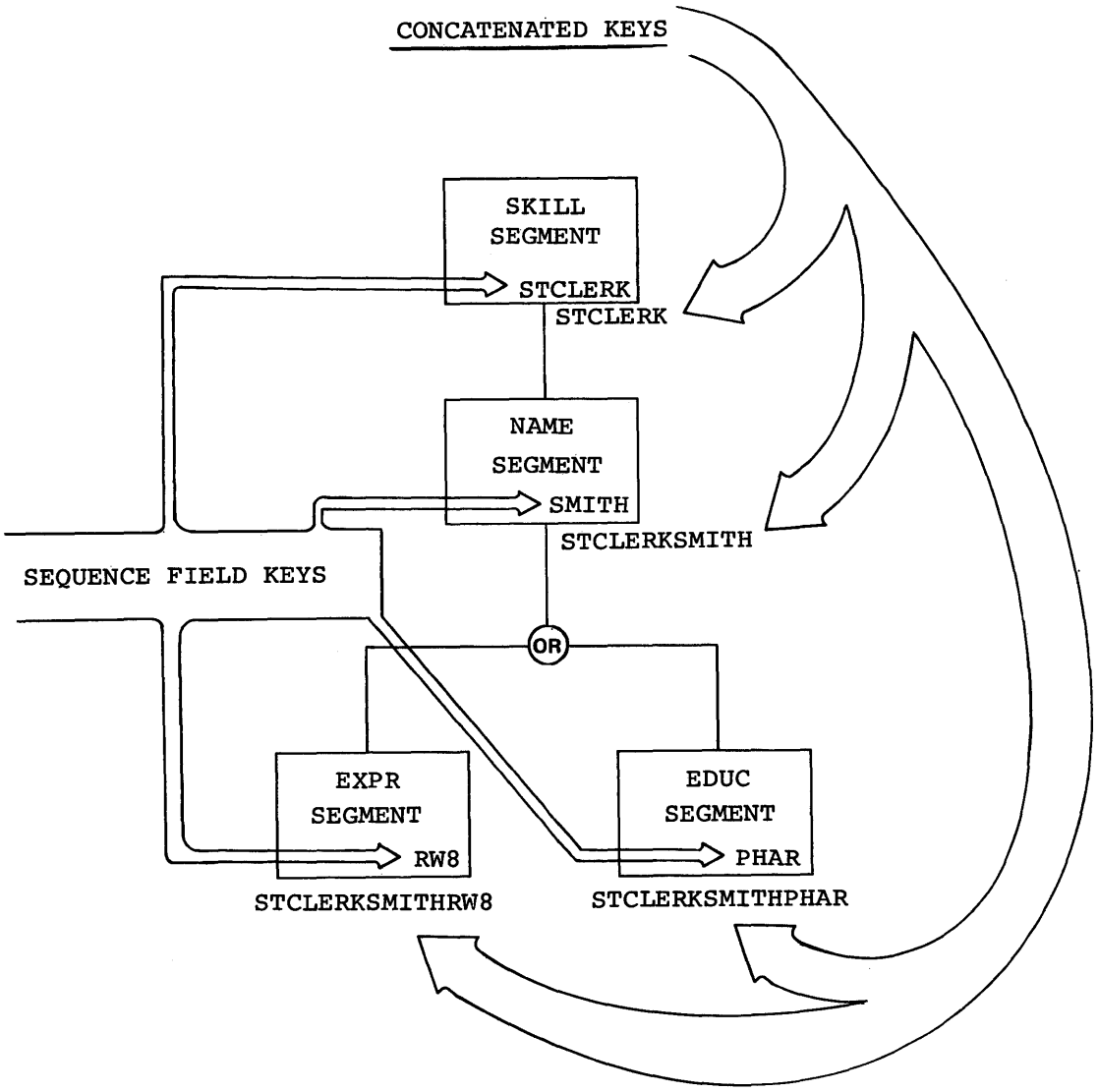


Figure 4-3. Concatenated Keys

STRUCTURE

The hierarchy of a data base is created by defining an crder of dependence for the segrent types it contains. To IMS/VS, the hierarchy represents the order in which the user wants his application data stored. To the applicatien program, it represents the order in which the segment types in a data base are available fcr processing. The criteria normally used in determining the hierarchy is how the data in one segment type relates to data in another segment type, and the frequency in which a segment type will be accessed by an application program.

To understand how a data base hierarchy is developed, we'll use as an example a skills inventory application. We'll determine what segment types a data base should contain for the applicatien and the hierarchic order of those segment types. In addition, we'll show the data base that could result in storage by defining the segment types and their hierarchic crder through the DBDGEN utility.

Let's assume in our example that an application program wants to locate a given skill, and then find out what emplcyees pcssess that skill. In addition, the application must have access to the experience and education records of each employee.

In the assumptions, fcur sets of data were required by the skills inventory application. Each will be defined as a segment type in our skills inventory data base. Let's now create a hierarchic data structure that reflects the crder in which the four segment types are required by the application. To do this, we must determine both the order in which the applicatien program must use each type of data, and the order in which each type of data must be presented to the applicatien program so that the data retains its meaning. In the assumptions, it was stated that the applicatien wanted to find a given skill and then find the names of the employees that possessed that skill. From this statement we know that skill shculd be the first type of data in our structure and that name should follow skill. For the remaining types of data, experience and educatien, nc indication was given as to how they should fit in our structure, but we can determine their position in the structure if we can establish how they should relate to the skill and name types of data. Fcr cur application, experience and education data have no meaning by themselves, to each other, or in relation to skill. When related to rame data however, the experience and educatien types of data do have meaning since they are the experience and education records of specific employees. We can now complete our data structure. At the top is skill, below skill is name, and name in turn is followed by experience and education as shown in Figure 4-4. Experience and education are below name since they are dependent cn name for the skills inventory application. An employees name must be estaklished before his experience and educatien records have meaning. In turn, name is dependent on skill since the application will locate a given skill and then asscciate employee names to that skill. In summary, the data structure shown in Figure 5.4 represents the sets of data and the order of dependence fcr those sets required by the skills inventory application. It contains four sets of data arranged in three levels of dependence. An IMS/VS data base can contain a maximum cf 255 sets of data arranged in up to 15 levels of dependence.

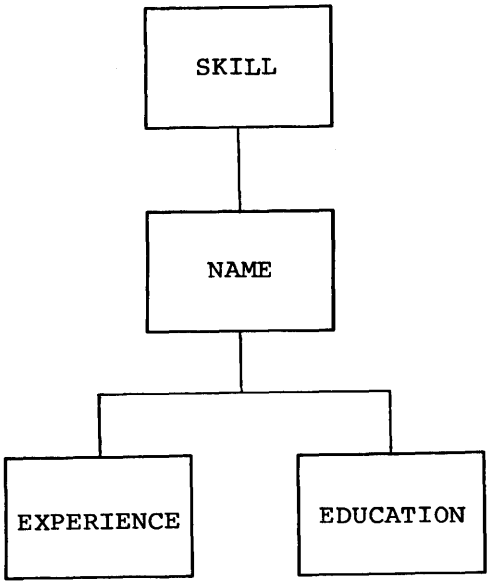


Figure 4-4. Hierarchy of Segment Types

The data structure just created is called a hierarchy of segment types. It represents the segment types and the hierarchic arrangement of those segment types that would be defined through DBDGEN to define our skills inventory data base. If we now assume data for each segment type, Figure 4-5 shows the data base that would result in storage.

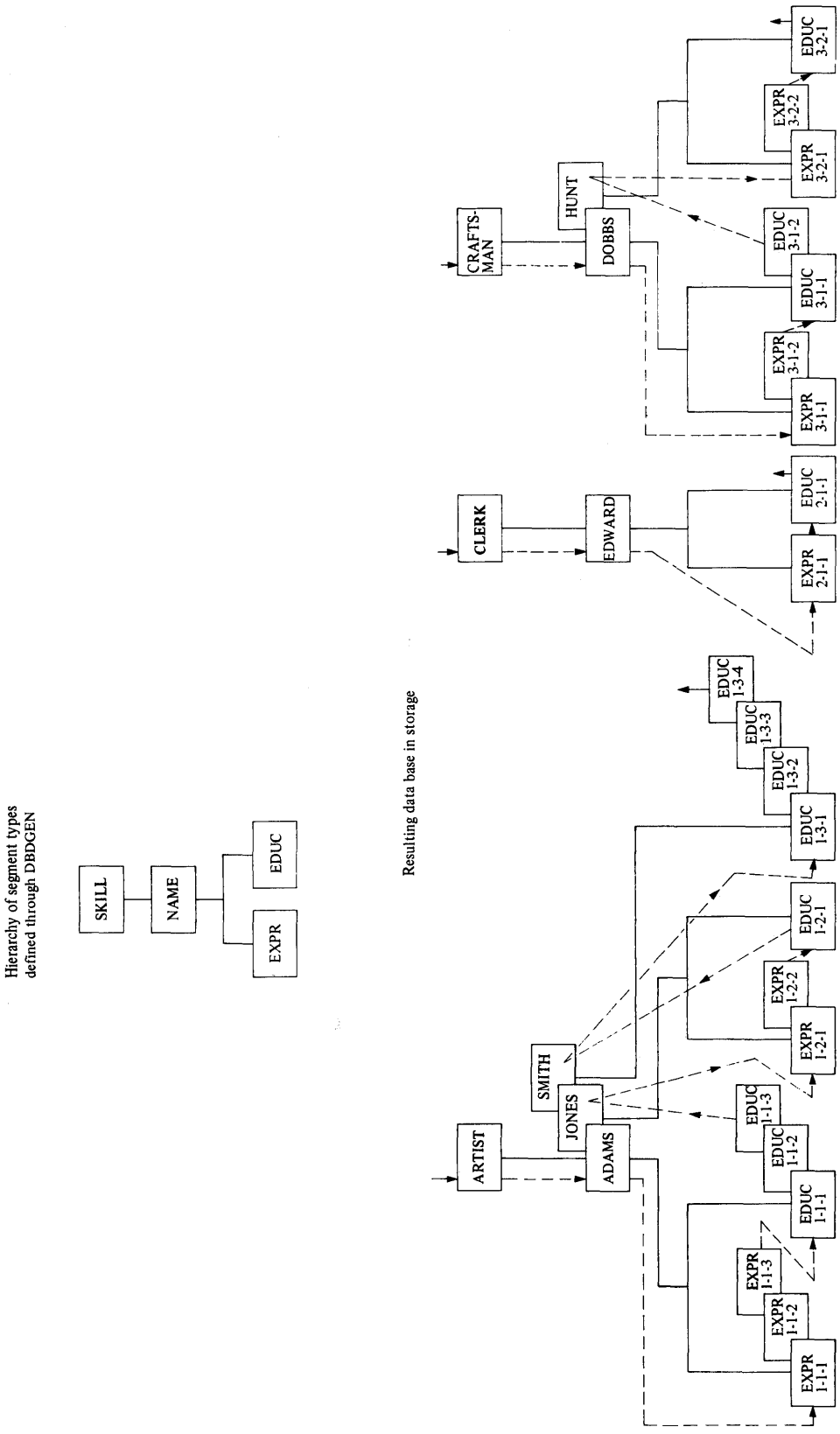


Figure 4-5. Data Base in Storage

Figure 4-5 shows how segments of each type can be loaded in a data base. Three occurrences of the skill segment type are shown. Related to each are the specific occurrences of the Name segment type that contain the names of the employees who possess that particular skill. Related to each Name segment in turn, are the specific segments of the Experience and Education segment types that contain the experience and education records of each employee. Skill is the root segment type in this data base, and each data base has only one. The root segment type is always the first segment type defined in a data base, and, as shown, it is the only segment type that occupies the first level of a data base hierarchy. Segments of all other types in a data base are stored in relation to an occurrence of the root, and as such are termed dependents of the root. In addition, occurrences of the Experience and Education segment types, shown at the third level of the hierarchy, are dependents of the Name segment type since they are stored in relation to occurrences of the Name segment type. When a data base hierarchy is read from top to bottom with the root at the top, each lower level in the hierarchy contains the dependent segments of the segments at the next higher level. Before any dependent segment is loaded in a data base, the segments on which it is dependent must be loaded. In all cases, a dependent segment in a data base is dependent on one segment at each higher level in the hierarchy.

The major unit of organization for segments within a data base is the data base record. A data base is comprised of one or more data base records, and a data base record contains one occurrence of the root segment type and all of its dependents arranged in hierarchic sequence. Hierarchic sequence for the segments in a data base record is top to bottom, then left to right passing through each segment only once. The skills inventory data base shown in Figure 4-5 contains three data base records, and the hierarchic sequence of each is shown.

The hierarchy of a physical or logical data base can contain a maximum of 15 levels. The order of dependence for segment types in the hierarchy is from level one, or the top of the hierarchy, to level 15, the bottom of the hierarchy. The top level of the hierarchy of any data base can contain only one segment type. It is called the root segment type for that data base. Subsequent levels below the root can contain any number of segment types such that the maximum of 255 total segment types in the data base is not exceeded.

Defining a Physical Data Base Hierarchy

The input to the DBDGEN utility that defines the segment types a data base contains, their physical characteristics and their hierarchic position is the SEGM statement. (The SEGM statement is described in the IMS/VS Utilities Reference Manual.) For our explanation here, it is only necessary to know that a data base hierarchy is defined through the order in which SEGM statements are arranged for input to DBDGEN, and through use of the PARENT= operand of the SEGM statement.

The PARENT= operand of the SEGM statement is used to define the physical relationships that exist between the segment types on each two adjacent levels in a data base hierarchy. The two segment types involved in the relationships are called a physical parent and a physical child. A physical parent is a segment type that has a dependent segment type defined at the next lower level in the data base hierarchy. A physical child is a segment type that is dependent on a segment type defined at the next higher level in the data base hierarchy. These two terms are used to state the order of dependence for the segment types in a data base. In a data base with multiple segment types defined, the root segment type is the physical parent of all segment types defined at the second level of the data base. In turn, the segment types on the second level are physical children of

the root. Each level of a data base contains the physical parent segment types of any segment types defined at the next lower level, and the physical child segment types of any segment types defined at the next higher level. The PARENT= operand of the SEGM statement is used to state specifically which segment type at the next higher level is the physical parent of a physical child. All segment types in a data base, except the root, are physical children since each is dependent on at least the root. On the SEGM statement that defines each physical child, the PARENT= operand is used to specify the name of the physical parent segment type.

The PARENT= operand of the SEGM statement defines the top to bottom order of segment types. The arrangement of SEGM statements for input to the DBDGEN utility defines the left to right arrangement of segment types. For input to DBDGEN, SEGM statements must be arranged in hierarchic sequence. Hierarchic sequence is defined as top to bottom, then left to right passing through each segment type only once. The segment types in the hierarchic structure shown in Figure 4-6 are numbered to show the order in which SEGM statements to define that structure must be arranged for DBDGEN input.

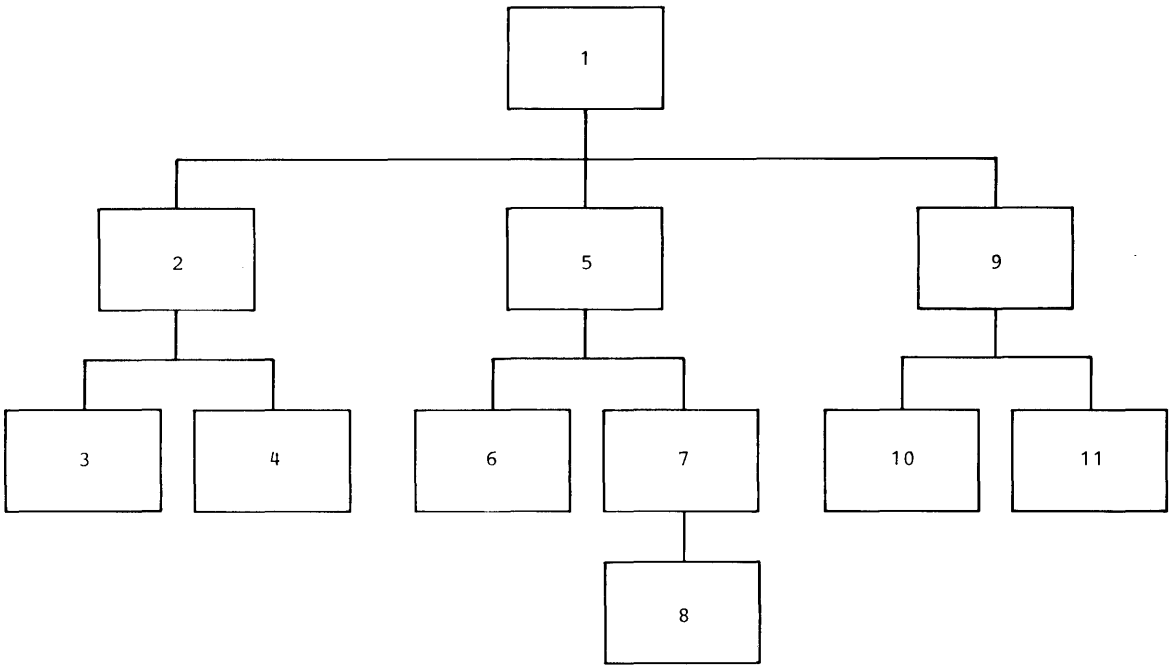


Figure 4-6. Segment Types Numbered in Hierarchic Sequence

Previously, the terms physical parent and physical child were defined. The remaining term used to describe physical relationships is physical twin. Physical twins are occurrences of the same segment type that are dependent on the same occurrence of the physical parent segment type. In Figure 4-7, the three Name segments Adams, Jones and Smith are physical twins since all three are dependent on the skill segment that contains the data artist. Under Adams, the three Experience segments are physical twins and the three Education segments are physical twins since, in each case, the three are of the same segment type under the same occurrence of the physical parent segment type.

The term sibling, used frequently in data base literature, refers to the relationship between two or more segment types at the same level that are dependents of the same parent type segment.

The hierarchies of all four physical data base types are defined as just described, but in auxiliary storage, each of the four physical data base types is organized differently. To understand the advantages of one data base organization over another, a basic understanding of the DL/I calls used to process segments in a data base is required, since the primary trade off between the four types is auxiliary storage space versus the performance of application programs processing a data base through DL/I calls.

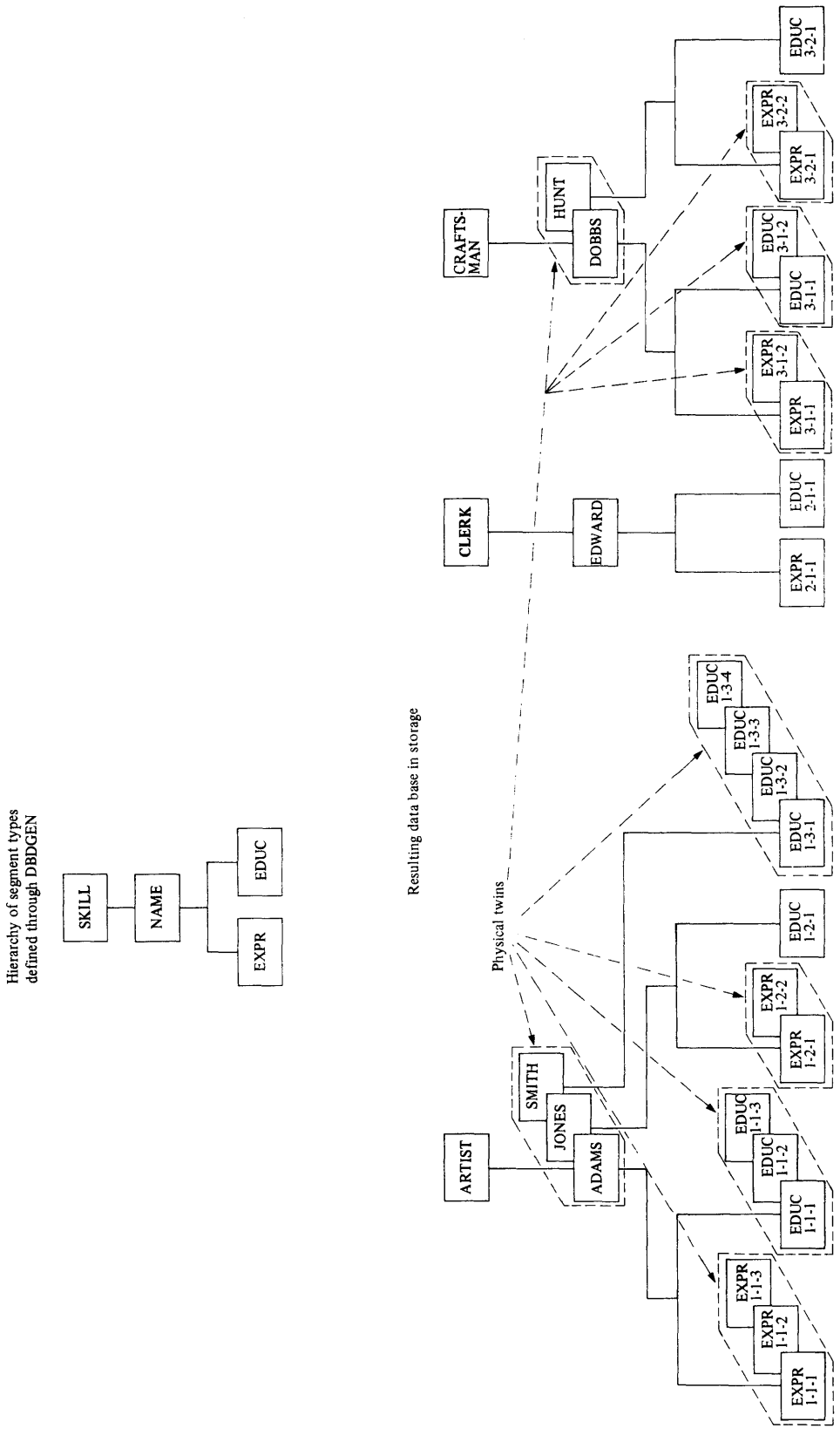


Figure 4-7. Physical Twins

CALLS

The segments in an IMS/VS data base are processed through calls issued by an application program. Calls are issued to get, insert, delete or replace a segment or a path of segments at a time. A call references a parameter list which includes all data required by IMS/VS to complete the call. Included in the list are a function code and an SSA (segment search argument). The function code states the call to be performed, and the SSA states the segment or path of segments to be processed. A call is unqualified when no SSA is included with the call, and a call is qualified when an SSA is provided for the call. A brief description of the primary calls used in processing a data base and a brief description of SSAs follows. For more detailed information, refer to the IMS/VS Application Programming Reference Manual.

The direction of movement in a data base is called forward when the search for a given segment is going away from the first occurrence of the root stored in a data base towards the last segment stored in the data base. Backward movement is movement in the opposite direction. Position in a data base is the segment or segments in a data base from which the search for another segment starts.

Get Unique

The GU (get unique) call is used to retrieve a specific segment or path of segments from a data base, and at the same time establishes a position in a data base from which additional segments can be processed in a forward direction.

Get Next

The GN (get next) call is used to retrieve the next desired segment or path of segments from a data base. The get next call normally moves forward in the hierarchy of a data base from current position. It can be modified to start at an earlier position than current position in the data base through a command code, but its normal function is to move forward from a given segment to the next desired segment in a data base.

Get Next within Parent

The GNP (get next within parent) call is used to retrieve dependent segments of a given parent segment from a data base. A get unique or get next call is used to establish parentage for the get next within parent. After a get unique or get next retrieves a given parent segment, successive get next within parent calls retrieve the dependent segments of that parent in hierarchic sequence.

Hold Form of Get Calls

Another form of the three get calls is the hold form. A GHU (get hold unique), GHN (get hold next), or GHNP (get hold next within parent) indicates the intent of the user to issue a subsequent delete or replace call. A get hold call must be issued before issuing a delete or replace call.

Insert

The ISRT (insert) call is used to insert a segment or a path of segments into a data base. It is used to initially load segments in

all data base types, and tc add segments to existing HISAM, HDAM and HIDAM data bases. Segments can not be inserted cr added into an HSAM data base except at lcad time.

To control where cccurrences of a segment type are inserted into a data base, the user can define a unique sequence field in the segment type, or specify insert rules that control placement of occurrences of a segment type that has no sequence field or a ncn unique sequence field defined. When a urique sequence field is defined in a root segment type, the sequence field of each occurrence of the rcot segment type must contain a unique value. When defined for a dependent segment type, the sequence field of each occurrence under a given physical parent must contain a unique value.

Following are the insert rules the user can specify to control the placement of segments in a data base. They are used to ccntrol the placement of occurrences of a segment type with non-unique sequence field values and for placement of all occurrences of a segment type when no sequence field has been defined.

FIRST - States that a new occurrence of a segment type is inserted before the first existing occurrence of this segment type. If this segment has a non-unique key, a new occcurrence is inserted before all existing cccurrences of the same type that contain the same sequence field key.

IAST - States that a new occurrence is inserted after the last existing cccurrence of this segment type. If this segment has a non-unique key, a new cccurrence is inserted after all existing occurrences of the same type that contain the same sequence field key.

HERE - Assumes the user has established position on the specified segment type by a previous Data Language/I call and the new occurrence is inserted before the segment which satisfied the last call. If current position is not within occurrences of the segment type being inserted, the new occurrence is inserted before all existing occurrences of that segment type. If this segment has a non-unique key and data base position is not within occurrences of the segment type with equivalent key value, a new occurrence is inserted before all existing occurrences that contain the same sequence field key.

Delete

The DLET (delete) call is used to delete a segment or path of segments from a data base. It should be noted that, due to the hierarchic arrangement of segments in a data base, the deletion of a parent segment implies the deletion of that parent's dependents. When a parent segment is deleted in an IMS/VS data base, all of its dependents are deleted.

Replace

The REPL (replace) call is used to replace the data in the data portion of a segrent or path cf segments in a data base.

SSA (Segment Search Argument)

An SSA identifies a segment or group of segments that are to be processed by a call. An SSA can contain three parts. As a minimum, it contains the name of the segment type to be processed. Optionally, an SSA can contain command codes and/or qualification statements. Command codes, when used, specify a functional variation of a call. Qualification statements identify through fields which segment or segments of the specified segment type are to be processed by the call. A qualification statement contains a field name, relational operator and comparative value. When occurrences of the segment type are searched, the specified field is compared to the comparative value in accordance to the relational operator specified.

PHYSICAL DATA BASE ORGANIZATION IN STORAGE

HIERARCHIC SEQUENTIAL AND DIRECT METHODS OF STORING A DATA BASE

Two storage organization methods are used to create the hierarchic arrangement of segments in storage for the four physical data base types. The hierarchic sequential method is used for HSAM and HISAM data bases, and the hierarchic direct method is used for HDAM and HIDAM data bases. The hierarchic sequential method consists of using physically adjacent storage locations to store all segments within a data base record in hierarchic sequence. This creates a hierarchy for the occurrence of the root and all of its dependents within each data base record in which each segment is related to the segment that hierarchically follows it through physical adjacency in storage. The hierarchic direct method consists of placing four byte direct address pointers in the prefix of each segment stored in the data base to establish the hierarchy of segments in each data base record. A description of the types of pointers used in HDAM and HIDAM data bases follows.

Pointers

To relate each segment in an HDAM or HIDAM data base to its related segments, direct address pointers are used. The pointers are four bytes long, and they are placed in the prefix of each segment stored in the data base. A direct address pointer consists of the relative byte address of a segment from the beginning of a data set. Either one of two methods of direct address pointing can be specified for each segment type in an HDAM or HIDAM data base. The two methods are hierarchic pointing, or the combination of physical child/physical twin pointing. Figure 4-8 should be referred to when reading the following descriptions of the types of pointers.

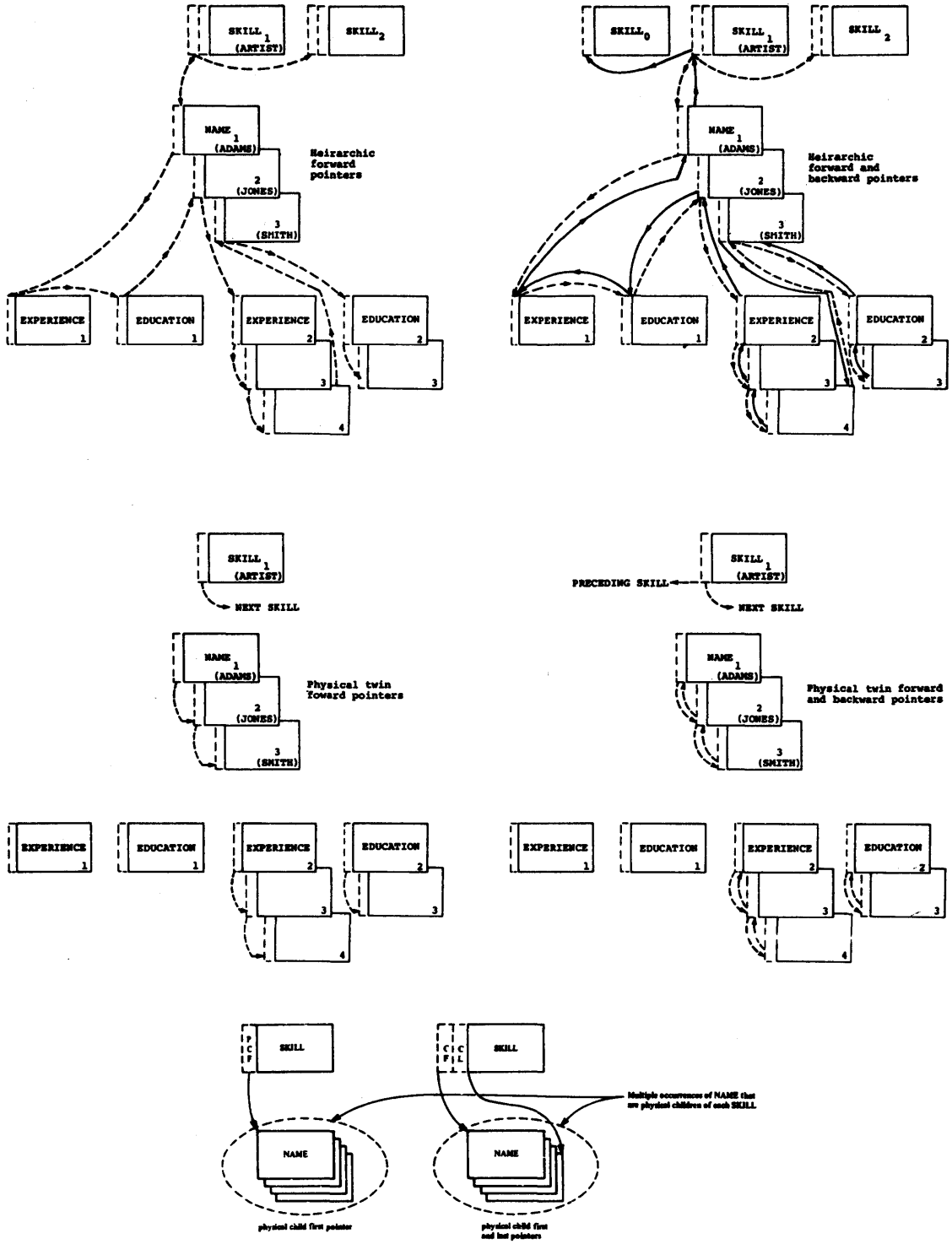


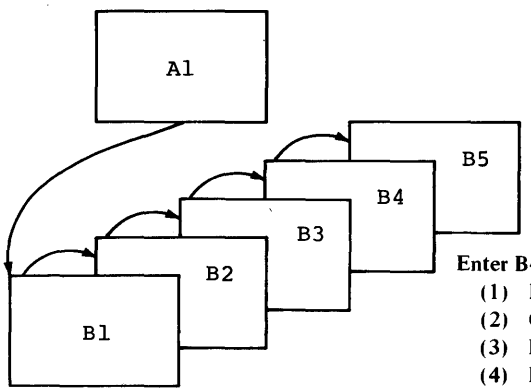
Figure 4-8. Direct Address Pointers

Hierarchic Pointers

Two options for hierarchic pointing can be specified for each segment type in an HDAM or HIDAM data base. They are hierarchic forward, or hierarchic forward and backward pointing. When hierarchic forward pointers are specified for all segment types in a data base, each segment in a data base record points to the segment that hierarchically follows it through a four byte hierarchic forward pointer. When forward and backward pointers are specified, the backward pointer points from each segment in a data base record to the segment that hierarchically precedes it. The use of hierarchic pointers in an HDAM or HIDAM data base results in the same arrangement of segments within each data base record as the hierarchic sequential method provides in an HSAM or HISAM data base, but rather than segments being related through physical adjacency, they are related through pointers that require additional auxiliary storage space. For most data bases with high update activity, the additional auxiliary storage space used for pointers is more than compensated for through the space reuse facilities gained in HDAM and HIDAM data bases.

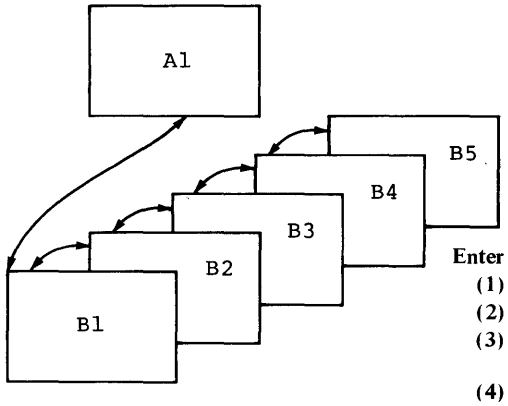
In a data base that contains hierarchic pointers, when a call is issued to process a segment in the data base, the hierarchic forward pointers are followed in searching for the segment to be processed. Hierarchic backward pointers are used only when a segment is being deleted. For delete, the backward pointers improve performance by enabling the pointers in the segments that hierarchically precede and follow the segment to be deleted to be updated without first going to the physical parent of the segment being deleted. With forward only pointers, deletion of a dependent segment requires going to the physical parent of the dependent, and then searching forward to update the pointer in the segment that precedes the segment being deleted as shown in Figure 4-9.

Delete segment B4:



Enter B4 to delete B4:

(1) Place pointer to B5 that is in B4 in work area.
(2) Go up to A1.
(3) Follow pointers forward to B3.
(4) Replace pointer to B4 that is in B3 with pointer to B5 from work area.
(5) Delete B4.



Enter B4 to delete B4:

(1) Place Forward & Backward pointers that are in B4 in work area.
(2) Follow backward pointer to B3.
(3) Replace pointer to B4 that is in B3 with pointer to B5 from work area.
(4) Follow Forward pointer to B5.
(5) Replace pointer to B4 that is in B5 with pointer to B3 from work area.
(6) Delete B4.

Figure 4-9. Use of Backward Pointers for Delete

Physical Child/Physical Twin Pointers

Physical child/physical twin pointers benefit applications that process the segment types in a data base randomly. They allow the most direct paths to the dependent segment types in a data base. Two options for physical child and/or physical twin pointers can be specified for each segment type in a data base. The physical child pointers that can be specified are physical child first, or both physical child first and last. The physical twin pointers that can be specified are physical twin forward, or both physical twin forward and backward. When specified for all physical child segment types, physical child pointers are stored in the prefix of each physical parent segment, and they point to each of the physical child segment types of that physical parent segment. A physical child first pointer points from a physical parent segment to the first occurrence of a physical child segment type in a data base that is a dependent of that physical parent. A physical

child last pointer points from a physical parent segment to the last occurrence of a physical child segment type in a data base that is a dependent of that physical parent. If a physical parent segment has multiple physical child segment types, its prefix contains physical child first, or first and last pointers to each of those physical child segment types. Physical twin pointers are used to relate all occurrences of the same physical child segment type that are dependents of the same physical parent segment. Physical twins are multiple occurrences of the same segment type that are dependent on one occurrence of a physical parent segment type. A physical twin forward pointer points from a given twin to the twin following it in the data base, and a physical twin backward pointer points from a given twin to the twin before it in the data base.

In searching for a given segment in a physical data base using physical child/physical twin pointers, the physical child first and physical twin forward pointers state the hierarchic path to be followed in search of the segment. The normal path followed in locating a desired segment is from a given physical parent segment to the first occurrence of one of its physical child segment types, and then forward through all occurrences of that segment type to the last occurrence following physical twin forward pointers.

A physical child last pointer enables a search to go directly from a physical parent to the last occurrence of one of its physical child segment types as shown in Figure 4-10. In so doing, the physical child last pointer eliminates the forward search of all occurrences of a segment type under one physical parent when only the last occurrence of the physical child is desired. A physical child last pointer is used when inserting a new segment with no sequence field and the insert rule specified is last, or for get or insert, when command code L is specified for the call and the SSA for the call has no qualification statement. When a physical child last pointer is followed to the last occurrence of a dependent segment, any further movement in the data base is forward. A physical child last pointer does not enable searching from the last to the first occurrence of a dependent segment under one physical parent segment.

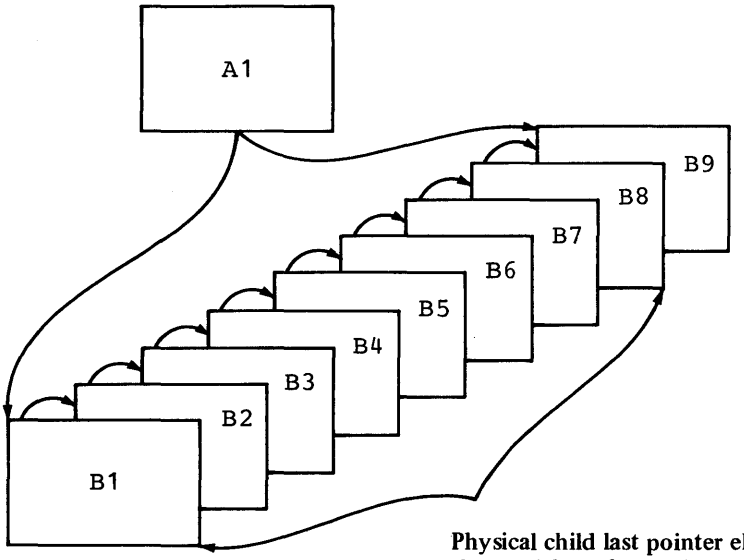


Figure 4-10. Use of Physical Child Last Pointer

Physical twin backward pointers in dependent segment types are used to improve delete performance as described for hierarchic backward pointers. In addition, when physical twin forward and backward pointers are specified for the root segment type of a HIDAM data base, they enable sequential processing across data base records without intervening references to the HIDAM index. When only physical twin forward pointers are specified for the root segment type of a HIDAM data base, sequential processing across data base records requires intervening references to the HIDAM index.

DATA SET GROUPS

To describe what data sets are used for storing the segment types in a data base, and to describe the physical characteristics of those data sets, data set groups are defined through the DBDGEN utility using DATASET statements. For an HSAM data base, one data set group is defined. For HISAM, HDAM and HIDAM data bases, from one to 10 data set groups can be defined. The terms used to describe data set groups are primary and secondary. A primary data set group contains the root segment type. All other data set groups are called secondary data set groups. A primary data set group must be defined for each data base type. A secondary data set group is normally defined to enable using data sets with different logical record and control interval or block lengths to enhance auxiliary storage space utilization. In a HISAM data base, a secondary data set group offers one additional advantage. It enables direct access to a segment type at the second level of a HISAM data base without first accessing a root.

Rules for Dividing a Data Base into Data Set Groups

HISAM, HDAM and HIDAM data bases can be divided into a maximum of 10 data set groups according to the rules that follow.

For HISAM data bases, secondary data set groups cannot be defined when VSAM is used as the OS/VS access method for the data base, or when a HISAM data base is indexed by a secondary index. HISAM data bases using ISAM/OSAM as the OS/VS access methods and not indexed by a secondary index can only be divided into multiple data set groups at the second level of its hierarchy. The first segment type defined in a secondary data set group must be a segment type defined at the second level of the hierarchy of a HISAM data base. Included in a secondary data set group, are all segment types dependent on the first segment type defined in that secondary data set group.

For HDAM or HIDAM data bases, secondary data set groups can be started with a segment type defined at any level of the hierarchy and the secondary data set group can contain any combination of the segment types in the data base. However, the following restriction must be met. A physical parent and its physical children must be connected by PC/PT pointers when they are in different data set groups; a PC/PT pointer means that each parent must be a physical child (PC) pointer to the first occurrence of each child type, and that the children must be connected to each other by physical twin (PT) pointers.

HSAM STORAGE ORGANIZATION

In an HSAM data base, all data base records within a data base, as well as all segments within each data base record are related through physical adjacency in storage as shown in Figure 4-11. An HSAM data base is stored on a tape, or a direct access storage device as a sequential data set. The data set is loaded in chronological sequence and it uses a fixed length unblocked format (RECFM=F). Since the data set is loaded in chronological sequence, the order in which the user presents each segment to be stored in the data set establishes the hierarchic arrangement of segments in the data base. A sequence field is not required in the root segment type of an HSAM data base.

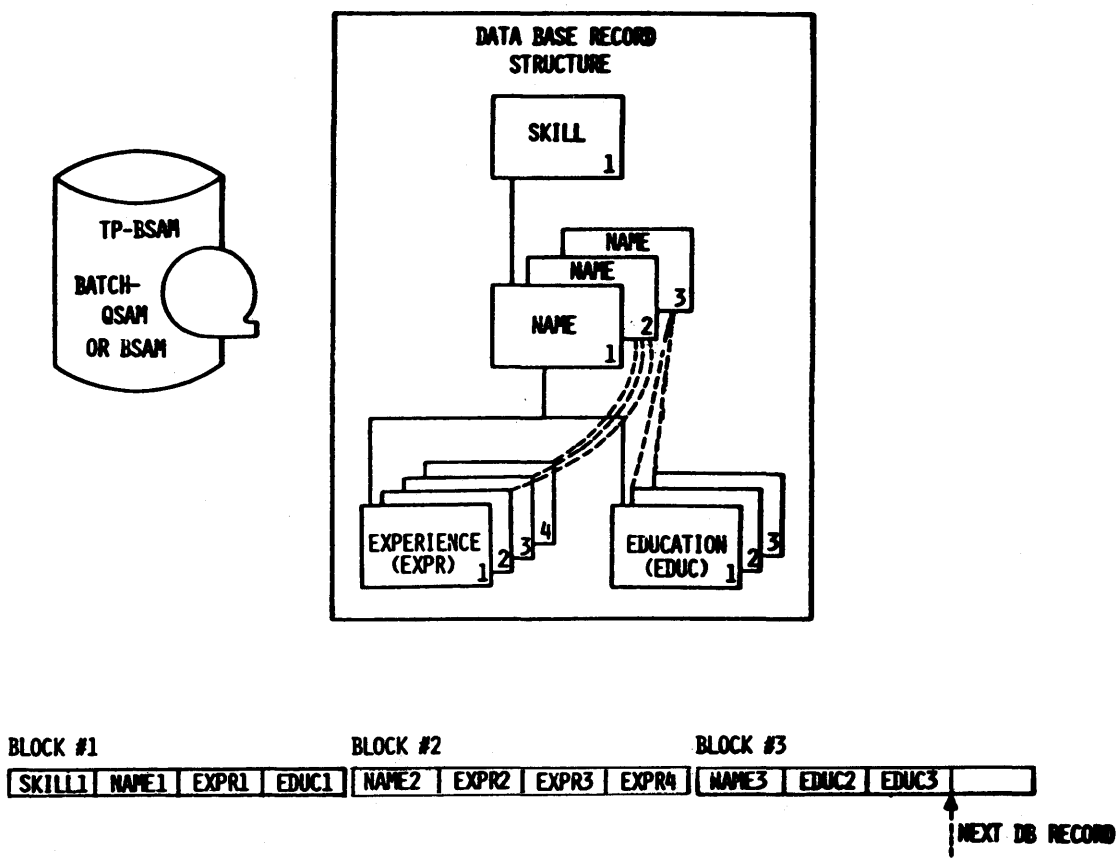


Figure 4-11. One Data Base Record of HSAM Data Base on Tape

When a sequence field is defined in the root segment type, each data base record must be presented for loading in ascending key sequence. Within each data base record, all segments must be presented for loading in hierarchic sequence.

In the data set, one or more consecutive blocks are used to store a data base record. Each block is filled with segments of a data base record until the remaining space is not sufficient for the next segment to be stored. When not sufficient for the next segment to be stored, the remaining space in the block is padded with zeros and the segment is stored in the next consecutive block. When the last segment of a data base record has been stored in a block, any unused space, if sufficient, is filled with segments from the next data base record.

Initial entry to an HSAM data base is through get unique or get next calls. When the first call is issued, the search for the desired segment starts at the beginning of the data base in storage, and passes sequentially through all segments stored in the data base until the desired segment is reached. After the desired segment is reached, the position it occupies is used as the starting position for any additional calls that process the data base in a forward direction. From current position in an HSAM data base that has a unique sequence field defined in the root segment type, if a get unique is issued to retrieve a segment that is forward in the data base, the search starts from current position and moves forward to the desired segment. If the desired segment requires backward movement in the data base, the processing option parameters G or GS, which are specified during PSBGEN, determine how backward movement is accomplished. The G processing option

specifies the get function only, whereas the GS processing option specifies get segments in ascending sequence only. If GS has been specified and backward movement in a data base is required to satisfy a get unique, the search for the desired segment will start at the beginning of the data base and move forward. Under the same conditions when the G processing option is specified, from current position the search will move backwards in the data base. This is accomplished by backspacing over the block just read on tape or disk, and the block previous to the block just read, then reading the previous block forward until the desired segment is found.

An HSAM data base can be randomly processed through get unique calls within one volume. When no sequence field has been defined in the root segment type of an HSAM data base, each get unique causes the search for the desired segment to start at the beginning of the data base regardless of current position.

Insert, delete and replace calls cannot be used when processing an existing HSAM data base. The only calls that are valid for processing an existing HSAM data base are the get calls which enable retrieval of segments from the data base only. To update an HSAM data base, it must be reloaded.

Simple HSAM

A simple HSAM data base is an HSAM data base that contains only one segment type. When a simple HSAM data base is defined, occurrences of the segment type are loaded into the data base without prefixes.

HISAM STORAGE ORGANIZATION

In a HISAM data base, segments within each data base record are related through physical adjacency in storage as with an HSAM data base. Unlike HSAM however, in a HISAM data base each data base record is indexed. In defining a HISAM data base, the user must define a unique sequence field in the root segment type of the data base. When defined, the sequence field values in occurrences of the root are used to index to each data base record in the data base.

In defining a HISAM data base, the user can specify VSAM or the combination of ISAM/OSAM as the access methods to be used for the data base. When ISAM/OSAM are specified, he can also specify that the HISAM data base be stored as one to 10 data set groups. If VSAM is specified, a HISAM data base can have only one data set group. When VSAM is specified as the access method, a data set group contains one key sequenced data set and one entry sequenced data set. When ISAM/OSAM are specified as the access methods each data set group contains one ISAM data set and one OSAM data set. In both cases, one data set, key sequenced or ISAM, is used for primary storage of segments and as an index. The other data set, entry sequenced or OSAM, is used for overflow storage of segments. The terms used to describe data set groups are primary and secondary. A primary data set group contains the root segment type. All other data set groups are called secondary data set groups.

HISAM Data Base Stored as One Data Set Group

When only one data set group is defined for a HISAM data base, the data base is stored as shown in Figure 4-12. Each key sequenced or ISAM logical record will contain in hierarchic sequence an occurrence of the root, plus all dependents of the root that there is sufficient space for in the logical record. When no space remains for the

remaining segments in a data base record, the remaining segments are stored in hierarchic sequence in one or more logical records of the entry sequenced or OSAM data set. To relate all logical records in both data sets that contain segments in one data base record, a direct address pointer is stored in each logical record to chain them in hierarchic sequence.

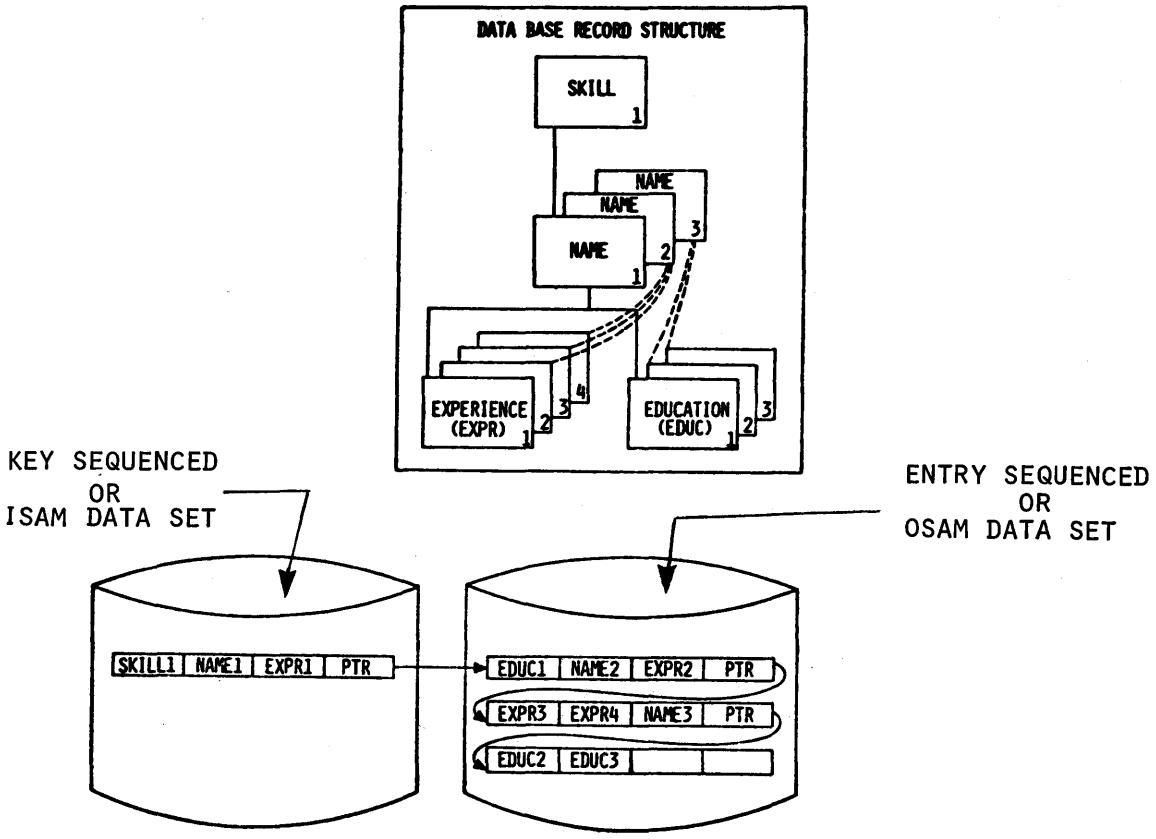


Figure 4-12. HISAM Data Base Record in Storage (Single Data Set Group)

The structures of logical records for VSAM, and ISAM or OSAM data sets are shown in Figure 4-13. The first 3 bytes of each logical record for ISAM or OSAM, and the first 4 bytes of each logical record for VSAM are used for a direct address pointer. The pointer is used to maintain root segments in ascending key sequence and to maintain all segments within a data base record in hierarchic sequence when new segments are inserted into a data base after initial load. Following the pointer are one or more segments of a data base record in hierarchic sequence. At the end of the last segment in the logical record for VSAM, ISAM or OSAM, a one byte segment code of zero is stored to indicate that the last segment in the logical record has been reached. Following the zero segment code for VSAM, remaining space in a logical record is residual. Following the zero segment code for ISAM or OSAM, there are three bytes of zeros, or a 3 byte direct address pointer.

ISAM/OSAM Format

↑ NOTE 1	SEGMENT	SEGMENT	SEGMENT	SEGMENT CODE O (1 BYTE)	OOO OR NOTE 2	RESIDUAL

VSAM Format

↑ NOTE 3	SEGMENT	SEGMENT	SEGMENT	SEGMENT CODE O (1 BYTE)	RESIDUAL

NOTES:

1. The pointer is comprised of the 3 byte relative record number of the OSAM data set logical record that contains a root inserted after initial load.
2. The pointer is comprised of the 3 byte relative record number of the OSAM data set logical record that contains the next dependents in hierarchic sequence.
3. The pointer is comprised of the 4 byte relative byte address of the entry sequenced data set logical record that contains the next dependents in hierarchic sequence.

Figure 4-13. HISAM Data Base VSAM, ISAM and OSAM Logical Record Formats

Three bytes of zeros indicate that this logical record contains the last segment in a data base record. If not zeros, a three byte pointer points to the logical record that contains the next segments in the data base record in hierarchic sequence.

In a VSAM data set, one or more logical records are contained in each control interval. In an ISAM or OSAM data set, one or more logical records are contained in each block. A control interval or block is the unit of data transferred between an I/O device and main storage. For VSAM and ISAM data sets, the respective access method uses an index to address a specific control interval or block in a data set. For an entry sequenced data set or an OSAM data set, direct addresses are used to address each control interval or block respectively in a data set.

To load a HISAM data base, occurrences of the root segment type must be arranged in ascending key sequence, and all dependents of each root must follow that root in hierarchic sequence. In the key sequenced or ISAM data set, consecutive logical records within a control interval or block are used to store root segment occurrences in ascending key sequence. The first logical record contains the root segment with the lowest key, the next consecutive logical record contains the root segment with the next higher key, and the last logical record contains the root segment with the highest key in that control interval or block. In addition, control intervals or blocks within a data set are loaded in ascending root segment key sequence, this enables a given data base record to be accessed directly through the key of its root.

HISAM Logical Record Lengths

Logical record lengths are a major consideration in a HISAM data base. They affect space and access time.

Extremely short or long logical records tend to increase wasted space. Since only complete segments are stored in a logical record, a gap of space is usually unused at the end of each logical record. The number of gaps increases as the LRECL becomes small, and the size of gaps increases as the logical record length becomes large (if data base records are shorter than the logical record length, the remaining space is lost).

All segments in a logical record are accessed with one read of an I/O device. Accessing additional logical records may require additional reads and seeks depending on physical positioning. The number of seeks and reads to access an entire data base record is in proportion to the number of logical records which comprise that data base record, and therefore increases as the logical record length decreases.

To choose a value for LRECL, several choices should be tried with the following restrictions:

1. The minimum length must be at least equal to the maximum segment length, including prefix, plus 5 for VSAM or 7 for ISAM.

2. For ISAM/OSAM the maximum length cannot exceed the physical block size of the I/O device used. For VSAM, the maximum logical record length is 30720.

3. For ISAM/OSAM, LRECLs must be evenly divisable into physical block size (1/2, 1/4, etc.)

4. VSAM LRECL values must be an even length.

For each LRECL value chosen, the average usable space within a record can be calculated as follows:

$$u = (LRECL - \frac{A}{2} - B)$$

where:

u = usable data characters per logical record.

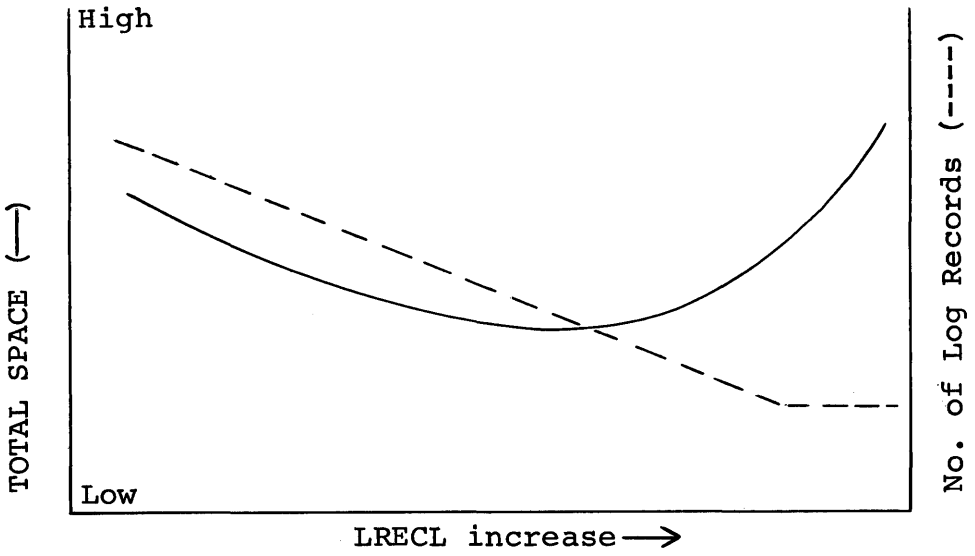
A = weighted average of segment lengths not including the root segment.

B = 7 for ISAM/OSAM, and 5 for VSAM.

The number of logical records required for a particular type data base record is then calculated by dividing the usable logical record length into the total length of the data base record. By breaking the file into a number of typical record types and calculating the space required for each, the total space requirements can be approximated.

As stated before, the number of reads required to obtain an entire data base record is proportional to the number of logical records it requires. By using "typical" records the number of logical records required for the entire file can be calculated. Due to record blocking and the IMS/VS buffer pool management, the actual number of accesses required will be less than the number of logical records. A file requiring fewer (large logical record length) logical records can be accessed faster than the same file with an LRECL value requiring more logical records (small logical record length).

If the number of logical records (relative access time) and total space are plotted against several trial LRECL values, the graph should take the following general form:



As shown, as the value of LRECL gets larger, the number of logical records decreases continually, until the LRECL specification equals the largest data base record length. At this point, the number of logical records equals the number of data base records.

The total space requirements tend to rise if the value of LRECL is either too long or too short. Once several trial LRECL values have been plotted, it should be possible to pick a good one for the file. Consideration should be given to the relative importance of space and access time in the individual situation.

The ISAM and OSAM portions of the data base need not have the same LRECLs. To determine the effect of different values for LRECL, each portion of the data base must be figured separately as above.

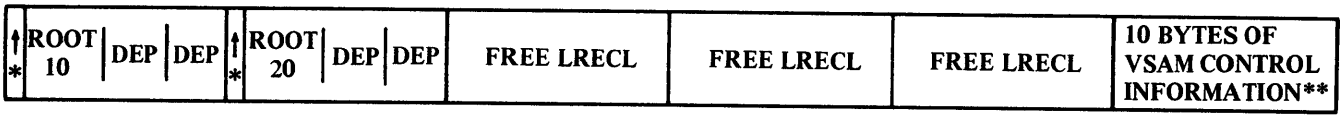
HISAM Root Segment Insertion

To maintain root segment in ascending key sequence when new roots are inserted after initial load of a HISAM data base, one method is used when VSAM is specified as the access method for the data base and another method is used when the combination of ISAM and OSAM are specified as the access methods.

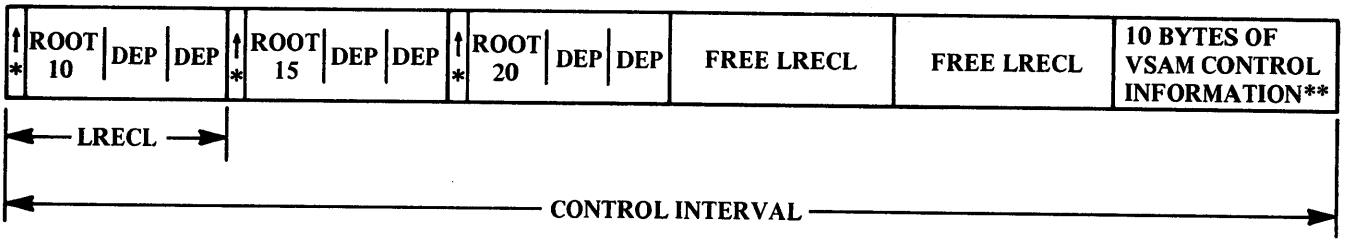
The method shown in Figure 4-14 is used when VSAM is specified. The proper control interval in the key sequenced data set for the new root is obtained, and if the control interval has a free logical record, the new root is stored in ascending key sequence in the control interval by pushing all logical records that contain roots with higher keys to the right one position. If no free logical record exists in the control interval, the control interval is split forming two control intervals that are both equal in size to the original.

Insert root with key of 15

BEFORE



AFTER



* The pointer is comprised of the 4 byte relative byte address of the entry sequenced data set logical record that contains the next dependents in hierarchic sequence.

** For unblocked data sets, the VSAM control information is only 7 bytes.

Figure 4-14. Root Segment Insertion into Key Sequenced Data Set Control Interval

Each new control interval will contain approximately one half of the logical records that were stored in the original control interval which results in free logical records in the last half of each new control interval. After the control interval has been split, the new root is stored in the proper control interval in ascending key sequence by pushing all logical records that contain roots with higher keys to the right one logical record.

To maintain root segments in ascending key sequence when ISAM and OSAM are specified as the access methods for a HISAM data base, the method shown in Figure 4-15 is used. Each new root is stored in an OSAM logical record. To maintain root key sequence, a direct address pointer is placed at the beginning of the ISAM logical record that contains the root segment with the next higher key to point to the OSAM logical record that contains the inserted root as shown in Example 1. Example 2 shows a second root segment being inserted in the OSAM data set. The logical record that contains the root with the next higher key in the ISAM data set points to the OSAM logical record that contains the root with the lowest key. That OSAM logical record in turn points to the OSAM logical record that contains the next higher key.

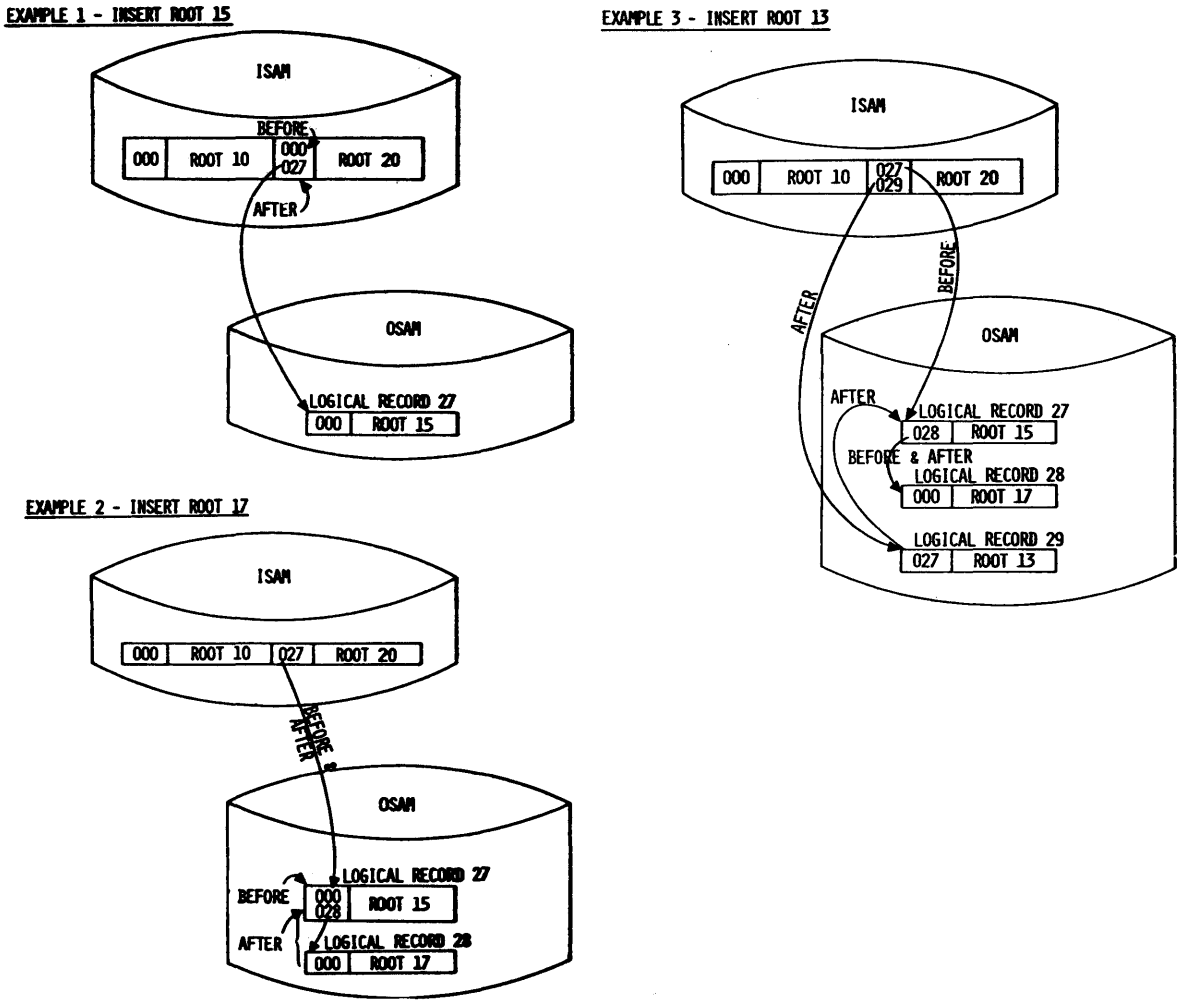


Figure 4-15. Root Segment Insertion When ISAM/OSAM are HISAM Data Base Access Methods

In a HISAM data base, the order of chaining a series of root segments can significantly impact updates. If the addition of root segments is a part of the update, insertions should be made in descending sequence, highest key first when ISAM/OSAM are the OS access methods. This reduces the number of reads necessary to find a point at which to insert a new root. It can be seen in Figure 4-16 that, even with a short chain, the insertion of higher root keys requires a larger number of accesses than the insertion of lower keys. For example, to insert Root 46 it was necessary to read both Roots 34 and 36. The insertion of 32, however, only required the reading of Root 34. Note that the building of long chains of roots occurs only when a large number of updates affects the same area of the data base. The need for descending insertions is less if the inserts are distributed over the data base.

When VSAM is used for a HISAM data base, new roots can be inserted in either ascending or descending sequence. Ascending sequence should provide sightly faster performance.

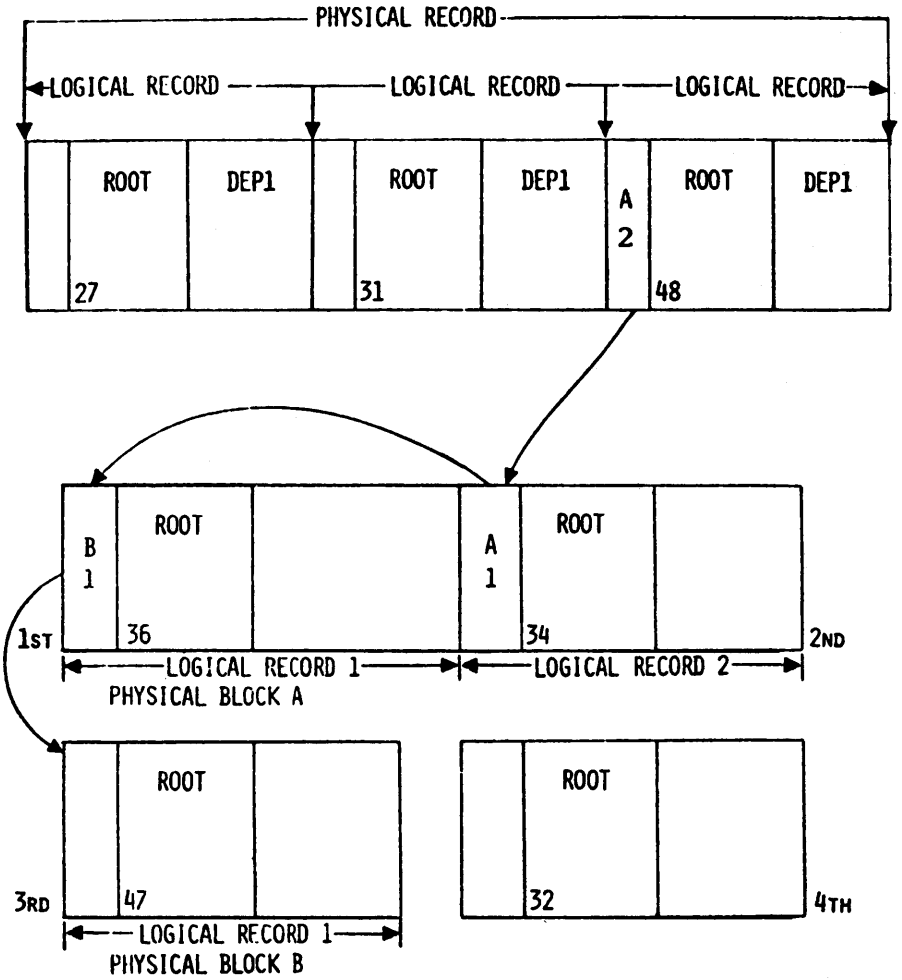


Figure 4-16. HISAM Root Segment Insertion Sequence

HISAM Dependent Segment Insertion

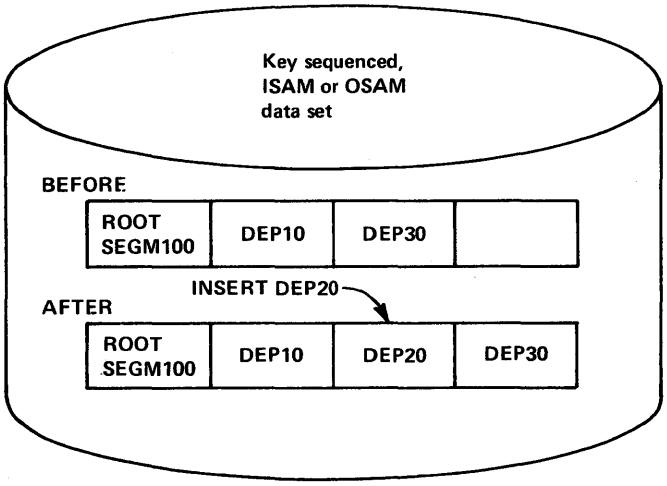
The method used to maintain the hierarchic sequence of segments within a data base record when new dependent segments are inserted into a HISAM data base is essentially the same for both VSAM, and the combination of ISAM and OSAM access methods.

In a HISAM data base, one logical record in the primary storage data set and, if necessary, one or more logical records in the overflow storage data set, are used to store each data base record. Within each logical record and across all logical records that contain segments in one data base record, segments are hierarchically related through their physical sequence in storage. Within each logical record, segments are physically stored in hierarchic sequence, and across logical records, a direct address pointer relates each logical record to the next in hierarchic sequence.

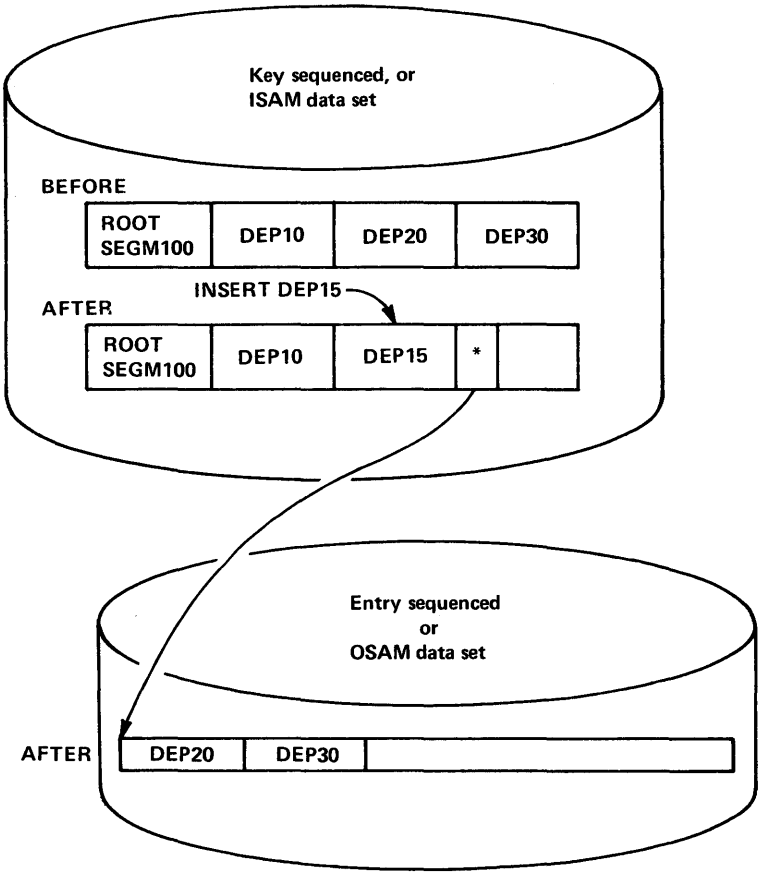
Figure 4-17 shows how the physical sequence of segments within a data base record in storage is maintained when inserting dependents into a data base after intial load. Example 1 shows a dependent segment being inserted into a logical record that contains sufficient space for the new dependent. The new dependent is stored in its proper hierarchic position within the logical record by shifting the segments that will hierarchically follow it to the right within the logical record. Example 2 shows segments displaced to a logical record at the end of the overflow data set when the inserted segment did not leave sufficient space for them at the end of the original logical record. In Example 3, the length of the segment being inserted is greater than the space remaining in the original logical record even after displacing following segments in that logical record, so all are placed in an overflow storage logical record. Example 4 shows an inserted segment that will not fit into the original logical record, and a displaced segment that will not fit into the first overflow logical record with the inserted segment. Two overflow logical records are used, and they are chained together in hierarchic sequence.

In the previous examples, the overflow logical records referred to are at the end of the entry sequenced data set when VSAM is the access method specified, and they are at the end of the CSAM data set when ISAM/OSAM are specified as the access methods. In both cases, logical records at the end of the respective data sets are used for newly inserted or displaced segments from both the primary storage data set and the overflow storage data set.

Example 1- Space available in logical record for dependent being inserted.



Example 2- Space available in logical record for dependent being inserted by displacing existing segments in logical record to an OSAM or entry sequenced data set logical record.



*The pointer is at the beginning of VSAM logical records.

Figure 4-17 (Part 1 of 3). Dependent Segment Insertion into a HISAM Data Base with One Data Set Grcup

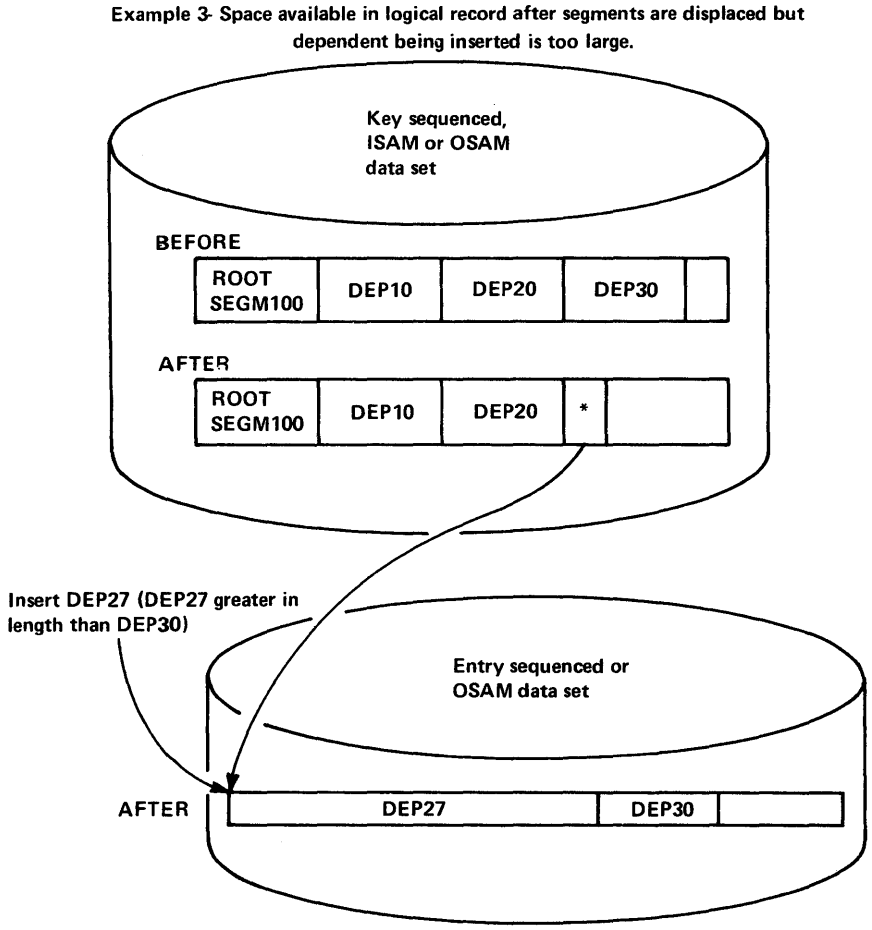
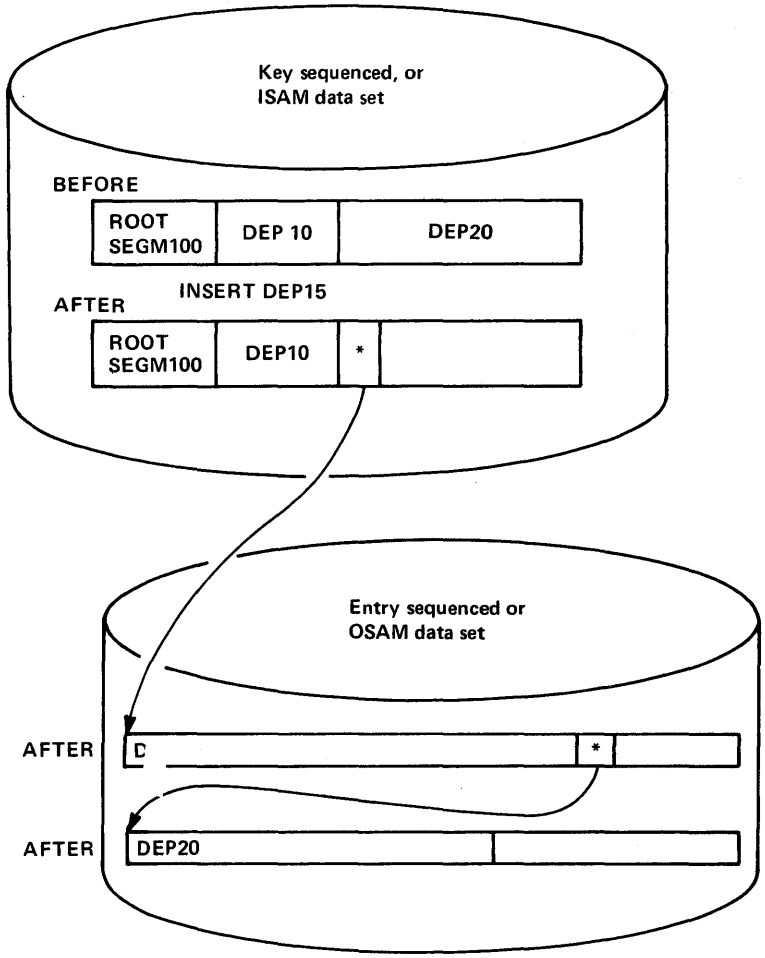


Figure 4-17 (Part 2 of 3). Dependent Segment Insertion into a HISAM Data Base with One Data Set Group

Example 4- Space available in logical record after segments are desplaced to overflow, however, segment being inserted is too large and segment displaced will not fit in 1st overflow.



*The pointer is at the beginning of VSAM logical records

Figure 4-17 (Part 3 of 3). Dependent Segment Insertion into a HISAM Data Base with One Data Set Grcup

HISAM Segment Deletion

ISAM/OSAM: When segments are deleted in a HISAM data base that uses ISAM/OSAM, segments are simply marked as being deleted in the delete byte contained in their prefix. They are not physically deleted from a data base. To regain space occupied by deleted segments, a HISAM data base must be unloaded and reorganized by the user through the HISAM reorganization unload and reload utilities.

VSAM: Segment deletion in a HISAM data base using VSAM is the same as stated for ISAM/OSAM except as follows. When a root segment is deleted from a HISAM data base that uses VSAM, the logical record in the key sequenced data set that contains the root is either erased or the delete byte is marked as when using ISAM/OSAM. An erase is only done when processing the data base in batch mode, the root or any dependent of the root is not involved in an active logical relationship, and there is only one PCB per data base within the PSB.

Secondary Data Set Groups

Secondary data set groups should be considered for HISAM data bases using ISAM/OSAM as the OS/VS access methods in two cases. They should be considered when storage space is wasted because an efficient logical record length cannot be found for the primary data set group due to segment types in the data base whose lengths vary considerably. And, when access to a dependent segment type in the data base is too time consuming through the primary data set group.

As in a primary data set group, each secondary data set group uses two data sets. An ISAM data set is used as the first storage data set and as the index to the first segment type defined in that data set group. Ann OSAM data set that is used as the overflow storage data set. The benefit gained in defining multiple data set groups is that each data set group defined can have different logical record and block lengths. In addition, the occurrences of the first segment type defined in each secondary data set group are indexed through the key of the root segment they follow in a data base record. When defining a secondary data set group, the minimum LRECL must be expanded by the amount necessary to append sequence field keys of the root segment type onto occurrences of the first segment type defined in the secondary data set group.

When only one data set group is defined for a HISAM data base, the segments in each data base record are stored in hierarchic sequence using one logical record in the first storage data set and, if necessary, one or more logical records in the overflow data set. To index each data base record, the key of the root that starts each data base record is used. When multiple data set groups are defined, one logical record in the first storage data set of each data set group and, if necessary, one or more logical records in each overflow data set are used to store the segments of one data base record as shown in Figure 4-18. To index each data base record, the key of the root that starts each data base record is duplicated and is used to index the segments in each secondary data set group that are in the same data base record. In the figure, the secondary data set group contains a duplicate of the key of the root that starts that data base record. The duplicate key is followed by the first occurrence of the description segment type in the data base record, which in turn, is followed by all other segments in that base record in hierarchic sequence.

The use of multiple data set groups to store a HISAM data base has an affect on main storage requirements. Each data set group requires additional space in the DMB pool.

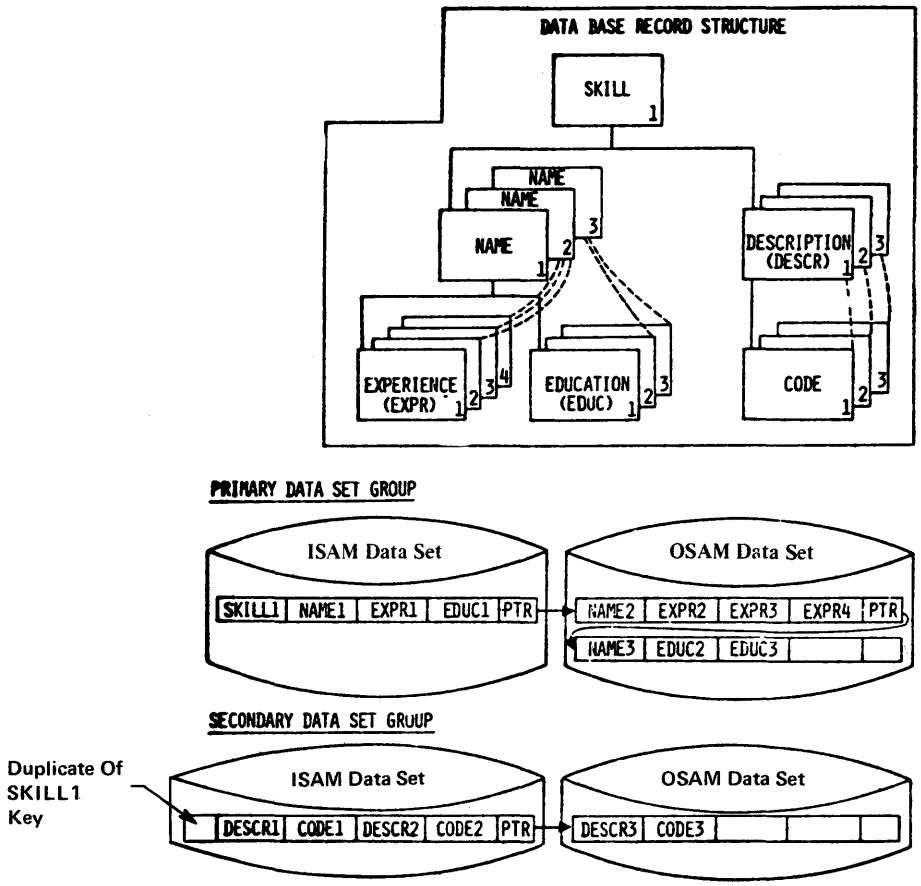


Figure 4-18. One Data Base Record in a HISAM Data Base (Multiple Data Set Group)

Simple HISAM

A simple HISAM data base is a HISAM data base that contains only one segment type. When defining a simple HISAM data base, VSAM must be the access method specified. When defined, occurrences of the segment type are loaded into the data base without prefixes, thus making the data sets that contain the data base compatible for use by VSAM as well as IMS/VS.

HDAM AND HIDAM STORAGE ORGANIZATIONS

Two of the primary advantages gained with HDAM and HIDAM data bases are space reuse and the ability to access segments within the data base through direct address pointers. Either data base type is stored in one or more VSAM entry sequenced or OSAM data sets depending on the number of data set groups defined. Space within each data set is managed through free space elements and bit maps. When segments are inserted or deleted from either data base type, the space used or freed by those segments is recorded in a bit map to enable its reuse when inserting new segments. To hierarchically relate segments in HDAM and

HIDAM data bases, direct address pointers are used. In either data base type, hierarchic, physical child/physical twin or any combination of the two types of pointers can be specified.

The storage organization methods used for HDAM and HIDAM data bases are essentially the same. The primary difference between HDAM and HIDAM data bases is that access to occurences of the root segment type is through a user randomizing module for an HDAM data base, and through an index for a HIDAM data base. To access a given root in an HDAM data base, the randomizing module examines the key of the root, and through hashing or some other arithmetic technique, computes the address of the root and passes it to IMS/VS. To access the same root in a HIDAM data base, an index must be searched by IMS/VS to find the address of the root. When found, the root is accessed. By using a randomizing module to locate roots, the I/O operations required to search the index are eliminated.

HDAM

To use an HDAM data base, the user must supply a randomizing module. The randomizing module determines where each root should be stored in the data base, and supplies the address of each root stored to IMS/VS each time that root must be accessed. Addresses supplied by a randomizing module consists of a relative block number and an anchor point number. Anchor points are stored in the anchor point area of each control interval or block, and each is a four byte direct address pointer to a root. To access a given root, the relative block number locates a specific control interval or block in relation to the start of the data set, and the anchor point number locates a specific anchor point in the anchor point area of that control interval or block.

Figure 4-19 shows the organization of an HDAM data base in storage. The entry sequenced or OSAM data set in the primary data set group is divided into two areas called the root addressable area and the overflow area. The root addressable area is the first n control intervals or blocks in the data set, and the overflow area is the remaining portion of the data set.

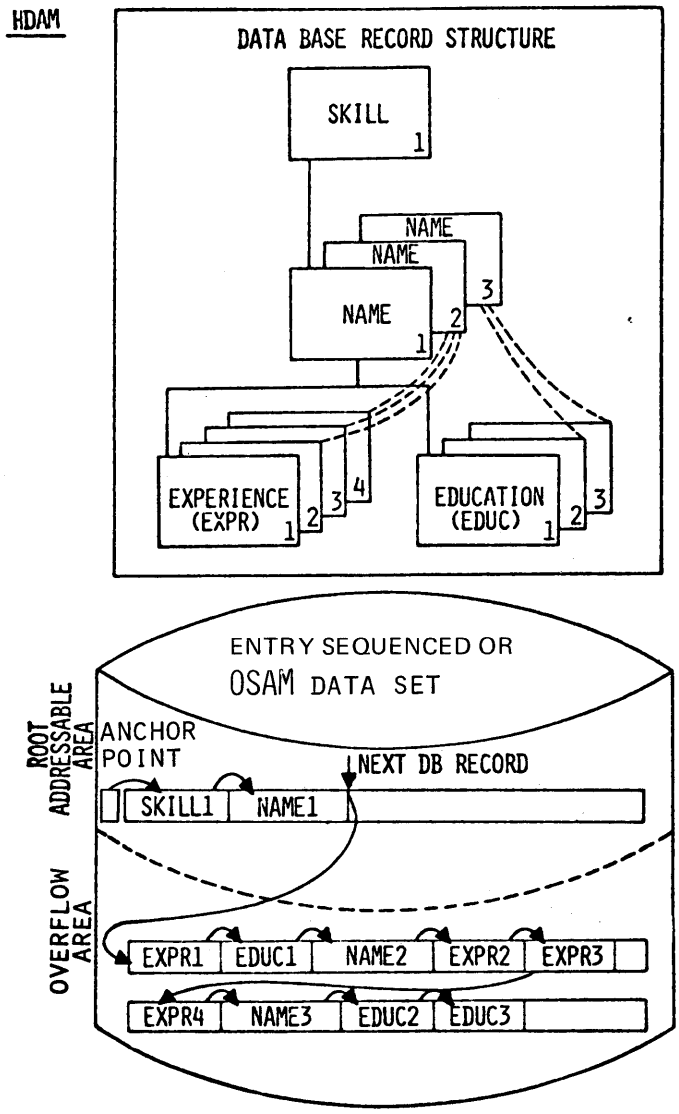


Figure 4-19. HDAM Data Base Record in Auxiliary Storage

The root addressable area is the area in which the user randomizing module assigns roots. The length of the root addressable area is specified by the user through the DBDGEN utility. Also specified is the number of anchor points to be placed in the anchor point area of each control interval or block in the root addressable area. A third parameter specified is the maximum number of bytes of a data base record to be stored in the root addressable area. The root addressable area is used as the primary storage area for segments in each data base record, and the overflow area is used for overflow storage. Since data base records vary in length, the bytes parameter is used to control the amount of space used for each data base record in the root addressable area. The bytes parameter limits the number of segments of a data base record that can be consecutively inserted into the root addressable area. When consecutively inserting a root and its dependents, each segment is stored in the root addressable area until the next segment to be stored will cause the total space used to exceed the bytes parameter. The total space used for a segment is the combined lengths of the prefix and data portions of the segment. When exceeded, that segment and all remaining segments in the data base record are

stored in the overflcw area. It should be noted that the bytes parameter only controls segments consecutively inserted in one data base record. Consecutive inserts are inserts to cne data base record that are not intervened by any call to process a segment in a different data base record.

Size of Root Addressable Area

The general criteria to determine the size of the root addressable area is:

$$\frac{\text{Number of bytes of a data base record to be stored in the root addressable area} \times \text{the expected number of data base records}}{\text{(Number cf bytes per block)}} = \text{required size of the root addressable area in blocks}$$

In addition, if distributed free space is specified, the space estimate obtained must be multiplied by one factcr fcr free blocks and another for free space within each block as shown in the following formula:

$$\text{(Total Space)} = \text{(Minimum Space)} \times \frac{\text{fbff}}{\text{fbff}-1} \times \frac{1}{1-\frac{\text{fspf}}{100}}$$

where:

$$2 \leq \text{fbff} \leq 100 \text{ or } \text{fbff} = 0 \text{ and } 0 \leq \text{fspf} < 100$$

See "Distributed Free Space" in this chapter fcr definitions of fbff and fspf.

At least root segments should be stored in the root addressable area. In addition, active dependent segments should be placed in the root addressable area since this will provide fast access to them because of their physical proximity to the root segment. When all space in the root addressable area is occupied, all segments inserted (roots included) are placed in the overflow area.

The size of the root segment addressable area is fixed with DBD generation. The overflow area however, can be dynamically extended if the overflow storage data set allocation is specified as secondary allocation.

Loading an HDAM Data Base

To load each data base record into an HDAM data base, the user randomizing module generates a relative block and anchor point number for the root that starts that data base record and passes them to IMS/VS. IMS/VS in turn, attempts to store the root in the control interval or block specified. If space is available in that ccntrol interval or block, the rcot is stored and a four byte direct address pointer to the root is stored in the specified anchcr point position in the anchor point area of that control interval or block. When space is not available in the control interval or block specified, IMS/VS uses the HD space search algorithm to find the available space nearest to the control interval or block specified by the randomizing module. When found, the root is stored and a pointer to that root is stored in

the original anchor point position and relative block number specified by the randomizing module.

When a randomizing module produces the same relative block and anchor point number for multiple roots, the specified anchor point points to one, and the rest are chained through physical twin pointers. When a unique sequence field has been defined in the root segment type, the anchor point points to the root with the lowest key and the rest are chained in ascending key sequence through physical twin pointers. When a unique sequence field is not defined, the insert rules of FIRST, LAST or HERE determine the sequence in which the roots are chained. All roots chained from a given anchor point are called synonyms since all have the same relative block and anchor point number. To reduce the length of root segment synonym chains if they affect performance, the user should increase the number of root anchor points specified for each control interval or block in the root addressable area. The user randomizing routine can then distribute the roots across more anchor points, thereby reducing the number of synonyms per anchor point.

After a root is loaded into the root addressable area, the next segments in a data base record are stored following the root until the bytes parameter causes the remaining segments in a data base record, if any, to be stored in the overflow area.

HIDAM

A HIDAM data base in auxiliary storage is actually comprised of two data bases that are normally referred to collectively as a HIDAM data base. When defining each through the DBDGEN utility, one is defined as the primary HIDAM index data base and the other is defined as the HIDAM data base. In the following discussion the terms 'HIDAM data base' will refer to the HIDAM data base defined through DBDGEN.

The primary HIDAM index data base is used to index to the data base records stored in a HIDAM data base. When a HIDAM data base is defined through DBDGEN, a unique sequence field must be defined in the root segment type. The resulting key in the sequence field of each occurrence of the root is used by IMS/VS to create an index segment for each root that is stored in the index data base. To identify which root an index segment indexes, the key in the sequence field of a root is stored in the data portion of an index segment. To index to that root, the address of the root in the HIDAM data base is stored as a direct address pointer in the prefix of its index segment.

Loading a HIDAM Data Base

When the user loads a HIDAM data base, the primary HIDAM index data base is loaded automatically by IMS/VS. In loading a HIDAM data base, all roots must be inserted in ascending key sequence, and all dependents of a root must be inserted following that root in hierarchic sequence. As each root is stored in the HIDAM data base, IMS/VS generates the index segment for that root and stores it in the index data base.

The index data base consists of an ISAM and an OSAM data set when ISAM/OSAM are specified as the access methods for the data base, or it consists of a key sequenced data set when VSAM is specified as the access method as shown in Figure 4-20. When ISAM/OSAM are specified for the index data base, the ISAM data set is used for storage of index segments created during initial load of a HIDAM data base, and it is called the primary data set. The OSAM data set is used for storage of index segments created when new roots are added to a HIDAM data base after initial load, and it is called the overflow data set. When VSAM is specified for the index data base, the key sequenced data set is

used for both index segments created during initial load and after initial load.

When ISAM/OSAM are used for an index data base, all index segments for roots initially loaded are stored in ascending key sequence in the ISAM data set. When roots are added after initial load, the index segment for that root is stored in the first available logical record in the OSAM data set. When this occurs, a pointer is stored at the beginning of the logical record in the ISAM data set that contains the next higher key. The pointer points to the logical record in the OSAM data set that contains the added index segment. When multiple index segments have to be chained from the same logical record in the ISAM data set, the logical record in the ISAM data set points to the OSAM logical record that contains the index segment with the lowest key. Any additional OSAM logical records to be chained are chained from the first OSAM logical record in ascending key sequence. Since index segments added after initial load are stored in the OSAM data set, their access requires additional I/O operations. To improve performance degraded by root inserts, the index data base should be reorganized through the HISAM Reorganization Unload and Reload utilities.

A HIDAM data base is stored in from one to ten entry sequenced or OSAM data sets depending on the number of data set groups defined and the access method specified. Each data set group uses one entry sequenced data set when VSAM is specified as the access method, and one OSAM data set when OSAM is the access method specified. When initially loading segments into a HIDAM data base or when inserting segments into a HIDAM data base after initial load, the HD space search algorithm is used to find the most suitable space available.

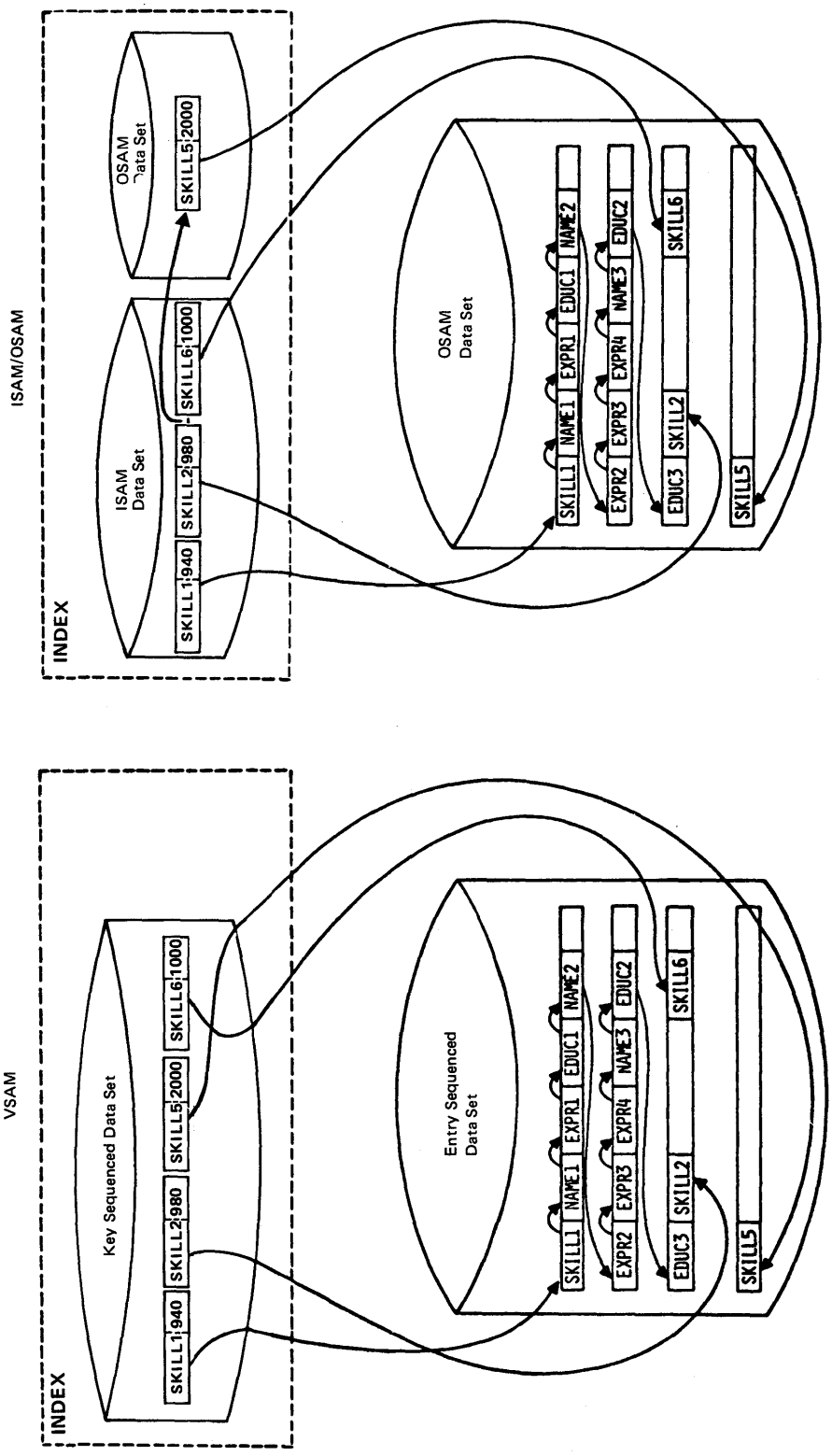


Figure 4-20. Insert of a Root Segment intc a HIDAM Data Base after Initial Lcad

HIDAM Data Base Root Segment Type Pointer Options

If forward only hierarchic or physical twin pointers are specified for the root segment type of a HIDAM data base, sequential access to each root segment is accomplished by using the index. When forward and backward hierarchic or physical twin pointers are specified for the root segment type, for sequential processing the index is only used to access the first root segment. From the first root, additional roots can be processed sequentially without further reference to the index.

Format of Data Sets Used for HDAM and HIDAM

In defining an HDAM or HIDAM data base, the user can specify VSAM or OSAM as the access method to be used for the data base, and he can also specify that the data base be stored as one to ten data set groups. When VSAM is specified, each data set group consists of one entry sequenced data set. When OSAM is specified, each data set group consists of one OSAM data set. In either case, the resulting data set will have an unblocked format. When not specified by the user, DBDGEN generates logical record lengths for the data sets that are equivalent to a quarter-track, third-track, half-track, or full-track block.

Direct address pointers within the prefix of a segment and the anchor point(s) of a block are constructed by the following formula:

Pointer Value = (Block Size) X (Block Number) + (Offset within Block).

This formula is also used for pointers in the prefix of segments of an INDEX data base when relating to segments in a HIDAM data base.

In order that all segments will be on half word alignment, within the data set a slack byte is added to the end of any segment in HDAM data bases or HIDAM whose total length is an odd number.

The control fields used in managing entry sequenced or OSAM data sets for HDAM and HIDAM data bases are (See Figure 4-21):

- Free space anchor point
- Free space element
- Anchor point area
- Bit map control interval or blocks

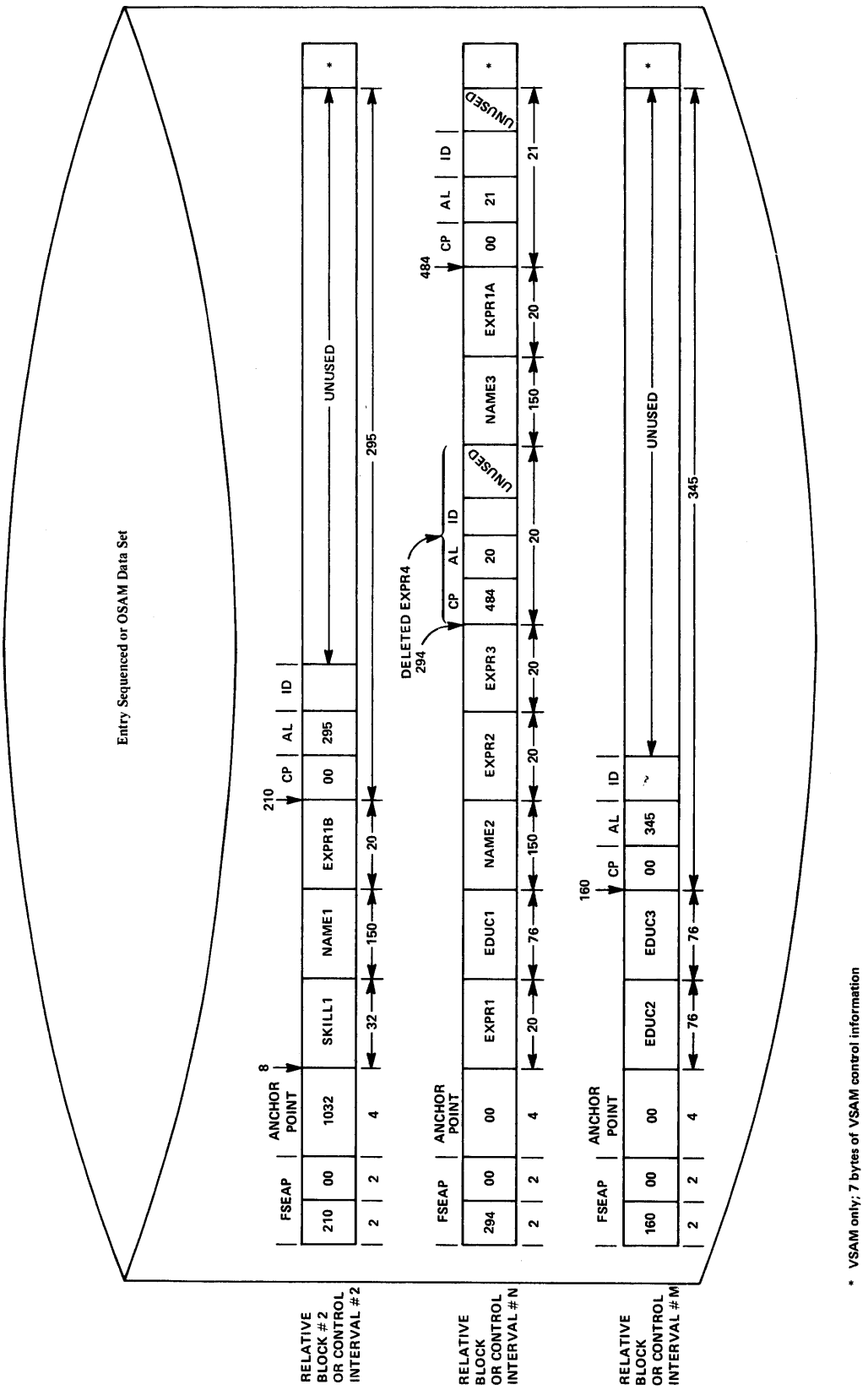


Figure 4-21. Control Fields Used to Manage Entry Sequenced or OSAM Data Sets Used for HDAM or HIDAM Data Bases

Free Space Anchor Point

Each control interval or block in an entry sequenced or OSAM data set respectively starts with a four byte free space anchor point (FSEAP) field. The field contains, in the first two bytes, the offset in bytes to the first free space element in that control interval or block. The second two bytes contain a flag that identifies bit map blocks. For blocks other than bit map blocks, the second two bytes of the field contain zeros.

Free Space Element

A free space element identifies each area of free space in a control interval or block that is eight bytes or more in length. To identify each area of free space, a free space element occupies the first 8 bytes of each area of free space. The fields in each free space element are:

- Free space chain pointer field (CP) -- This field contains, in bytes, the offset to the next free space element in the control interval or block from the beginning of the control interval or block.
- Available length field (AL) -- This field contains in bytes the length of the vacant space that this free space element identifies. The value in the available length field includes the length of the free space element itself.
- Task ID field (ID) -- This field contains the task ID of the program that freed the space that is identified by this free space element. The task ID enables a given program to free and reuse the same space during a given scheduling without contending for that space with other programs.

The task ID consists of a one-byte calendar date followed by a three byte cumulative sync point count for the day.

Anchor Point Area

For an HDAM data base, the user specifies the number of four byte direct address root anchor points desired in each control interval or block in the root addressable area. For each anchor point specified, four bytes of space is reserved in the anchor point area of each control interval or block in the root addressable area. The space for anchor points is not reserved in those control intervals outside the root addressable area, including the bit map control intervals. This space is initially made free space and is available just as other free space in a control interval.

For a HIDAM data base, when forward-only hierarchical or physical twin pointers are specified for the root segment type, one 4-byte anchor point is present in each control interval or block. The anchor point addresses the last root inserted into the control interval and the roots are chained in the reverse order from the sequence in which they were inserted into the control interval. With a forward-only (not forward and backward) pointer at the root level, it is impossible for IMS/VS to keep the roots chained in key sequence when new roots are inserted into an existing data base. Because the forward pointer chains roots in reverse time sequence and not in key sequence, it is not used by IMS/VS to obtain the next sequential root. The index is used to do sequential processing. For a HIDAM data base we recommend that you use no-twin, twin forward and backward, or hierarchical forward and backward pointers at the root level. When one of these options is

used, no anchor point is placed in the control interval. If your processing is primarily random, no-twin is best because all accesses to the root segments are via the index. If your processing is primarily sequential, use physical or hierarchical forward and backward. With these pointers the roots are chained in key sequence.

Bit Map Block

A bit map control interval or block starts with a two byte free space chain pointer field. The field always contains zeros in a bit map control interval or block in the root addressable area of an HDAM data base since no space is available in those bit map control intervals or blocks for segments. The next two bytes contain a bit map flag. A low order one in the second two bytes of a control interval or block indicates that the control interval or block contains a bit map. The same two bytes in all other control intervals or blocks in a data set will contain zeros. The next 0 to n bytes of a bit map control interval or block are for root anchor points. Following the anchor point area when present, the remainder of the control interval or block is a bit map.

Bit Map

The first control interval or block of the first extent of the data set specified for each data set group in an HDAM or HIDAM data base is used for a bit map. If, however, the organization is VSAM, the second block is used for the bit map and the first block is reserved. In the bit map, each bit is used to describe whether or not space is available in a particular control interval or block. The first bit, corresponds to the block the bit map itself is in, and each consecutive bit corresponds to the next consecutive block in the data set. When the bit value is one, it indicates that the associated block has sufficient space remaining to store an occurrence of the longest segment type defined in this data set group. When the bit value is zero, sufficient space is not available for an occurrence of the longest segment type defined in this data set group. The first bit map in a data set contains n bits that describe space availability in the next n consecutive control intervals or blocks in the data set. After the first bit map, another bit map is stored at every nth control interval or block to describe space availability in the remaining control intervals or blocks in the data set.

Inserts and Deletes in HDAM and HIDAM Data Bases

The techniques used to insert or delete segments are the same for both HDAM and HIDAM data bases. The techniques involve use of bit maps, space available chains and available length fields. The three are used to find space when inserting a segment, or to record free space when a segment is deleted.

Inserts

The following HD space search algorithm is used to find the most suitable space for a segment being inserted into an HDAM or HIDAM data base.

HD Space Search Algorithm

SEARCH CRITERIA: When searching for space, if space the exact size of the request is found, it is used; otherwise, three free areas are selected in the following order of preference:

1. Smallest ≥ REQUEST + min. segment in data set
2. Smallest ≥ REQUEST *2
3. Smallest ≥ REQUEST From this set, the first area that does not alter bit map setting is selected, if there is one. Otherwise, the first area found is selected.

1. SAME BLOCK
2. ANY BLOCK CURRENTLY IN BUFFER POOL ON SAME TRACK
3. ANY BLOCK CURRENTLY IN BUFFER POOL ON SAME CYLINDER
4. ANY BLOCK ON SAME TRACK WHERE SPACE FOR MAXIMUM SEGMENT LENGTH EXISTS (Based on bit map)
5. ANY BLOCK ON SAME CYLINDER WHERE SPACE FOR MAXIMUM SEGMENT LENGTH EXISTS (Based on bit map)
6. ANY BLOCK CURRENTLY IN BUFFER POOL WITHIN ± N CYLINDERS
7. ANY BLOCK ON ± N CYLINDERS WHERE SPACE FOR MAXIMUM SEGMENT LENGTH EXISTS (Based on bit map)
8. ANY BLOCK IN BUFFER POOL AT END OF DATA SET
9. ANY BLOCK AT END OF DATA SET (Based on bit map)
10. ANY BLOCK IN THE DATASET WHERE SPACE FOR MAXIMUM SEGMENT LENGTH EXISTS (Based on bit map)

In the algorithm, the same block as that which contains the immediately preceding segment in the hierarchy of a data base record is chosen for the new segment insertion under search criteria one. If not satisfied, search criteria two through ten are used in sequence in attempting to obtain space for insertion.

Deletes

Deletion of a segment from an HDAM or HIDAM data base involves:

- Updating the pointers in any other segments that point to the deleted segment.
- Accumulating the space occupied by the deleted segment on the space available chain for the block and possible adjustment to the bit map. If a deleted segment is adjacent to an already available area of space, the two areas are combined into one.

Figure 4-22 illustrates the deletion of segment EXPR4 and the incorporation of the space it occupied on the space available chain.

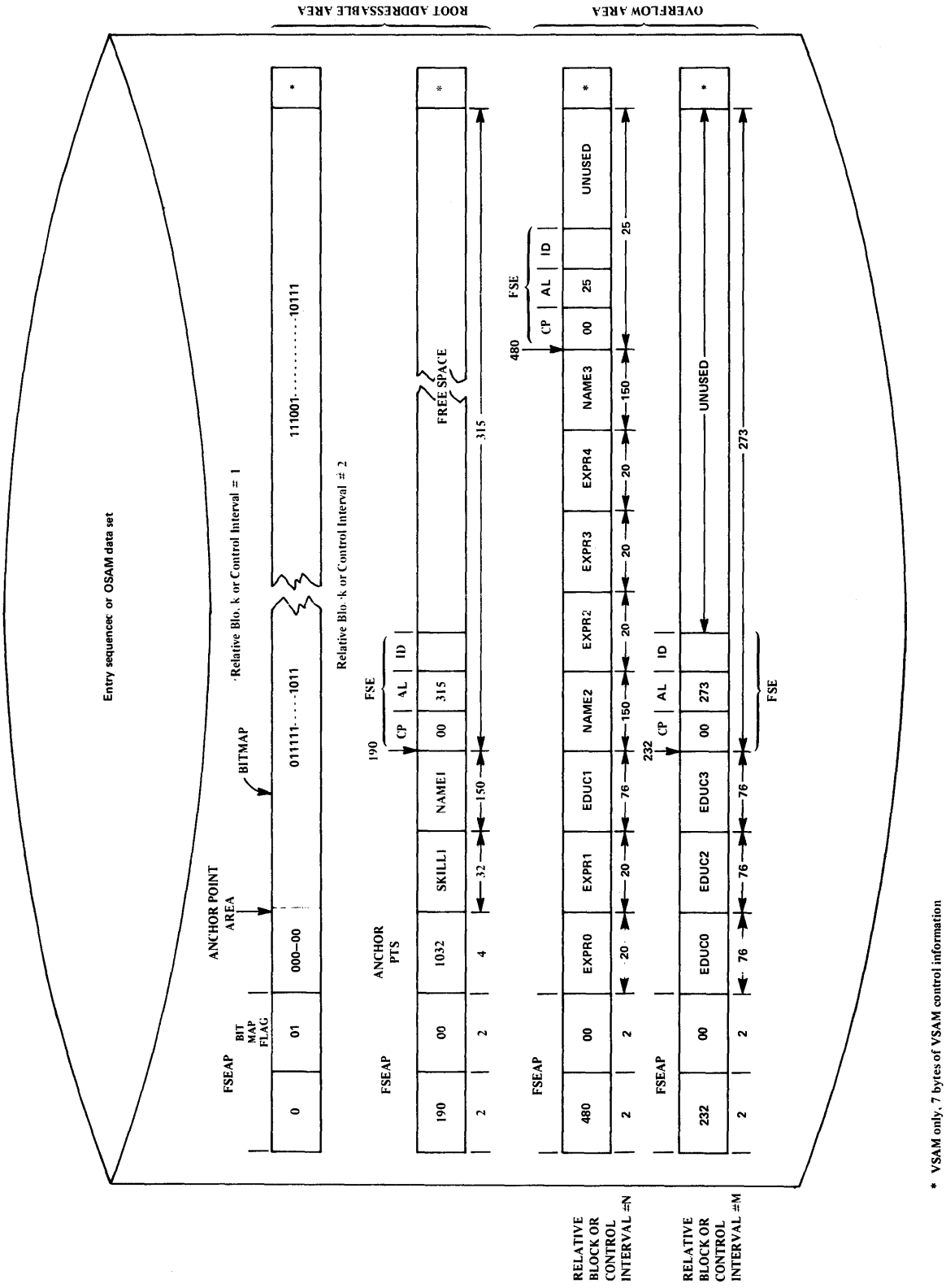


Figure 4-22. Hierarchic Direct Deletion of Dependent Segment

Distributed Free Space

A consideration affecting HDAM or HIDAM data base performance is a result of certain types of dependent segment insert activity. After an HDAM or HIDAM data base is initially loaded or reorganized, high segment insert activity may degrade performance. This degradation occurs when added segments are not placed physically adjacent to their related segments. For HDAM or HIDAM, segments inserted after a data base is initially loaded or reorganized are stored at the end of their data set group, or in the position of previously loaded segments that have been deleted from that data set group as follows:

Space for an inserted segment in an HDAM or HIDAM data base is acquired by scanning a user specified number of disk cylinders to locate the free space nearest to its related segments. If no space is found, the segment is inserted at the end of that data set group. This method attempts to place added segments in the position physically closest to their related segments to keep direct access storage device access time to a minimum. However, since this method does not always place added segments in space physically adjacent to their related segments, data bases must be reorganized periodically to eliminate the degradation to performance.

The distributed free space option can be selected during DBDGEN for HDAM and HIDAM data bases. It is intended to minimize degradation to performance caused by insert activity, and in so doing, decrease the frequency in which HDAM or HIDAM data bases must be reorganized. It accomplishes this by allowing the user to specify, on the DATASET statement for each data set group, periodic blocks of free space and/or a percentage of free space in each block to be left vacant during initial load or reorganization of the data base. These free spaces are then used after data base initial load or reorganization to store added segments physically close to their related segments.

The FRSPC= operand on the DATASET statement is used to specify free space in each data set group of an HDAM or HIDAM or data base. In the operand, any combination of two parameters can be specified. One is the fbff (free block frequency factor). The fbff specifies that every nth block in a data set group will be left as free space during data base load or reorganization (where fbff=n). The range of fbff includes all integer values from 0 to 100 excluding fbff=1. The second parameter that can be specified is the fspf (free space percentage factor). The fspf specifies the minimum percentage of each block in a data set group that is to be left as free space during load or reorganization. The range of fspf is from 0 to 99.

HISAM AND HIDAM KEY SEGMENTS

For HISAM or HIDAM index data bases using ISAM/OSAM, IMS/VS generates an additional root segment and stores it as the last root segment in the data base. This additional root segment has the sequence field filled with X'FF's. It is generated and placed in the data base by IMS/VS because added root segments are chained from the root with the next higher sequence field key.

HIDAM data bases using VSAM also contain an X'FF' key segment. It is used for twin backward pointing at the root level.

For variable length or compressed segments, an X'FF' key segment is allocated the maximum length specified for the segment type, and the size field of the segment has the high order bit turned on (X '8XXX'). This segment is never compressed.

OPTIONS AVAILABLE IN DEFINING PHYSICAL DATA BASES

Following is a summary of the characteristics of the four physical data base types.

HSAM

- All segments and data base records are related through physical adjacency.
- Stored as a sequential data set.
- Can only retrieve segments from existing data base. To update requires reload.
- Can be stored on tape.

HISAM

- Segments within data base records are related through physical adjacency.
- Indexed access to data base records.
- User can specify multiple data set groups.
- Space occupied by deleted segments is not reusable, except when root segments are deleted in a key sequenced data set.
- VSAM or the combination of ISAM/OSAM can be specified as the operating system access method.
- Logical relationships using symbolic pointers.
- When VSAM is specified as the operating system access method for a HISAM data base, the additional options available are:
  - Secondary Indexing using symbolic pointers
  - Variable Length Segments
  - User Segment Compaction/Expansion Routines

HDAM OR HIDAM

- Segments within data base records are related through hierarchic and/or physical child/physical twin direct address pointers.
- Access to the root in each data base record is through a user randomizing module for HDAM and through an index for HIDAM.
- User can specify multiple data set groups.
- Space occupied by deleted segments is reusable.
- VSAM or OSAM (combination of ISAM/OSAM for HIDAM index) can be specified as the operating system access method.
- Logical relationships using direct address or symbolic pointers.
- Distributed Free Space.

- When VSAM is specified as the operating system access method for the data base, the additional options available are:

  - Secondary Indexing using direct address or symbolic pointers

  - Variable Length Segments

  - User Segment Compaction/Expansion Routines

LOGICAL RELATIONSHIPS

In multi-application data management systems, data duplication is often a problem. Duplicates in storage waste storage space and cause duplicate maintenance. Duplicates are caused when a given type of data is common to several applications, but each application requires the common data stored in relation to different types of data, or in combination with different types of data. To eliminate storage duplication, logical relationships are used. Logical relationships enable the user to store a given segment type once and to define that segment type as dependent on one segment type in a physical data base and a different segment type in a logical data base. Logical relationships are also used to create logical data bases that contain a combination of segment types from different physical data bases without duplicating them. This means the segment types in two different physical data bases, for two different applications, can be combined into a logical data base for a third application without creating a third physical data base.

All logical relationships establish a relationship between two segment types in one or more physical data bases. They are defined in the physical data bases of the segment types they relate to, and they are used when the related segment types are processed through a logical data base. When defined between segment types in the same physical data base, a logical relationship enables a different hierarchy of segment types to be defined for the segment types in that physical data base. When defined between segment types in different physical data bases, it enables a hierarchy of segment types to be defined that combines the segment types in both data bases into a single data base. In each case, the new hierarchy of segment types is defined through the DBDGEN utility to create a logical data base. The hierarchy of segment types for a logical data base is comprised of a subset of the physical and logical relationships defined between the segment types in their physical data bases.

Logical relationships enable occurrences of two segment types to be stored independently of each other, yet enable an application program to process them through a logical data base as if stored in relation to each other. Through the logical data base, the relationship between the two segment types appears to be that of a physical parent segment type and one of its physical child segment types in a physical data base. An application processes occurrences of the related segment types through their logical data base in the same manner as occurrences of a physical parent segment type and a physical child segment type are processed in a physical data base.

A logical relationship is defined in the physical data base or data bases of the segment types being related. Through a logical relationship, segment types in the same or different physical data bases are related in a manner that is in most cases transparent to application programs using the physical data bases. To enable use of a logical relationship defined between two segment types in one or more physical data bases, a logical data base must be defined.

The terms used to describe the segment types involved in logical relationships are physical parent, logical parent, and logical child. The terms physical parent and logical parent are used to describe the two segment types being related through a logical relationship. The term logical child is used to describe one or both of the additional segment types that are required to relate two segment types through a logical relationship.

METHODS OF RELATING SEGMENT TYPES THROUGH A LOGICAL CHILD

Three types of logical relationships can be defined in IMS/VS data bases. The three types are unidirectional, physically paired bidirectional, and virtually paired bidirectional logical relationships. For each of the three types of logical relationships, a logical child segment type relates two segment types by one of two methods. The first method described in the following text is used for unidirectional and physically paired bidirectional logical relationships. The second method described is used for virtually paired bidirectional logical relationships. In both methods, a logical child is physically related to one of the segment types being related through a logical relationship. In addition for the first method, the logical child points to the other segment type. In the second method the logical child points to the other segment type, and is pointed to by the other segment type. Figure 4-23 shows the first method of relating segment types through a logical child segment type.

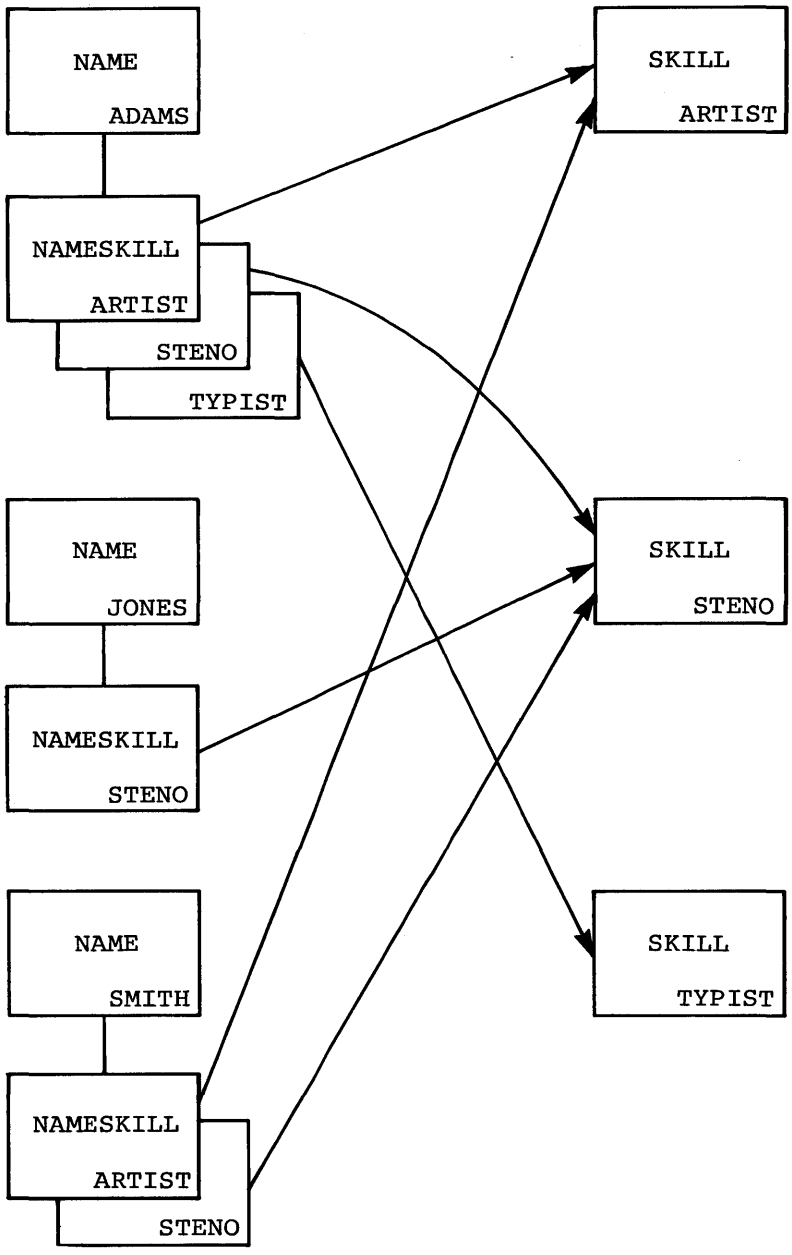


Figure 4-23. Relating Occurrences of SKIII to Cccurrences of NAME

Method One

Figure 4-23 shows occurrences of the SKILL segment type being related to occurrences of the NAME segment type through occurrences of an additional segment type that is required to relate NAME and SKILL segments. A logical child is an additional segment type that is required to relate two segment types through a logical relationship. A logical child segment type relates two segment types by being physically related to one segment type and by pointing to the other segment type. The segment type that the logical child segment type is physically related to is called the physical parent of the logical child. The segment type that the logical child segment type points to is called the logical parent of the logical child. The pointer in a logical child that points to a logical parent is called a logical parent pointer. In Figure 4-23, NAME is the physical parent, SKILL is the logical parent, and NAMESKILL is the logical child segment type. To establish the physical relationship between the NAME and NAMESKILL segment types shown in Figure 4-23, NAMESKILL is defined as a physical child segment type of NAME in the physical data base of the NAME segment type. Since NAME and NAMESKILL are a physical parent and a physical child segment type in the same physical data base, occurrences of each are related when loaded into their physical data base. To relate an occurrence of SKILL to an occurrence of NAME in storage, the user loads an occurrence of NAMESKILL, the logical child segment type, as a physical child of a given NAME segment. This process is repeated for each occurrence of the logical parent that is to be related to that NAME segment. When loading a logical child segment into its physical data base, the user identifies which logical parent segment the logical child points to, by placing the concatenated key of the logical parent in the data portion of the logical child segment. Since the concatenated key of a logical parent segment is the symbolic pointer to that segment in its physical data base, when the user loads logical child segments as physical children of a given physical parent segment, the respective logical parent segment pointed to by each logical child is related to the physical parent segment. When processing the related segment types through a logical data base, it is the physical relationship between occurrences of the physical parent and logical child segment types in their common physical data base, plus the logical parent pointer contained in each logical child segment, that enables access to related occurrences of the logical parent segment type from each occurrence of the physical parent segment type.

Method Two

In the second method of relating two segment types through a logical child segment type, all of the conditions described for the first method remain the same. The logical child segment type is physically related to its physical parent segment type and points to its logical parent segment type. One occurrence of the logical child segment type is loaded as a physical child of a given physical parent segment for each occurrence of the logical parent segment type that is to be related to that physical parent. To identify which logical parent segment is being related to a physical parent segment through a logical child segment, the user places the concatenated key of the logical parent in the data portion of each logical child segment loaded. Through the relationship of physical parent and logical child segments in their physical data base, and the logical parent pointer in each logical child segment, related occurrences of the logical parent segment type can be accessed from physical parent segments. In addition, logical child pointers are used in the logical parent segment type, and logical twin and physical parent pointers are used in the logical child segment type, as shown in Figure 4-24. The additional pointers are used to enable accessing specific occurrences of the physical parent segment type that are related to each occurrence of the logical parent. Logical

twins are multiple occurrences of the same logical child segment type that point to the same occurrence of the logical parent segment type. When specified in the logical child segment type, logical twin pointers point from each logical twin to the next to chain together all logical twins that point to a given logical parent. The physical parent pointer in each occurrence of the logical child segment type is generated automatically by IMS/VS to enable access to the physical parent segment of each logical child from that logical child. A logical child pointer is specified for the logical parent segment type to enable accessing a logical child segment from a logical parent segment. A logical child pointer points from a given logical parent segment to one of the logical twins, which also points back to that logical parent segment. Since all logical twins that point to the same logical parent are chained through logical twin pointers, and each logical child contains a physical parent pointer, the specific physical parent segments that are related to a given logical parent segment can be accessed from that logical parent segment.

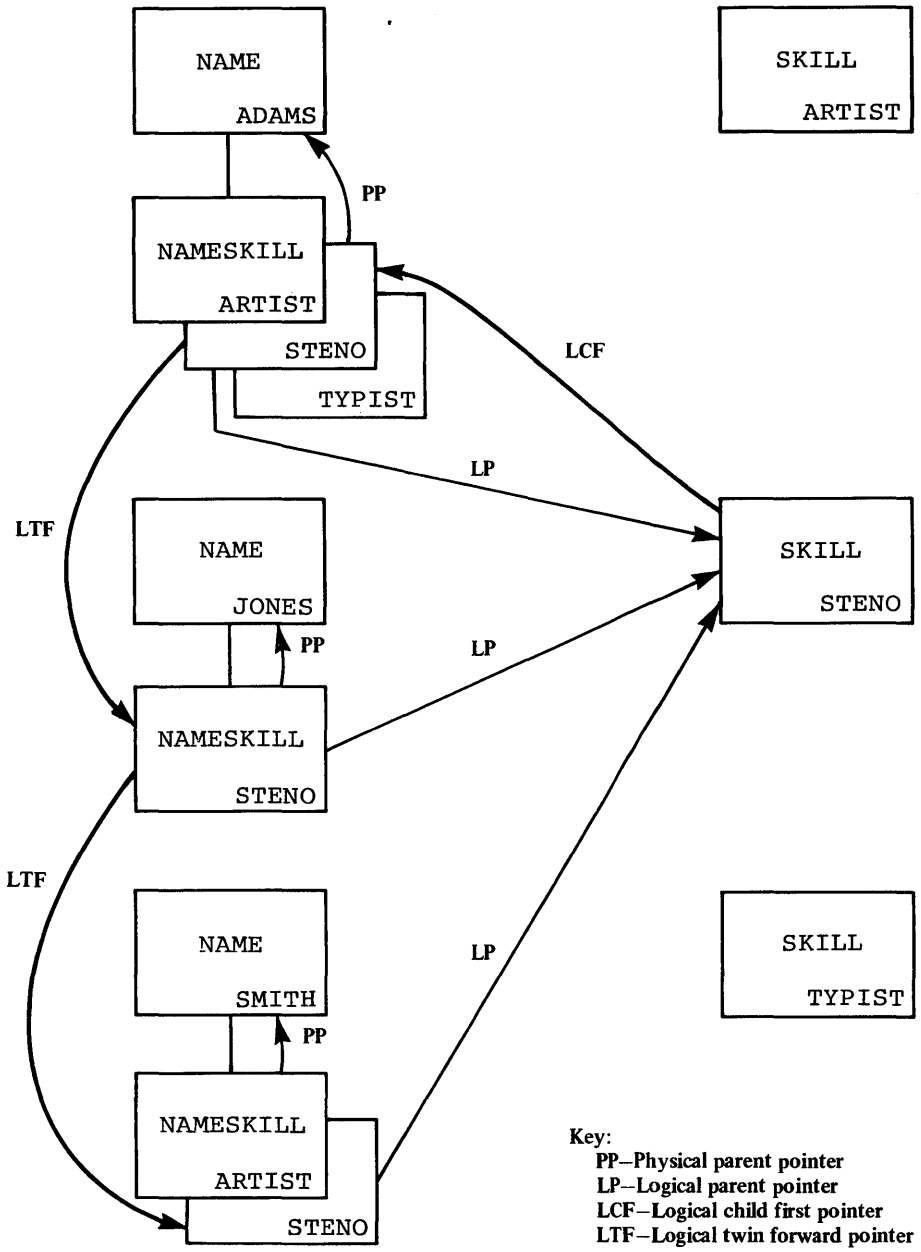


Figure 4-24. Relating Occurrences of NAME tc Occurrences of SKILL

Logical Relationship Paths

The physical relationship between physical parent and logical child segments in their physical data base, and the logical parent pointer in each logical child segment creates a physical parent to logical parent path between each physical parent segment and the logical parent segments related to each physical parent segment. To define use of the path in a logical data base, the logical child and logical parent segment types are defined as one concatenated segment type that is a physical child of the physical parent segment type as shown in Figure 4-25.

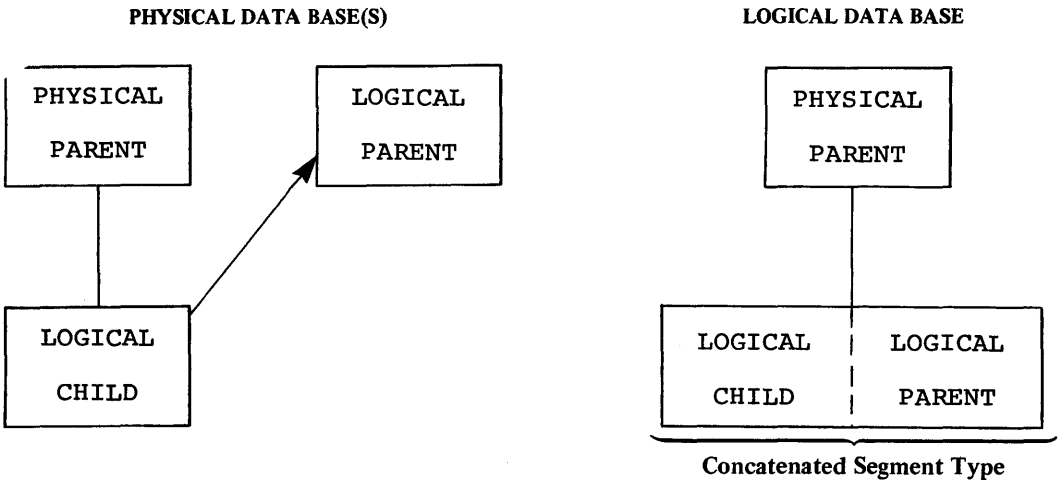


Figure 4-25. Defining a Physical Parent to Logical Parent Path in a Logical Data Base

In addition, when logical child pointers are used in the logical parent segment type, and logical twin and physical parent pointers are used in the logical child segment type, a logical parent to physical parent path is created between each logical parent segment and the physical parent segments related to each logical parent segment. To define use of the path in a logical data base, the logical child and physical parent segment types are defined as one concatenated segment type that is a physical child of the logical parent segment type as shown in Figure 4-26.

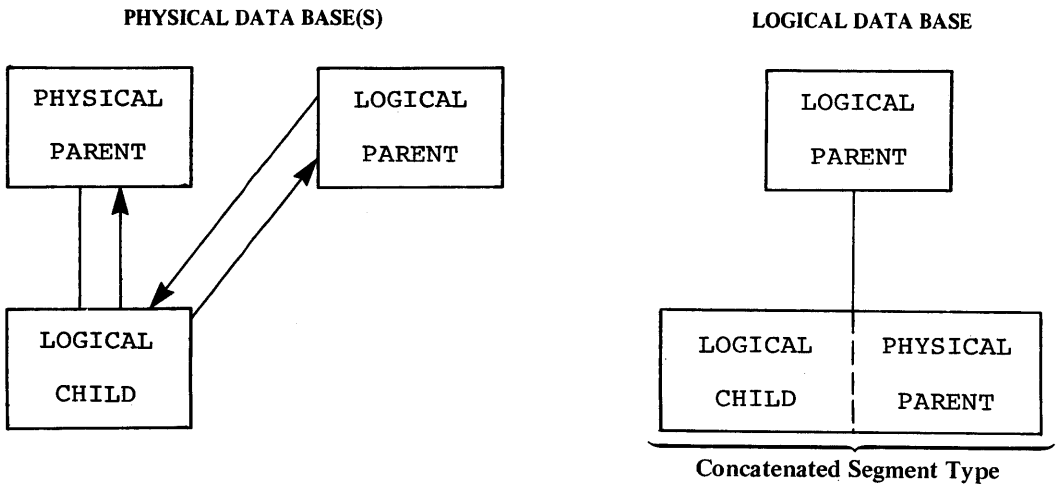


Figure 4-26. Defining a Logical Parent to Physical Parent Path in a Logical Data Base

When use of a physical parent to logical parent path between segment types is defined in a logical data base, the physical parent segment type involved in the logical relationship is the physical parent of the concatenated segment type. When an application program retrieves an occurrence of the concatenated segment type from a physical parent segment, an occurrence of the logical child and the respective logical parent pointed to by the logical child are concatenated and presented to the application program as one segment. When use of a logical parent to physical parent path is defined in a logical data base, the logical parent segment type is the physical parent of the concatenated segment type. When an application program retrieves an occurrence of the concatenated segment type from a logical parent segment, an occurrence of the logical child and the physical parent segment pointed to by the logical child are concatenated and presented to the application program as one segment.

In each case the physical parent or logical parent segment type included in the concatenated segment type is called the destination parent. For a physical parent to logical parent path, the logical parent is the destination parent in the concatenated segment type. For a logical parent to physical parent path, the physical parent is the destination parent in the concatenated segment.

Logical Child Segment

By definition, a logical child segment contains the concatenated key of the destination parent followed by intersection data, if any. A logical child segment relates a specific physical parent segment to a specific logical parent segment. Since a logical child is the point of intersection for a physical parent and logical parent segment, any data contained in a logical child segment in addition to the concatenated key of a destination parent is called intersection data. When defining a logical child segment type in its physical data base, the length specified for the segment type must be sufficient to contain the concatenated key of a logical parent. Any length greater than that required for the concatenated key can be used for intersection data.

To identify which logical parent segment will be pointed to by a logical child segment, the concatenated key of the logical parent segment must be present, with each logical child segment, in the user's I/O area when the logical child segment is initially presented for loading into a data base. However, if the logical parent segment is in a HDAM or HIDAM data base, its concatenated key may not be written to storage when the logical child segment is loaded. If the logical parent is in a HISAM data base, a logical child in storage must contain the concatenated key of its logical parent.

When a concatenated segment is retrieved through a logical data base, it contains the concatenated key of the destination parent, followed by intersection data in the logical child segment, which in turn is followed by the data in the destination parent segment. Figure 4-27 shows the format of a retrieved concatenated segment in the user I/O area. The concatenated key of the destination parent is returned with each concatenated segment to identify the destination parent retrieved. IMS/VS obtains the concatenated key of the destination parent from the logical child in the concatenated segment, or by constructing the concatenated key. If the destination parent is the logical parent of the logical child and the concatenated key of the logical parent has not been stored with the logical child, IMS/VS constructs the concatenated key of the logical parent segment and presents it to the user as a part of the concatenated segment. If the destination parent is the physical parent of the logical child, IMS/VS must always construct the concatenated key of the physical parent.

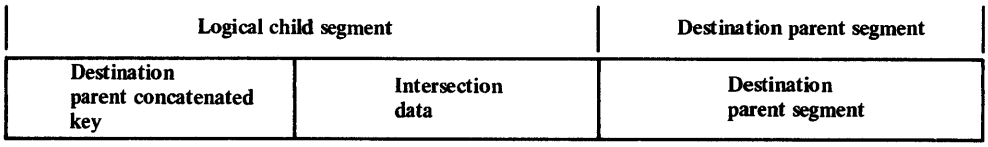


Figure 4-27. Format of Concatenated Segment Returned to User I/O Area

UNIDIRECTIONAL LOGICAL RELATIONSHIP

A unidirectional logical relationship is used to relate two segment types in one direction. Figure 4-28 shows the schematic view of a unidirectional logical relationship that is defined between two segment types in the same or different physical data bases, and the resulting view of the segment types involved that is defined in a logical data base. In a physical data base, a logical child segment type is defined as a physical child of one segment type, and a direct address or symbolic pointer is specified in the logical child segment type to point to the other segment type. This results in creating a physical parent to logical parent path between occurrences of the two segment types when they are loaded into storage.

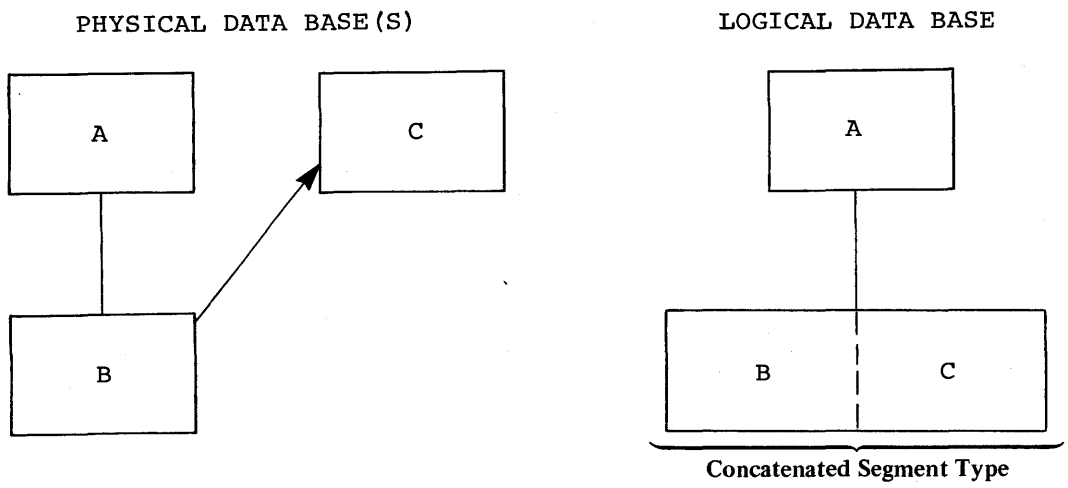


Figure 4-28. Unidirectional Logical Relationship

PHYSICALLY PAIRED BIDIRECTIONAL LOGICAL RELATIONSHIP

A physically paired bidirectional logical relationship is used to relate two segment types in two directions, and to provide the same intersection data in both directions. Figure 4-29 shows the schematic view of a physically paired bidirectional logical relationship that is defined in a physical data base or data bases, and the resulting views of the segment types involved that are defined in a logical data base. In a physical data base or data bases, a logical child segment type is defined as a physical child of each of the two segment types being related, and a direct address or symbolic logical parent pointer is specified for each logical child segment type. One logical child segment type creates a physical parent to logical parent path between occurrences of the two segment types in storage in one direction, and the other logical child segment type creates a physical parent to logical parent path between occurrences of the two segment types in storage in the other direction as shown in Figure 4-29. When defining each logical child segment type in its physical data base, the user specifies that each logical child segment type is paired with the other logical child segment type to enable IMS/VS to maintain the same intersection data in paired logical child segments. In storage, paired logical child segments provide two different paths between the same two segments, and both logical child segments contain the same intersection data. For example, in Figure 4-30 under the NAME segment ADAMS, the occurrence of NAMESKILL that points to ARTIST, and under the SKILL segment ARTIST, the occurrence of SKILLNAME that points to ADAMS are physically paired logical child segments since they provide two different paths between the same two segments and they contain the same intersection data. In a physically paired logical relationship, if the user updates intersection data in one logical child segment, IMS/VS automatically updates the intersection data in the paired logical child segment. When initially loading paired logical child segments, the user must place the same intersection data in each of the paired logical child segments.

During the initial load of a data base that contains physically paired logical children, the application program must load (using an ISRT call) both sides of the physical pair. The intersection data for the paired segments must be identical. After the initial load, in any update step, if an insert, delete, or replace is done for one of the paired segments, IMS/VS performs the same function for the paired segment.

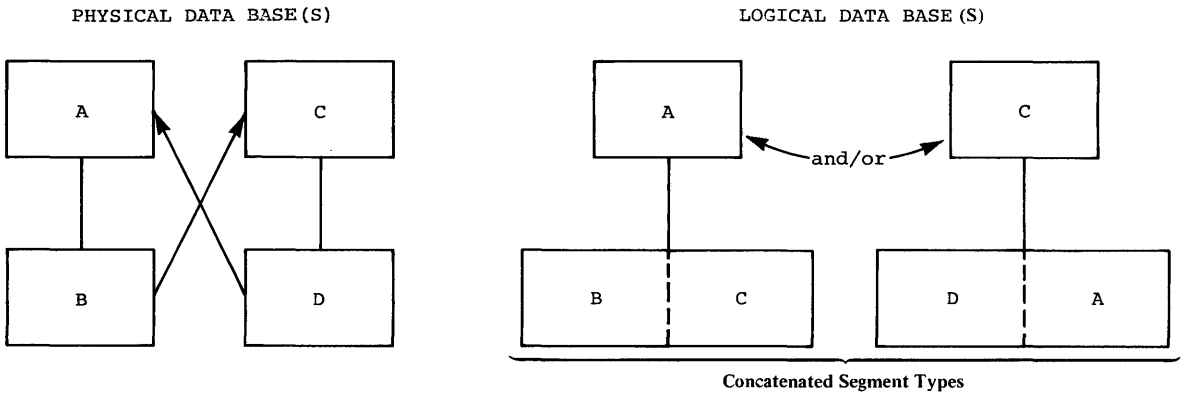


Figure 4-29. Physically Paired Bidirectional Logical Relationship

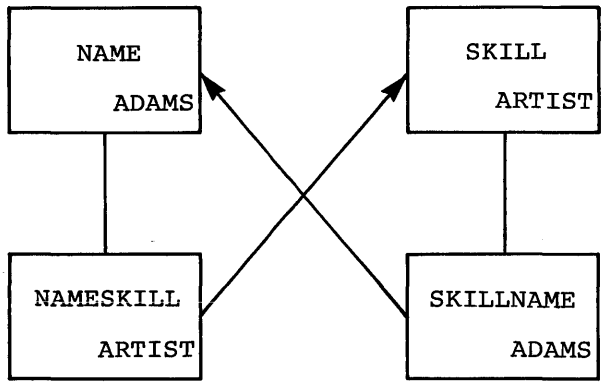


Figure 4-30. Physically Paired Logical Child Segments

VIRTUALLY PAIRED BIDIRECTIONAL LOGICAL RELATIONSHIP

In a virtually paired bidirectional logical relationship, one logical child segment type in storage relates two segment types in two directions, and provides the same intersection data in both directions. Figure 4-31 shows the schematic view of a virtually paired bidirectional logical relationship that is defined in a physical data base or bases, and the resulting views of the segment types involved that are defined in a logical data base.

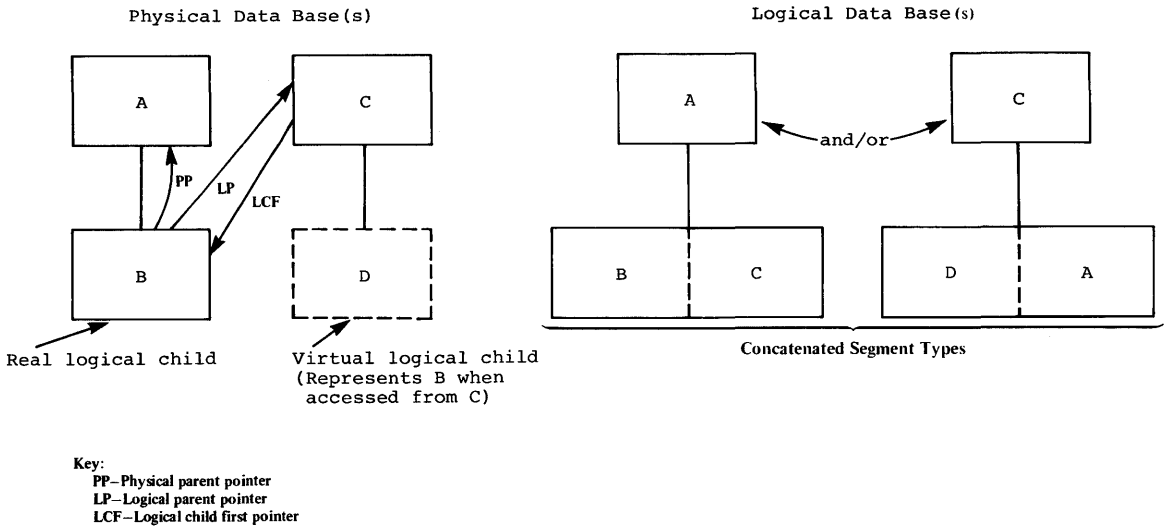


Figure 4-31. Virtually Paired Bidirectional Logical Relationship

To define a virtually paired bidirectional logical relationship, two logical child segment types are defined in the physical data bases involved in the logical relationship, but only one is actually placed in storage. The logical child segment type that is defined and results in storage is called the real logical child. The logical child segment type that is defined, but does not result in storage is called the virtual logical child.

In a virtually paired bidirectional logical relationship, occurrences of the real logical child create physical parent to logical parent, and logical parent to physical parent paths between occurrences of the two segment types being related. To accomplish this the real logical child is defined as a physical child segment type of one of the segment types being related, and a symbolic or direct address logical parent pointer is specified for the real logical child segment type. This creates a physical parent to logical parent path between occurrences of the two segment types being related. In addition, logical child pointers are specified for the logical parent segment type of the real logical child, and logical twin pointers are specified for the real logical child segment type to create a logical parent to physical parent path in storage between occurrences of the two segment types being related. The physical parent pointers required in occurrences of the real logical child for a logical parent to physical parent path are generated automatically by IMS/VS.

For the physical parent to logical parent path, the user controls the sequence in which occurrences of the real logical child are accessed from their physical parent segment by defining a sequence field in the real logical child segment type, or by specifying use of the insert rule of first, last or here when defining the real logical child in its physical data base. For the logical parent to physical parent path, the user controls the sequence in which occurrences of the real logical child are accessed from their logical parent by defining a virtual logical child segment type as a physical child of the logical parent of the real logical child, and in addition, by defining a sequence field in the virtual logical child. Or, the user can specify a second insert rule of first, last or here that controls the sequence of real logical child segments as viewed from their logical parent segment. The insert rule that controls the sequence of real logical child segments as viewed from their physical parent segment is specified on the SEGM statement that defines the real logical child segment type

in its physical data base. The insert rule that controls the sequence of real logical child segments as viewed from their logical parent is specified on an ICHILD statement. As input to DBDGEN when defining a segment type in a physical data base that is used as a logical parent in one or more logical relationships, an ICHILD statement must follow a SEGM statement that defines a logical parent segment type for each logical child segment type of that logical parent. LCHILD statements identify the logical child segment types of a logical parent by following a SEGM statement that defines a logical parent. For a virtually paired bidirectional logical relationship, when no sequence field or a non-unique sequence field is defined for the real logical child segment type as viewed from its logical parent segment type, the insert rule of first, last or here specified on an LCHILD statement controls the sequence in which occurrences of the real logical child are accessed from their logical parent segment.

To enable using a sequence field for sequencing occurrences of the real logical child from its logical parent segment type, a virtual logical child segment type is defined. A virtual logical child segment type is defined as a physical child of the logical parent segment type of the real logical child. A virtual logical child segment type is defined in the physical data base of the logical parent of the real logical child to represent the view of the real logical child when accessing the real logical child from its logical parent. In defining a virtual logical child segment type, a name is specified for the virtual segment type and the name of the real logical child segment type is associated to the name specified. To enable sequencing occurrences of the real logical child through sequence field values from the logical parent, a sequence field is defined in the virtual segment type. Since the virtual segment type represents the real logical child as viewed from its logical parent, the sequence field defined represents fields in the real logical child segment type as viewed from its logical parent type.

Defining Fields in Logical Child Segment Types

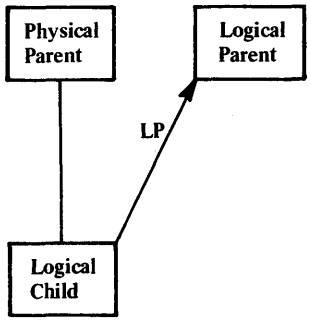
Since a logical child segment, by definition in a logical data base contains the concatenated key of a destination parent, followed by intersection data, if any, the concatenated key of the destination parent is included as a part of the logical child segment type when defining fields within the logical child segment. For a physical parent to logical parent path, fields can be defined within the logical child segment type that are comprised of the concatenated key of the logical parent. For a logical parent to physical parent path, fields can be defined within the logical child segment type that are comprised of the concatenated key of the physical parent. In addition for a logical parent to physical parent path, fields defined within the logical child segment type can be comprised of non-contiguous data in the logical child.

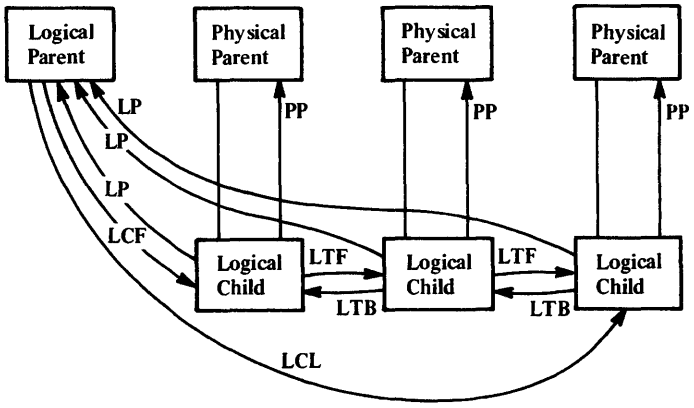
POINTERS AND THE COUNTER USED IN LOGICAL RELATIONSHIPS

Logical relationships can be defined in HISAM, HDAM and HIDAM data bases, or between any combination of the three. In defining logical relationships in each type or between types, the data organization methods used for the data bases must be considered when specifying the pointers used in logical relationships. Physical adjacency in storage is used to relate segments in a HISAM data base which means that all pointers to segments stored in a HISAM data base must be symbolic. In HDAM and HIDAM data bases, segments in storage are related through direct address pointers. Segments stored in HDAM and HIDAM data bases can be pointed to by symbolic or direct address pointers.

The following pointers are used in defining logical relationships (see Figure 4-32):

- Logical Parent Pointer
- Logical Child Pcinter
- Logical Twin Pointer
- Physical Parent Pcinter





Key:
PP–Physical parent pointer
LP–Logical parent pointer
LCF–Logical child first pointer
LCL–Logical child last pointer
LTF–Logical twin forward pointer
LTB–Logical twin backward pointer

Figure 4-32. Pointers Used in Logical Relationships

Logical Parent Pcinter

A logical parent pcinter points from a logical child segment to a logical parent segment. To point to a logical parent segment type in a HISAM data base, a symbolic pointer must be stored with each logical child segment. To point to a logical parent segment type in an HDAM or HIDAM data base, a symbclic pointer can be stored with each logical child segment and/or a direct address logical parent pcinter can be specified.

Logical Child/Logical Twin Pointers

Logical child and logical twin pointers are only specified in virtually paired bidirectional logical relationships. The logical child pointers that can be specified are logical child first, or logical child first and last pointers. A logical child first, or a combination of logical child first and last pointers are stored in the prefix of a logical parent segment to point to each of its logical child segment types. A logical child first pointer points to the first occurrence of a logical child segment type, and a logical child last pointer points to the last occurrence of that segment type when viewed from the logical parent.

The logical twin pointers that can be specified are logical twin forward or the combination of logical twin forward and backward pointers. Logical twins are multiple logical child segments of the same type that point to the same occurrence of a logical parent. A logical twin forward pointer points from a given logical twin to the logical twin stored after it and a logical twin backward pointer points from a given logical twin to the logical twin stored before it. Use of the logical twin backward pointer improves delete performance.

Physical Parent Pointers

In HDAM and HIDAM data bases involved in logical relationships, physical parent pointers are generated automatically by IMS/VS. IMS/VS places physical parent pointers in the prefix of all logical child and logical parent segments, and in the prefix of all segments on which a logical child or logical parent segment is dependent in its physical data base. This creates a path from a logical child or logical parent segment to the root segment on which the logical child or logical parent segment is dependent. Since all segments on which a logical child or logical parent segment is dependent are chained through physical parent pointers from the logical child or logical parent segment to its root, access to those segments in reverse order is enabled through a logical data base.

Counter

A four-byte counter is required in all logical parent segments that do not contain logical child pointers. It is stored in the prefix of a logical parent segment to maintain a count of how many logical child segments point to the logical parent. When required, it is placed in logical parent segments automatically by IMS/VS.

DEFINING SEQUENCE FIELDS FOR DATA BASES INVOLVED IN LOGICAL RELATIONSHIPS

To avoid potential problems in processing data bases involved in logical relationships, unique sequence fields should be defined in all logical parent segment types, and in all segment types that a logical parent is dependent on in its physical data base. When unique sequence fields are not defined in all segment types on the path to and including a logical parent segment type, multiple logical parent segments in a data base can have the same concatenated key. When multiple logical parent segments have the same concatenated key, problems can arise during initial data base load, and after initial data base load when symbolic logical parent pointers in logical child segments, are used to establish position on a logical parent segment to be processed.

At initial data base load time, if logical parent segments with nonunique concatenated keys exist in a data base, the logical relationship resolution utilities attach all logical child segments that contain the same concatenated key to the first logical parent segment in a data base that has that concatenated key.

When inserting or deleting a concatenated segment and position for the logical parent portion of the concatenated segment is determined by the logical parents concatenated key, positioning for the logical parent steps on the first segment at each level of the logical parents data base that satisfies the key equal condition for that level. For insert when using this method of establishing position in the logical parents data base, if a segment is missing on the path to the logical parent segment to be inserted, a GE status code is returned to the application program. Under the same conditions for deletion of a logical parent segment a U803 abnormal termination occurs.

RULES FOR DEFINING LOGICAL RELATIONSHIPS IN PHYSICAL DATA BASES

Following are the rules that must be followed when defining logical relationships in physical data bases.

Logical Child

1. A logical child segment type must have a physical parent segment type and a logical parent segment type.

2. A logical child segment type can have only one physical parent segment type and one logical parent segment type.

3. A logical child segment type is defined as a physical child segment type in the physical data base of its physical parent.

4. A logical child segment type is always a dependent segment type in a physical data base, and as such, it can be defined at any level except the first level of a data base.

5. In its physical data base, a logical child segment type can not have a physical child segment type defined at the next lower level in the data base that is also a logical child.

6. A logical child segment type can have physical child segment types. However, if a logical child segment type is physically paired with another logical child segment type, only one of the paired segment types can have physical child segment types.

Logical Parent

1. A logical parent segment type can be defined at any level of a physical data base including the root level.

2. A logical parent segment type can have one or multiple logical child segment types. Each logical child segment type related to the same logical parent segment type defines a logical relationship.

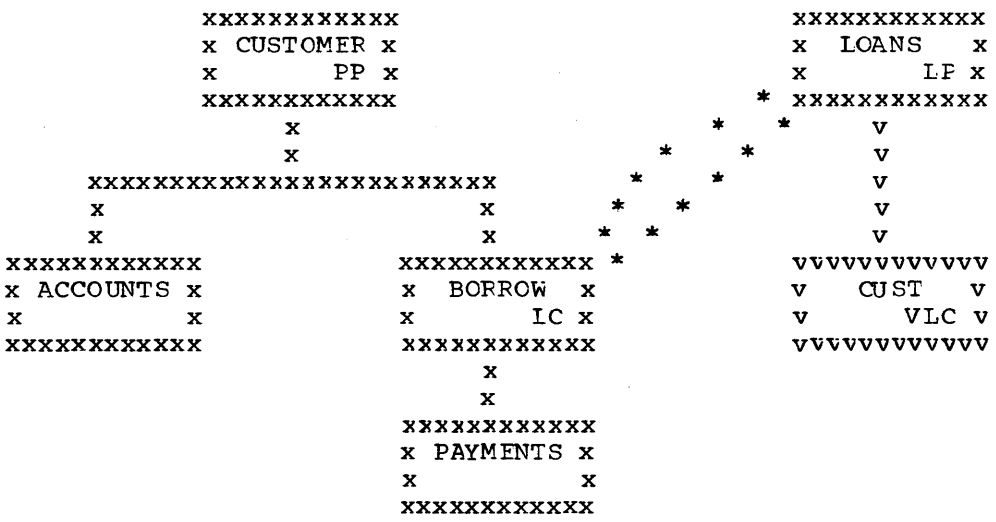
3. A segment type in a physical data base can not be defined as both a logical parent and a logical child.

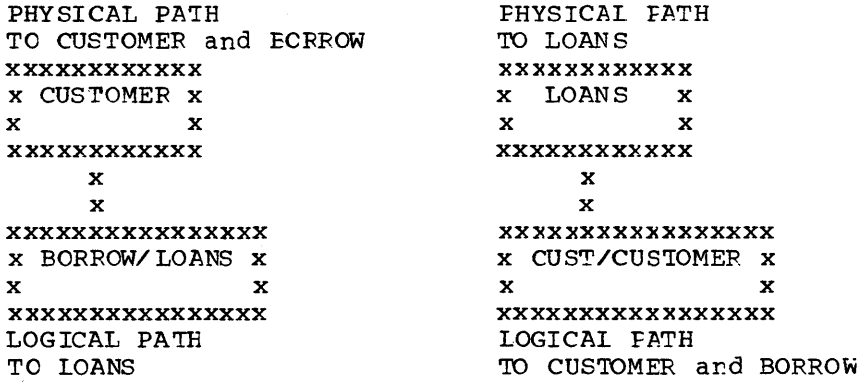
4. A logical parent segment type can be defined in the same physical base as its logical child segment types, or in a different physical data base.

Physical Parent

1. A physical parent segment type of a logical child cannct also be a logical child.

REPLACE, INSERT AND DELETE RULES





Insert, Delete, and Replace rules are needed when a segment is involved in a logical relationship because that segment is updatable from two paths; a physical path and a logical path.

Think a minute about the following questions:

1. Should the CUSTOMER segment be insertable by both its physical and logical paths?

2. Should the BORROW segment be replaceable via only the physical path, or by both the physical and logical paths?

3. If the LOANS segment is deleted by its physical path, should it be erased from the data base or should it be marked as physically deleted but remain accessible by its logical path?

4. If the logical child segment BORRCW or the concatenated segment BORROW/LCANS is deleted from the physical path, should the logical path CUST/CUSTOMER also be automatically deleted or should the logical path remain?

The answer to these questions depends on the application, but the enforcement of the answer depends on choosing the correct insert/delete/replace rules for the logical child, logical parent and physical parent segments.

The application processing requirements must be determined first, and the rules that support (enforce) those application processing requirements must be determined second.

For instance, the answer to question one depends on whether or not the application defines that a CUSTOMER segment must have been previously inserted into the Data Base prior to accepting the loan. An insert rule of physical (P) on the CUSTOMER segment would prohibit the insertion of the CUSTOMER segment except by the physical path. While an insert rule of virtual (V) would allow inserting the CUSTOMER segment by either the physical or logical path.

It probably makes sense for a customer to be checked (past credit, time on current job, etc) and the CUSTOMER segment inserted prior to approving the loan and inserting the BORROW segment. Thus, the insert rule of the CUSTOMER segment should be physical (P) to prevent this segment from being inserted logically (which incidentally provides better control of the application).

Consider question three. We can reason two ways: (1) If it is possible for this load institution to terminate a type of loan (cancel 7% car loans -- create 9% car loans) before everyone who has that type of loan has fully paid the loan, then we are saying that it's possible for the LOANS segment to be physically deleted and still be accessible from the logical path. This condition is supportable by specifying the delete rule for LOANS as either logical (L) or virtual (V) but not as physical (P).

The physical (P) delete rule prohibits physically deleting a logical parent segment prior to all its logical children having been physically deleted (which means the logical path to the logical parent is deleted first).

INTRODUCTION SUMMARY

Data Base Administrators should examine all application needs and decide who may insert, delete, and replace segments involved in logical relationships and how those updates are to be made (physical path only or physical and logical path). The insert/delete/replace rules in the physical DBD and the PROCOPT parameter in the PCB are the means of control. These rules are explained in detail in the following pages.

```
                                        PPP FIRST
SEGM  NAME=  , , , , , , ,RULES=(LLL,LAST )
                                        VVV HERE
                                         B
                          insert_____///
                          delete_____//
                          replace____/
```

P = PHYSICAL
L = LOGICAL
V = VIRTUAL
B = BIDIRECTIONAL VIRTUAL

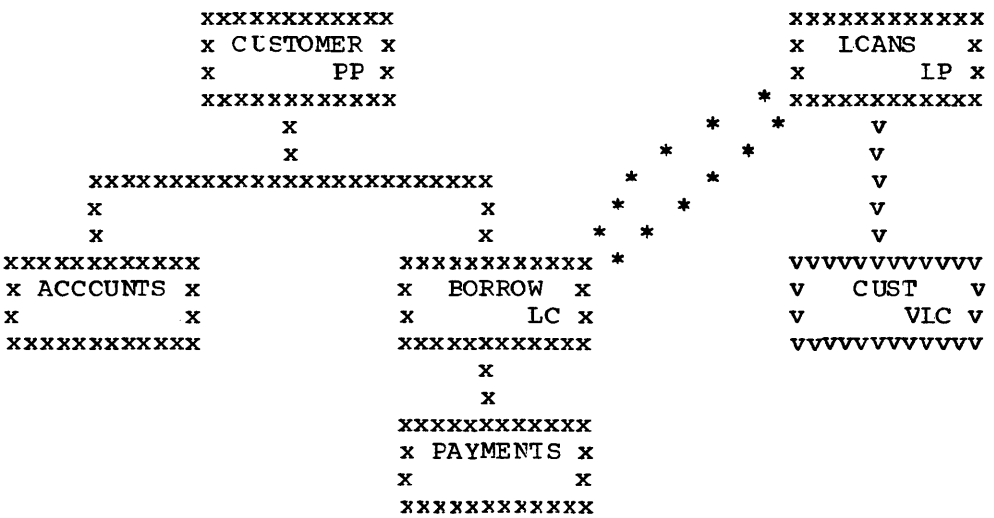
The operands of the RULES parameter are positional. Position one defines the INSERT rule, position two defines the DELETE rule and position three the REPLACE rule.

For example, RULES=PLV says the insert rule is physical, the delete rule is logical and the replace rule is virtual. Notice the "B" rule is only applicable for delete.

The second positional operand (FIRST,LAST,HERE) does not apply in any way to a discussion concerning LOGICAL UPDATE RULES and was only included to maintain the correctness of the coding example.

In general the "P" rule (physical) is the most restrictive and the "V" rule (virtual) the least restrictive with the "L" rule (logical) somewhere in between.

RULES are applicable only to the segments involved in logical paths; the Logical Child segment and its Logical Parent and Physical Parent segments. Rules are not coded for the virtual logical child.



THE REPLACE RULES

Applicable to the Physical Parent, Logical Parent and Logical Child segments of a Logical Path.

1. PHYSICAL: The segment can only be replaced when retrieved via a physical path. If this rule is violated, no data is replaced and an 'RX' status code is returned.

2. LOGICAL: The segment can only be replaced when retrieved via a physical path. If this rule is violated, no data is replaced, however, an 'RX' status code is not returned. A 'bb' status code is returned.

3. VIRTUAL: The segment can be replaced when retrieved by either a physical or logical path.

The Replace Call

A replace can be performed only on that portion of a concatenated segment to which an application program is data sensitive.

If no data is changed in a segment, no data is replaced and no replace rule is violated.

If data in a concatenated segment has been changed, data is replaced only if neither portion of the concatenated segment violates its replace rule.

The replace rule is not checked for a segment which is part of a concatenated segment but was not retrieved.

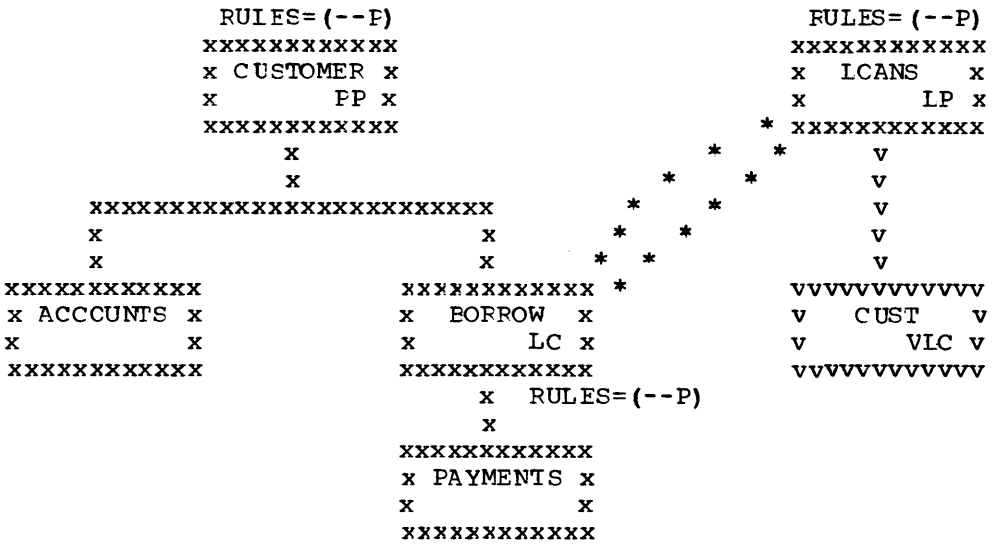
The status code returned to an application program will indicate the first violation that was detected. These status codes are:

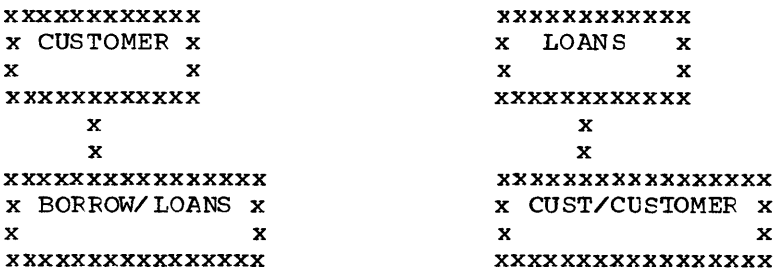
'AM' - Replace attempted and PROCOPT≠R

'DA' - Key field of segment was changed

'RX' - Replace rule violated

Physical Replace Rule Example

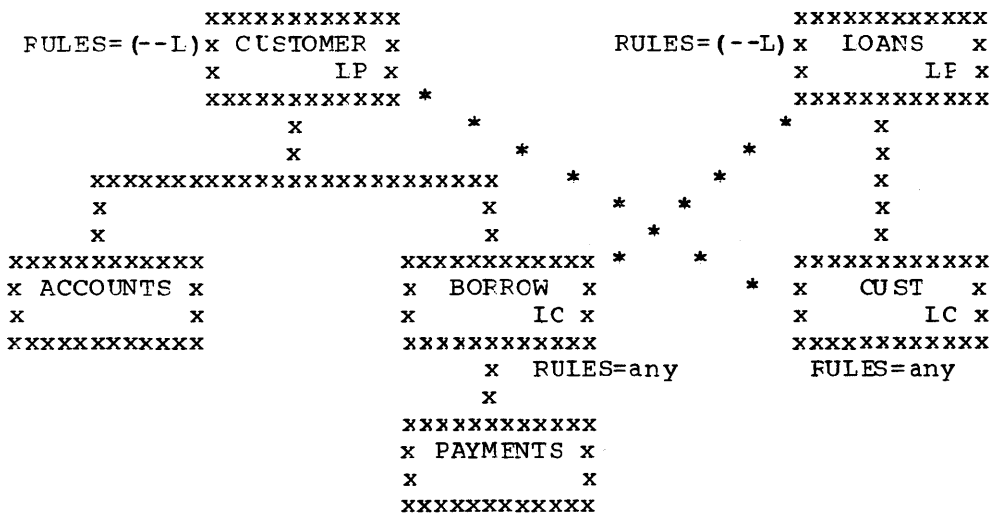


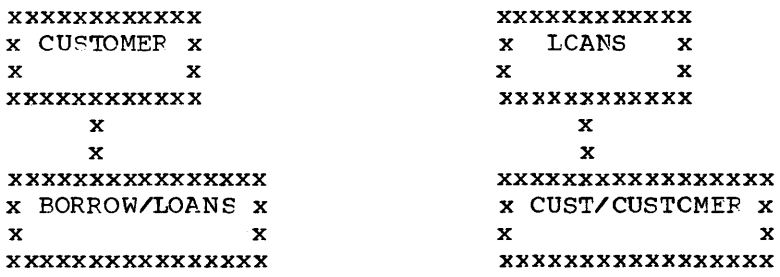



```
GHU   'CUSTOMER'          STATUS CODE='bb'

REPL                      STATUS CODE='bb'

GHN   'BORROW/LCANS'      STATUS CODE='bb'

REPL                      STATUS CODE='RX'
```

The physical replace rule prevents replacing the LOANS segment as part of a concatenated segment. Replacement must be by the segment's physical path.

Logical Replace Rule Example



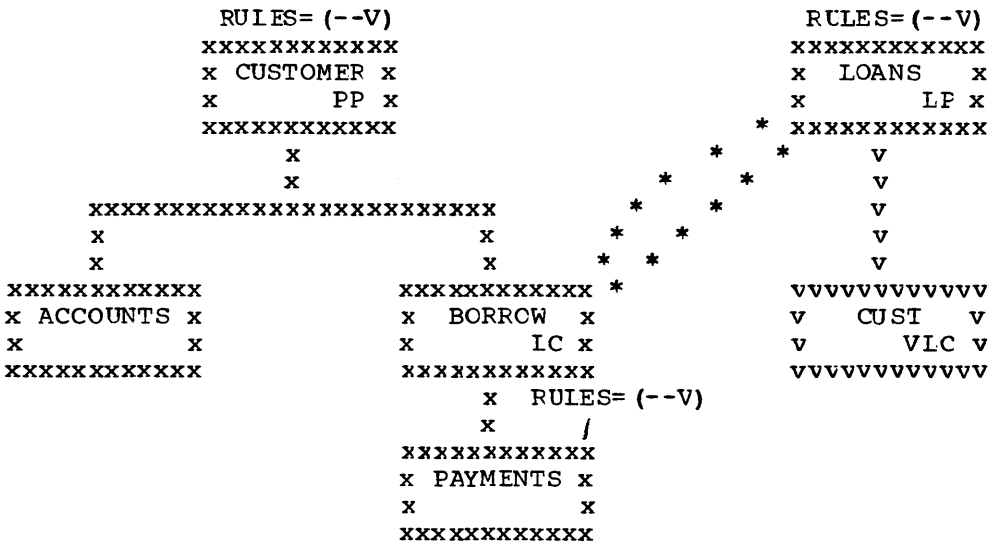


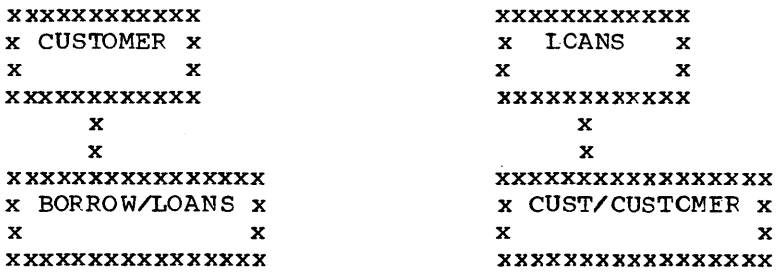

```
GHU   'CUSTOMER'
      'BORRCW/LOANS'      STATUS CODE='ØØ'

REPL                      STATUS CODE='ØØ'
```

The logical replace rule prevents replacing the LCANS segment as part of a concatenated segment, since replacement must be by the segment's physical path. However, the status code returned is 'ØØ'. The BORROW segment, being accessed by its physical path, is replaced. Since the access of the logical child is by its physical path, it does not matter what replace rule is selected.

The LOGICAL replace rule provides for the special case of allowing the replacement of only the logical child half of the concatenation, and the return of a 'ØØ' status code.

Virtual Replace Rule Example






```
GHU   'LOANS'
      'CUST/CUSTOMER'     STATUS CODE='bb'

REPL                      STATUS CODE='bb'
```

The virtual replace rule allows replacing the CUSTOMER segment via its logical path as part of a concatenated segment.

Replace Rules Summary

Specifying the replace rule as virtual, on any of the segments involved in the logical relationship, allows replacing that segment by either its physical path or logical path.

Specifying the replace rule as physical, on any of the segments involved in the logical relationship, prevents the replacement of that segment except when retrieved via its physical path.

The logical replace rule provides for a special case. Specifying the replace rule for the logical parent as LOGICAL, allows IMS/VS to return a 'bb' status code but without replacing any data when the logical parent is accessed concatenated with the logical child. Since the logical child has been accessed by its physical path, its replace rule may be any of the three. Thus using the LOGICAL replace rule allows the selective replacement of the logical child half of the concatenation and a 'bb' status code.

Figure 4-33 shows all possible combinations of the replace rules that can be specified, and the resulting actions that take place for each combination when a call is issued to replace a concatenated segment in a logical data base.

SEGMENT REPLACE RULES

Logical View 1

Replace rule specified	B	P	P	P	P	P	P	P	P	P	L	L	L	L	L	L	L	L	V	V	V	V	V	V	V	V
	C	P	P	P	L	L	L	V	V	V	P	P	P	L	L	L	V	V	V	P	P	P	L	L	L	V	V	V
Denotes segment you are attempting to replace	B	X		X	X		X	X		X	X		X	X		X	X		X	X		X	X		X	X		X
	C		X	X		X	X		X	X		X	X		X	X		X	X		X	X		X	X		X	X
Status code			RX	RX								RX	RX								RX	RX						
Data replaced? Y = yes N = no	B	Y		N	Y		Y	Y		Y	Y		N	Y		Y	Y		Y	Y		N	Y		Y	Y		Y
	C		N	N		N	N		Y	Y		N	N		N	N		Y	Y		N	N		N	N		Y	Y

Logical View 2

Replace rule specified	B	P	P	P	P	P	P	P	P	P	L	L	L	L	L	L	L	L	V	V	V	V	V	V	V	V
	A	P	P	P	L	L	L	V	V	V	P	P	P	L	L	L	V	V	V	P	P	P	L	L	L	V	V	V
Denotes segment your are attempting to replace	B	X		X	X		X	X		X	X		X	X		X	X		X	X		X	X		X	X		X
	A		X	X		X	X		X	X		X	X		X	X		X	X		X	X		X	X		X	X
Status code		RX	RX	RX	RX		RX	RX		RX		RX	RX								RX	RX						
Data replaced? Y = yes N = no	B	N		N	N		N	N		N	N		N	N		N	N		N	Y		N	Y		Y	Y		Y
	A		N	N		N	N		Y	N		N	N		N	N		Y	Y		N	N		N	N		Y	Y

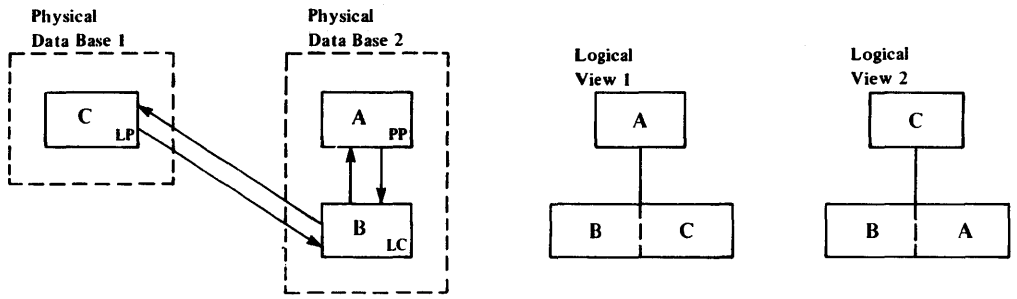


Figure 4-33. Replace Rules

THE INSERT RULES

Applicable to the Destination Parent (Logical Parent and Physical Parent) segments, but not to the Logical Child segment. See "Logical Child Insertion" below.

1. PHYSICAL: The destination parent may be inserted only via its physical parent path.

   This means that the destination parent must exist prior to inserting a logical path. A concatenated segment is not needed; the logical child is inserted by itself.

2. LOGICAL: The destination parent may be inserted either via its physical path or concatenated with the logical child via the logical path.

   When a logical child/destination parent concatenated segment is inserted, the destination parent is inserted provided it does not already exist and the I/O area key check does not fail (see 'DA' status code). If the destination parent does exist, it will remain unchanged and the logical child will be connnected to it.

3. VIRTUAL: The destination parent may be inserted via its physical path or concatenated with the logical child via the logical path.

   When a logical child/destination parent concatenated segment is inserted, the destination parent is replaced if it already exists, and is inserted if it does not.

Logical Child Insertion

The RULES operand must be coded to supply replace and delete rules for the logical child. However, the insert rule has no meaning except to satisify the RULES macro's coding scheme, so any insert rule (P,L,V) may be coded.

1. A logical child will be inserted provided that the insert rule of the destination parent is not violated, and

2. The logical child to be inserted does not already exist (i.e., is not a duplicate).

The Insert Call

The I/O area in an application program must contain either the logical child or the logical child/destination parent concatenated segment in accordance with the destination parent's insert rule.

The logical child/destination parent concatenated segment insert operation is performed only if both components of the concatenated segment can be inserted.

The insert operation is not affected by KEY or DATA sensitivity as specified in a logical DBD or a PCB. This means that if a program is other than DATA sensitive to both the logical child and the destination parent of a concatenated segment, that program must nevertheless supply both in the I/O area when inserting a logical path, and the insert rule is logical or virtual. Thus maintenance programs that insert concatenated segments should be DATA sensitive to both segments in the concatenation.

Status Codes

'AM' - An insert was attempted and PROCOPT≠I.

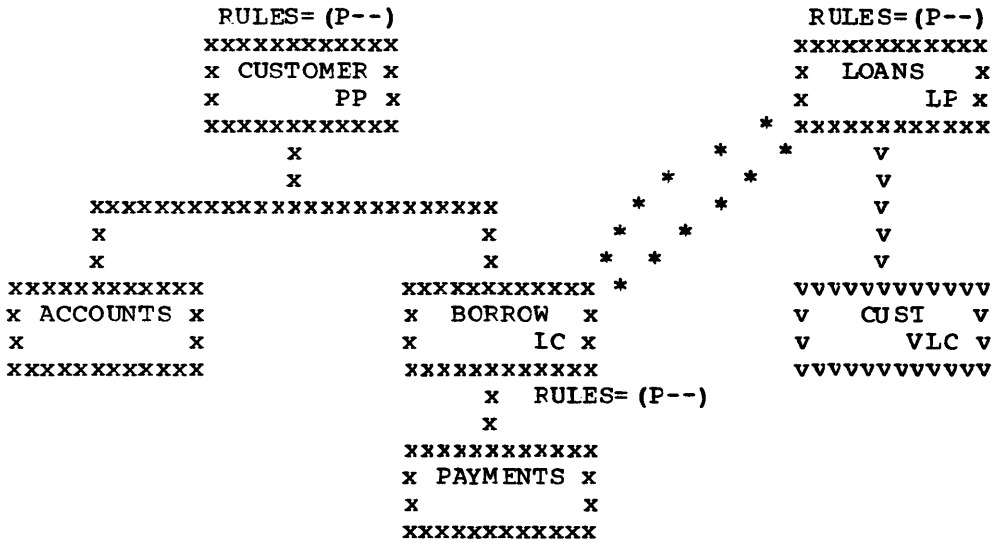
'GE' - Parent of the destination parent or logical child was not found.

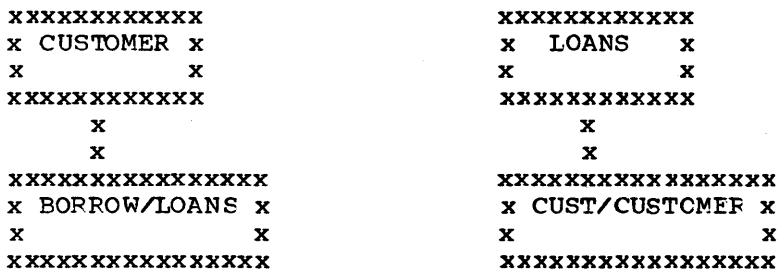
'II' - Attempt to insert duplicate segment.

'IX' - Physical rule and destination parent not found.

- I/O area key check fails. Concatenated segments contain the destination parent's key twice -- once as part of ICHILD's LPCK and second as a field in the parent. The keys must be equal.

Physical Insert Rule Example





If the LOANS segment does exist then:

ISRT 'CUSTOMER' STATUS CODE='bb'

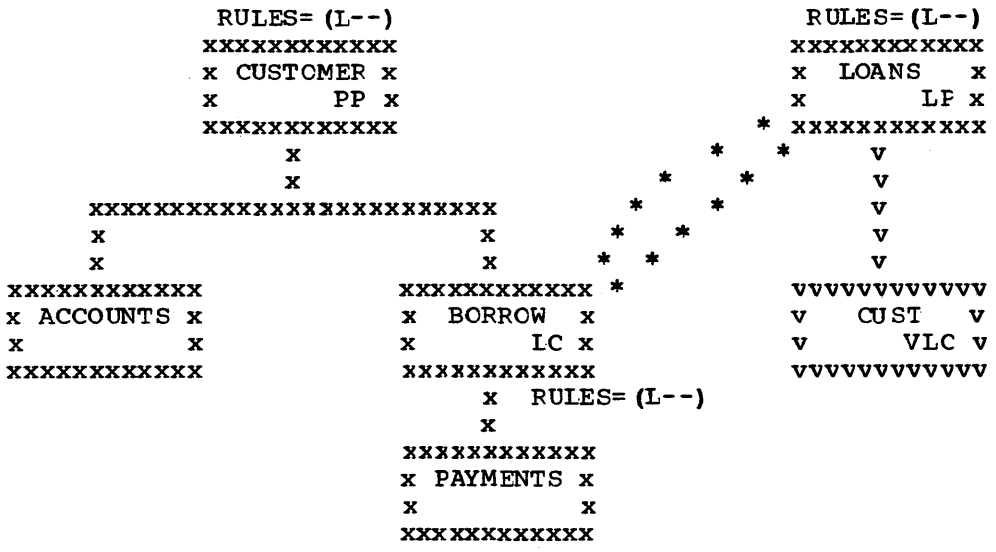
ISRT 'BORROW' STATUS CODE='bb'

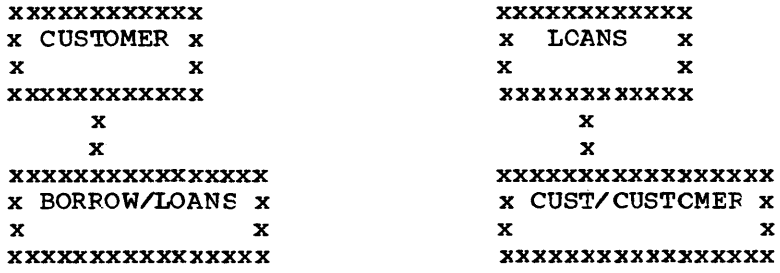
However, if LOANS does not exist, then:

ISRT 'CUSTOMER' STATUS CODE='bb'

ISRT 'BORROW' STATUS CODE='IX'

Logical Insert Rule Example

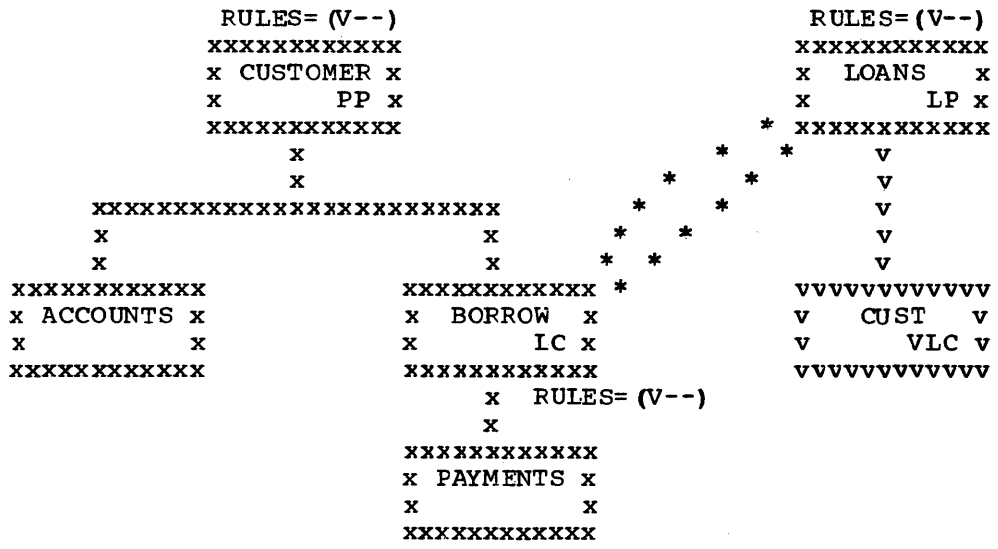


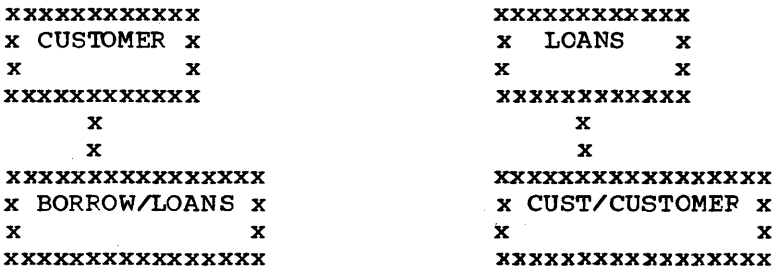



```
   ISRT  'LOANS'       STATUS CODE='ØK'

   ISRT  'CUST'        STATUS CODE='IX'
```

The 'IX' status code is the result of omitting the concatenated segment CUST/CUSTOMER in the second call. IMS/VS checked for the key of the CUSTOMER segment (in the I/O area) and failed to find it. With the logical insert rule, the concatenated segment must be inserted to create a logical path.

Virtual Insert Rule Example





ISRT 'CUSTOMER' STATUS CODE='bb'

ISRT 'BORROW/LCANS' STATUS CODE='bb'

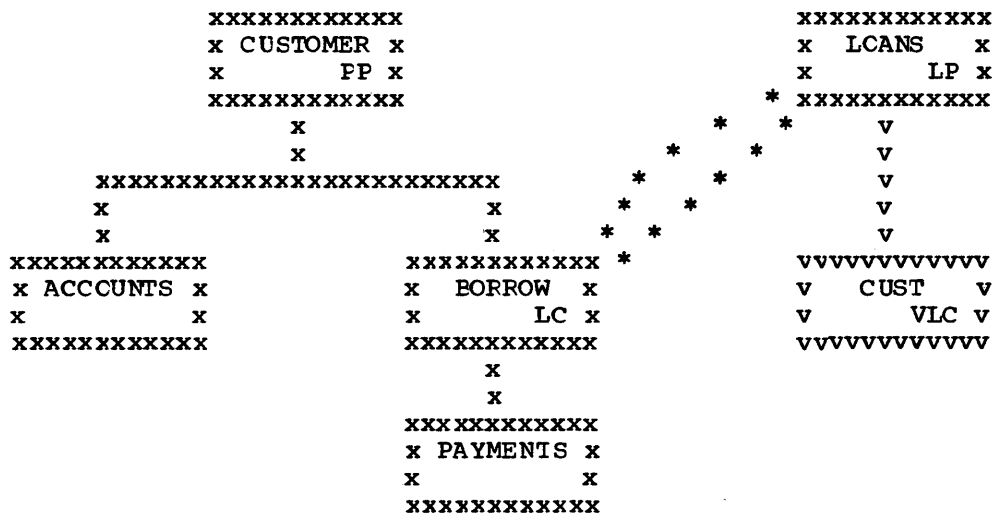
Remember this action will replace the LOANS segment if present, and insert the LOANS segment if not, so the virtual insert rule is a very powerful option.

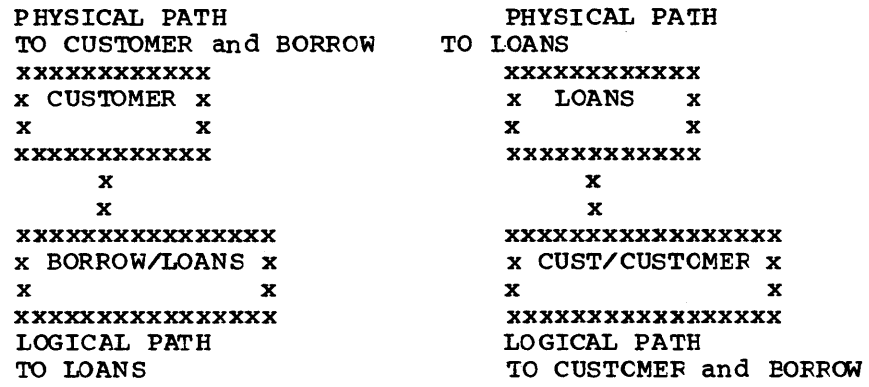
Insert Rules Summary

The virtual insert rule is the most powerful of the three rules in that it will insert the destination parent (inserted as a concatenated segment via the logical path) if the parent didn't previously exist, and replace the existing destination parent with the inserted destination parent ctherwise.

Specifying the insert rule as logical on the logical parent and the physical parent, allows insertion via either its physical path or its logical path as part of a concatenated segment. When inserting a concatenated segment, if the destination parent already exists, it will remain unchanged and the logical child will be connected to it. If it does not exist, it will be inserted. In either case, the logical child will be inserted provided that the segment is not a duplicate and that the destination parents insert rule is not violated.

Specifying the insert rule as physical prevents inserting the destination parent as part of a concatenated segment. This means that a destination parent may be inserted only by its physical path. If the insert creates a logical path, only the logical child needs be inserted.





The DLET call is a request for access path deletion not DASD space release of a segment. Delete rules are needed when a segment is involved in a logical relaticnship because that segment belongs to two paths; a physical path and a logical path.

The selection of the delete rules for the logical child and its logical and physical parent (or two logical parents if physical pairing), determines whether one or two DLET calls are necessary to delete the two access paths.

Physical and Logical Deletion

1. PHYSICAL DELETION: Physically deleting a segment prevents further access to that segment via its physical parents. Physically deleting a segment also physically deletes its physical dependents.

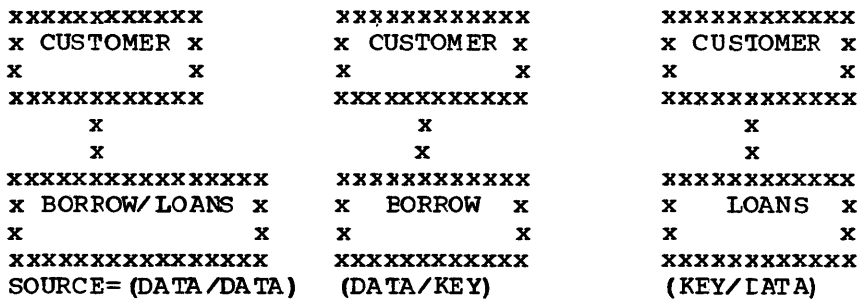
   EXCEPTION: If one of the physical parents of the physically deleted segment is a logical child segment which has been accessed from its logical parent, then the physically deleted segment is accessible from that logical child since the physical dependents of a logical child are "Variable Intersection Data."

2. LOGICAL DELETION: Logically deleting a logical child prevents further access via its logical parent. Unidirectional logical child segments are assumed to be logically deleted.

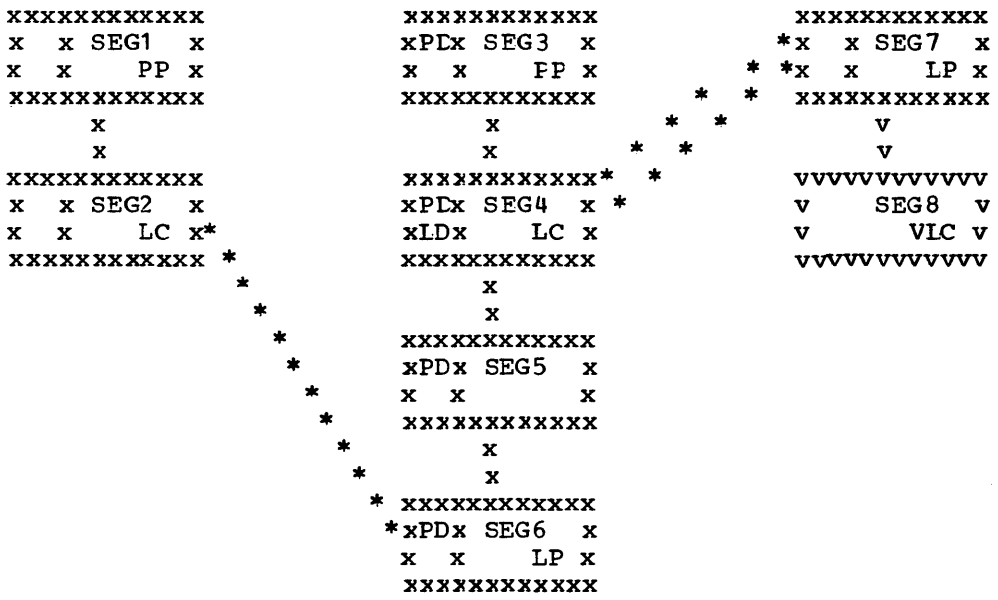
   A logical parent is considered logically deleted when all of its logical children are physically deleted. For physically paired logical relationships, the physical child paired to the logical child must also be physically deleted, before the logical parent is considered logically deleted.

Deleting Concatenated Segments

The picture below shows that an application program can be sensitive to either the concatenated segment (SOURCE=(DATA/DATA), (DATA/KEY), (KEY/DATA) or only the logical child, since it is the logical child that is either physically or logically deleted (depending on the path accessed) in all cases.



The Third Access Path



There are three paths to the logical child segment SEG4. The physical path from its physical parent SEG3, the logical path from its logical parent SEG7, and a third path from its physical dependents (SEG5 and SEG6) because segment SEG6 is a logical parent accessible from its logical child SEG2.

These paths are "full-duplex" paths, meaning that accessibility is two way (up and down). There are two delete bits that control access along the paths, but they are "half-duplex," meaning that they only block half of each respective path. There is not a bit that blocks the third path. If SEG4 were both physically and logically deleted (PD and LD bits set), it would still be accessible from the third path and so would both of its parents.

Neither physical nor logical deletion prevents access to a segment from its physical or logical children. Logically deleting SEG4 prevents access to SEG4 from its logical parent SEG7, but does not prevent access from SEG4 to SEG7. Likewise, physically deleting SEG4 prevents access to SEG4 from its physical parent SEG3, but does not prevent access from SEG4 to SEG3.

DELETE BYTE DEFINITION

Segment Prefix -- Delete Byte

BIT	INTERPRETATION
0	segment deleted (HISAM)
1	DB record deleted (HISAM)
2	-
3	-
4	-
5	segment deleted/physical path (PD bit)
6	segment deleted/logical path (LD bit)
7	-

The logical delete bit is only meaningful for logical child segments and their logical parents. The PD and LD bits are set or assumed set as follows:

- If a segment is physically deleted (prevent further access from its physical parent), then delete processing scans downward from that segment through its dependents, turns upward and either releases each segment's DASD space or sets the PD bit. HISAM is an exception -- the delete bit is set in the segment specified by the DLET call and processing terminates.
- If the PD bit is set in a logical parent, then the LD bit is set in all logical children that can be reached from that logical parent.
- In physical pairing when the PD bit is set in the physical child of a pair of logical children, the LD bit is set in its pair.
- When a virtually paired logical child segment is logically deleted (prevent further access from its logical parent), the LD bit is set in the logical child. If physical pairing, the LD bit is set in the logical child and the PD bit is set in its pair (a physical child of the logical parent).
- The LD bit is assumed to be set in all logical children of unidirectional logical relationships.
- The LD bit is assumed set in a logical parent when the PD bit is set in all of its logical children. If physical pairing, the PD bit must be set in both paired logical children.

The Delete Call

A DL/I delete call may be issued against a segment defined in either a physical or logical DBD. The call can be issued against either a physical segment or a concatenation.

A delete call issued against a concatenated segment is a request for the deletion of the logical child along the accessed path.

If a concatenated segment or a logical child is accessed from its logical parent, then the DLET call is a request fcr logical deletion. In all cther cases, a DLET call is a request for physical deletion.

Physical deletion of a segment propagates logical deletion request to its logical children and propagates physical deletion request to its physical children and to any index pointer segments for which it is the source segment.

Delete sensitivity must be specified in the PCB for each segment against which a DLET call may be issued, but need not be specified for the physical dependents of those segments.

Delete operations are not affected by KEY/DATA sensitivity as specified in either the PCB or logical DBD.

Status Codes

'DX' - A delete rule is violated

'DA' - Key changed in the I/O area

DASD SPACE RELEASE

The delete call is not a request for DASD space release. Depending on the data base organization, DASD space may or may not be reused when it is released. DASD space for a segment is released when the following conditions are met:

- Space has been released for all physical dependents of the segment.
- The segment is physically deleted (PD bit set cr being set).
- If the segment is a logical child or a logical parent, then it must be physically and logically deleted (PD bit set/being set, and LD bit set/assumed set).
- If the segment is a dependent of a logical child (variable intersection data) and the DLET call was issued against a physical parent of the logical child, then the logical child must be both physically and logically deleted.
- If the segment is a primary index pointer segment, the space has been released for its target segment.

DELETE RULES

Logical Parent

1. PHYSICAL: The logical parent must be previously logically deleted before a DLET call is effective against the segment or any of its physical parents. Otherwise the call results in a 'DX' status code and no segments are deleted.

   However, if a delete request is made against the segment as a result of propagation across a logical relationship, then the PHYSICAL rule acts like the following LOGICAL rule.

2. LOGICAL: Either physical or logical deletion can occur first. All logical children are logically deleted. The logical parent remains accessible from its logical children.

3. VIRTUAL: A logical parent will be physically deleted:

   a. Explicitly when deleted by a DLET call. Logical children are logically deleted. The logical parent remains accessible from its logical children.

   b. Implicitly when it is logically deleted. When the last logical child is physically deleted, as the result of a DLET call, the logical parent will be logically deleted and physically deleted. If physical pairing, the physical child paired to the last logical child must also be physically deleted before the logical parent is implicity deleted.

Physical Parent (Virtual Pairing Only)

1. PHYSICAL/LOGICAL/VIRTUAL: Meaningless.

2. BIDIRECTIONAL VIRTUAL: When all physical child segments of a physical parent which are virtually paired logical children have been logically deleted, the physical parent is automatically deleted.

Logical Child

1. PHYSICAL: The logical child segment must be logically deleted first and physically deleted second. If physical deletion is attempted first, the DLET call issued against the segment or any of its physical parents results in a 'DX' status code and no segments are deleted. If a delete request is made against the segment as a result of propagation across a logical relationship, or if the segment is one of a physically paired set, then the rule acts like the following LOGICAL rule.

2. LOGICAL: Deletion of a logical child is effective for the path for which the delete was requested. Physical and logical deletion of the logical child can be performed in any order.

   The logical child and any physical dependents remain accessible from the non-deleted path.

3. VIRTUAL: A logical child is both logically and physically deleted when it is deleted through either its logical or physical path (setting either the PD or LD bits, sets both). If this rule is coded on only one logical child segment of a physically paired set, it acts like the LOGICAL rule.

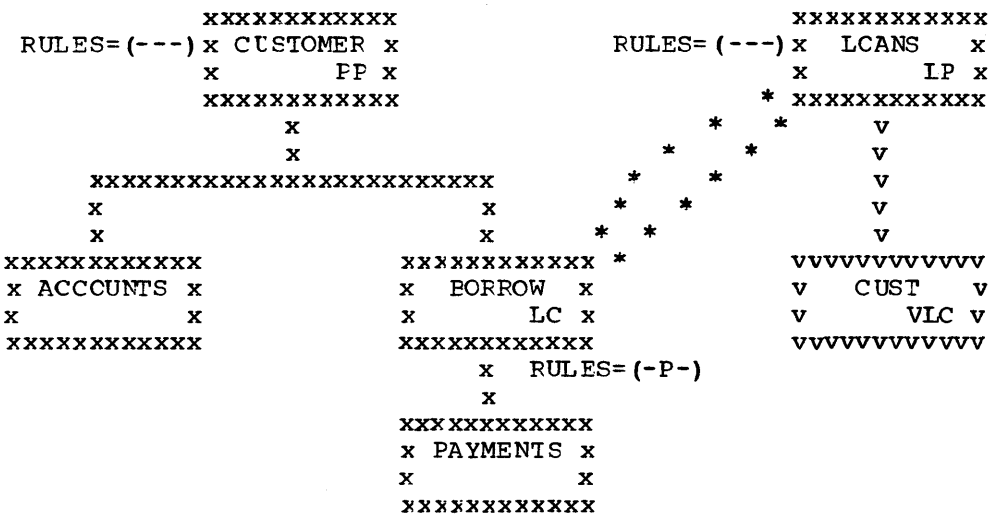
For logical children involved in unidirectional logical relationships, the meaning of all three rules are the same, so any of the three rules can be specified.

EXAMPLES

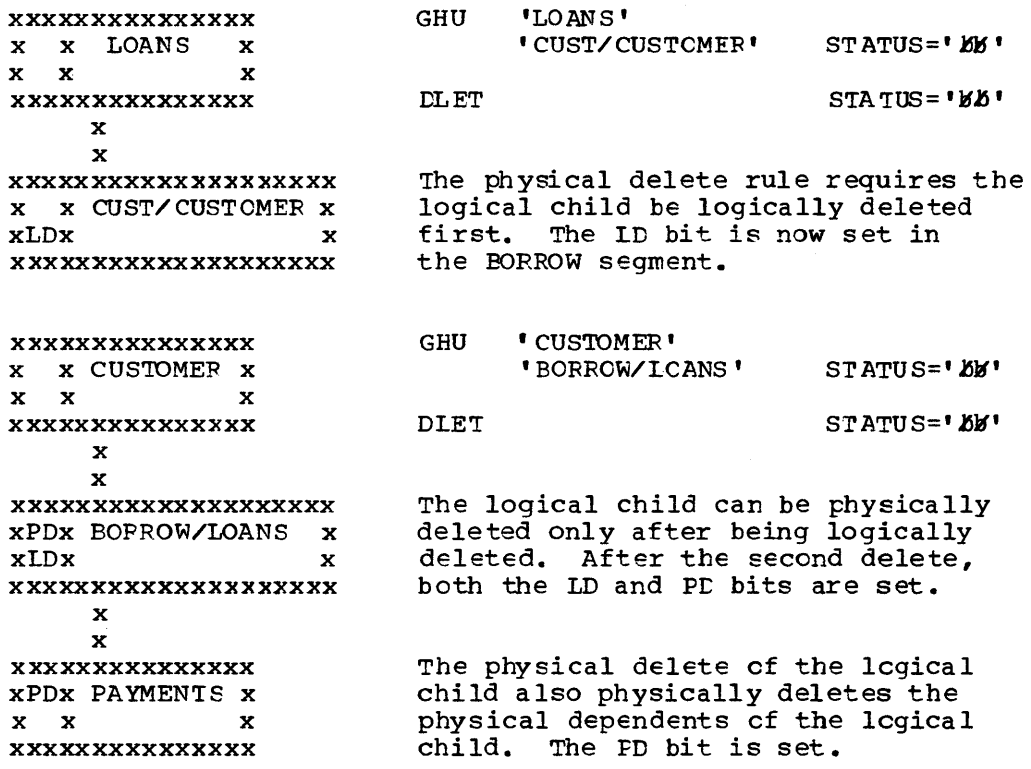
The following examples illustrate the use of the delete rules individually for each of the segment types that the rule can be coded for (logical children, and their logical and physical parents).

Only the rule pertinent to the examples are shown in each figure. The explanation applies to the specific example.

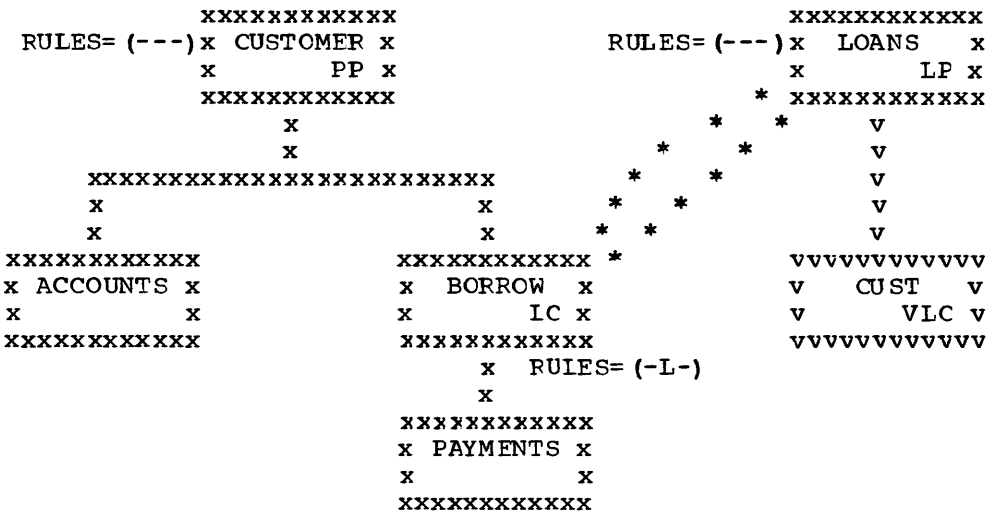
Logical Child, Virtual Pairing -- Physical Delete Rule Example



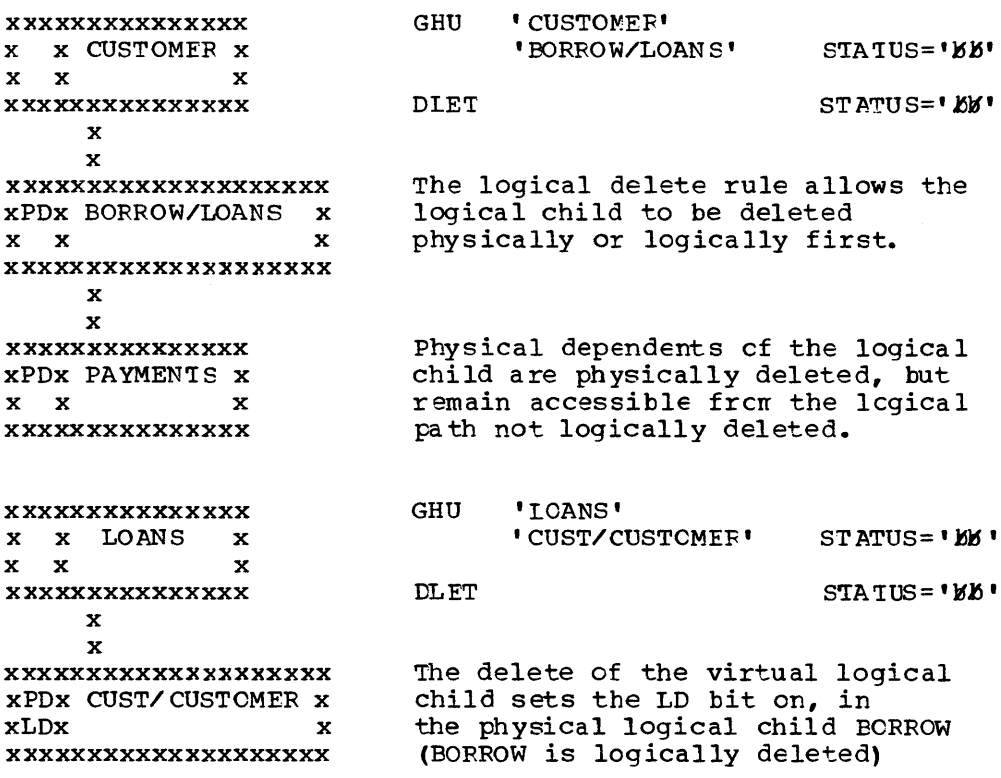
To Delete the Logical Child



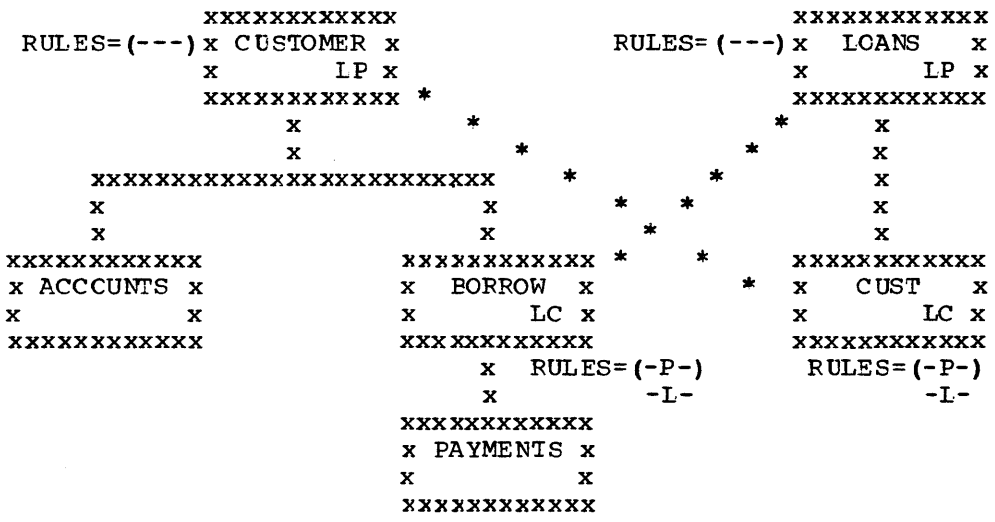
Logical Child, Virtual Pairing -- Logical Delete Rule Example



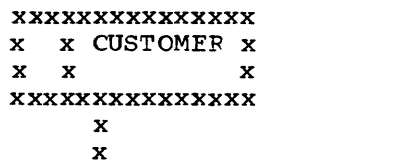
To Delete the Logical Child



Logical Child, Physical Pairing -- Physical/Logical Delete Rule Example

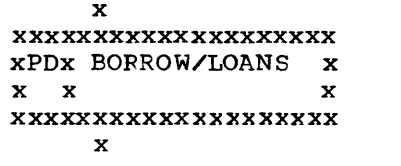


To Delete the Paired Logical Children

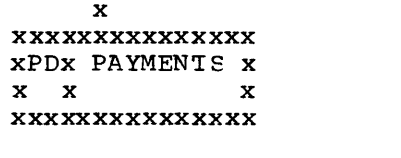


GHU 'CUSTOMER'
'BORROW/LOANS' STATUS='bb'

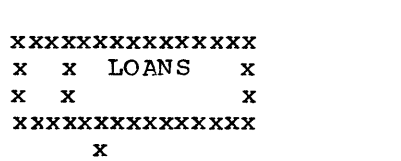
DLET STATUS='bb'



With the physical or logical delete rule, each logical child must be deleted from its physical path.

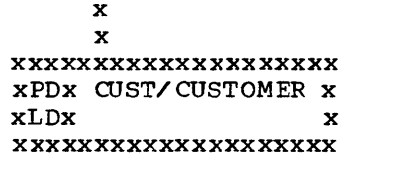


Physical dependents of the logical child are physically deleted, but remain accessible from the paired logical child not deleted.



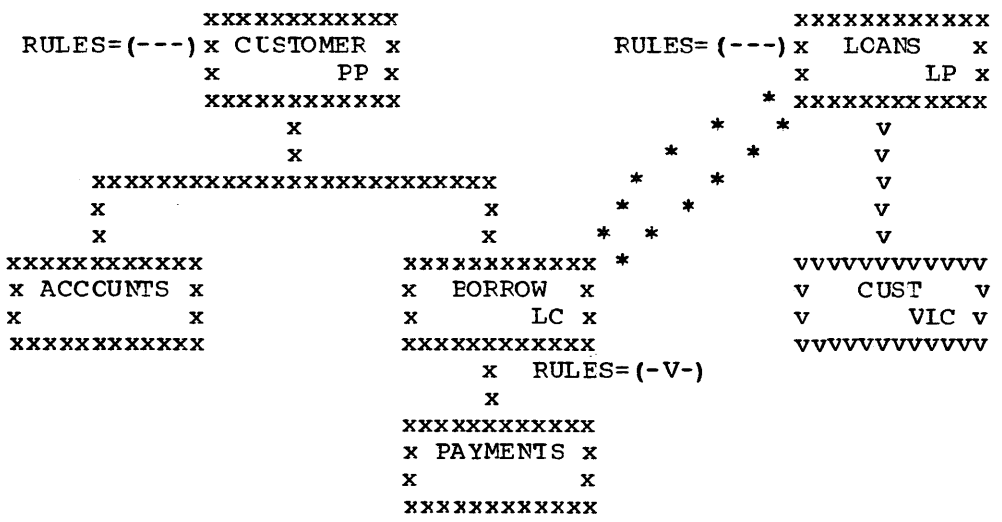
GHU 'LCANS'
'CUST/CUSTCMER' STATUS='bb'

DLET STATUS='bb'



Physically deleting BORROW set the LD bit in CUST. Physically deleting CUST will set the LD bit in the BORROW segment.

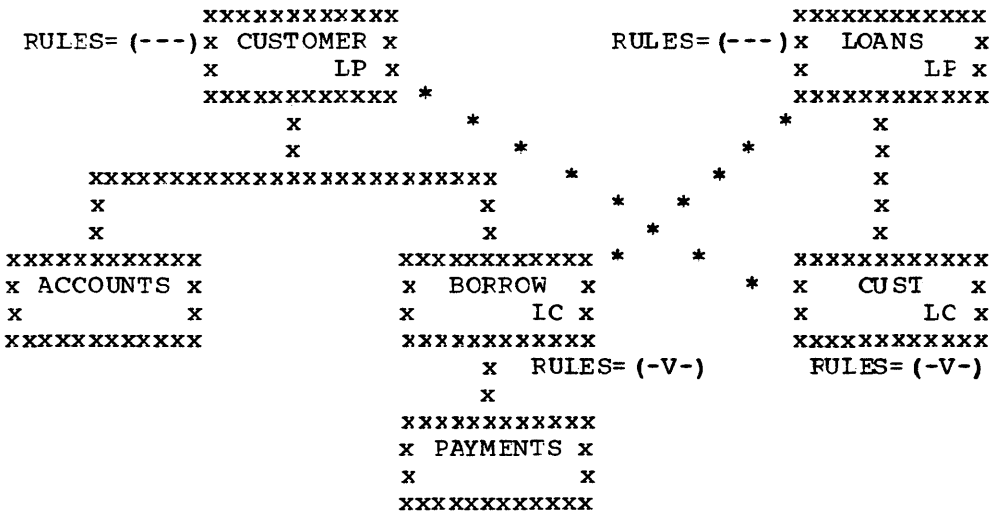
Logical Child, Virtual Pairing -- Virtual Delete Rule Example



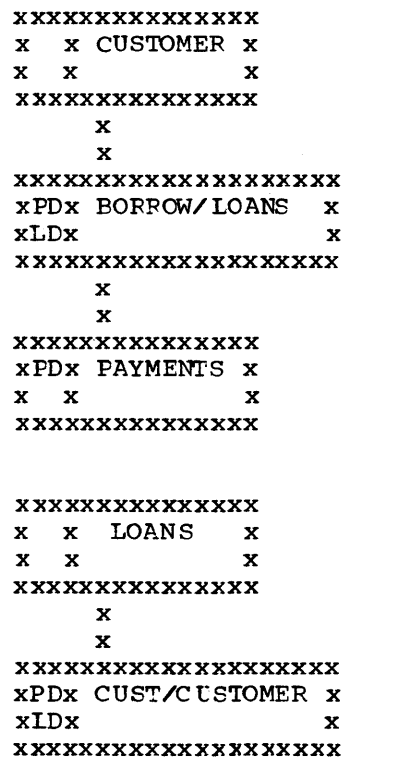
To Delete the Logical Child

```
xxxxxxxxxxxxxxx          GHU   'CUSTOMER'
x  x CUSTOMER x                '8ORRCW/LCANS'     STATUS='ØØ'
x  x          x
xxxxxxxxxxxxxxx          DLET                     STATUS='ØØ'
     x
     x
xxxxxxxxxxxxxxxxxxx      The virtual delete rule allows the
xPDx BORROW/LOANS  x     logical child tc be deleted
xLDx               x     physically and logically. Deleting
xxxxxxxxxxxxxxxxxxx      either path, deletes both paths.
     x
     x
xxxxxxxxxxxxxxx          Physical dependents of the logical
xPDx PAYMENTS x          child are physically deleted.
x  x          x
xxxxxxxxxxxxxxx


xxxxxxxxxxxxxxx          GHU   'LOANS'
x  x  LOANS   x                'CUST/CUSTCMER'    STATUS='GE'
x  x          x
xxxxxxxxxxxxxxx
     x
     x
xxxxxxxxxxxxxxxxxxx      The previous physical delete,
xPDx CUST/CUSTOMER x     deleted both paths, because the
xLDx               x     delete rule is virtual.  Deleting
xxxxxxxxxxxxxxxxxxx      either path, deletes both.
```

Logical Child, Physical Pairing -- Virtual Delete Rule Example



To Delete the Paired Logical Children

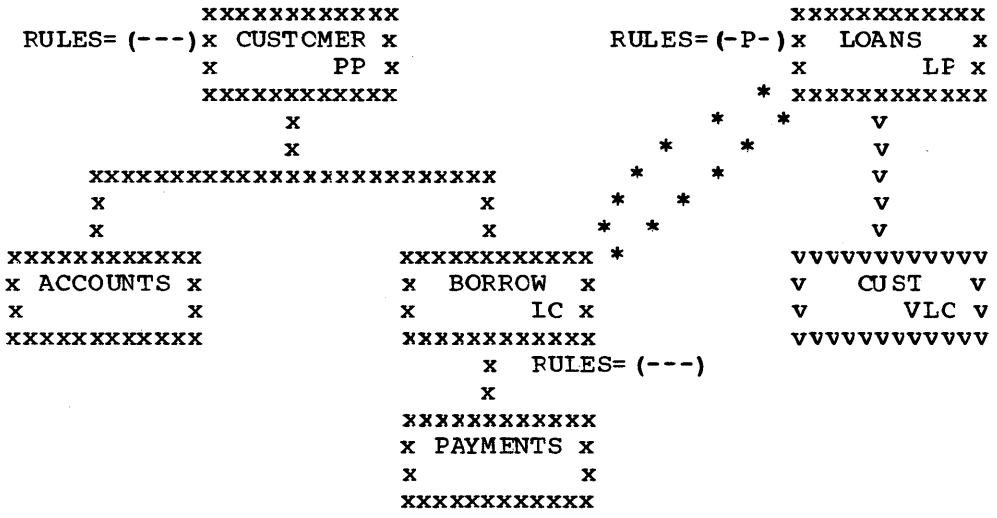


GHU 'CUSTOMER'
'BORROW/LCANS' STATUS='bb'

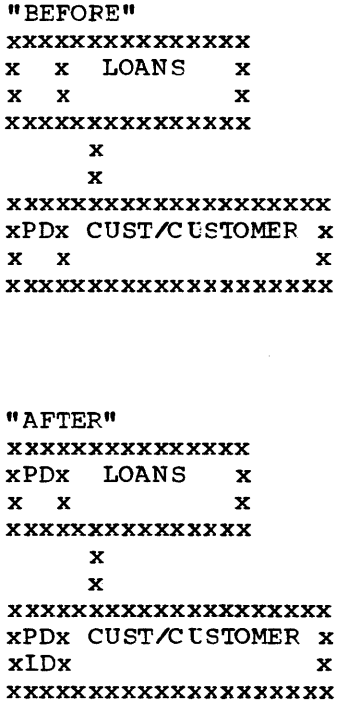
DLET STATUS='bb'

GHU 'LOANS'
'CUST/CUSTOMER' STATUS='GE'

Logical Parent, Virtual Pairing -- Physical Delete Rule Example



To Delete the Logical Parent



GHU 'LOANS' STATUS='bb'

DLET STATUS='bb'

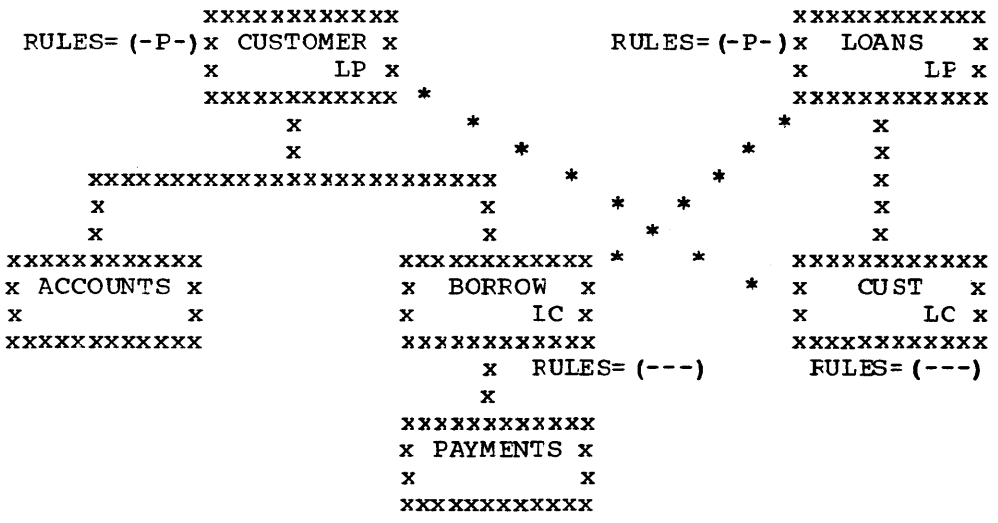
The physical delete rule requires that all logical children be previously physically deleted.

Physical dependents of the logical parent are physically deleted.

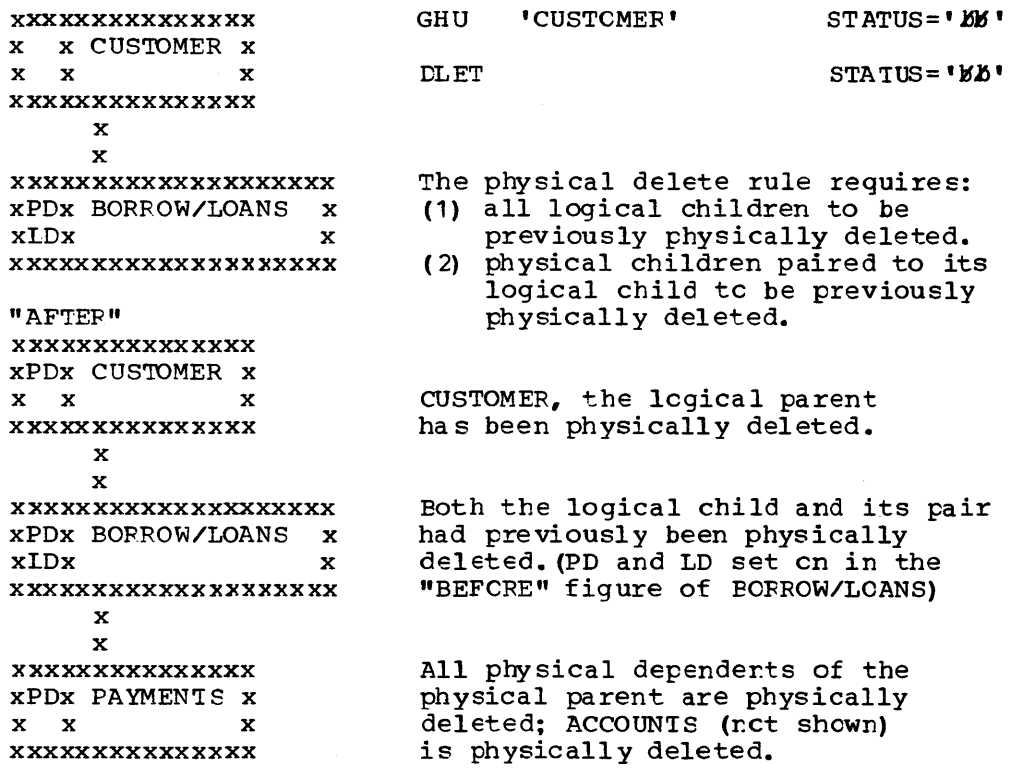
The DLET status code will be 'DX' if all of its logical children were not previously physically deleted.

All logical children are logically deleted. LD bit is set in the physical logical child BORROW.

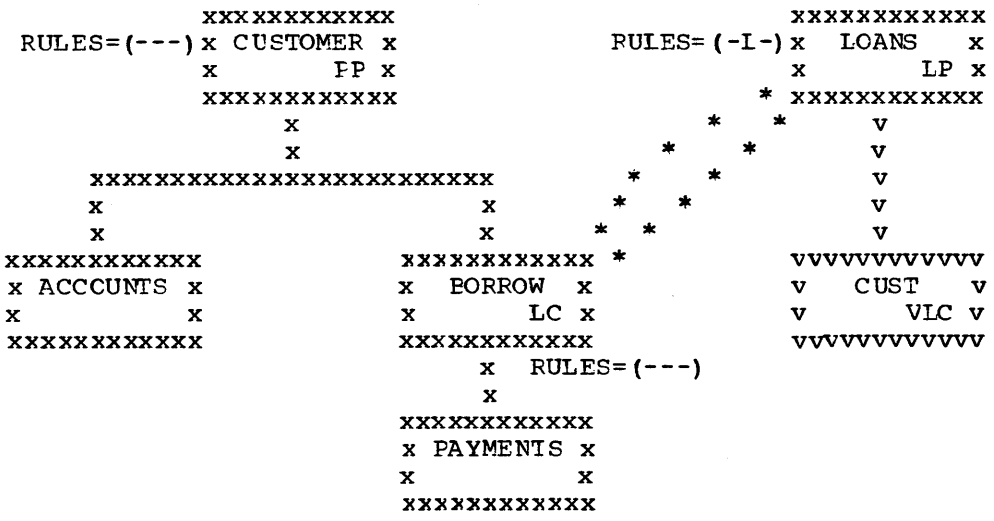
Logical Parent, Physical Pairing -- Physical Delete Rule Example



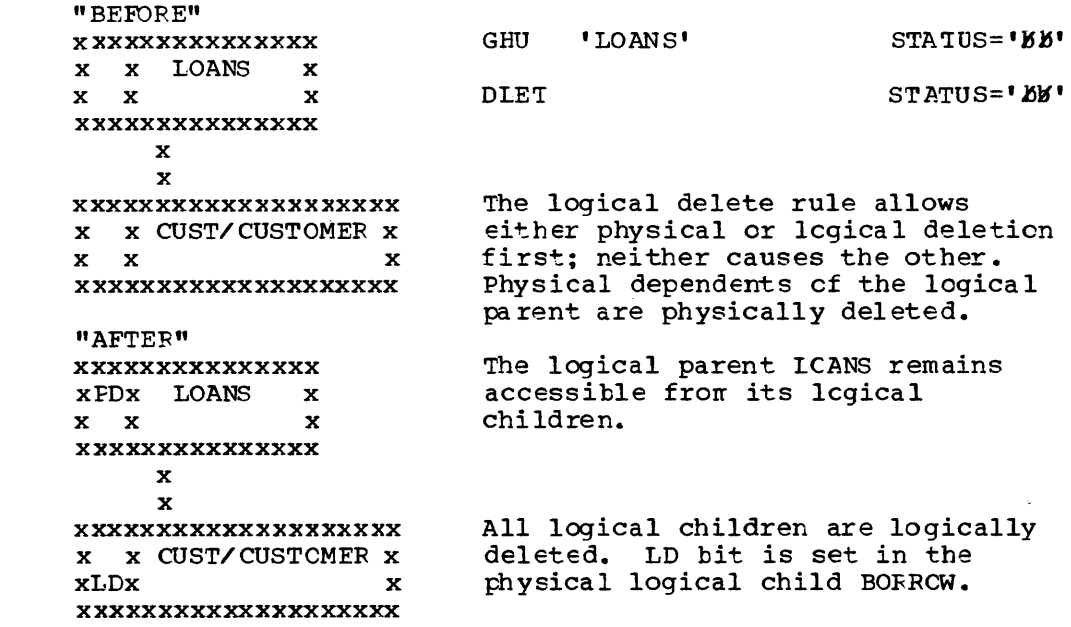
To Delete Either of the Logical Parents



Logical Parent, Virtual Pairing -- Logical Delete Rule Example

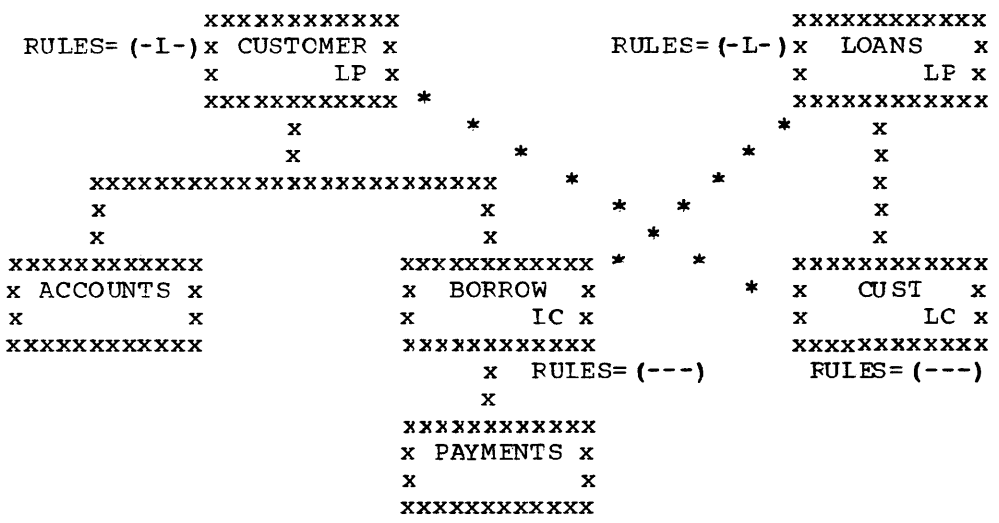


To Delete the Logical Parent

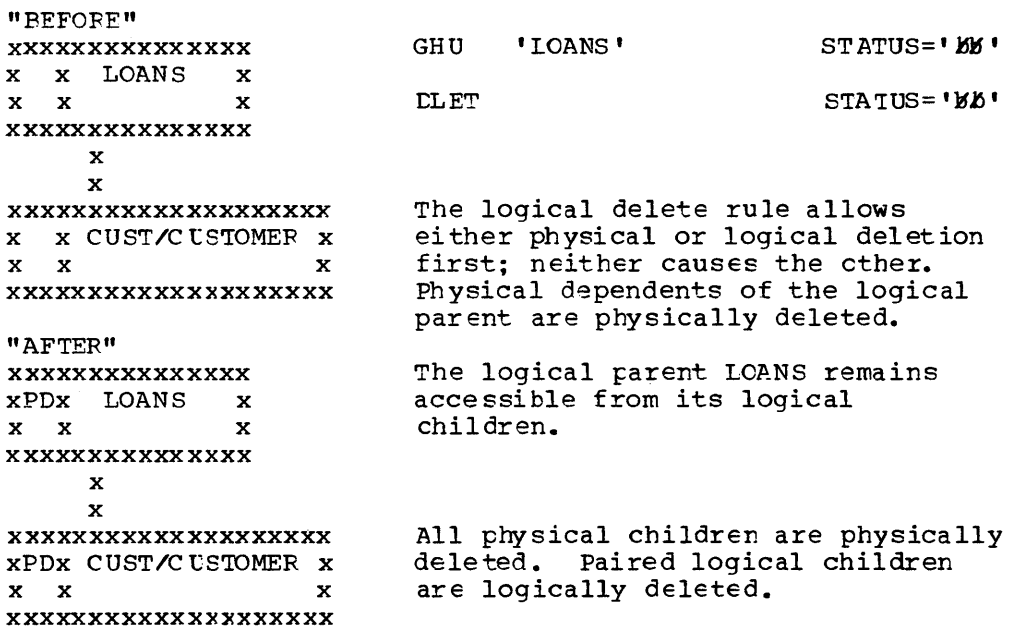


The above processing and results would be the same if the logical parent LOANS delete rule were virtual instead of logical. An additional example to explain the virtual delete rule follows.

Logical Parent, Physical Pairing -- Logical Delete Rule Example

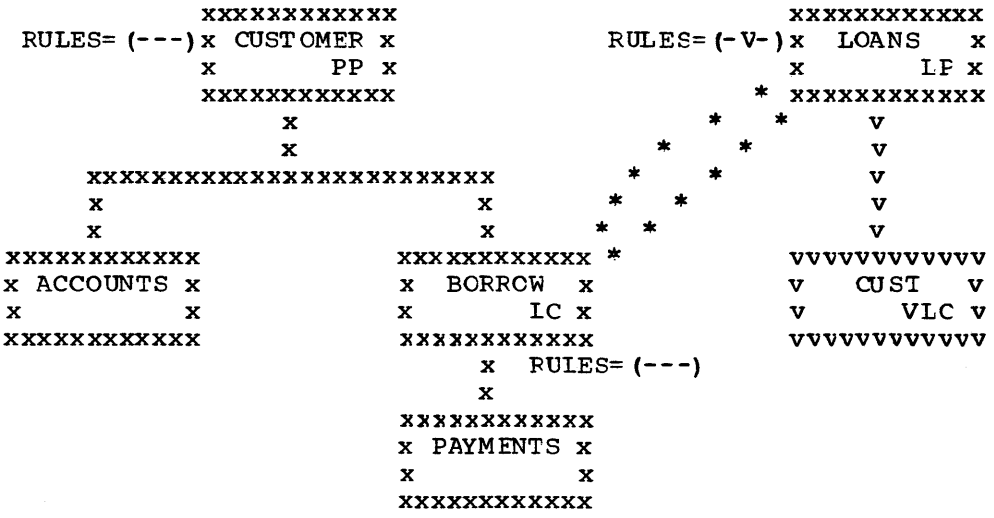


To Delete Either of the Logical Parents

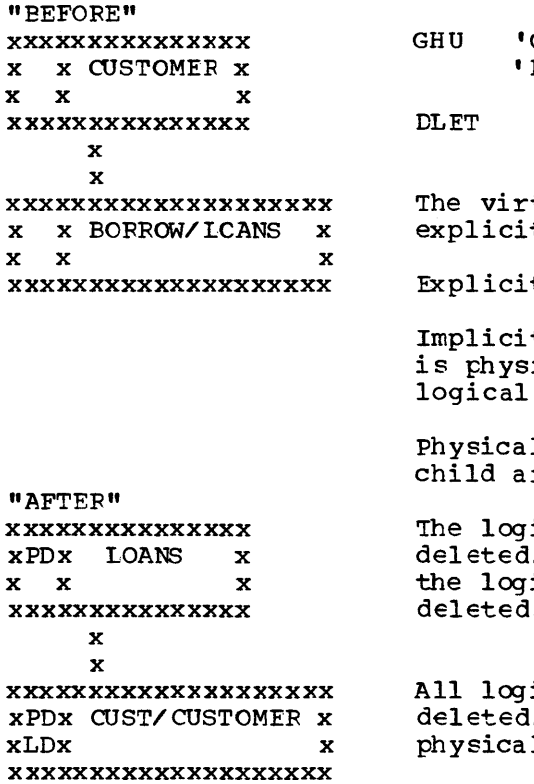


The above processing and results would be the same if the logical parent LOANS delete rule were virtual instead of logical. An additional example to explain the virtual delete rule follows.

Logical Parent, Virtual Pairing -- Virtual Delete Rule Example



Deleting Last Logical Child Deletes Logical Parent



GHU 'CUSTOMER' 'BORROW/LOANS' STATUS='bb'

DLET STATUS='bb'

The virtual delete rule allows explicit and implicit deletion.

Explicit is same as logical rule.

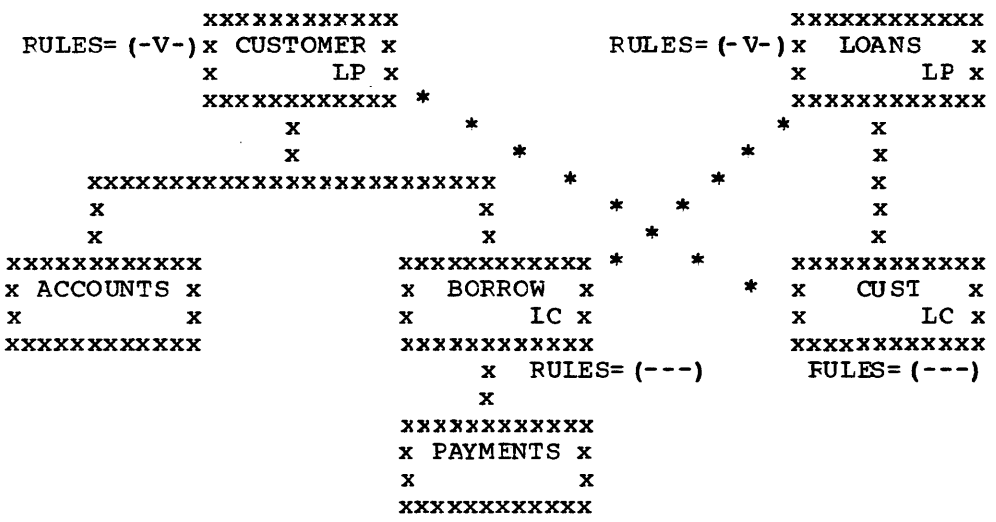
Implicit means the logical parent is physically deleted when the last logical child is physically deleted.

Physical dependents of the logical child are physically deleted.

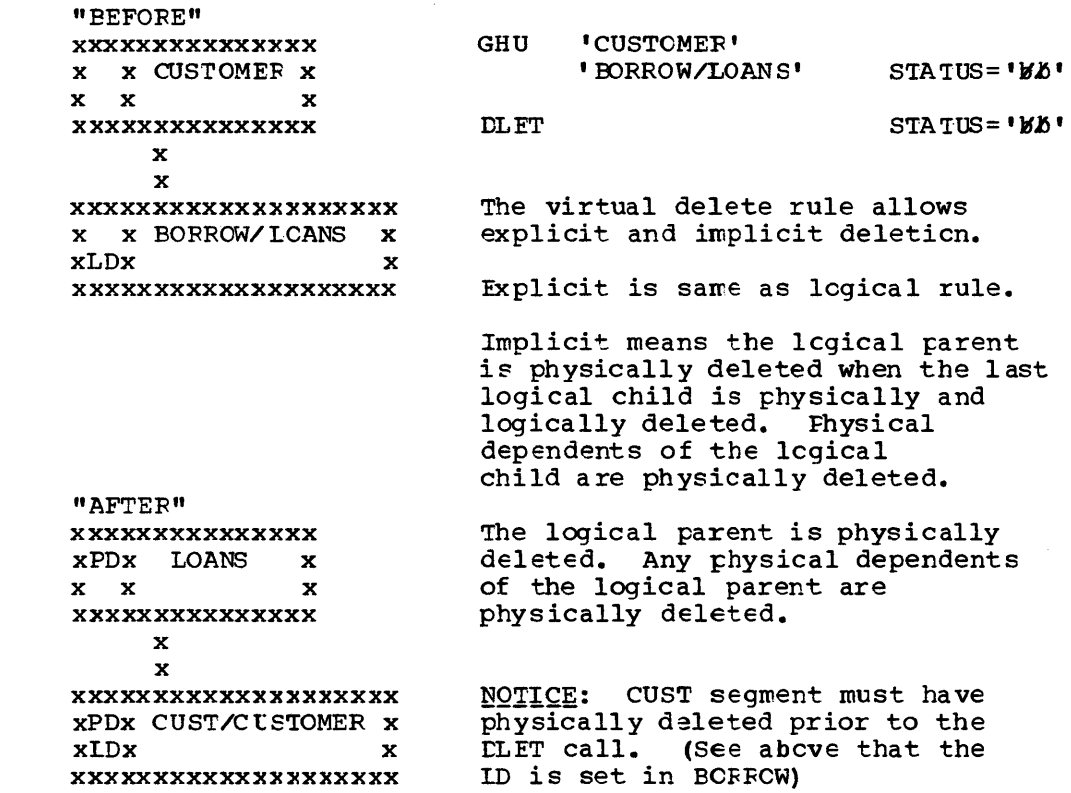
The logical parent is physically deleted. Physical dependents of the logical parent are physically deleted.

All logical children are logically deleted. LD bit is set in the physical logical child BORROW.

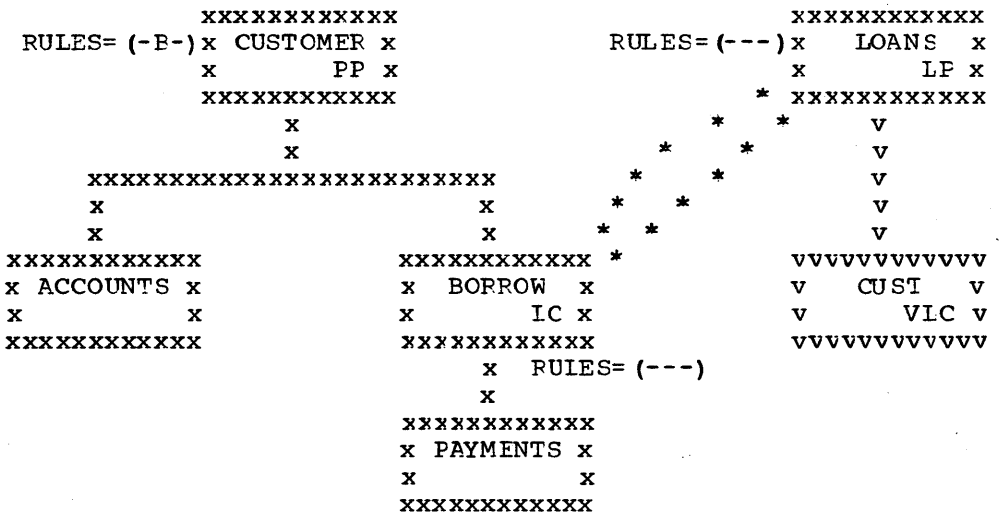
Logical Parent, Physical Pairing -- Virtual Delete Rule Example



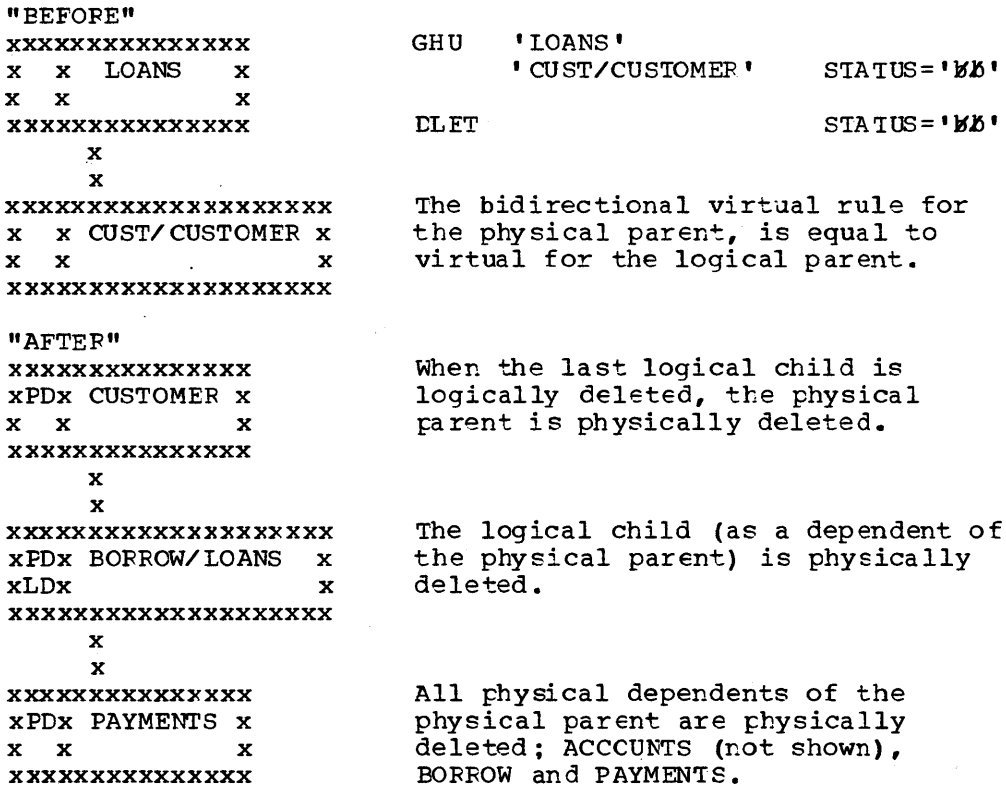
Deleting Last Logical Child Deletes Logical Parent



Physical Parent, Virtual Pairing -- Bidirectional Virtual Example



Deleting Last Logical Child Deletes Physical Parent



Accessibility of Deleted Segments

A physically deleted segment remains accessible under the following circumstances:

1. A physical dependent of the deleted segment is a logical parent which is accessible from its logical children.

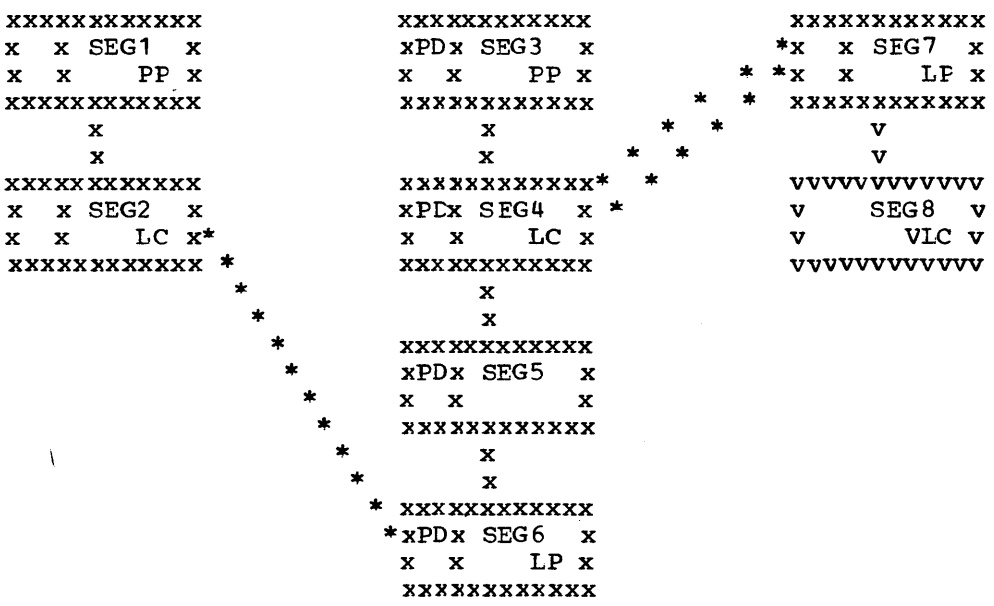
2. A physical dependent of the deleted segment is a logical child which is accessible from its logical parent.

3. A physical parent of the deleted segment is a logical child which is accessible from its logical parent. The deleted segment is this case is variable intersection data of a bidirectional logical relationship.

A logically deleted logical child cannot be accessed from its logical parent.

Neither physical nor logical deletion prevents access to a segment from its physical or logical children. Since logical relationships provides for inversion of the physical structure, a segment may be either physically or logically deleted or both and still be accessible from a dependent segment, because of an active logical relationship. A physically deleted root segment can be accessed when it is, defined as a dependent segment in a logical DBD. The logical DBD defines the inversion of the physical DBD.

1. EXAMPLE OF DELETED SEGMENTS ACCESSIBILITY: When the physical dependent of a deleted segment is a logical parent with logical children not physically deleted, the logical parent and its physical parents are accessible from those logical children.

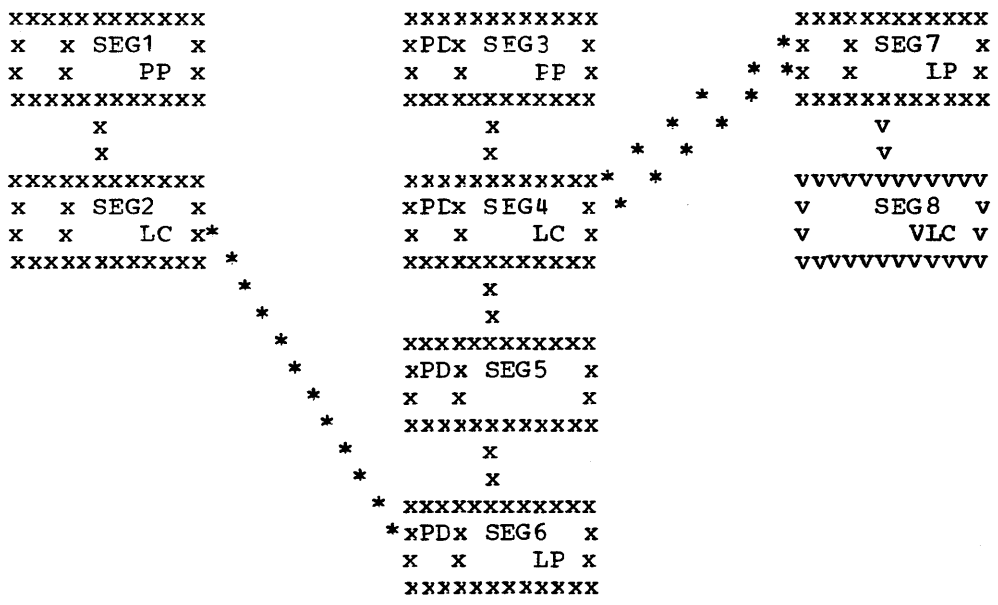


The above physical structures show that SEG3, SEG4, SEG5, and SEG6 have been physically deleted. Probably by issuing a DLET call for SEG3. This resulted in all of SEG3's dependents being physically deleted. (SEG6's delete rule ≠ PHYSICAL or a 'DX' status code would be the result).

SEG3, SEG4, SEG5, and SEG6 remain accessible from SEG2, the logical child of SEG6, because SEG2 is not physically deleted.

However, physical dependents of SEG6 cannot be accessible, and their DASD space is released unless an active logical relationship prohibits such release.

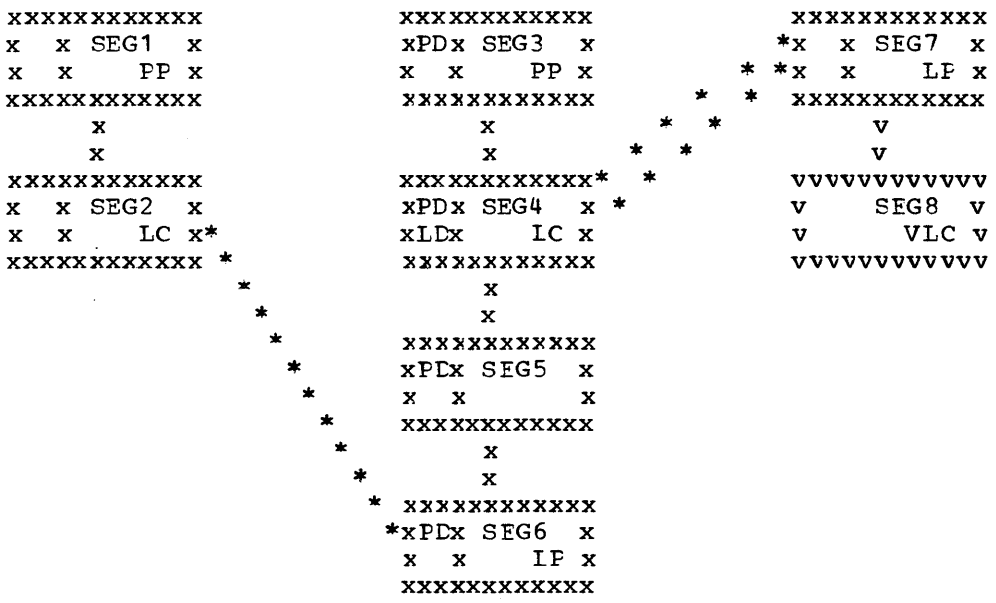
2. EXAMPLE OF DELETED SEGMENTS ACCESSIBILITY: When the physical dependent of a deleted segment is a logical child whose logical parent is not physically deleted, the logical child, its physical parents and its physical dependents are accessible from the logical parent.



The above physical structures show that SEG3, SEG4, SEG5, and SEG6 have been physically deleted.

The logical child segment SEG4 remains accessible from its logical parent SEG7 (note that SEG7 is not physically deleted). Also accessible are segments SEG5 and SEG6, which are variable intersection data. The physical parent of the logical child (SEG3) is likewise accessible from the logical child (SEG4).

3. EXAMPLE OF DELETED SEGMENTS ACCESSIBILITY: A physically and logically deleted logical child can be accessed from its physical dependents.

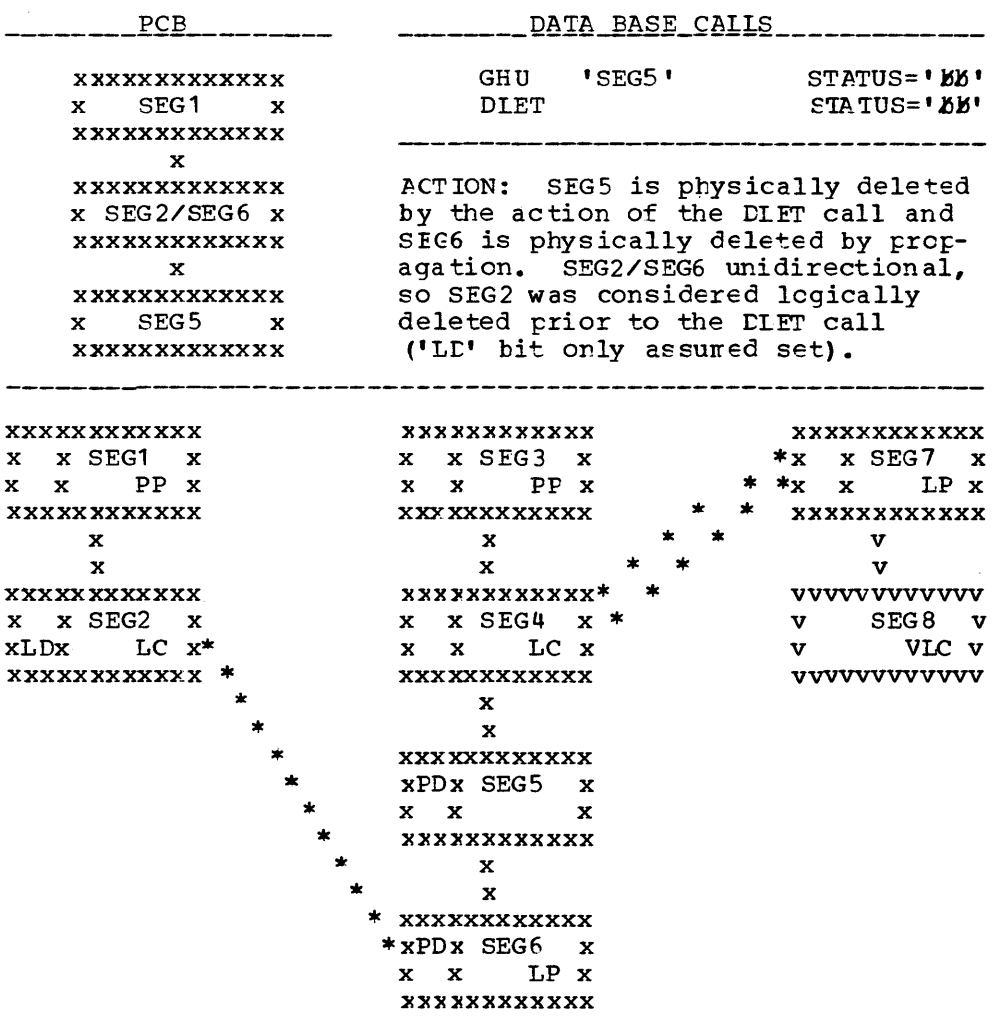


The above physical structures show that the logical child SEG4 is both physically and logically deleted.

From a previous example (number 1) we know that SEG6 (a logical parent) is accessible from SEG2, if that segment (its logical child) is not physically deleted. Likewise we know that once we have accessed SEG6, its physical parents (SEG5, SEG4, SEG3) are accessable. It does not matter that the logical child is logically deleted (which is the only difference in this example from example 1).

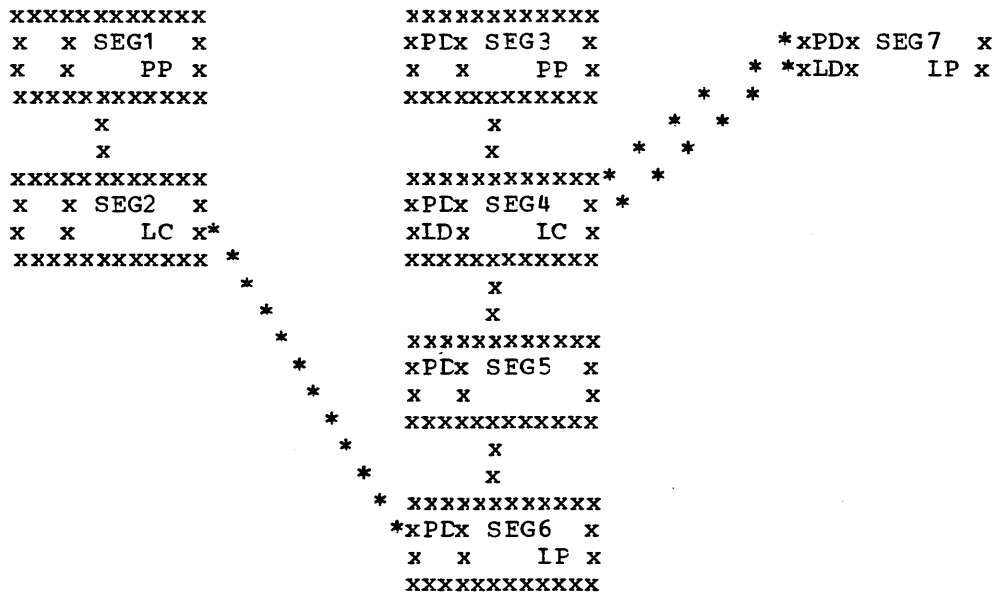
The third path cannot be blocked because a delete bit for this path does not exist. Thus the logical child SEG4 is accessible from its dependents regardless of its being physically and logically deleted.

4. EXAMPLE OF DELETED SEGMENTS ACCESSIBILITY: When a segment accessed by its THIRD path is deleted, it is physically deleted in its physical data base, but remains accessible from its THIRD path.



The results are interesting. SEG5 is unaccessible when accessed by its physical parent path (from SEG4) unless SEG4 were accessed by its logical parent SEG7 (SEG5 & SEG6 are accessible as variable intersection data). SEG5 is still accessible from its third path (from SEG6) because SEG6 is still accessible from its logical child. Thus a segment can be physically deleted by an application program and still be accessible to that application program, using the same PCB used to delete the segment.

5. 801 ABNORMAL TERMINATION POSSIBILITY If a logical parent is physically and logically deleted, it's DASD space will be released. For this to occur, all of its logical children must be physically and logically deleted. However, these logical children may not be DASD space released because of physical dependents with active logical relationships. Accessing such a logical child from its physical dependents may result in an 801 abnormal termination if the LPCK is not stored in the logical child or if the concatenation definition is data sensitive to the logical parent.



The logical parent SEG7 has been physically and logically deleted (the LD bit is never really set, but only assumed set. It is shown for the purpose of illustration). All of the logical children of the logical parent have also been physically and logically deleted. However, the logical parent has had its segment space released, whereas the logical child (SEG4) still exist due to an active logical relationship that precludes releasing its space.

If an application program accesses SEG4 from its dependents (SEG1 to SEG2/SEG6 to SEG5) IMS must build the logical parents concatenated key if that key is not stored in the logical child. When IMS/VS attempts to access the logical parent SEG7, the results will be an 801 abnormal termination. 801 says that IMS/VS followed a pointer that did not lead to the segment expected.

AVOIDING POSSIBLE 801 ABNORMAL TERMINATION

We must avoid creating a physically deleted logical child which can be accessed from below in the physical structure (its third path). A logical child can be accessed from below if any of its physical dependents are accessible through logical paths.

First Solution

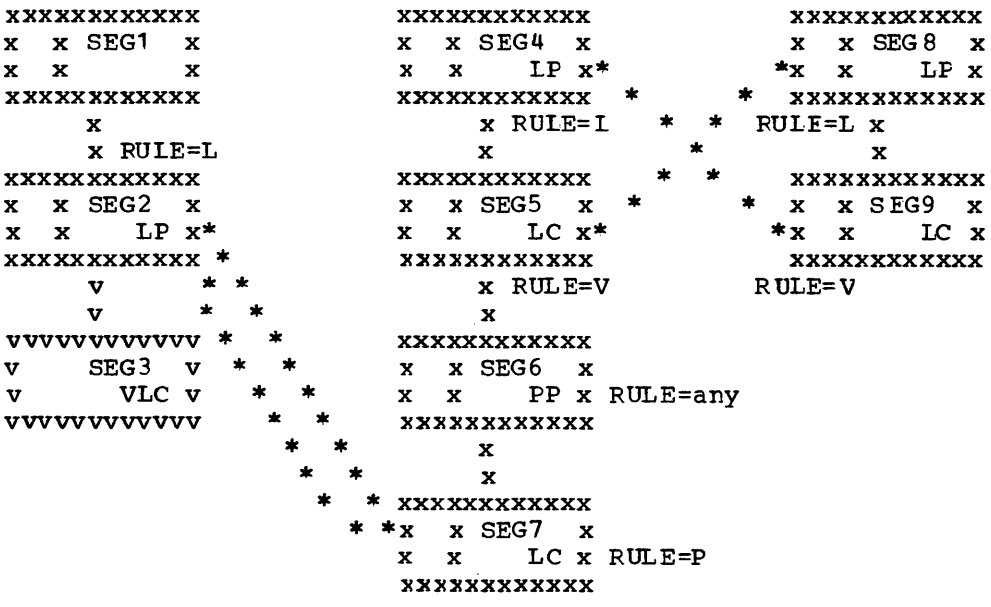
One solution is to require the logical paths to dependents to be broken prior to physically deleting the logical child. This can be done by using a PHYSICAL rule for the dependents as long as no physical deletes are allow to propagate into the data base. Therefore no VIRTUAL rules on logical children can be allowed at or above THE LOGICAL CHILD, since with the V rule a propagated logical delete causes a physical delete without a P rule violation check (see DETECTION OF PHYSICAL DELETE RULE VIOLATION). The LOGICAL rule will also cause propagation if the PD bit is already, but the dependents PHYSICAL rule will prevent that case. Similarly, no VIRTUAL rule can be allowed on any logical parent above the logical child, since the logical delete condition would cause the physical delete.

Second Solution

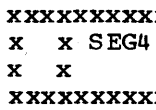
A second solution is to break the logical path whenever the logical child is physically deleted. This can be accomplished for subordinate logical child segments with the VIRTUAL delete rule. Subordinate logical parent segments need to have bidirectional logical children with the VIRTUAL rule (must be able to reach the logical children) or physcially paired logical children with the VIRTUAL rule. This solution will not work with subordinate logical parent's pointed to by unidirectional logical children.

DETECTION OF PHYSICAL DELETE RULE VIOLATION

The delete routine scans the physical structure containing the requested segment to be deleted to determine if any segment in the physical structure has the physical delete rule and whether that rule is violated.



PCB



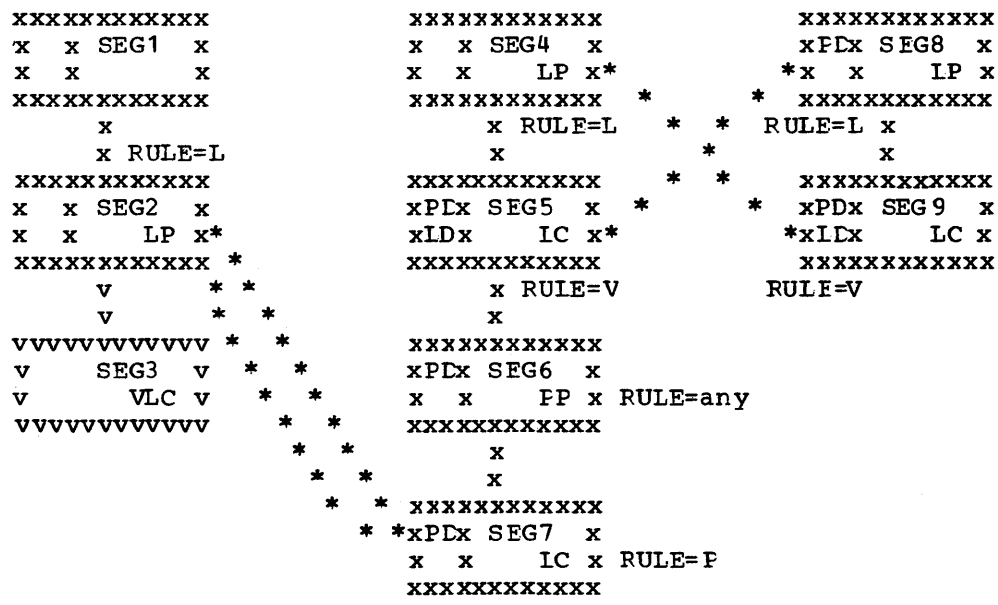
DATA BASE CALLS

GHU	'SEG4'	STATUS='bb'
DLET		STATUS='DX'

SEG7 (logical child of SEG3) has the physical delete rule and it has not been logically deleted (the ID bit has not been set) so the physical rule is violated and a 'DX' status code is returned to the application program and no segments are deleted.

PHYSICAL DELETE RULE TREATED AS LOGICAL

After the delete routine determines that neither the segment specified in the DLET call nor any physical dependent of that segment in the physical structure has the physical delete rule, any physical rule encountered later (logical deletion propagated to logical child or logical parent causing physical deletion (V rule) in another data base) is treated as LOGICAL.



PCB	DATA BASE CALLS		
SEG8	GHU	'SEG8'	STATUS='bb'
	DLET		STATUS='bb'

SEG8 and SEG9 are both physically deleted, and SEG9 is logically deleted (V rule). SEG5 is physically and logically deleted because it is the physical pair to SEG9 (with physical pairing setting the LD bit in one set the PD bit in the other and vice verse). Physically deleting SEG5 causes propagation of the physical delete to SEG5's physical dependents, thus SEG6 and SEG7 are physically deleted. Notice that the physical delete of SEG7 is prevented if the physical deletion had started by issuing a DLET call for SEG4. But the physical rule of SEG7 is treated as logical in this case.

INSERTING PHYSICALLY AND/OR LOGICALLY DELETED SEGMENTS

When a segment is inserted, a replace operation will be performed (i.e., space will be reused) and existing dependents of that segment remain, provided:

- The segment to be inserted already exists (same segment type and same key field value for both the physical and logical sequencing), and
- The delete bit is set on for that segment along the path of insertion.

If the DB organization is HD, the logical twin chain will be established as required, and existing dependents of that segment will remain.

If the DB organization is HISAM, and the root segment is physically and logically deleted before the insert is attempted, then the first LRECL for that root in primary and secondary DSGs is reused and remaining LRECLs on any OSAM chain are dropped.

DELETE RULES SUMMARY

The DLET Call

A DLET call issued against a concatenated segment (SOURCE=DATA/DATA, DATA/KEY, KEY/DATA) is a DLET call against the logical child only.

A DLET call against a logical child which has been accessed from its logical parent is a request that the logical child be logically deleted.

In all other cases a DLET call issued against a segment is a request for that segment to be physically deleted.

Physical Deletion

A physically deleted segment cannot be accessed from its physical path with one exception -- if one of the physical parents of the physically deleted segment is a logical child segment which can be accessed from its logical parent, then the physically deleted segment is accessible from that logical child since the physical dependents of a logical child are "variable intersection data."

Logical Deletion

By definition, a logically deleted logical child cannot be accessed from its logical parent. Unidirectional logical child segments are assumed to be logically deleted.

By definition, a logical parent is considered logically deleted when physical deletion has occured for all of its logical children and for all of its physical children which are part of a physically paired set.

Access Paths

Neither physical nor logical deletion of a segment prevents access to the segment from its physical or logical children, or from the segment to its physical or logical parents. A physically deleted root segment can be accessed only from its physical or logical children.

Propagation of Delete

In bidirectional physical pairing, physical deletion of one of the pair of logical children causes logical deletion of its pair. Likewise, logical deletion of one causes physical deletion of the other.

Physical deletion of a segment propagates logical deletion requests to its bidirectional logical children and propagates physical deletion requests to its physical children and to any index pointer segments for which it is the source segment.

DELETE RULES

Further delete operations are governed by the following delete rules:

Logical Parent

1. PHYSICAL: If the segment is not already logically deleted, then a DLET call against the segment or any of its physical parents results in a 'DX' status code and no segments are deleted. If a request is made against the segment as a result of propagation across a logical relationship, then the rule acts like the LOGICAL rule.

2. LOGICAL: Either physical or logical deletion can occur first and neither causes the other.

3. VIRTUAL: Either physical or logical deletion can occur first. If the segment becomes logically deleted as the result of a DLET call, then it will be physically deleted also.

Physical Parent of a Virtually Paired Logical Child

1. PHYSICAL/LOGICAL/VIRTUAL: Meaningless.

2. BIDIRECTIONAL VIRTUAL: Whenever all physical children which are virtually paired logical children are logically deleted, the physical parent segment is physically deleted.

Logical Child

1. PHYSICAL: If the segment is not already logically deleted, then a DLET call requesting physical deletion of the segment or any of its physical parents results in a 'DX' status code and no segments are deleted. If a delete request is made against the segment as a result of propagation across a logical relationship, or if the segment is one of a physically paired set, then the rule acts like the LOGICAL rule.

2. LOGICAL: Either physical or logical deletion can occur first and neither causes the other.

3. VIRTUAL: Either physical or logical deletion can occur first and either causes the other. If this rule is used on only one segment of a physically paired set, it acts like the LOGICAL rule.

Space Release

Depending on the data base organization, DASD space may or may not be reused when it is released. DASD space for a segment is released when the following conditions are met:

- Space has been released for all physical dependents of the segment.
- The segment is physically deleted.
- If the segment is a logical child or a logical parent, then it must be physically and logically deleted.

- If the segment is a dependent of a logical child (variable intersection data) and the DLET call was issued against a physical parent of the logical child, then the logical child must be both physically and logically deleted.

- If the segment is a primary index pointer segment, the space must have been released for its target segment.

DEFINING A LOGICAL DATA BASE

To identify which segment types in one or more physical data bases are used in a logical data base, the segment types in the physical data bases are redefined in a logical data base through DBDGEN using SEGM statements. The NAME= operand of the SEGM statement is used to specify the name used for the segment type in the logical data base, and the SOURCE= operand is used to identify which segment type or types in physical data bases are represented by the name specified in the logical data base. On the SOURCE= operand, the user specifies the name of the segment type in a physical data base that is represented in the logical data base through the name specified on the NAME= operand. Also specified on the SOURCE= operand is the name of the physical data base that contains the segment type being defined in a logical data base. When defining a concatenated segment type in a logical data base, the names of both segment types that comprise the concatenated segment type, and the name of the physical data base that contains each segment type to be concatenated are specified on the SOURCE= operand.

As in the definition of a physical data base, the hierarchy of segment types in a logical data base is defined through use of the PARENT= operand of the SEGM statement, and through the order in which SEGM statements are presented as input to DBDGEN. The PARENT= operand is used to specify the physical parent segment type of each dependent segment type defined in the logical data base, and the order in which SEGM statements are arranged determines the left to right order of physical child segment types of each physical parent segment type.

A concatenated segment type is defined in a logical data base to enable use of a logical relationship. When a concatenated segment type is defined in a logical data base, the concatenated segment type enables access to the destination parent in the logical relationship. In addition, subject to rules for defining logical data bases, the concatenated segment type enables crossing a logical relationship to access segments that are in the hierarchic path of the destination parent in the physical data base of the destination parent.

Before the rules for defining logical data bases can be understood, crossing a logical relationship, and the first and additional logical relationships crossed in a hierarchic path of a logical data base must be understood.

Definition of Crossing a Logical Relationship

A logical relationship is considered crossed when it is used in a logical data base to access a segment that is a physical parent or a physical dependent of a destination parent in the destination parents physical data base. If a logical relationship is used in a logical data base to access a destination parent only, the logical relationship is considered not to be crossed. In Figure 4-34, DBD1 and DBD2 are two physical data bases with a logical relationship defined between them. DBD3 through DBD6 are the four logical data bases that can be defined as a result of the logical relationship between DBD1 and DBD2. If the structure shown for DBD3 is defined as a logical data base, no logical relationships are crossed since no physical parent or physical

dependent of a destination parent is included in the logical data base. If DBD4 through DBD6 were defined as logical data bases, a logical relationship is crossed in each case since each logical data base contains a physical parent or a physical dependent of the destination parent.

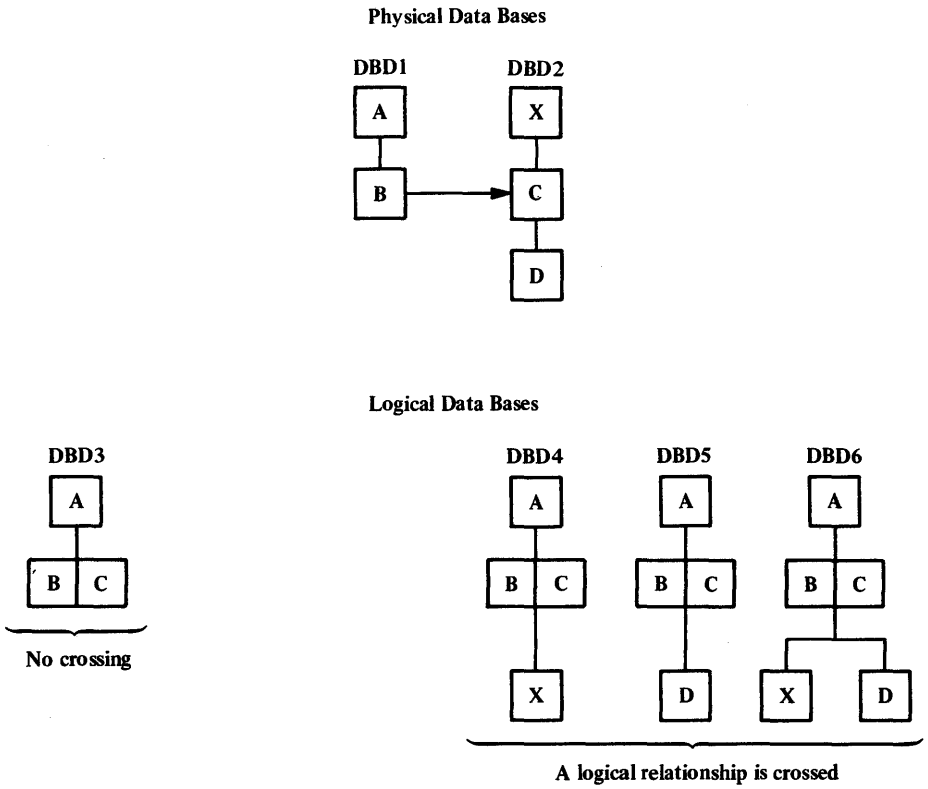


Figure 4-34. Definition of Crossing a Logical Relationship

Definition of First and Additional Logical Relationships Crossed

Multiple logical relationships can be crossed in a hierarchic path of a logical data base. Figure 4-35 shows three physical data bases (DBD1, DBD2 and DBD3) in which logical relationships have been defined. Also in the figure are two logical data bases (DBD4 and DBD5) that can be defined as a result of the logical relationships defined in the physical data bases. In DBD4, the two concatenated segment types (BF and DI) that have been defined enable access to all segment types in the hierarchic paths of their respective destination parents. If either logical relationship or both are crossed, each is considered to be the first logical relationship crossed in the hierarchic path of logical data base DBD4 since each concatenated segment type is reached by following the physical hierarchy of segment types in DBD1. In logical data base DBD5, an additional concatenated segment type (GI) has been defined that was not included in DBD4. The additional concatenated segment type GI in DBD5 enables access to segments in the hierarchic path of the destination parent if crossed. When the logical relationship enabled by the concatenated segment GI is crossed, this constitutes an additional logical relationship crossed in the hierarchic path of the logical data base since, from the root of the logical data base, the logical relationship enabled by the concatenated segment type BF must be crossed to enable access to the concatenated segment type GI.

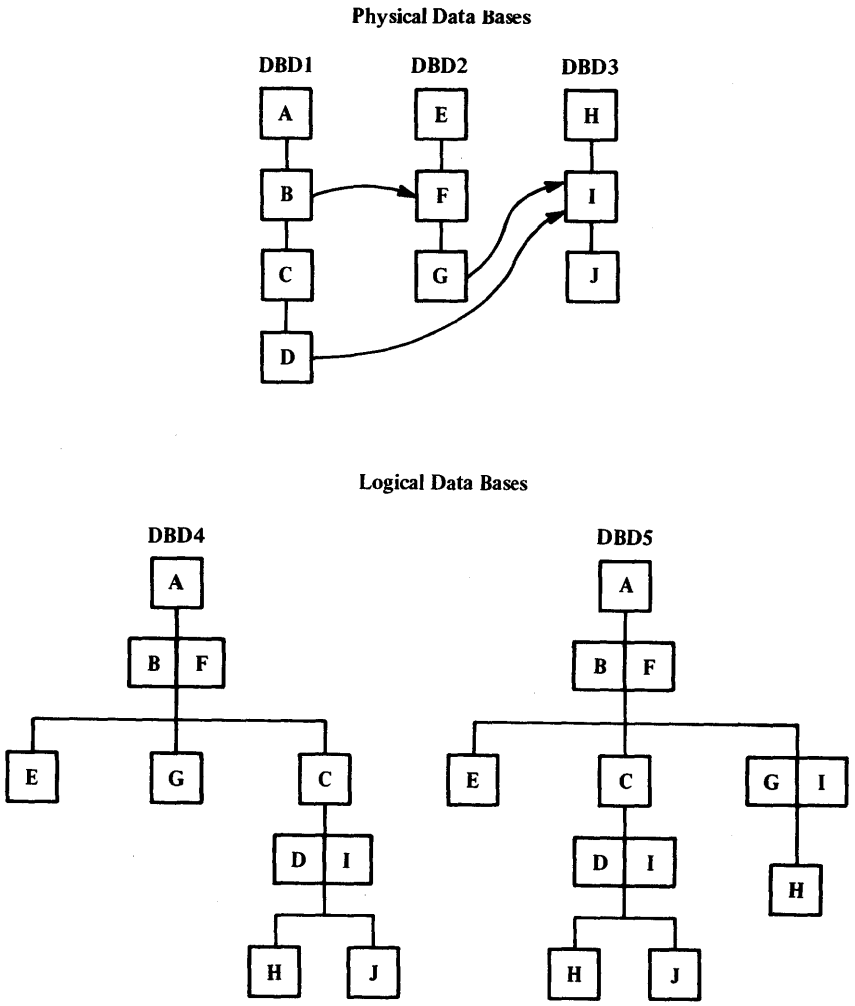


Figure 4-35. The First Logical Relationship Crossed in a Hierarchic Path of a Logical Data Base

When the first logical relationship is crossed in a hierarchic path of a logical data base, access tc all segment types in the hierarchic path of the destination parent is enabled as fcllcws:

- Parent segment types of the destination parent are included in the logical data base as dependents of the destination parent in reverse order as shown in Figure 4-36.
- Dependent segment types of the destination parent are included in the logical data base as dependents of the destination parent without change in their order as shown in Figure 4-36.

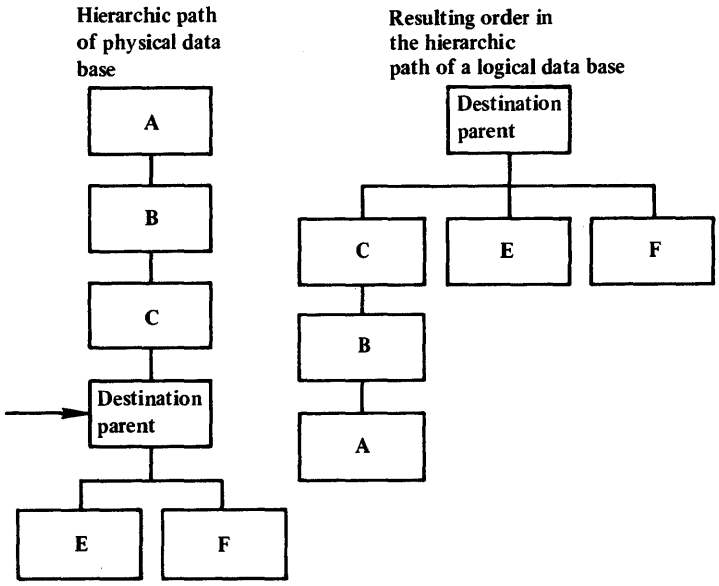


Figure 4-36. Logical Data Base Hierarchy Enabled by Crossing the First Logical Relationship

When an additional logical relationship is crossed in a hierarchic path of a logical data base, access to all segments in the hierarchic path of the destination parent is enabled as in the first crossing.

Rules for Defining Logical Data Bases

1. The root segment type of a logical data base must be the root segment type of a physical data base.

2. A logical data base must be supported by one or more physical data bases. A logical data base must use only those segment types, and physical and/or logical relationship paths that are defined in the physical DBD generation(s) referenced by the logical DBD generation.

3. The path used to connect two segments in a logical data base (i.e., a parent and a child) must have been defined as a physical relationship path or a logical relationship path in the physical DBD generation(s) referenced by the logical DBD generation.

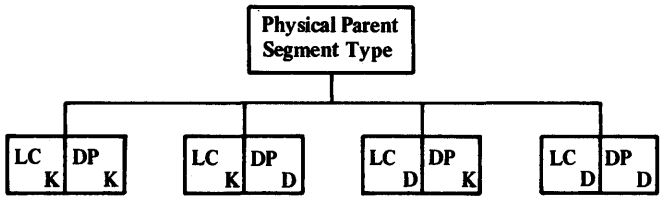
4. Physical relationship paths and logical relationship paths may be intermixed in a hierarchical segment path of a logical data base.

5. After a logical relationship has been crossed in a hierarchic path of a logical data base, additional physical relationship paths and/or logical relationship paths may be included by proceeding in an upward and/or downward direction from the destination parent. When proceeding downwards along a physical relationship path from the destination parent, direction may not be changed except by crossing a logical relationship. When proceeding upwards along a physical relationship path from the destination parent, direction may be changed.

6. Dependents in a logical data base must appear in the same relative order as they appear under their parent in their

physical data base. If a segment in a logical data base is a concatenated segment (i.e., a logical child concatenated with either its physical or logical parent), the physical children of each of the concatenated segments may not be intermixed. The relative order of the children of each of the concatenated segments must remain unchanged.

7. Different variations of a concatenated segment type can be defined as physical child segment types of a physical parent segment type, but only one variation can have dependent segment types. Figure 4-37 shows the four variations of a concatenated segment type that can be defined in a logical data base. When multiple variations are defined under a single physical parent segment type, the variation of the concatenated segment type under the physical parent that has dependents must be the left most variation of the concatenated segment type. A PCB for the logical data base can be sensitive to only one variation of the concatenated segment type.



Key:
LC–Logical child segment type
DP–Destination parent segment type
K–KEY sensitivity specified for the segment type
D–DATA sensitivity specified for the segment type

Figure 4-37. Variations of a Concatenated Segment Type Enabled by Specification of KEY and DATA Sensitivity

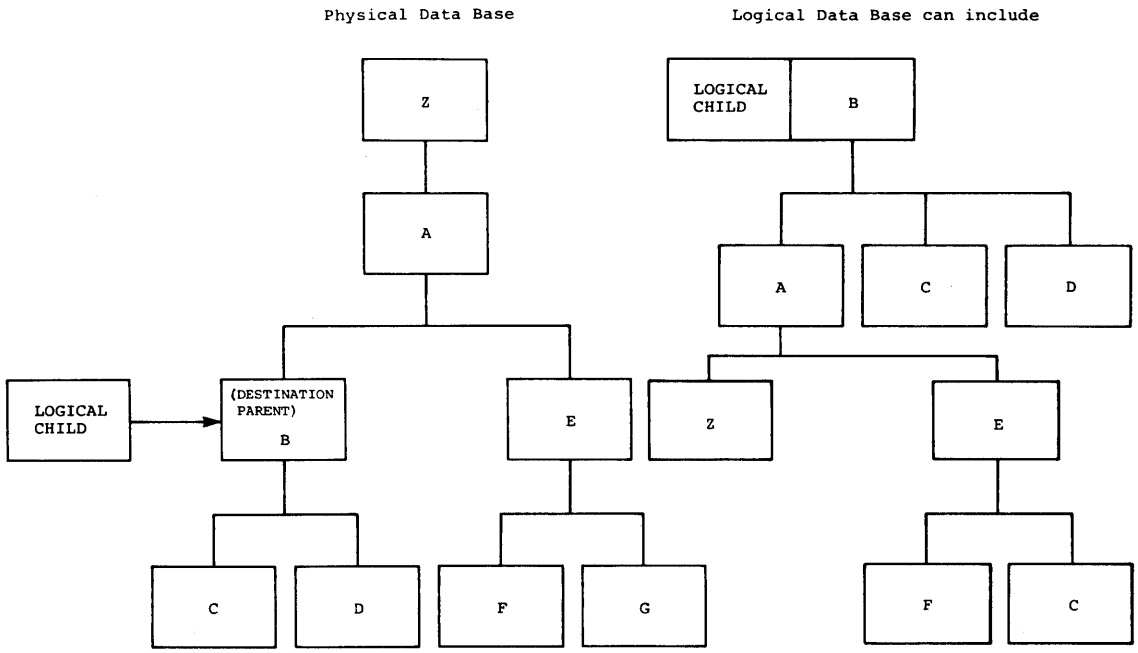
Example 1

When only one logical relationship is crossed in a hierarchic segment path of a logical data base to reach a destination parent:

a. All segment types on which the destination parent is dependent in its physical data base can be included in the logical data base as dependents of the destination parent in inverted order.

b. All segment types that are dependent on the destination parent in its physical data base can be included in the logical data base as dependents of the destination parent without change in their order.

c. All segment types that are dependent on any of the inverted order segment types defined in (a) can be included without change in their order.


Physical Data Base
Logical Data Base can include
Z
A
LOGICAL CHILD
(DESTINATION PARENT)
B
E
C
D
F
G
LOGICAL CHILD
B
A
C
D
Z
E
F
C

Example 1.

SECONDARY INDEXING

Secondary indexes are used to establish alternate entries to physical or logical data bases for application programs. Following are definitions of the terms used for secondary indexing:

- Secondary Index

  A secondary index is comprised of an index pointer segment type defined in a secondary index data base that provides an alternate entry into a physical or logical data base.

- Index Pointer Segment Type

  A segment type defined in a secondary index data base that contains the data and pointers used to index an "index target segment type" in a physical or logical data base (see Figure 4-38).

- Index Target Segment Type

  A segment type defined in a physical or logical data base that is pointed to by an index pointer segment type (see Figure 4-38).

- Index Source Segment Type

  A segment type that is the source from which a secondary index is created (see Figure 4-38).

- Secondary Processing Sequence

  The sequential order in which occurrences of an index target segment type are accessed through a secondary index.

- Secondary Data Structure

  The hierarchic order of segment types in a physical or logical data base that results automatically when a data base is accessed through a secondary index.

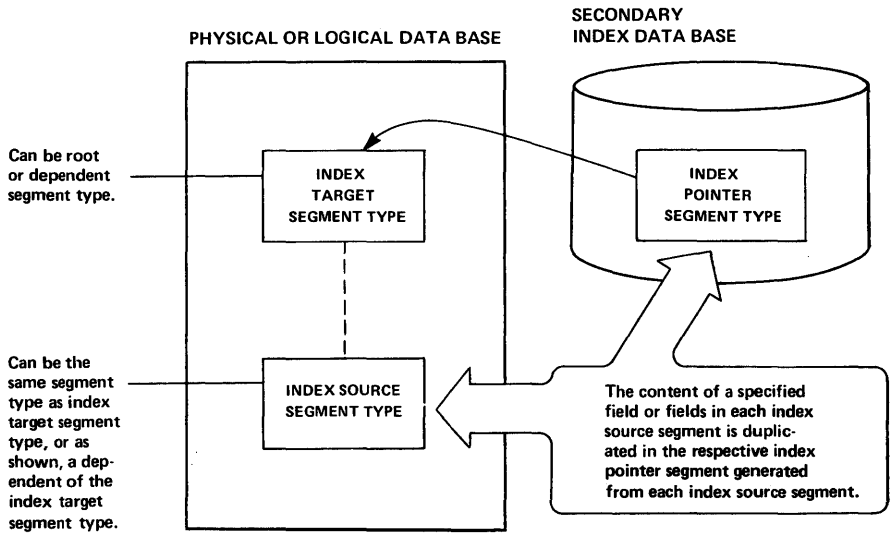


Figure 4-38. Segment Types Associated with a Secondary Index

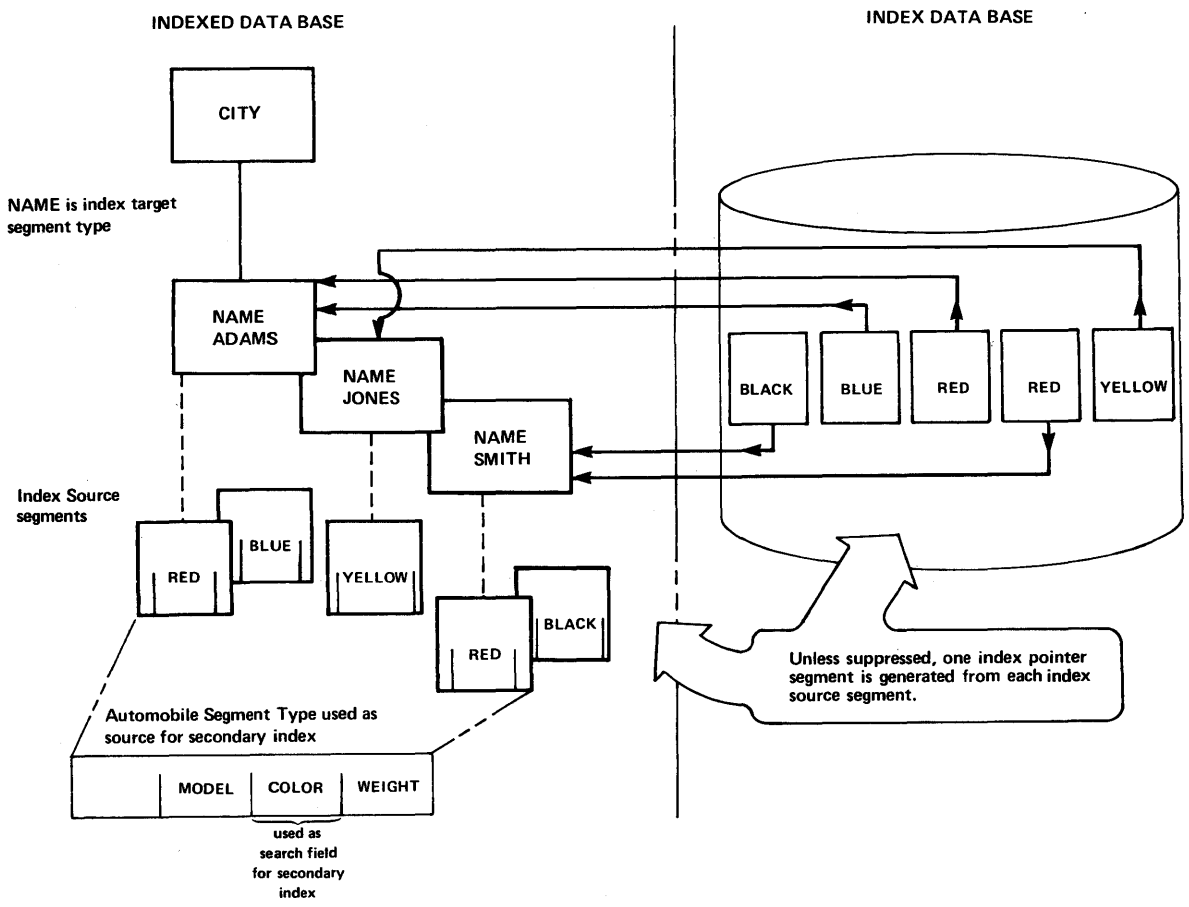


Figure 4-39. Indexing to NAME Segments Based on the Color Field of a Dependent

An index target segment type can be the root or a dependent of a physical or logical data base, and an index source segment type can be either the index target segment type itself or any physical dependent of the index target segment type. A secondary index contains one index pointer segment for each occurrence of the index source segment type in a physical or logical data base. If the same segment type in a data base is used as both the index target and index source segment types, the secondary index contains one index pointer segment that points to each index target segment. If a dependent of an index target segment type is used as the source segment type for a secondary index, the secondary index contains one index pointer segment that points to an index target segment for each source segment that is a dependent of that index target segment as shown in Figure 4-39.

The user specifies through DBDGEN what data within the index source segment type is to be used to index occurrences of the index target segment type. From one to five fields, that can be non-contiguous, within the index source segment type can be specified for use as search data in a secondary index. When specified, the content of each field specified is copied in the search field of the respective index pointer segment generated from each source segment.

SECONDARY PROCESSING SEQUENCE

Entry to a data base through a secondary index enables access to the index target segment type and all segment types in the hierarchic path of the index target segment type. Through the secondary index, the order in which index target segments are accessed is called their secondary processing sequence. The secondary processing sequence of index target segments is determined by the search field values placed in index pointer segments.

SECONDARY DATA STRUCTURE

The order in which segment types in the hierarchic path of an index target segment type are accessed is called their secondary data structure. The hierarchic arrangement of segment types for the secondary data structure is created automatically by IMS/VS. To enable use of the secondary data structure, the user must define sensitivity to the segment types in the secondary data structure through PSBGEN.

Figure 4-40 shows a physical data base hierarchy in which a dependent segment type is indexed through a secondary index. Also shown is the secondary data structure for the segment types in the hierarchic path of the index target segment type.

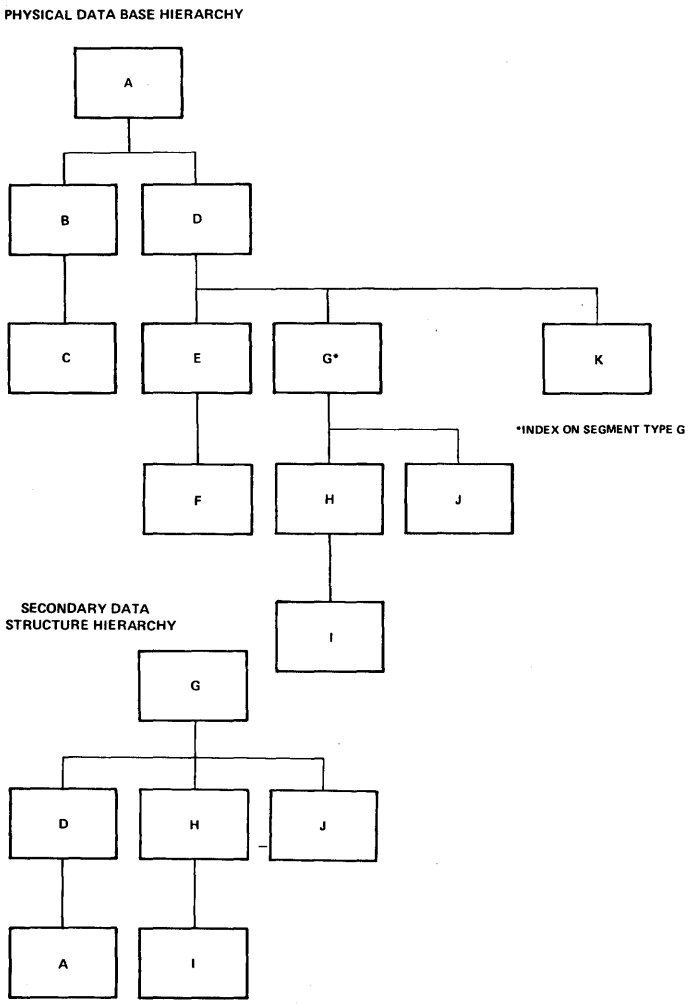


Figure 4-40. Secondary Data Structure

The secondary data structure for segment types in a data base is determined as follows:

1. The index target segment type is the root.

2. Parent segment types of the index target segment type in a data base become the left most dependents of the index target segment type in reverse order.

3. All dependents of the index target segment type are included in the secondary data structure without change in their order except when a parent of the index target segment type is included in the secondary data structure as stated in item 2. In this case, dependents of the index target segment type are displaced one position to the right in the secondary data structure.

4. Only those segment types in the hierarchic path of the index target segment type are included in the secondary data structure.

5. When the root segment type of a data base is indexed through a secondary index, the hierarchy of the data base is unchanged for the secondary data structure.

Options and Rules for Secondary Indexing

A secondary index can be defined using:

- Fields in the index source segment type that contain unique or non-unique data for the search field of the secondary index.
- Up to 5 non-contiguous fields in the index source segment type as the search field of a secondary index.

To enable processing a secondary index as a data base itself:

- The user can specify that data in fields of index source segments is to be duplicated in the index pointer segment generated from each index source segment.
- Index pointer segments can contain any additional user data desired.

A secondary index can be used:

- To index selectively or sparsely by using an option and/or exit provided to enable suppressing the creation of index entries for desired index source segments.
- To 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.
- To selectively access a given segment, through data contained in that segment or a dependent of that segment.
- To access a given dependent in an HDAM or HIDAM data base in less time than is normally required through the primary addressing 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. You cannot declare a secondary processing sequence on an index if the target is a concatenated segment type or a dependent of a concatenated segment type in a logical data base.

3. When using a secondary processing sequence, you cannot insert or delete an index target segment, or any segment on which an index target segment is dependent in a 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.

5. If a variable length segment type is used as an index source segment and an attempt is made to insert an occurrence of the segment type whose length does not include any fields or portions of any fields specified for use in the search, subsequence or duplicate data fields of an index pointer segment, one of the following actions occur:

   a. If the missing index source segment data is used in the search field of an index pointer segment, generation of the index pointer segment for that source segment is suppressed.

   b. If the missing index source segment data is used in the subsequence or duplicate data fields of an index pointer segment, the index pointer segment field will contain one of the three following representations of zero for the missing data (P='000F', X=X'00', or C='0'). The representation used will be the type specified on the FIELD statement that defined that index source segment field.

6. When symbolic pointing only is used to point to index target segments from index pointer segments, unique sequence fields must be defined in the index target segment type and all segment types on which the index target segment type is dependent in it's physical data base.

7. DL/I does not assume responsibility for the order of index pointer segments that contain non-unique keys after a reorganization of the secondary index.

8. A logical child segment type cannot be used as an index source segment type. However, a dependent of a logical child can be used as an index source segment type.

9. In a logical data base, no qualification on indexed fields is allowed in the SSA for a concatenated segment. However, an SSA for any dependent of a concatenated segment can be qualified on an indexed field.

10. The insert rule of FIRST is always followed when entries are added to secondary indexes for data base maintenance.

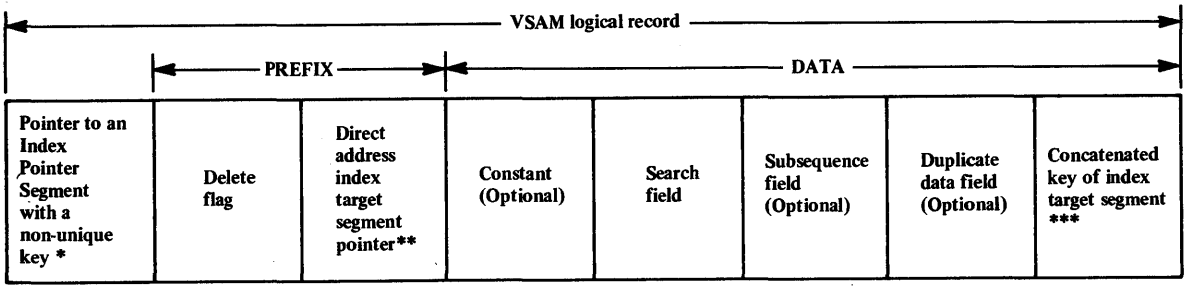
Organization of Secondary Indexes in Auxiliary Storage

A secondary index is stored using a VSAM key sequenced data set only if index pointer segment keys are all unique. If not unique, the index is stored in a key sequenced and an entry sequenced data set pair. The key sequenced data set is used to store the first occurrence of an index pointer segment with a given key, and the entry sequenced data set is used to store additional index pointer segments that contain

the same key. Within both data sets, one logical record is used to store each index pointer segment. When multiple index pointer segments with the same key are stored in the secondary index data base, a pointer is placed at the beginning of the logical record that contains each to chain them together. In the chain, the key sequenced data set logical record always contains the first index segment with a given key, and it points to one of the duplicates in the entry sequenced data set. The sequence in which logical records that contain duplicates are chained in the entry sequenced data set is determined by the insert rule of FIRST.

Index Pointer Segment Format

Figure 4-41 shows the structure of an index pointer segment within a VSAM logical record. In a logical record, the optional non-unique pointer is used to chain logical records that contain index pointer segments with duplicate keys. The remaining portion of each logical record contains an index pointer segment.



* Not present if a unique sequence field is defined in the index pointer segment type.

**Not present when symbolic pointing to the index target segment type is specified

*** Present when symbolic pointing to the index target segment type is specified, and the concatenated key is not present in the subsequence or duplicate data fields.

Figure 4-41. VSAM Logical Record and Index Pointer Segment Formats

An index pointer segment is a fixed length segment that contains a prefix and a data portion. The prefix contains a one byte delete flag and a four byte pointer field when the index uses direct address pointers to point to an index target segment. If symbolic pointing is designated in the index data base, the pointer field is omitted from the prefix. The delete flag is used to mark an index pointer segment as being deleted when the respective index source segment from which an index pointer segment was created is deleted. For HDAM and HIDAM data bases, the pointer field contains a direct address pointer that is used in secondary indexes to point to the occurrence of the index target segment type that is indexed by an index pointer segment. Symbolic pointers may also be specified for HDAM and HIDAM data bases if the user desires. In a secondary index for a HISAM data base, the concatenated key of the index target segment must be stored in the data portion of the index pointer segment to point to the index target segment. The user can specify its position within the data portion by defining the concatenated key as a system related field. When not defined as a system related field, IMS/VS automatically places the concatenated key in a predetermined position in the index pointer segment as shown in Figure 4-41.

The data portion of an index pointer segment contains 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. Constant, subsequence, and duplicate data are optional.

Three fields and a constant can be defined in an index pointer segment type through an XDFLD statement. The fields defined through an XDFLD statement are called search, subsequence and duplicate data (DDATA) fields. A search, subsequence or duplicate data field in an index pointer segment type is comprised of one to five fields that are defined in the index source segment type through FIELD statements. For each respective field in the index pointer segment type, a list of names of one to five fields defined in the index source segment type can be specified for use as that index pointer segment field. The content of the index source segment fields specified are duplicated in the respective index pointer segment field when each index pointer segment is created. The sequence of field names in a list determines the order in which the fields are stored in the index pointer segment field. Names of source segment fields can be specified in any desired order.

In auxiliary storage, the key of an index pointer segment consists of the values in the constant, search field, and subsequence field when all three are specified on an XDFLD statement. Of the three, only the search field is required in each index pointer segment. The constant and subsequence field are optional. When only the search field is specified for an index pointer segment type, the search field contains the entire key of each index pointer segment. When a constant is specified, the key of an index pointer segment consists of the constant followed by the search field value. When a subsequence field is specified, the subsequence field value is appended to the search field value and it becomes a part of the key of the index pointer segment. The combined length of the constant, search and subsequence fields that make up a key must not exceed 240 bytes.

The use of each field in an index pointer segment is as follows.

Constant

When specified, a one byte constant occupies the first byte of the data portion of each occurrence of the index pointer segment type. The constant is used to identify all index pointer segments associated with each secondary index when multiple secondary indexes are defined in the same index data base. The use of the shared index option is described under the heading of "Shared Index Data Bases."

Search Field

The data in the search field of an index pointer segment is a collection of the data in from one to five index source segment fields. In an index pointer segment, the search field is used as all or a part of the key of that index segment. The search field contains the value(s) in a field or fields of an index source segment on which an index target segment is indexed. This means to the user, the presence of the constant and/or subsequence fields in the keys of index pointer segments are transparent to calls. To process a specific index target segment through a secondary index, the SSA of a call is qualified on the search field value only.

Subsequence Field

The data in the subsequence field of an index pointer segment is a collection of the data in from one to five index source segment fields. The purpose of the subsequence field is to extend the key of index segments to prevent storing index segments in an overflow data set in cases where search field values are non-unique. Index segments in an overflow data set can degrade performance. One example could be that of indexing a personnel data base segment type by birth data. If the last name of the individual were specified as subsequence data then all segments with identical birth date fields would be stored in alphabetical order by last name in the primary storage data set. This would not necessarily eliminate all synonyms, but most keys would now be unique. The extension of the index pointer segment key by adding the subsequence field is transparent to the caller since calls are qualified on search field values only.

When the key of an index pointer segment type is comprised of search field values only, index pointer segments with the same search field value are stored in one key sequenced data set logical record, plus the required number of entry sequenced data set logical records. By specifying a subsequence field, the key of index pointer segments is extended. When subsequence field values are unique, multiple occurrences of the index pointer segment type with the same search field value are stored in consecutive logical records in the key sequenced data set. Increased performance results since no searches among the duplicates in the entry sequenced data set are required.

Duplicate Data Field (DDATA)

Data in the DDATA field of an index pointer segment is a collection of the data in one to five index source segment fields. When specified, space for its contents is alloted adjacent to the subsequence or search field value in an index pointer segment. The DDATA field is defined to prompt the system to duplicate data contained in index source segments in index pointer segments to enable using that data when a secondary index is processed as a data base itself.

Additional Data in Index Pointer Segments

The user can include any additional data desired in index pointer segments by specifying a length for the index pointer segment type that is sufficient to include the additional data. When included, the additional data is available to the user when processing the secondary index as a data base itself. The user should note however, that initial loading of additional data, and maintenance of the additional data when reorganizing an indexed data base maintenance is his responsibility. During reorganization of an indexed data base, the secondary index(s) for that data base are recreated. When each secondary index is recreated, any additional user data that existed in the original secondary index is lost.

System Related Fields

System related fields are defined in index source segment types for use in secondary indexing. They are defined using FIELD statements, and can only be defined for index source segment types. Two types of system related fields can be defined for use in secondary indexing.

The first type of system related field consists of defining a portion or all of the concatenated key of an index source segment as a field within the index source segment. The name of this field can be up to

eight characters long, and its name must begin with the three characters /CK. It may appear in the field list for either subsequence or DDATA fields defined by the XDFLD statement.

The name of the second type of system related field begins with the three characters /SX. A /SX field is a four byte field that contains an IMS/VS generated value that uniquely identifies a source segment. It may appear only in the subsequence field of an index pointer segment, and may only appear if the index source segment is in an HDAM or HIDAM data base.

System related fields are defined in the index source segment type, but do not physically exist as fields within that segment type. System related fields are defined in the source segment type for use in the subsequence or DDATA fields of index pointer segments. The subsequence and DDATA fields of the index pointer segment type are defined through an XDFLD statement. In defining each, the names of up to five fields defined in the index source segment type can be specified for each on the XDFLD statement.

The /CK or /SX fields are used in the subsequence field of index pointer segments to make their keys more unique. In addition, a /CK field can be used to store portions of the concatenated keys of occurrences of the index source segment type in their respective index pointer segments. This may reduce space requirements where pointing is symbolic and part of the concatenated key is to be used as subsequence data. In a secondary index for a HISAM data base, the concatenated key of each index target segment must be included in its' respective index pointer segment for use as a symbolic pointer. When not included in the subsequence or DDATA fields of the index pointer segment type used for a HISAM data base, IMS/VS automatically appends the concatenated key of each index target segment to any data in the subsequence or DDATA fields of the respective index pointer segment.

Suppression of Index Entries

Two operands, that can be specified during DBDGEN, can be used to suppress the creation of index pointer segments. The two are the NULLVAL=, and the EXTRTN= operands on the XDFLD statement.

In the NULLVAL= operand, a one byte self-defining term, or the words BLANK or ZERO can be specified. If the NULLVAL operand is specified, all fields in an index source segment that comprise the search field in the index pointer segment are checked to see if each byte within the field(s) contains the specified value. When the field or fields in the index source segment are filled with the specified character, an index pointer segment is not created.

The EXTRTN= operand is used to specify the name of an index maintenance exit routine that is supplied by the user to suppress the creation of selected index entries.

Index Maintenance Exit Routine

Secondary indexing allows specification of a user supplied index maintenance exit routine which can selectively suppress the creation of index segments. The routine enables the user to control the density of a secondary index. One exit routine is allowed for every secondary index or a generalized routine may be written to serve several indexes. For detailed information on index exit maintenance routines, see the IMS/VS System Programming Reference Manual.

The name given to the load module used for controlling index maintenance must be the value of the EXTRTN= operand on the XDFLD statement in the DBD generation for the indexed data base.

Index Maintenance Processing

When an index source segment is inserted, deleted, or replaced in a data base, DL/I index maintenance keeps the index synchronized with the contents of the data base. The action taken depends on the operation being performed: insert, delete, or replace.

When a source segment is inserted, a copy of the proposed indexing segment is constructed during index maintenance. The NULLVAL test and exit routine test are performed on the copy to determine the suppression status of the indexing segment. If no suppression status is indicated, the indexing segment is inserted into the index.

When a source segment is deleted, a copy of the alleged existing indexing segment is constructed during index maintenance. The NULLVAL and exit routine tests are performed to determine the suppression status of the indexing segment. If no suppression status is indicated, the matching segment is found in the index and deleted.

If suppression status is indicated for an insert or delete, no further processing is required for that entry.

When a source segment is replaced, an index entry may or may not be affected. The indexing segment may be replaced or it may be deleted and a new indexing segment inserted. It is also possible that no action is required. The action taken is determined by comparing the constructed copies of the old and new indexing segments. The following describes the action to be taken:

- If no suppression is indicated for either segment and:
  - there is no change to indexing segment, no action is taken.
  - only the data in the indexing segment is changed, the indexing segment is replaced.
  - the key in the indexing segment is changed, the old segment is deleted and a new segment is inserted.
- If suppression is indicated :
  - for the old indexing segment but not the new, the new indexing segment is inserted.
  - for the new indexing segment but not the old, the old indexing segment is deleted.
  - for both the old and new indexing segment, no action is taken.

The question asked by the DL/I index maintenance routine when it invokes the user index exit routine is, "Will this index pointer segment appear in the index?" The exit routine answers the question through a return code.

Suppression of indexing by the exit routine must be consistent. The same index pointer segment cannot be examined at two different times and have suppression indicated only once. Also, user data cannot be used to evaluate suppression, since the actual index pointer segment is only seen by the exit routine just before insertion of the new one. In the cases of replace and delete, only a prototype is passed which

contains the constant, search data, subsequence data, and source data, plus any symbolic pointer which may have been added.

Shared Index Data Bases

Multiple secondary indexes can be placed in a single shared index data base. A shared index data base is created, accessed, and maintained in the same manner as a data base containing only one secondary index. To be eligible for combining, all indexes must be comprised of segments of equal length, with key fields of equal length, and with equal key offset positions. A maximum of 16 indexes can share a single shared index data base. Each secondary index in a shared index data base must have a constant specified which uniquely identifies that index. The advantage of a shared index data base is a reduction in the number of control blocks for VSAM and DL/I.

Processing a Secondary Index as a Data Base

A secondary index can be processed as a data base by providing a PCB which references the DBD of the secondary index. The purpose of processing a secondary index as a data base could be to scan the subsequence or duplicate 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 a secondary index separately, the following guidelines and restrictions apply:

- No changes to system-maintained data fields in the index pointer segment will be allowed unless ACCESS=(,,NOPRCT) is specified in the index DBD. Attempts to change system maintained data without the NOPROT option specified 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 unuseable and unmaintainable.

- Deletion of index entries by the user when the associated index source segments exist will result in NE status codes if the user makes updates to the index source segment 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 data base, the secondary index is regarded as a separate index data base. A series of GN calls will not violate the boundaries of the secondary index for which they are intended. Each secondary index in the shared index data base has its unique DBD name and root segment name.

Secondary Indexes and Segment Search Arguments

SSAs of calls for index target segments can be qualified on the search field of one or more secondary indexes when accessing index target segments through their primary or secondary processing sequence. This is accomplished by using indexed fields defined within the index target segment type to qualify SSAs. An indexed field is defined in

name only for an index target segment type. During DBDGEN, one indexed field is defined in the index target segment type for each secondary index that points to that segment type. The name specified for the indexed field actually represents the search field of the associated secondary index. Since the name specified for the indexed field of an index target segment type represents the search field of a secondary index, when an SSA is qualified on the indexed field of an index target segment, the search field of the associated secondary index is searched to satisfy the call.

When a secondary index is searched to see if an index pointer segment satisfies a call, the call is satisfied when an index pointer segment contains the specified search field value and points to the index target segment under consideration.

In cases where index source segments are several levels below index target segments, qualifying calls on the search field of a secondary index can prove to be an efficient means of selecting index target segments based on data in index source segments. In no case should this use be made of a secondary index when the index target segments and index source segments are the same segment type, and the indexed data base is being processed through its primary processing sequence. Even where the index target and index source segment types are different, the following guideline should be used. The method should be chosen which causes the fewest accesses to the data base or index.

Considerations

In using secondary indexing, consideration shculd 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 a source segment that is used in the search, subsequence or ddata 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 a 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 a 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 re-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 "IMS/VS Data Base Buffer Pools" the storage estimates chapter of the IMS/VS System Programming Reference 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 a key sequenced data set only if all index pointer segment keys are unique, and 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.

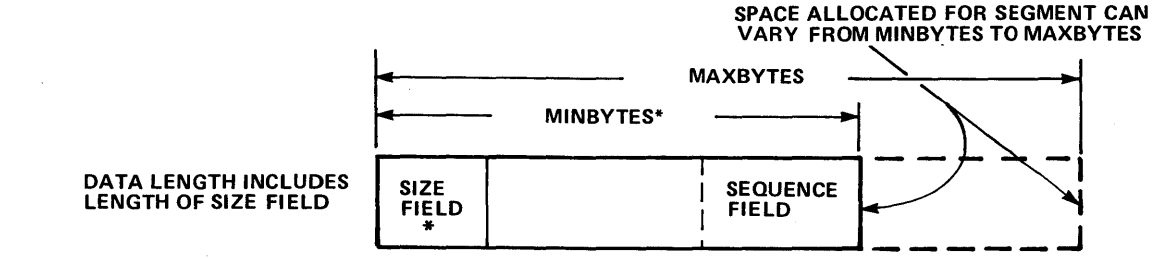
VARIABLE LENGTH SEGMENTS

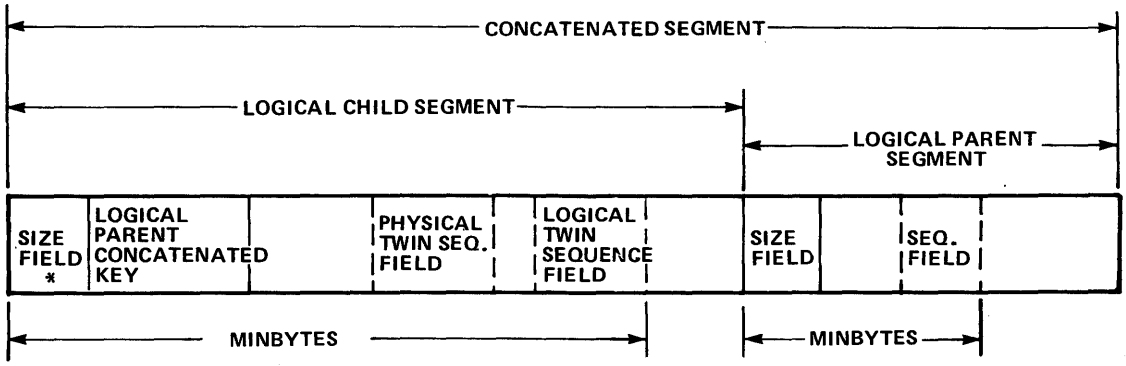
Variable length segments enable the user to vary the amount of storage space used to store different occurrences of the same segment type. They are intended for use by application programs that process variable length text or descriptive data. In addition in some cases, they can be used to enhance utilization of secondary storage. Variable length segments enable the user to vary the space used for each occurrence of a segment type between a minimum and maximum number of bytes through a two byte size field loaded with each occurrence.

Variable length segments can be used in HISAM, HDAM and HIDAM data bases when VSAM is specified as the access method for the data base. To use variable length segments, a segment type must be defined as variable in length when defining the segment type through the DBDGEN utility. The variable length of a segment type is specified using the BYTES= keyword of a SEGM statement as shown in Figure 4-42. MAXBYTES specifies the maximum length used for the data portion of occurrences of a segment type and MINBYTES specifies the minimum length used. In addition, the user must include a two byte size field in the first two bytes of the data portion of each occurrence of that segment type when loading it into the data base. The size field is loaded with each segment to tell IMS/VS the length of data in that segment. Since the size field is in the data portion of a segment, the data length placed in the size field must include the length of the size field itself. In addition, if a sequence field is defined in the segment type, the minimum length specified in the size field must include at least the size field and all data to the end of the sequence field.

When initially loading occurrences of a variable length segment type, the space used to store the data portion of an occurrence is the length specified in MINBYTES or the length specified in the size field, whichever, is greater. When MINBYTES is greater than the length specified in the size field of an occurrence, more space is allocated for the segment than is required for the segment. The additional space allocated is free space that can be used when existing data in the segment is replaced with data that is greater in length.

SEGM NAME=SEGNAME, BYTES=(MAXBYTES, MINBYTES)





LOGICAL CHILD SEGMENT IN PHYSICAL DATA BASE

*MINBYTES MUST BE ⩾ 4 BYTES AND MUST INCLUDE:

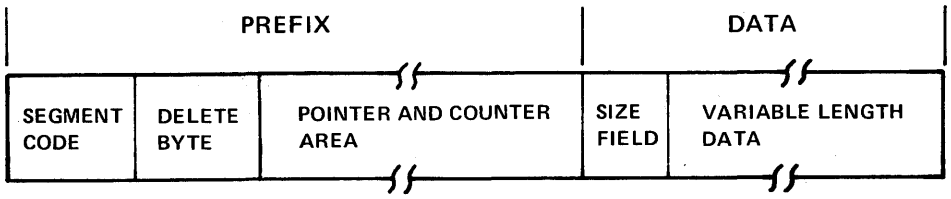
1. PHYSICAL TWIN SEQUENCE FIELD, IF PRESENT, AND KEY COMPRESSION IS NOT ENABLED BY A SEGMENT EDIT/COMPRESSION ROUTINE.
2. LOGICAL PARENT CONCATENATED KEY.
3. LOGICAL TWIN SEQUENCE FIELD, IF PRESENT.

Figure 4-42. Variable Length Segments

Variable length segment formats are shown in Figure 4-43. Fixed length and variable length segment formats are the same except for a size field placed in the data portion of a variable length segment. In addition for HDAM and HIDAM data bases, when the prefix and data portions of a segment are separated in storage due to update activitiy, the first four bytes of space following the prefix are used for a direct address pointer to the separated data portion of the segment.

The user can load segments initially with a given number of bytes of data, and then either increase or decrease the length of data in each by replacing the data. When the length of data in an existing segment in a HISAM data base is increased, the logical record containing the segment is rewritten to acquire the additional space required. Any segments displaced through the increased data length are placed in overflow storage. When the length of data in an existing segment in a HISAM data base is decreased, the logical record is rewritten to make all segments contiguous within the logical record. When the data in an existing segment in an HDAM or HIDAM data base is replaced with data that is greater in length and the space allocated for the existing segment is not sufficient for the new data, the prefix and data portions of the segment are separated in storage to obtain sufficient space for the new data. When separated, a pointer is placed in the first four bytes of space following the prefix, which remains in its original position, to point to the new data portion of the segment. When separated and existing data is replaced with data that fits into the original space allocated for the data portion of the segment, the new data portion is placed in the original space allocated overlaying the data pointer. When the prefix and data portions of a segment are not separated, and data in an existing segment in an HDAM or HIDAM data base is replaced with data that is shorter in length, the new data followed by free space occupies the position of the original data.

HISAM, OR HDAM AND HIDAM WHEN PREFIX AND DATA ARE NOT SEPARATED



HDAM AND HIDAM WHEN PREFIX AND DATA ARE SEPARATED

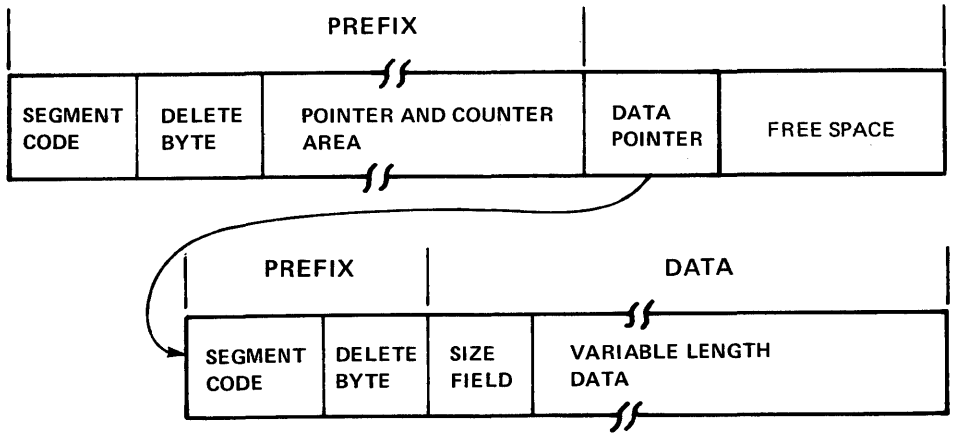


Figure 4-43. Variable Length Segment Formats

CONSIDERATIONS

For HISAM data bases, replacing existing data in segments with data that is greater in length can affect performance since this may require displacing segments to overflow logical records. Replacing data in an existing segment in a HISAM data base with data that is shorter in length has no affect on performance. The additional overhead required to use variable length segments in a HISAM data base consists only of the two byte size field loaded with each occurrence.

Additional storage requirements in HDAM and HIDAM data bases when variable length segments are initially loaded consists of the two byte size field in the data portion. In addition when the prefix and data portions of a segment are separated in storage, a one byte segment code and one byte delete byte are stored with the data portion of the segment. Performance can be affected when the prefix and data portions of segments are separated and stored in different blocks in a data set.

CONVERSION CONSIDERATIONS

When a segment type has been defined as variable in length, before initially loading or inserting any occurrence of that segment type into its data base, a size field must be present. To convert an existing data base from fixed length to variable length segments, a size field must be added to each segment before it can be loaded or inserted into a data base as a variable length segment.

The user must supply his own routine to convert an existing data base from fixed length to variable length segments. For the new data base, a new DBD must be created that identifies the variable length segment types. The routine could then sequentially retrieve segments from the existing data base, add the size field to variable length segments, and then insert the segments into the new data base.

A second method of converting from fixed length to variable length segments enables use of the IMS/VS Unload/Reload utilities; however, this method requires that an interim DBD be created. The interim DBD is required to enable a user routine to place a size field in fixed length segments before they can be loaded into a data base as variable length segments.

For the interim DBD, the user specifies a fixed length for the segment type that is being converted to variable length. In addition, the user specifies use of the segment edit/compression exit for that segment type. Using the reload utility, each time an occurrence of the segment type is presented to an IMS/VS action module for loading, the segment edit/compression exit enables a user routine to gain control to add a size field to the segment. After the size field is added, the user routine passes control back to the action module which then loads the segment into the data base. After the data base is reloaded by the reload utility, the user then creates a new DBD to define the variable length of the segment types converted.

SEGMENT EDIT/COMPRESSION EXIT

The segment edit/compression exit facility of IMS/VS enables the user to supply a routine to edit a segment during its movement between the application program input/output area and the data base buffer pool. The facility offers the user the ability to encode data for security purposes, to format data to be used by application programs, and to compress a segment to eliminate redundant characters. The edit/compression routine should be transparent to the application programs.

The routine to be used for edit/compression is named by the operand COMPRTN= routine-name on the SEGM statement in the DBDGEN operation for the data base. A segment work area is constructed by IMS/VS at initialization time, and the edit routine is loaded when the data base is opened. As a segment from that data base is requested by the user, its location in the buffer pool is obtained. If an edit routine has been specified, the address of the data portion of the segment, and the start of the segment work area is supplied and the routine is given control. On a retrieve operation, the edit routine is responsible for moving the data from the buffer pool to the segment work area. IMS/VS will move it from the segment work area to the application program I/O area. For load, insert, or replace operations, data is moved from the application program I/O area to the segment work area by the edit routine, then to the buffer pool by IMS/VS. See Figure 4-44 for a visual explanation of segment edit/compression.

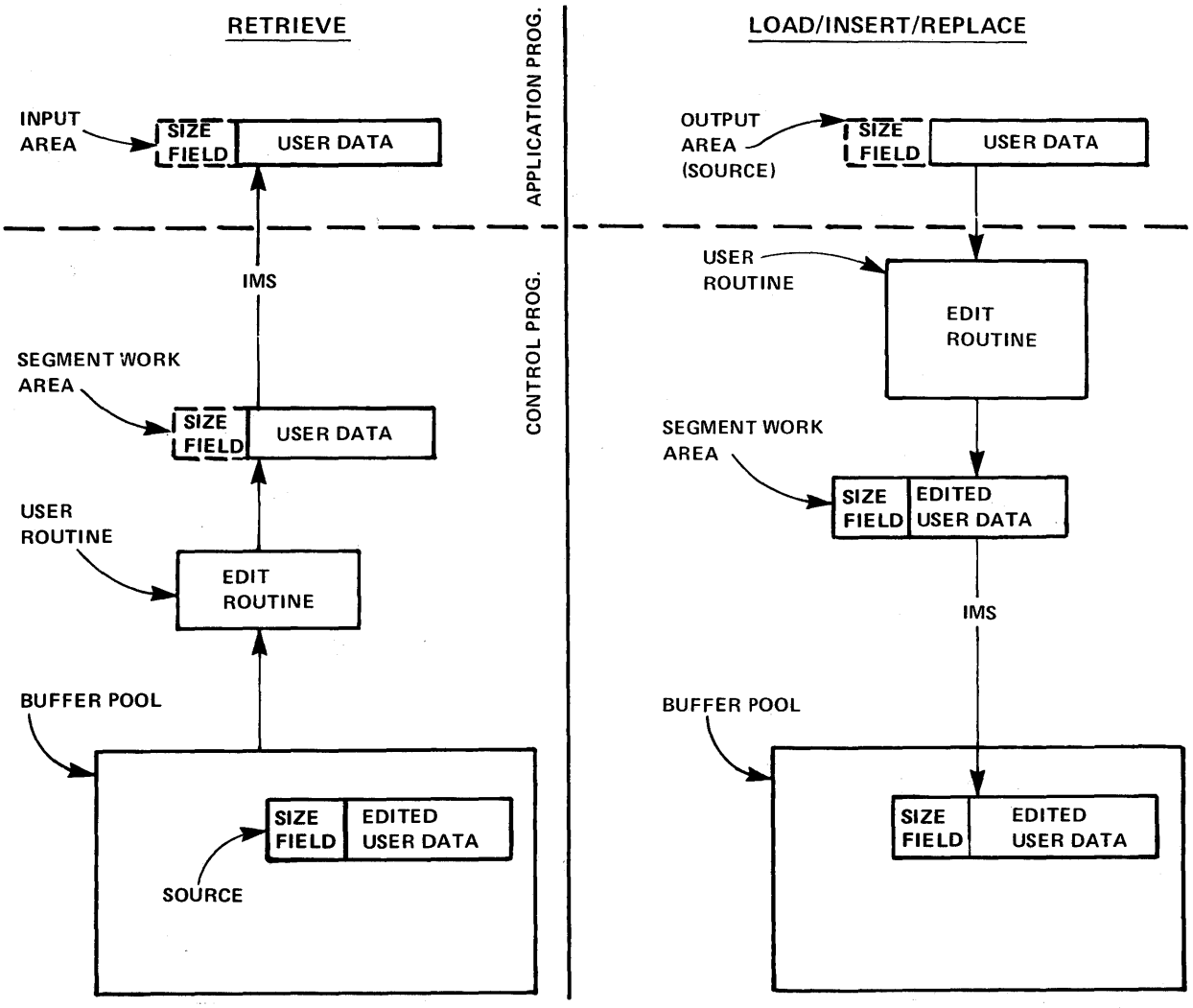


Figure 4-44. Segment Edit/Compression

To assist the user in providing parameters to his edit/compression routine, the DBD control block has a table appended to it in the form of assembly language control sections. One control section is developed for each segment type to be edited or compressed. These control sections contain information such as the edit/compression routine name, the name of the segment, and the total length of that control section. Each control section may be extended by the user to contain any desired data or algorithm information.

Although the segments may be defined as fixed or variable length to the application program, the segments to be processed by the edit/compression routine must be variable length in the data base. The data length is contained in a size field in the first two bytes of the segment. If the segment is defined as fixed length to the application program, the size field must be stripped off by the edit/compression routine before the segment is presented to the application program. In addition, if the segment was compressed, it must be expanded by the edit routine to the fixed length expected by the application program. In reverse, if the application program presents a fixed length segment, the edit/compression routine must append the size field prior to the segment being written to the data base. If the edit/compression routine compresses the segment, the size field must be updated to reflect the correct length.

To convert existing data bases to use this facility, the following steps must be performed:

1. Unload the current data base using the reorganization/unload utility and using the current DBD.

2. Define a new DBD which specifies VSAM as the access method and specifies a COMPRTN for those segments which are to be converted. Reload the data with the reorganization/reload utility.

3. The named COMPRTN provided during reload should encode, compress or edit the segment (as determined by the installation's requirements) and add the two byte size field.

There are two types of segment manipulation possible through the DL/I edit/compression facility:

> Data compression - movement or compression of data within a segment in a manner that does not alter the content or position of the key field. Typically, this type involves compression or encoding of data from the end of the key field to the end of the segment. This is the only time that the location of the fields may be altered. The segment size field of a variable length segment cannot be compressed.
>
> Key compression - movement or compression of any data within a segment in a manner that can change the relative position, value, or length of the sequence field as well as any other fields.

Any segment type in a physical data base can be specified during DBDGEN as being compressible with either the KEY or DATA option, with the following exceptions:

- Any segment type which is defined as a logical child may not be specified
- Segments residing in an INDEX data base may not be specified
- Segments defined as root segments of a HISAM data base may be specified for DATA compression only

Although the contents of the sequence field or the data may be modified by the edit/compression routine, the segment's position in the data base is determined by the original sequence field value. An example may help to explain this. If the defined sequence of a particular segment type is based on last names, and the data base contains segments for people named SMITH, JONES, and BROWN, the segments are maintained in alphabetical sequence -- BROWN, JONES, SMITH. Assume that an edit routine encodes these names as follows:

```
BROWN-------->29665

JONES-------->16552

SMITH-------->24938
```

The encoded value is placed in the sequence field. The segments will be maintained in the original sequence (BROWN, JONES, SMITH) rather than in the numerical sequence implied by the encoded values (16552, 24938, 29665). The records are maintained in the originally defined sequence so that if the application program issues a GET NEXT request, the correct segment is retrieved.

CONSIDERATIONS

General considerations which apply to using the segment edit/compression exit facility are:

- Any segment specified to be edited must reside in a VSAM data set
- All segment editing takes place on segments described in a physical data base only
- If the user routine is designed to edit more than one segment type, in one or more physical data bases, the routine must be link-edited as reentrant
- Adequate storage for the edit routine(s) must be provided for both batch and online systems
- Since this module becomes a part of the IMS/VS region, any abnormal termination on its part terminates the entire IMS/VS region
- The user routine cannot employ such Operating System macros as SPIE and STAE

The segment edit/compression exit provides the user with a valuable tool. However, several additional considerations are worth noting. Edit/compression processing of each segment on its way to or from an application program involves added CPU time. In addition, the search time required to locate the requested segment may be increased, depending upon the options selected. In the case of full segment compression, using the KEY compression option, every segment type that is a candidate to satisfy either a fully qualified key or data field request must be expanded or decoded to allow examination of the appropriate field by the IMS/VS retrieve module. For key field qualification, only those fields from the start of the segment through the sequence field are expanded during the search. For data field qualification, the total segment is expanded. In the case of data compression and a key field request, little more processing is required to locate the segment than that of non-compressed segments, since only the segment sequence field is used to determine if this segment occurrence satisfies the qualification.

Other considerations can impact total system performance, especially in an online teleprocessing environment. For example, being able to LOAD an algorithm table into memory will give the compression routine a large amount of flexibility. However, this action can place the entire IMS control region into a wait state until the requested member is present in main storage. It is suggested that all alternatives be explored to lessen the impact of situations such as this.

DATA BASE DESIGN CONSIDERATIONS

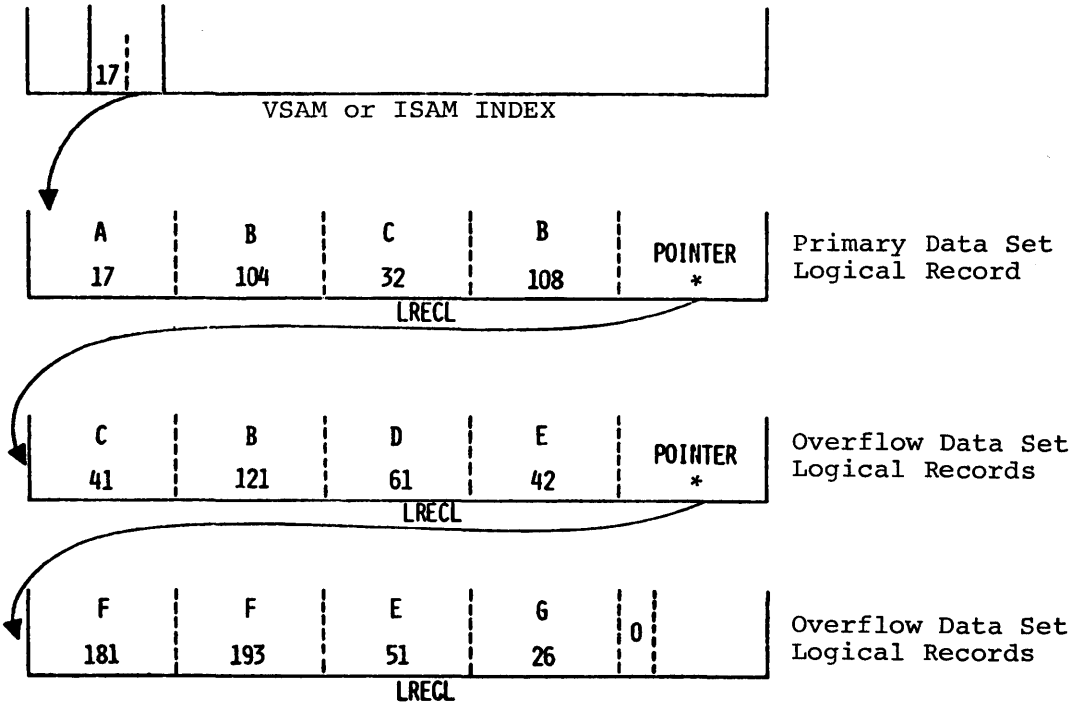
HIERARCHICAL SEQUENTIAL DESIGN CONSIDERATIONS

This section discusses the various data base design considerations with which programming personnel should be familiar to get the best use from the capabilities of the IMS/VS Hierarchic Sequential data base organization and in particular the HISAM data base access method.

PROCESSING TIME

Before performing I/O operations, IMS/VS uses information within its internal data base tables and data base buffers in an attempt at

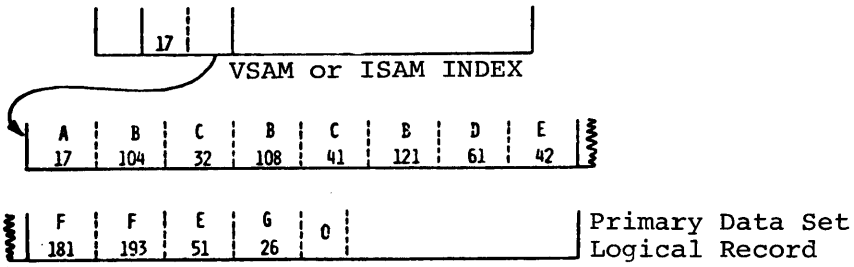
satisfying segment requests. If no infcrmaticr is found in main storage, the index and the necessary primary data set record are read from auxiliary storage. In accessing a segment of information, a VSAM or ISAM index is used to obtain the root segment cf the specified record. Dependent segments cf that root are reached in a sequential manner. If the information is nct found in a primary data set record, a pointer in the primary data set logical record causes a read to be performed on an overflow record.



* The pointer is at the beginning of VSAM logical records

Figure 4-45. HISAM Data Base Record in Auxiliary Storage

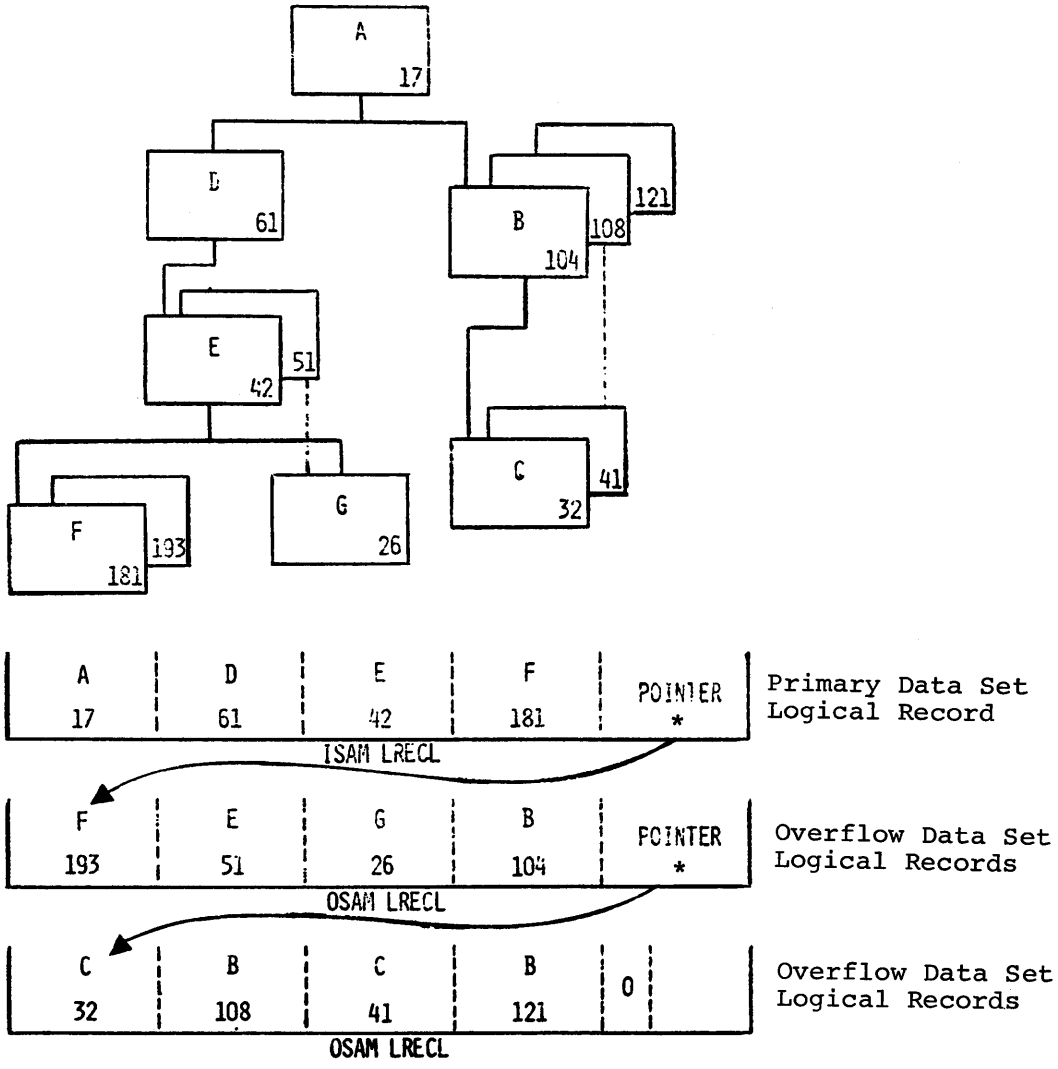
The first soluticn to reduce the number of direct access references (and hence processing time) is to increase the size of the primary data set logical record length, thus eliminating the need for overflow records.



(NO OVERFLOW RECORD REQUIRED)

Figure 4-46. HISAM Data Base Record -- Larger Primary Data Set Logical Record

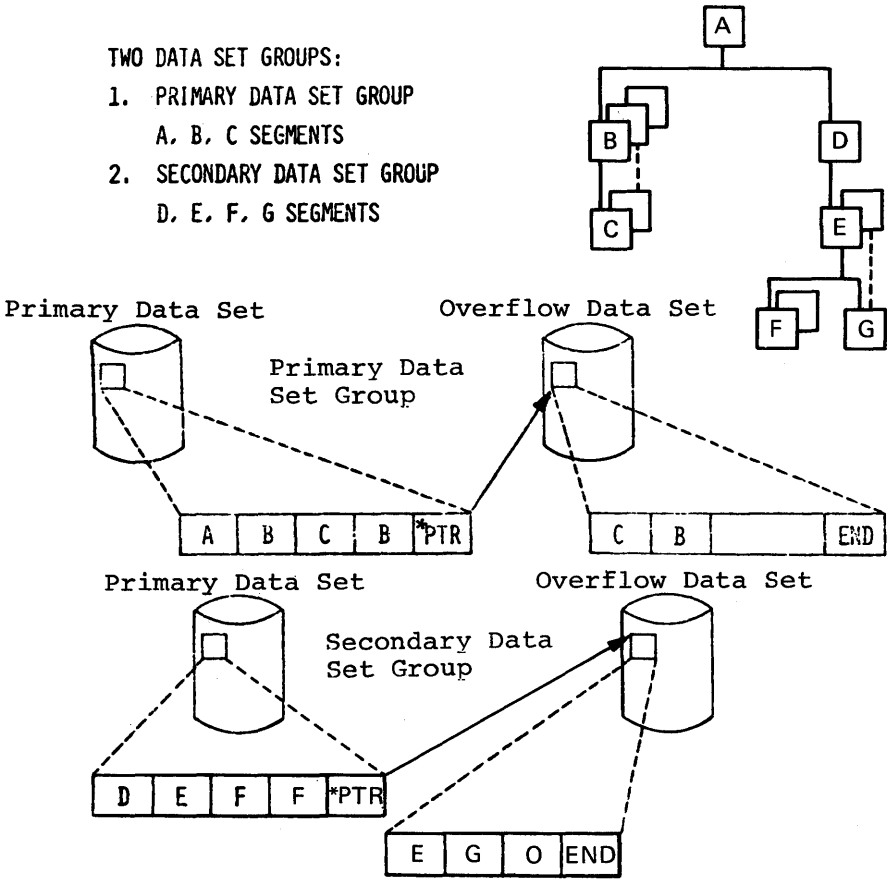
The segment which is logically farthest from the root in the top to bottom, left to right, hierarchy, is also physically farther from the root when the segment is stored. This indicates that the logical structure may be manipulated to reduce the number of direct access references required to obtain a particular segment.



* The pointer is at the beginning of VSAM logical records

Figure 4-47. Storage Sequence of Segments in HISAM Data Base Record

Another solution is the use of multiple data set groups. This allows second-level segments and their dependent segments to be accessed through an index with no need to access the root or intervening segments. The maximum number of data set groups for a data base is ten. See Figure 4-48.

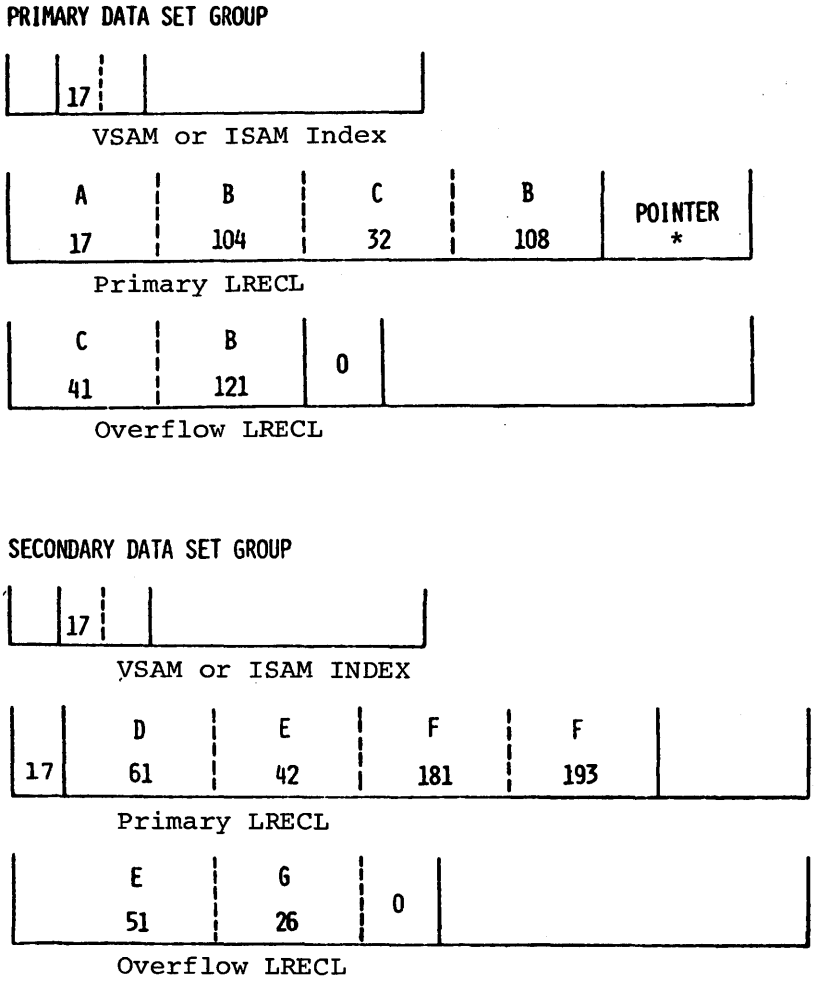


* The pointer is at the beginning of VSAM logical records

Figure 4-48. HISAM -- Multiple Data Set Groups

As can be seen, segments in the secondary data set group can be retrieved without reference to the primary data set group.

Figure 4-48 shows that segments D, E, F, and G are placed into a secondary data set group. Figure 4-49 shows that, because no references to the primary data set group are necessary, the number of references needed to obtain Segment F with key 181 has been reduced. Note that the index in both data set groups contains the key value of the root segment.



* The pointer is at the beginning of VSAM logical records

Figure 4-49. HISAM Segment Storage -- Multiple Data Set Groups

With multiple data set groups, we may also elect to expand the size of the logical record of the secondary data set group as shown in Figure 4-50. In this case, no overflow logical record would be required. Logical record sizes of data set groups representing a single data base may vary. In addition, the logical record length of each primary data set and of its associated overflow data set in the same data set group may be equal, or the overflow logical record length may be greater than the primary logical record length.

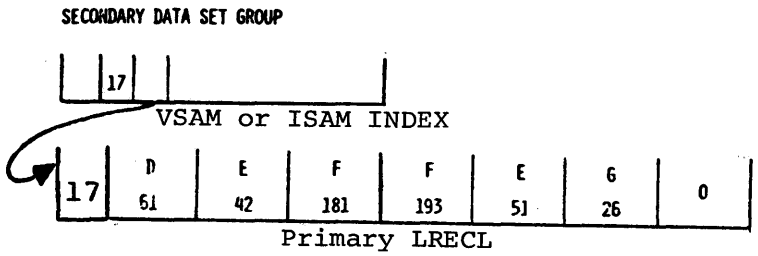


Figure 4-50. HISAM Secondary Data Set Group with a Larger Primary Data Set Logical Record Length

In summary:

- Logical record size can be expanded to accommodate more segments, thus eliminating or minimizing the need for overflow records.
- The logical structure can be manipulated to bring the most active segment types hierarchically closer to the root.
- Multiple data set groups can be used to avoid accessing a root segment and intervening dependent segments when accessing a particular segment type.
- The logical record length of primary and overflow data sets across data set groups may vary. The logical record length of an overflow data set must be equal to or greater than the primary logical record length in the same data set group.

DIRECT ACCESS STORAGE SPACE UTILIZATION

The percentage of utilization of direct access storage device space by an IMS/VS data base at load time is a function of the relationship between the logical record lengths and the size of the actual data base records being loaded. Data base records within a data base usually vary in size, but, since IMS/VS uses fixed-length logical records, the choice of a logical record length to contain the largest data base record results in unused space for the smaller data base records. Choice of a logical record length to hold the smaller data base records results in better space utilization in the primary data set, but parts of larger data base records are forced to the overflow data set on initial loading.

The choice of a logical record length must be made with appropriate consideration for the type of processing to be accomplished against the data base. For example, if new dependent segments are being created with great frequency, it may be a good idea to assign an oversized logical record length. This logical record length allows many dependent segments to be placed in the primary data set. Figure 4-51 shows what happens if a small logical record length is chosen for two records - Record 1 and Record 2. Two overflow records are required, and there is very little slack space.

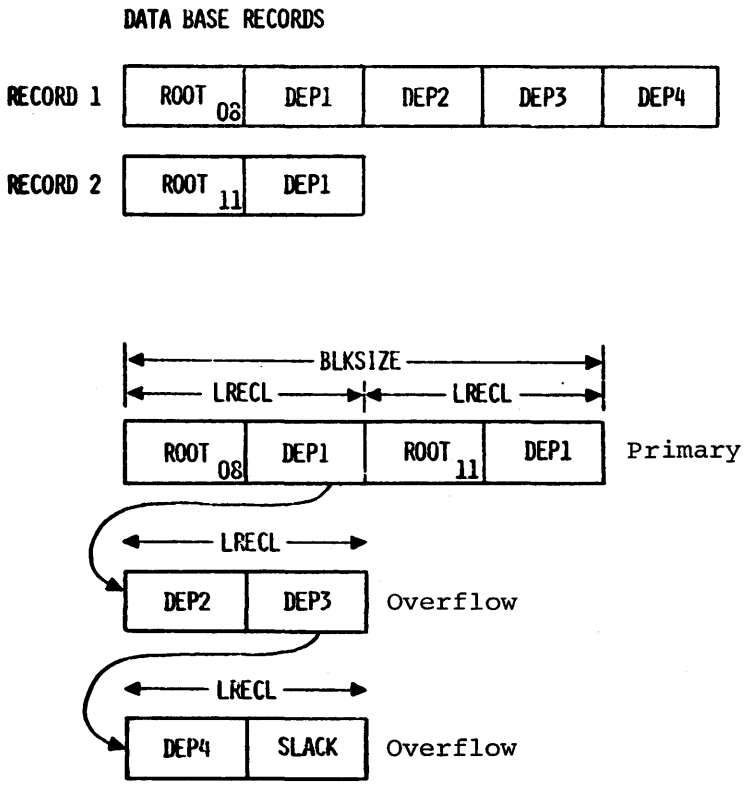


Figure 4-51. HISAM -- Small Logical Record Length

Figure 4-52 shows the same records with a large primary data set logical record length. There is no requirement for overflow records, but there may be a large amount of slack space in primary data set logical records. All unused space in the primary data set is tied to specific roots.

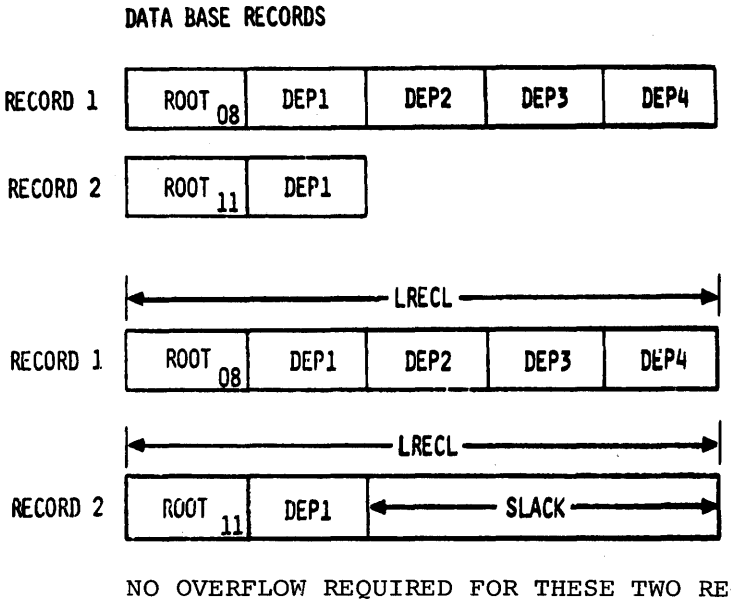


Figure 4-52. HISAM -- Large Logical Record Length

The slack space in Figure 4-53 can only be used by dependent segments of Root 24. New dependent segments of Root 18 would have to be put into overflow, even though the slack space related to Root 24 is not being used and is in the same physical block.

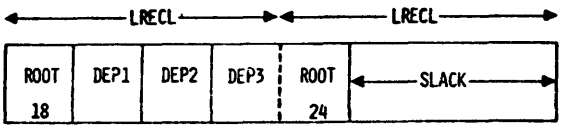


Figure 4-53. HISAM -- Utilizing Slack Space

Segmentation also influences utilization of direct access storage device space. See Figure 4-54. The minimum logical record length which can be assigned for a data base must be large enough to hold the largest segment defined for the data base. The following describes two different methods of segmenting for a 2000-byte record: Case A, where the ROOT and DEP1 segments are combined into one 1500-byte segment, yields a minimum logical record of 1500 bytes and produces 1000 bytes of slack in a OSAM record. This slack is only available for dependent segments which relate to a particular root. The logical record sizes for overflow must be at least as large as the primary logical records. The different method of segmenting the same record in Case B, where the ROOT and DEP1 segments are separate segments, yields a minimum logical record size of 1000 bytes, with no excess in the overflow record.

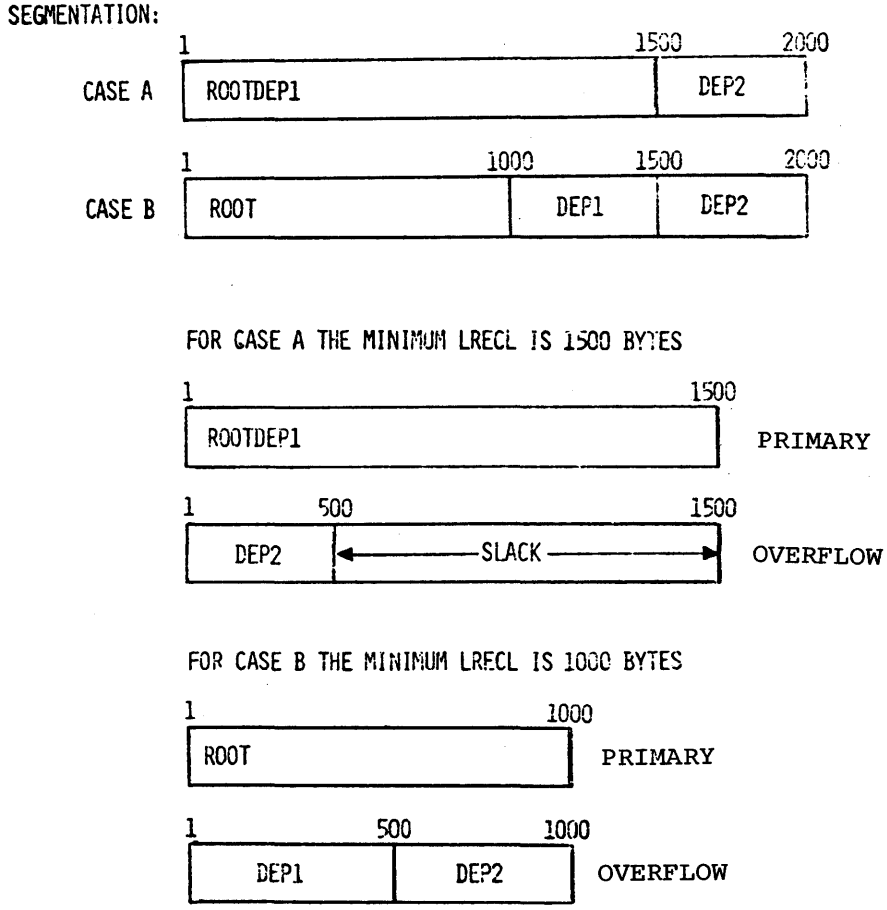
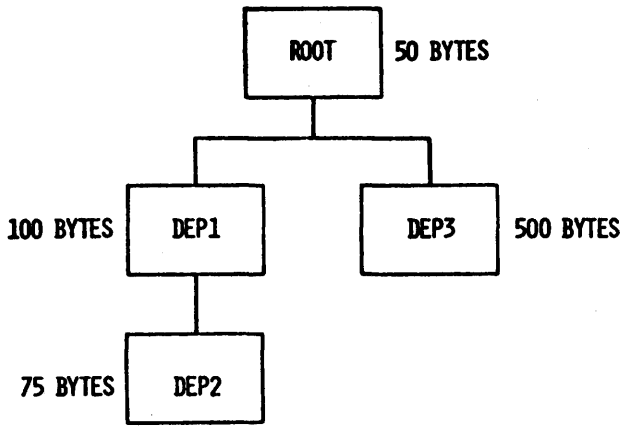


Figure 4-54. Data Base Record Segmentation Options

Since the sizes of logical records in data set groups will probably be different, multiple data set groups can also be used for better utilization of DASD space. Figure 4-55 shows a data base record for a single data set group. Again, since the logical record length must accommodate the largest segment, the minimum size of a logical record is 500 bytes. No slack remains in the overflow record. However, this choice leaves 275 bytes of slack in the primary record.



SINGLE DATA SET GROUP -- LRECL 500 BYTES

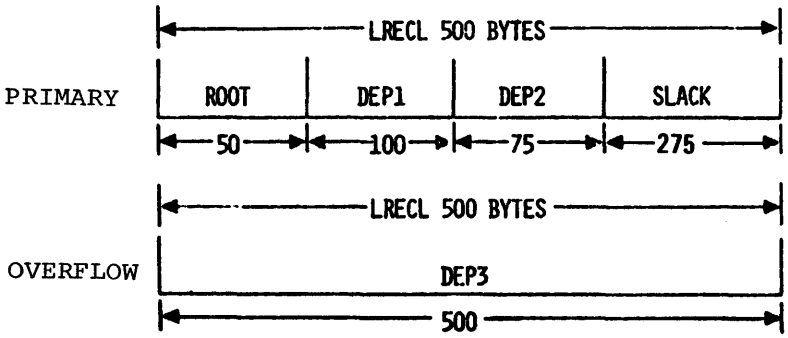


Figure 4-55. HISAM Single Data Set Group Segmentation

Figure 4-56 shows the same record as Figure 4-55, except that multiple data set groups are used. The primary data set group has a logical record size of 225 bytes. The secondary data set group has a logical record size of 500 bytes. There is no slack space, and no overflow records in the overflow data set are used. Note that 225 is not the minimum for the primary data set group; the minimum is 100, but this results in less efficient use of space.

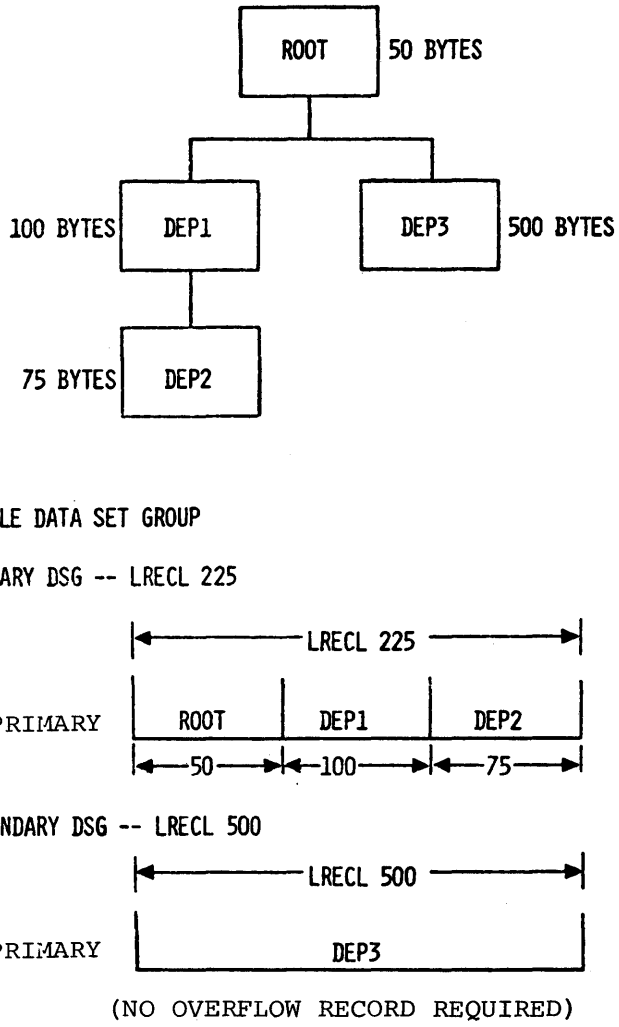


Figure 4-56. HISAM Multiple Data Set Group Segmentation

The reader shculd remember that the logical record length of the overflow data set within a data set group rust be equal tc or greater than the logical record length cf the primary data set. Both logical record lengths must accommodate the largest segment in the data set group.

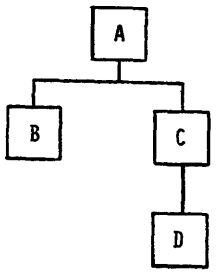
Both primary and overflcw logical records can be blocked one or multiple logical records to a physical blcck.

DESIGN TRADEOFFS

A review of the logical and physical structures supported by IMS/VS is recommended.

The IMS/VS Hierarchical Sequential organization supports the Hierarchical Sequential and the Hierarchical Indexed Sequential Access Methods (HSAM and HISAM). HSAM is based on BSAM and QSAM; HISAM, on VSAM or ISAM with an EXCP extension named CSAM (Overflow Sequential Access Method). The following discussion is based on HISAM.

In Figure 4-57, each block represents a segment type, with the block at the top referred to as the root segment (A). The other segments, B, C, and D, are dependent segments. Root segment A is also level one, segments B and C are level two, and segment D is level three.



1-255 SEGMENT TYPES

1-15 LEVELS

1 ROOT SEGMENT PER DATA BASE RECORD

0 TO N DEPENDENT SEGMENTS PER PARENT

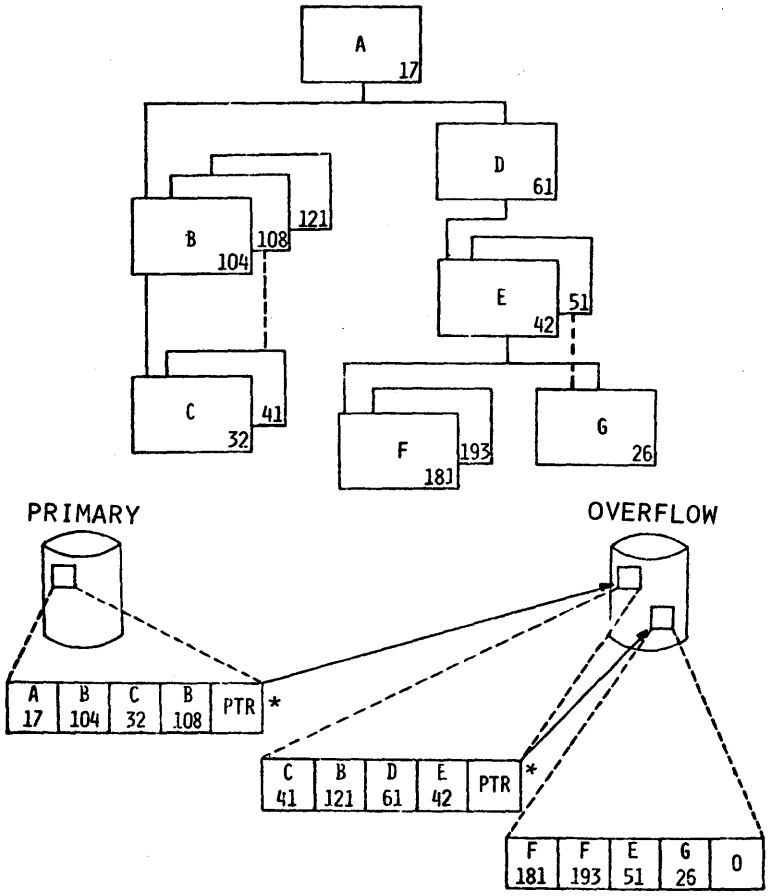
1 TO N SEGMENTS PER DATA BASE RECORD

1 TO N DATA BASE RECORDS PER DATA BASE

Figure 4-57. Data Base Structure Rules

The principal reason for the existence of dependent segments is that they give the user the ability (by variable-length records and dependent segments) to take care of information which it would otherwise be necessary to repeat from 0 to n times.

Figure 4-58 shows the physical order of storage -- top to bottom and left to right. As shown, this data base record begins in the primary data set and requires two additional logical records in the overflow data set. The two overflow records can be blocked within one overflow physical block. The technique of pointing from one record to another allows data base records within a data base to vary considerably in length, but for all practical purposes their size is unlimited.



*The pointer is at the beginning of VSAM logical records

Figure 4-58. HISAM Physical Storage -- ISAM, OSAM, or VSAM

Primary and overflow data set logical records are always blocked one or more for each physical block (see Figure 4-59). A data base record may be contained in one primary data set logical record, or it may consist of one primary data set logical record and one or more overflow logical records. The overflow logical records must be at least as large as the primary logical record. Overflow logical records may be unblocked or blocked. The number of overflow logical records within a physical block may be equal or not equal to the number of primary logical records within a physical block.

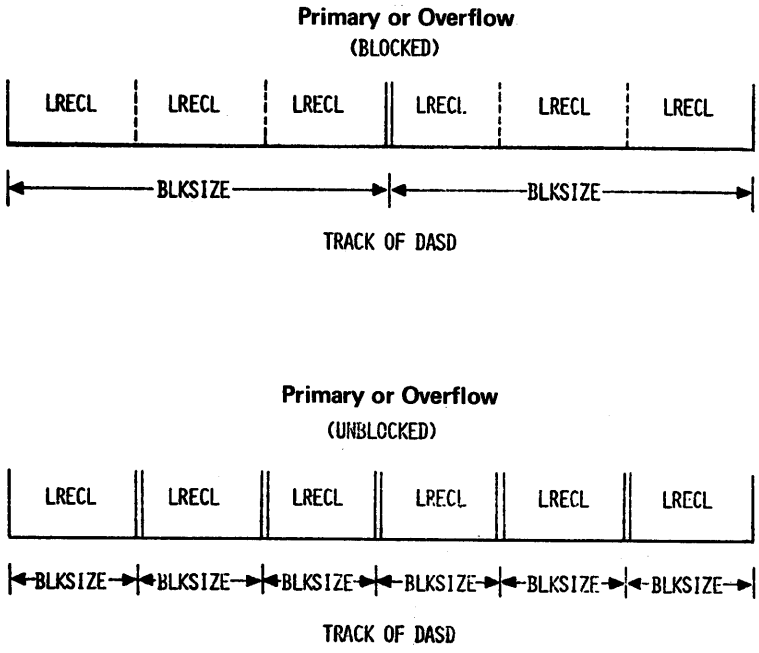


Figure 4-59. HISAM Physical Storage Blocked One or Multiple

VIABILITY OF DATA BASE DESIGN

In the design of a data base, the designer must consider the viability of that design. How can he anticipate, at least in some measure, changes such as the additions of new data, new applications for new data, new applications for existing data, discontinuance of existing data, in short, the change in the level of activity against the data?

In adding new data segment types, the simplest approach is to extend the data base to the right (see Figure 4-60). By the addition of new segment types in this manner, segments to the left and applications dealing with those segments need not be modified. An additional benefit for HISAM data bases, or HDAM and HIDAM data bases that use hierarchic pointers is that only the data base descriptions need to be regenerated. It is not necessary to unload and reload the data base. For HDAM and HIDAM data bases where the segment type being added will be pointed to by physical child pointers, it is necessary to unload and reload the data base to add physical child pointers to the prefix of the physical parent of the segment type being added.

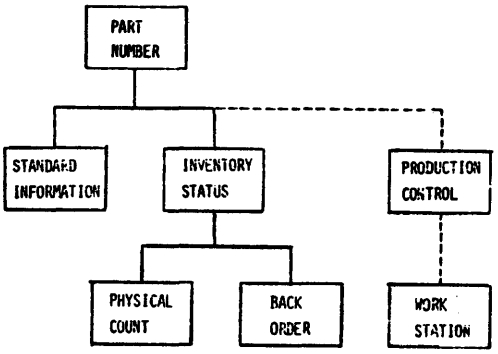


Figure 4-60. Data Structure Change -- New Segment Type Defined at End of Hierarchy

In Figure 4-61, the new data, "Production Control" and "Work Station," could be added to the data base at a different position. A disadvantage of adding new segment types in this way is that the physical code of each segment type changes. The order of segment insertion being top to bottom, left to right, such a change would require a reload of the file. Again, there would be no reprogramming required for existing application programs. One reason for this arrangement could be that the majority of the processing activity is to be against the Production Control information.

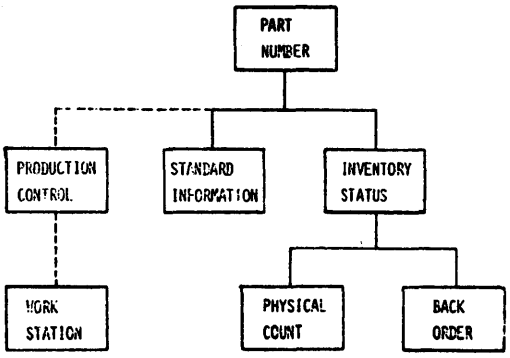
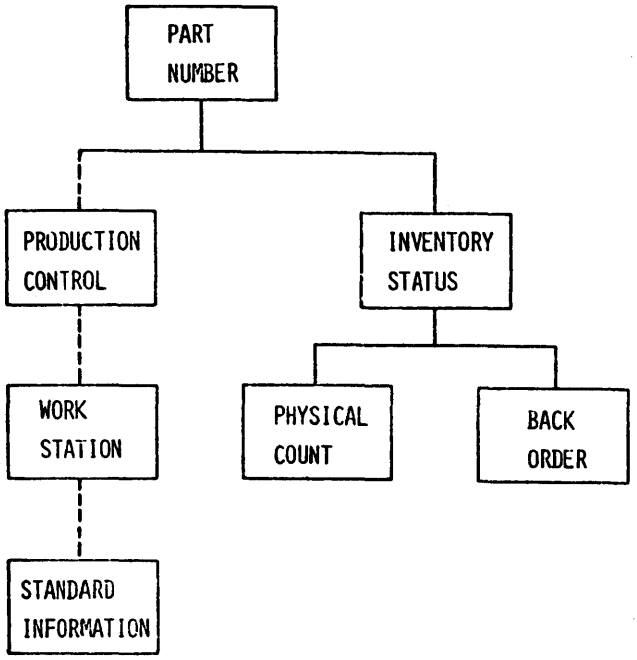


Figure 4-61. Data Structure Change -- New Segment Type Defined within Existing Hierarchy

Figure 4-62 shows an arrangement that will almost certainly impact existing application programs, by making it necessary to regenerate the PSB for any program which is sensitive to "Standard Information." The degree of modification to the application program will, of course, be a function of the type of calls made against the data base and the use made of the concatenated key feedback information. Assuming that no use is made of the concatenated key, the series of calls at the left would function properly without modification, after the PSB is made sensitive to "Production Control" and "Work Station."



GU	PART NUMBER	GU	PART NUMBER
GN	STANDARD INFORMATION	GN	

Figure 4-62. Data Structure Change -- New Segment Type Defined within a Leg of the Existing Hierarchy

It is evident that the series of calls on the right side of Figure 4-62 would not function properly. The unqualified GET NEXT call would obtain the "Production Control" segment instead of the "Standard Information" segment.

In user applications with existing data, if the activity becomes weighted to the right it may be desirable to move certain segment types, logically and physically, nearer the root. This would ensure that they were located in the record containing the root segment. In the data base of Figure 4-63, an increase in activity against the Production Control segment could make a new data base design more desirable.

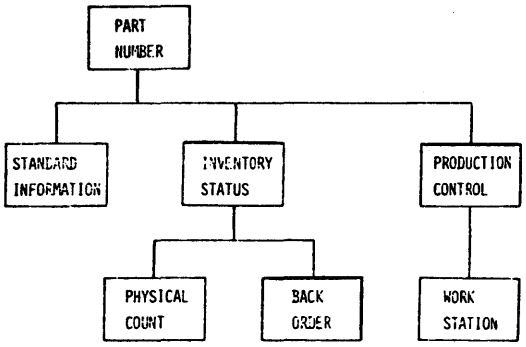


Figure 4-63. Data Base Structure -- Hierarchic Leg Independence

One solution, moving the segments logically and physically to the left, is shown in Figure 4-64. It is possible to plan for the freedom to move the legs of the data base around in this manner without affecting the functioning of application programs. If the data base and applications are designed in such a manner that each application program relates to the root and to only one leg of the data base, it is possible to manipulate the legs without impacting applications.

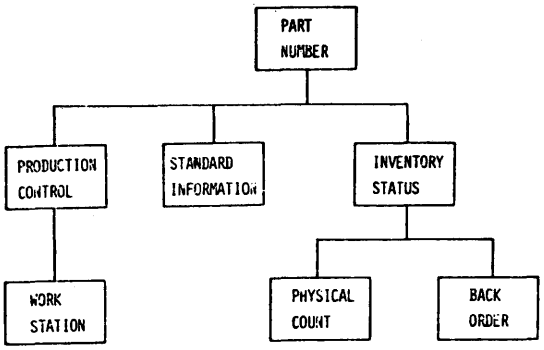


Figure 4-64. Restructured Data Base

Another solution to an increase in activity against certain segment types is to move those segment types to a secondary data set group. In this instance, however, the tradeoff against increased core buffer requirements would have to be considered.

When segments representing a particular segment type cease to exist, there is no mandatory adjustment necessary to the data base design nor is there any measurable penalty for not doing so. Eliminating segment types, however, requires a new data base description generation. If there are no occurrences of the segment type in the data base and the segment types are farthest away from the root in the top to bottom, left to right fashion, there is no need to unload and reload the data base. In Figure 4-65, "Production Control" and "Work Station" could be dropped.

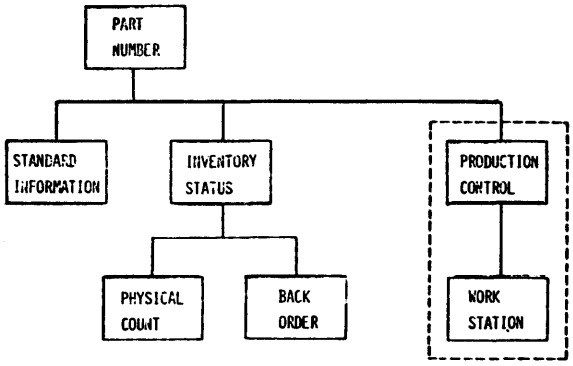


Figure 4-65. Data Base Structure -- Absence of Segment Types

Eliminating segment types which are not logically last requires that the data base be reloaded. Program specification blocks which were sensitive to the segment types being deleted would be regenerated. Since the segment types are deleted because the applications and data no longer exist, there is no need to modify the majority of remaining application programs. The extent of the modification would in all probability be only a change in some call parameters to Data Language/I.

Expanding the size of a segment can cause a change in the program (see Figure 4-66). On an individual application basis, this change can be avoided by using oversized work areas in the application program. As an example, a Data Language/I I/O area of 150 bytes could be defined when in reality the size of the segment to be operated on need only be 100 bytes. With this technique, the size of the segment could be expanded without affecting the application program. It should be noted that the size of the application program will be increased, but this added overhead is usually more desirable than the possible reprogramming effort.

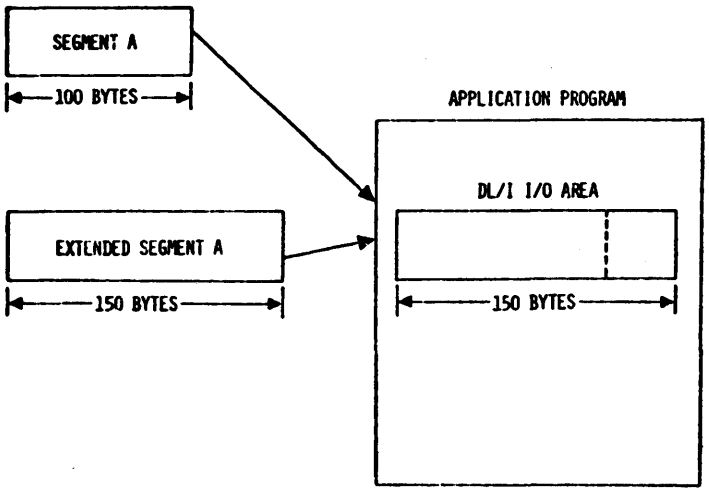


Figure 4-66. Application Program I/O Work Area Size Considerations

HIERARCHICAL DIRECT DESIGN CONSIDERATIONS

The Hierarchical Direct organization is composed of two data base access methods: HDAM and HIDAM. The HIDAM access method uses two physically distinct data bases for access and storage of the data. The INDEX data base is used to determine the sequence in which data is presented to an application program, but does not actually contain any of the data in the HIDAM data base. The HIDAM data base contains all the actual data and is physically distinct from the index. The HDAM access method uses one data base and a randomizing algorithm for accessing data.

Design Considerations for the Index of a HIDAM Data Base

The INDEX data base contains index segments which perform indexing. The content of the index segment is equivalent to the sequence field key in the root segment of a HIDAM data base. The INDEX data base used for HIDAM is composed of a single data set group which is similar to the HISAM organization.

Since the INDEX data base is basically a HISAM data base containing only index segments, the majority of the design considerations from the HISAM section apply equally well to the index. Of course, both the primary or primary and overflow data sets should have relatively high blocking factors. For example, a quarter track of a 2314 for block size would be appropriate. The HISAM unload and reload reorganization program should be run fairly frequently against an INDEX data base to reduce long OSAM chains when ISAM/OSAM are the access methods. This procedure should not require an excessive amount of time, since the INDEX data base is much smaller than the HIDAM data base it references.

Design Considerations for Data Portion of HIDAM Data Base

Considerations for the design of the data portion of the HIDAM data base involve the tradeoff between direct access space and access time. The most efficient organization for access time, when application programs do not access every segment in any data base record, is to choose the physical twin/physical child manner of relating segments of a data base record. If this option is chosen, any path through the

data base may be fcllcwed without looking at segments nct on the path. The negative aspect of this choice is that more storage is needed than if the user elects the hierarchical pointer apprcach to relate segments.

The hierarchical pointer optien reduces prefix size by stringing together all segments of a data base record, but IMS/VS must process it in much the same manner that it processes HISAM. That is, the segment on the right of the structure is at the end of the hierarchical pointer chain. All segments to the left of a desired segment have to be scanned to get to the desired segment. Therefore, this option should be used for those portions of a data base record that are normally accessed sequentially.

If root segments are often accessed sequentially, the user should probably select the bidirectional physical twin pcinter cption for root segments. If this option is chosen, and the user issues a data base call which references either implicitly or explicitly the next data base record, the index data base is not used to satisfy the request.

Design Considerations for an HDAM Data Base

The two major considerations in designing an HDAM addressing or randomizing algorithm are access time versus storage and reorganizatien considerations. Ideally, an addressing or randomizing algorithm spaces root segments across the root segment addressable area of a data base in such a manner that little storage is wasted and yet synonym chains are very short. Long synonym chains negate the savings made by not having to access an index. On the other hand, a sparse storage of root segments may waste direct access space that could be used if the organization were HIDAM.

If a data base is increasing in size, some thought should be given to reducing the problems when it must be reorganized. Some randomizing routines yield radically different results for the same set of keys when the size of the root segment addressable area is increased or decreased. If this is the case, the user is faced with the choice of loading randomly, which may be very slow, or doing an offline sort prior to reloading. Randcmizing routines can be constructed that will not seriously alter the sequence of data when the size of the root segment addressable area is changed. An example of this is the binary halving routine illustrated in the IMS/VS System Programming Reference Manual.

The user may wish to take advantage of the bytes parameter of the RMNAME= operand cn the DBD statements for a DBL generation. The use of this parameter reduces the inefficiency caused by dependent segments of a very large data base record taking excessive space in the root segment addressable area. If excessive space is used for dependent segments, other root segments are forced out of their prime blocks in the root segment addressable area and into overflow.

The use of bidirectional root segment physical twin chains is not recommended in HDAM, since roots are chained only off a root anchor point and thus do not tie the whole data base together as in HIDAM. Bidirectional pointers may cause additional processing time at insert time, since the NEXT root on the synonym chain must be updated to point back to the root being inserted.

HDAM -- HIDAM CONSIDERATIONS FOR DEPENDENT SEGMENTS

The user may wish to give some thought tc the use of additional data set groups in a HDAM or HIDAM data base to save access time and make better use of disk space. For example, a data base may contain a

segment-type which is quite bulky and infrequently accessed. This segment-type can be placed in a separate data set group, thus reducing access time among the more frequently used segment-types.

Another form of separating segments into data set groups the user may wish to investigate is separation by segment size. The Hierarchical Direct organization bit maps, which describe free space existing on blocks within the data base, contain information based on the maximum segment size within the data set group. If segments of about the same size are contained within each data set group, better space utilization may result.

Proper direct access data set block size is another factor to be considered in system performance. The larger the block size, the more data that is obtained with a single input/output operation. If the majority of the data is used, then larger block sizes have a good payoff. However, if the data is accessed randomly within a data base record and only small portions of any particular block are used, then the user has paid a penalty in terms of system channel time and core storage buffer space to access large blocks.

IMS/VS Use of BISAM/QISAM

The IMS modules concerned with management of DL/I data bases make use of the OS BISAM and QISAM access methods. These access methods are used to access data stored in OS ISAM data sets. OS ISAM data sets are used by IMS/VS to store data for HISAM data bases and for the index data base to a HIDAM data base. BISAM and QISAM are used by IMS under the following conditions:

1. BISAM is always used to manage an ISAM data set that is accessed from an IMS/VS CTL region (i.e., in a message processing environment).

2. The following rules apply to a batch processing environment only:

   a. QISAM is used to manage an ISAM data set of a HIDAM data base when the root segment of the HIDAM data base is referenced by one PCB only, and when the processing option for the root segment is retrieve or load only. In all other cases, BISAM is used to manage the ISAM data set of an index data base.

   b. BISAM is used to manage the ISAM data set of a HISAM data set group when (1) a PCB is sensitive to a logical parent that exists in the data set group, or (2) multiple PCBs are sensitive to segments within the data set group. Other data set groups will use QISAM. Note, therefore, that for a HISAM data base, BISAM may be used for some data set groups, while QISAM may be used for other data set groups.

   c. If use of both BISAM and QISAM is indicated for an ISAM data set by the rules presented above, use of BISAM takes precedence.

   d. If a user desires, for performance reasons, to cause IMS/VS to use BISAM to manage a particular ISAM data set, he can do so by ensuring the presence of one of the conditions outlined above that will cause use of BISAM. A user can cause IMS/VS to use QISAM for an ISAM data set only by ensuring the absence of any of the conditions that would cause IMS/VS to use BISAM.

e. Note that, because of the rules described above, both BISAM and QISAM may be used during execution of a particular application program.

I/O buffers for data sets using BISAM are always obtained from the IMS/VS data base buffer pool. I/O buffers for data sets using QISAM are always obtained by QISAM. If QISAM, use QISAM for read and writes; if BISAM, use ISAM or OSAM for read, OSAM for write.

INTERACTIVE QUERY FACILITY (IQF) DESIGN CONSIDERATIONS

The IMS/VS user who intends to incorporate the Interactive Query Facility (IQF) feature into his system can optimize future query performance by taking the time to understand the requirements of IQF before attempting to design new data bases. The user should, for instance, be aware of the following.

1. IQF provides two indexes, each capable of relating the values of fields to be indexed to the information necessary to locate the segment containing those values. The first index is provided for small indexed fields, and supports all indexed fields whose size is less than or equal to the size defined for the first index. The second index is provided for larger indexed fields, and supports all indexed fields whose size is less than or equal to the size defined for the second but larger than that specified for the small index. (The reader is referred to Chapter 4 of this manual for additional information on IQF's indexing capability.)

2. IQF permits naming of PCBs, and assigning synonyms to field names to distinguish fields in separate segments which have been defined in a physical DBD to have the same name. (See the "Interactive Query Facility" chapter in the IMS/VS Utilities Reference Manual.)

A number of restrictions apply to data base design for IQF. Of particular importance are the following facts:

1. An IQF query can access only one hierarchic path; that is, for any two segments used in the query, one must be an immediate or lower level dependent of the other as defined in the physical or logical DBD referenced by the PCB used for the query.

2. The parent of the lowest segment involved in a query path must be identified by a unique concatenated key within the data base used to process the query. One method that can be used to ensure compliance with this rule is to define a unique sequence field for every segment involved in a query path, except for the lowest. Root key values must be unique.

3. If a logical child is involved in a query path, the concatenated key of its destination parent must be unique. One method that can be used to ensure compliance with this rule is to define a unique sequence field for every segment whose sequence field is a component of the destination parent's concatenated key.

4. A field that is indexed by IQF can be no greater than 250 bytes in length.

5. Scattered sequence fields are not supported by IQF.

6. If KEY sensitivity is specified for a segment, the SEQUENCE field of that segment must not be used as an IQF field.

7. If a bidirectional logical relationship is implemented by using virtual pairing, and if the virtual logical child is defined, the user must reference the virtual logical child (via the SOURCE operand) when: (1) a segment is being defined in a logical DBDGEN, and (2) the segment consists of the concatenation of the virtual logical child and the physical parent of the real logical child.

8. All fields of a virtual logical child that are to be used in an IQF query must be defined by FIELD or *FIELD macro statements that refer to the data of the virtual logical child. (IQF does not automatically refer to field definitions provided for a real logical child and duplicate them under the virtual logical child at the appropriate offsets as does IMS/VS).

9. When a virtual logical child is defined, and when the user provides the virtual logical child in the input data stream provided to the IQF Utility before the corresponding definition of the real logical child, the user must provide a FIELD or *FIELD macro statement for the virtual logical child such that the last byte of the virtual logical child data is included within the range of data defined by the FIELD or *FIELD macro statement(s).

UTILITIES

DATA BASE RECOVERY

IMS/VS supplies a number of utility programs designed to provide a rapid recovery of a data base data set rendered unusable because of read/write errors. Although these programs will be used infrequently, judicious preplanning will enhance data base availability in the event of failure.

The data base recovery utility program set includes:

1. A program to create an image copy or dump of a data base.

2. A program to restore an image copy or dump of the data base.

3. Logging routines in the IMS/VS batch and control regions to record on system log tapes any changes made to a data base.

4. A program to accumulate information from system log tapes about the latest changes to a specific data base.

5. A program to rebuild the data base using, a) a previously created data base image copy, b) accumulated data base changes obtained from log tapes, and c) input from log tapes which have not been processed by the accumulation change program.

The initial consideration in the use of these programs should be the frequency of data base image creations. The amount of data base activity and size of data base will determine the intervals between data set image copy creations. Since image copy input to the Data Base Recovery Utility provides the most rapid means of recovery, the shortest interval possible between image copy time and data base recovery is desirable.

The second consideration should be the frequency of accumulation change creations. The accumulation change input provides a sorted, combined record of data base changes to be merged with the image copy. Since the sorted input is by physical block number within the data

base, the application of the accumulation change input can be accomplished almost as rapidly as an image copy only. In addition, the accumulation change utility operates independently of IMS/VS, allowing the log changes to be accumulated without impacting data base availability.

Since log change input is processed chronologically, random access to the data base being reconstructed is quite probable. The same physical record may be accessed multiple times in a single recovery. It becomes readily apparent that the accumulation of data base changes from log tapes with the accumulation program will greatly enhance performance of the data base recovery.

See the IMS/VS Utilities Reference Manual for additional information regarding the Data Base Recovery program set.

DATA BASE REORGANIZATION

Data base reorganization utility programs provide a convenient means for achieving physical and logical reorganization of IMS/VS data bases. Use of these facilities allows:

- Recovery of direct access space occupied by deleted segments in a HISAM data base
- Reorganization of a data base when the physical sequence of its segments has become different from its logical sequence because of insertion and/or deletion of segments.
- Physical and logical restructuring of a data base to better meet the requirements of IMS/VS application programs. Examples of restructuring of a data base include changing a data base access method and organization from HISAM to HIDAM, changing the physical placement of a segment type from one data set group to another, addition of segment-types, and addition or deletion of physical and/or logical intersegment relationships.

The details of data base reorganization are provided in the IMS/VS Utilities Reference Manual.

Data bases may be reorganized separately, or several may be reorganized concurrently. If nonreorganized data bases are logically related with direct addresses to those being reorganized, then these must be scanned for all logical children whose logical parents are in one of the reorganized data bases. Programs are supplied to scan the nonreorganized data bases, and where necessary update their direct address relationships to a reorganized data base.

UTILITY CONTROL FACILITY

IMS/VS also supplies the Utility Control Facility (UCF). The UCF directs the data base initial load, reorganization, recovery, and change accumulation utilities, and provides for restart processing after a utility failure or a user request to end processing.

The UCF provides a method of performing most data base utility operations and maintenance in preparation for recovery and reorganization under the control of one job, one step, and in one scheduling. It handles the data base recovery and data base data manipulations in reorganization, the combining of data base data changes into composite change records in change accumulation, and backup copy processing. Restart processing is invoked by an EXEC parameter or a

control statement, and normal processing is directed by control statements.

See the IMS/VS Utilities Reference Manual for a full description of the UCF.

Reorganization Interval

The IMS/VS statistics programs provide data that can help systems operation personnel determine whether a data base should be reorganized. Determination of an appropriate interval at which a data base is to be scheduled for reorganization depends, to a large extent, on systems operation personnel's knowledge of the overall activity on the data base (that is, the number of segment additions and deletions), of the physical organization of the data base, of the relationship of the data base to other data bases, and of the installation's planned use of the data base in the future.

Most data base reorganizations are done to recover space occupied by deleted segments and/or to physically resequence segments in their logical order. The number of segment insertions and deletions can be determined from data provided by the application accounting report, and the distribution of transaction response times as shown in the transaction response report. When segment chains become long, and when they involve segments that are in different areas of a storage device, response times tend to increase. Growing response times can indicate a need for data base reorganization.

Frequency of reorganization should be considerably less for HDAM and HIDAM than for HISAM data bases. HDAM and HIDAM both reuse space freed by deleted segments and both attempt to place inserted segments physically near their logically related segments (that is, near segments to which they are chained by physical child, physical twin forward, or hierarchical forward pointers).

Reorganization of HISAM Data Bases

Three methods can be used to reorganize HISAM data bases. A specially written user program can be provided, the IMS/VS-provided HISAM reorganization utilities can be used, or the IMS/VS-provided HD reorganization utilities can be used.

The first method must be used when you want to accomplish data base structural changes that cannot be accomplished through the use of the HD reorganization utilities. The special program involves use of GET NEXT calls to retrieve segments from a data base, use of special user routines to accomplish the desired changes to the data base, and use of INSERT calls to reload the data base. Care must be taken in writing a special reorganization program so that concatenated key pointers, used for logical relationships, are properly maintained.

The second method uses the HISAM reorganization utilities and should be used whenever a HISAM data base is to be reorganized with no changes. The HISAM reorganization is accomplished by a rapid unload/reload program that references data on a physical data base block level. The utilities drop deleted segments and then restore segments to the data base so that their physical sequence is the same as their logical sequence. External pointers that reference segments within the HISAM data base are unaffected, since they must be concatenated key pointers. Pointers within the HISAM data base that reference segments in other data bases are affected only if the other data bases are reorganized and if the pointers are direct pointers. Pointers contained in segments

stored in the HISAM data base can be updated as described in the IMS/VS Utilities Reference Manual.

If structural changes are required, the third method should be used. This is described under Reorganization of HDAM and HIDAM Data Bases, following.

Reorganization of HDAM and HIDAM Data Bases

Two methods can be used to reorganize HDAM and HIDAM data bases. The first method involves use of GET NEXT and INSERT calls with a user-written application program as described above.

The second method is to use the IMS/VS-provided HD reorganization unload and reload utilities. These utilities use unqualified GET NEXT calls to sequentially unload segments from the data base to be reorganized. Data is appended to each segment to resolve logical relationships after doing the reload. After unload is completed, the segments are reloaded using INSERT calls. At reload time, a check is made to determine if a segment is involved in logical relationships. If so, data describing the logical relationships is written to work tapes for later use in updating the logical relationship pointers. Note that the HD unload and reload utilities can be used to reorganize HISAM data bases, but that performance does not equal that of the rapid unload/reload utility. However, if structural changes are required, the HD reorganization utilities must be used.

If the data base has logical relationships, the HD reorganization utility must be used to reorganize the data base.

The reload utility also provides statistical data that can be used as described in the IMS/VS Utilities Reference Manual.

IMS/VS DATA BASE SPACE ALLOCATION

When direct access storage is required for a data base, the amount of space needed and the device type must be specified. IMS/VS follows the same approach as OS/VS.

The amount of space required can be specified in terms of blocks, tracks, or cylinders. If it is desired to maintain device-independence across direct access types, space requirements should be specified in terms of blocks. Otherwise, if the request is in terms of tracks or cylinders, such items as their capacity must be considered. ISAM data sets must be allocated by cylinder. The amount of space is supplied in the DD statement for the data set.

Allocating the space for an IMS/VS data base that uses ISAM and OSAM (HISAM and INDEX) is similar to allocating space for an operating system indexed sequential data set; similar because an operating system data set can be divided into three areas, prime, index, and overflow. The three areas of an IMS/VS data base are index, prime, and overflow. Each data set group of a HISAM data base requires a primary and overflow data set to be allocated.

Allocating the space for an IMS/VS data set that uses only OSAM (HDAM and HIDAM) is similar to allocating space for an operating system direct (BDAM) data set. Each data set group of an HDAM or HIDAM data base requires one OSAM data set to be allocated.

DBD generation computes, from the user's definition of segment frequency, the logical record length and/or block size of a data base.

It considers the device and rounds to the next higher one-fourth track, one-third track, one-half track, or full track.

DBD generation also computes from the user's definition of segment frequency the space allocation requirements for a HSAM, HISAM, or INDEX data base.

For use by Systems Operation personnel, IMS/VS has two parameters that can be inserted when a DBD generation is performed. These provide an additional means of specifying the LRECL or RECORD and blocking factor (BLOCK) for a data set with a data base. Instead of DBD generation specifying the LRECL, it can be overridden by specifying the RECORD and the BLOCK parameters in the DATASET control card. Refer to the IMS/VS Utilities Reference Manual for details.

ALLOCATION CONSIDERATIONS

Space allocation should be considered with regard to the data base structure, the application programs that will access that structure, and the tools of IMS/VS DBD generation. The following discussion deals with an IBM 2314 Direct Access Storage Facility.

When an IMS/VS data base is loaded on an IBM 2314 Direct Access Storage Facility, it is necessary to allocate space for that data base with JCL data definition statements. The creation of a HISAM or INDEX data base may require up to three DD statements, one each for the index, prime, and OSAM overflow areas. This discussion should provide assistance in initially determining the amount of space to allocate to these areas for any specific application.

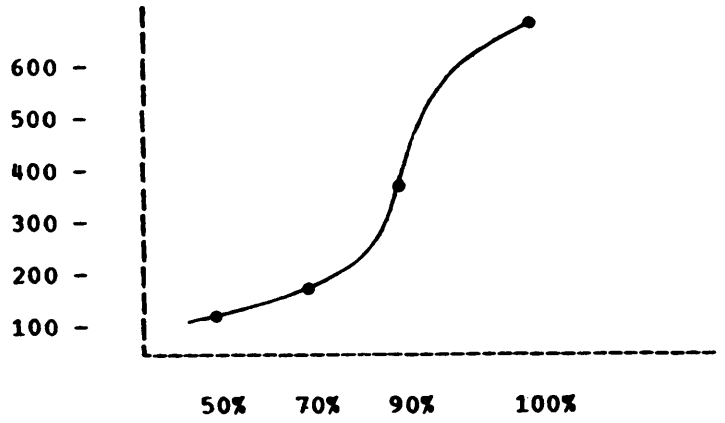


Figure 4-67. Logical Record Length Distribution

The graph in Figure 4-67 indicates that 50% of the logical records are 150 bytes or less in length, 70% of the logical records are 200 bytes or less in length, and 100% of the logical records are 600 bytes or less in length.

With fixed-length ISAM, it is necessary to establish a fixed value for LRECL in the prime area. If a value of 600 bytes is selected for LRECL, all logical records can fit in the prime area. However, 90% of the records then have at least 100 bytes of slack or wasted space; 70% of the logical records have at least 400 bytes of unused space.

In this example, if a value less than 600 bytes is selected for the LRECL value, the ISAM prime area is not capable of holding all the logical data base records. Those dependent segment occurrences that do not fit in an ISAM prime logical record are housed in OSAM overflow records. Therefore, the determination of an ISAM prime logical record length must consider the tradeoff between storage in the ISAM prime area and in the OSAM overflow area.

To determine a best balance between ISAM prime and OSAM overflow, the following points must be considered:

1. Access to data that is wholly contained in the prime area is more rapid than access to data contained in the two areas. Access is even slower to those logical data base records that require more than one overflow record.

2. Disk space allocated to OSAM overflow can be used to hold segment occurrences that overflow from any logical data base record. Unused space in the prime area is tied to specific roots.

3. Records may be blocked in the prime area and in the OSAM overflow area. The logical record length in the OSAM block must be equal to or greater than the ISAM logical record length in the same data set group.

4. The nature of the accesses to the large logical data base records also has an important effect. If the large logical data base records are highly used, the value of the prime LRECL should be increased to completely house more logical records, and the total size of overflow should be reduced. If the large logical data base records are infrequently accessed, an opposite shift should be made to increase the use of OSAM overflow.

Considering these relevant factors for a specific data base, a percentage balance must be established between the ISAM prime and the OSAM overflow. For example, it may be best, in the context of optimizing space and time utilization, that 90% of the logical data base records completely fit in the prime area and 10% require some OSAM overflow storage. After this percentage is selected, the frequency of dependent segment occurrences is developed for the 90% percentile of the parent segments. The 90% is an estimated value for this specific data base.

When space is allocated for a data base which uses only OSAM (HDAM or HIDAM) for data segment storage, DBD generation selects the appropriate physical block size for storage. Since space within physical blocks can be shared by segments from different data base records, logical record definition is not utilized.

The number of physical blocks, tracks, and/or cylinders which should be defined in the JCL statements for any IMS/VS data base allocation can be estimated in the following manner:

- Calculate the average data base record length (sum of segment lengths times frequency).

- Calculate the number of logical records and physical blocks (HISAM or INDEX) or physical blocks (HDAM or HIDAM) required to store a data base record.

- Multiply the number of physical blocks per data base record times the number of data base records.
- The result should provide an estimate for minimum space allocation. In addition to this value, space must be provided for additions to the data base after initial load.
- If distributed free space was specified in the DBD, then multiply the minimum space allocation by:

$$\frac{1}{1-\frac{fspf}{100}} \times \frac{fbff}{fbff-1}$$

where:

$2 \leq fbff \leq 100$ or $fbff = 0$ and $0 \leq fspf < 100$

See "Dataset Statement" in the IMS/VS Utilities Reference Manual for more information on fbff and fspf.

INTERACTIVE QUERY FACILITY DATA BASE SPACE ALLOCATION

The IQF (Interactive Query Facility), available as a feature for IMS/VS users, includes its own utility that provides the following functions.

- Allocation and data generation for the IQF system data base.
- Allocation and initialization for the IQF phrase data base.
- Allocation and/or data generation for either or both of the IQF index (QINDEX) data bases.
- Generation of PSBs (which describe the IQF and user data bases accessible through a given transaction code or set of codes) and the DBDs for the IQF data bases.

The IQF utility is used for three basic types of jobs -- System Creation, Index Creation, and Index Modification.

In a system creation run, the IQF system data base is allocated and loaded with data from the user's DBD and PSB decks, augmented with additional IQF input. The IQF phrase data base as well as the IQF index data bases are allocated during this run. The index data bases can also be loaded at this time.

An index creation run examines all occurrences of specified user data base fields, and loads the corresponding indexes with references to all occurrences of each value. For large indexes, the user should consider breaking the index load into separate runs; this is desirable to avoid a complete rerun if any failure occurs.

During an index modification run, previously allocated index data bases can be selectively updated by examining the user's data bases and recording the appropriate index values. At the same time, the capability to index specified user data base fields can be added or deleted.

Additional information on the IQF Utility is provided in the IMS/VS Utilities Reference Manual.

SPACE ALLOCATION GUIDELINES FOR IQF PROCESSOR DATA BASES

The direct access storage requirements for the IQF processor data bases vary with each installation. Guidelines for estimating size are provided below.

1. The system data base (sometimes referred to as the Field File) is a HISAM data base.

   The formula for the HISAM data portion is:

   (PSB * 12) + (PCB * 320) + (FLD * 86) + (SYN * 36) + 12 = number of bytes required

   The formula for the INDEX portion is:

   (PCB * 54) + 12 = number of bytes required

   where:

   PSB is the number of PSBs defined to IMS/VS for IQF.

   PCB is the number of PCBs sensitive to IQF (*QPCB statements in IMS PSBGEN).

   FLD for physical DBDs, the total number of FIELD and *FIELD statements in each segment referred to by a SENSEG statement in all PCBs sensitive to IQF. For logical DBDs, the total number of FIELD and *FIELD statements, determined by the number of FIELD and *FIELD statements in the corresponding physical segments.

   SYN is the actual number of synonyms defined in *QFIELD statements.

   The formula for calculating the space requirements for the IQF system data base creation utility program work data sets are listed below. The IQFFC procedure may require revision of the space allocation for the following DD statements:

   - UTDBD FLD x 96 bytes
   - SSYNIN SYN x 52 bytes
   - SSYNOUT SYN x 52 bytes
   - SPCBIN SENSEG x 44 bytes
   - SPCBOUT SENSEG x 44 bytes

- SWRKIN PCBFLD x 96 bytes
- SWRKOUT PCBFLD x 96 bytes
- UTSPL 10 x 32K maximum

where:

PCBFLD is the largest sum of FIELD and *FIELD statements defined by each SENSEG within one PCB.

SENSEG is the number of segments sensitive to IQF.

Sort work space allocations for SSYN, SPCE and SWRK should be increased accordingly. Check the OS/VS documentation on sort/merge for allocation algorithms.

2. The phrase data base is a HIDAM data base.

   The OSAM portion is determined by:

   Estimated number of phrases to be defined x 201 = number of bytes required

   The INDEX portion is determined by:

   Estimated number of phrases to be defined x 20 = number of bytes required

3. In the IQFIU utility procedure, space allocation for temporary data sets HOLDS and HOLDL in step IU1, and SHRTOUT and LONGOUT in step IU2, is based on a volume of 1000 records of 50 bytes each. An installation which will be indexing a volume of records in excess of this should use the following formulas to calculate the size (in bytes) of the temporary data sets required to hold the records being indexed.

   HOLDS and SHRTOUT (Formula 1) $B = (M + S + 6) * F * N$

   HOLDL and LONGOUT (Formula 2) $B = (M + L + 6) * F * N$

   where:

   B is the number of bytes required.

   M is the maximum root key length (MAXRTKEY).

   S is the maximum length of the short index value (INKEYLEN, value2).

   F is the frequency of the root in the IMS/VS data base.

   N is the number of fields being indexed.

   L is the maximum length of the long index value (INKEYLEN, value3).

- The result of these calculations should be translated to tracks or cylinders. (See OS/VS Data Management Services, GC26-3783.)

- In addition, corresponding adjustments may be necessary to the SORT work data sets SHRTWK01 - 03 and LONGWK01 - 03 used for intermediate storage. If more SORT intermediate storage is needed, the user has the option of increasing the space parameter of the existing work data sets, allocating up to three more work data sets for both SHRT and LONG, or a combination of the two. (See OS/VS sort/merge documentation for details on calculating intermediate storage space requirements.)

The IQF index data bases are HISAM files. Space requirements for these data bases depend on several factors. These factors include the logical record size, normal quantity of records in the indexes, frequency of updating, and frequency of file reorganization. For example, if the data base is updated several times before being reorganized, the space required is greater than if a reorganization were run following each update. To determine the minimum size (in bytes) of the short index data bases, use Formula 1 for the short index and Formula 2 for the long. N, in this case, should represent the normal number of indexed fields to be contained in the index data bases. Guidelines for determining the cylinders or tracks needed for the ISAM and OSAM portions required by the IQF index data bases are provided earlier in this chapter (see discussion of "IMS/VS Data Base Space Allocation").

CHAPTER 5. DESIGN CONSIDERATIONS FOR THE MULTIPLE SYSTEMS COUPLING (MSC) FEATURE

This chapter describes the Multiple Systems Coupling (MSC) feature and contains design considerations for its use. Familiarity with the preceding chapters of this manual is assumed.

The MSC feature provides the ability to connect geographically dispersed IMS/VS systems in such a way as to allow programs and operators of one system access to programs and operators of the connected systems. Communication between two or more (up to 255) IMS/VS systems running on any combination of OS/VS1 and OS/VS2 MVS systems in one or more System/370 CPUs is permitted.

The MSC feature also provides a way to extend the throughput of an IMS/VS system beyond the capacity of a single System/370 Model 168MP. This is possible if the IMS/VS applications can be partitioned among systems such that either:

- Applications execute in more than one system with data base contents split between systems (horizontal partitioning).
- Applications execute in one system with the complete data base, that they reference, attached to that system (vertical partitioning); the transactions can originate in any system.

RELATIONSHIP OF A DB/DC/MSC SYSTEM TO A SINGLE DB/DC SYSTEM

In addition to the considerations for a single DB/DC system described in the previous chapter, major design considerations for MSC are: determining how to distribute functions among systems and obtain acceptable performance, defining valid connections between the systems, and implementing operating procedures for maintaining consistent connections.

The flow of a transaction through processing in a multisystem environment requires a few steps in addition to those required in a single system environment. The additional steps can be identified by comparing Figures 5-1 and 5-2.

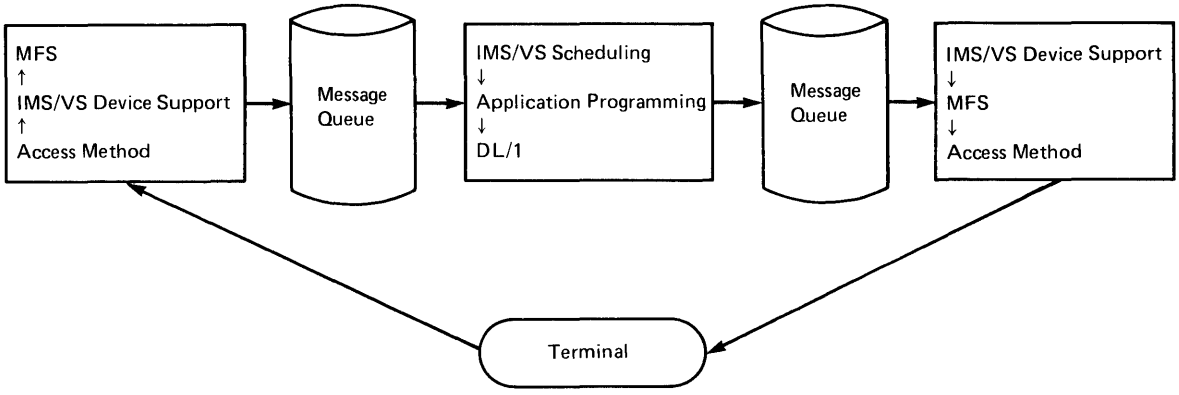


Figure 5-1. Single DB/DC System Transaction Flow

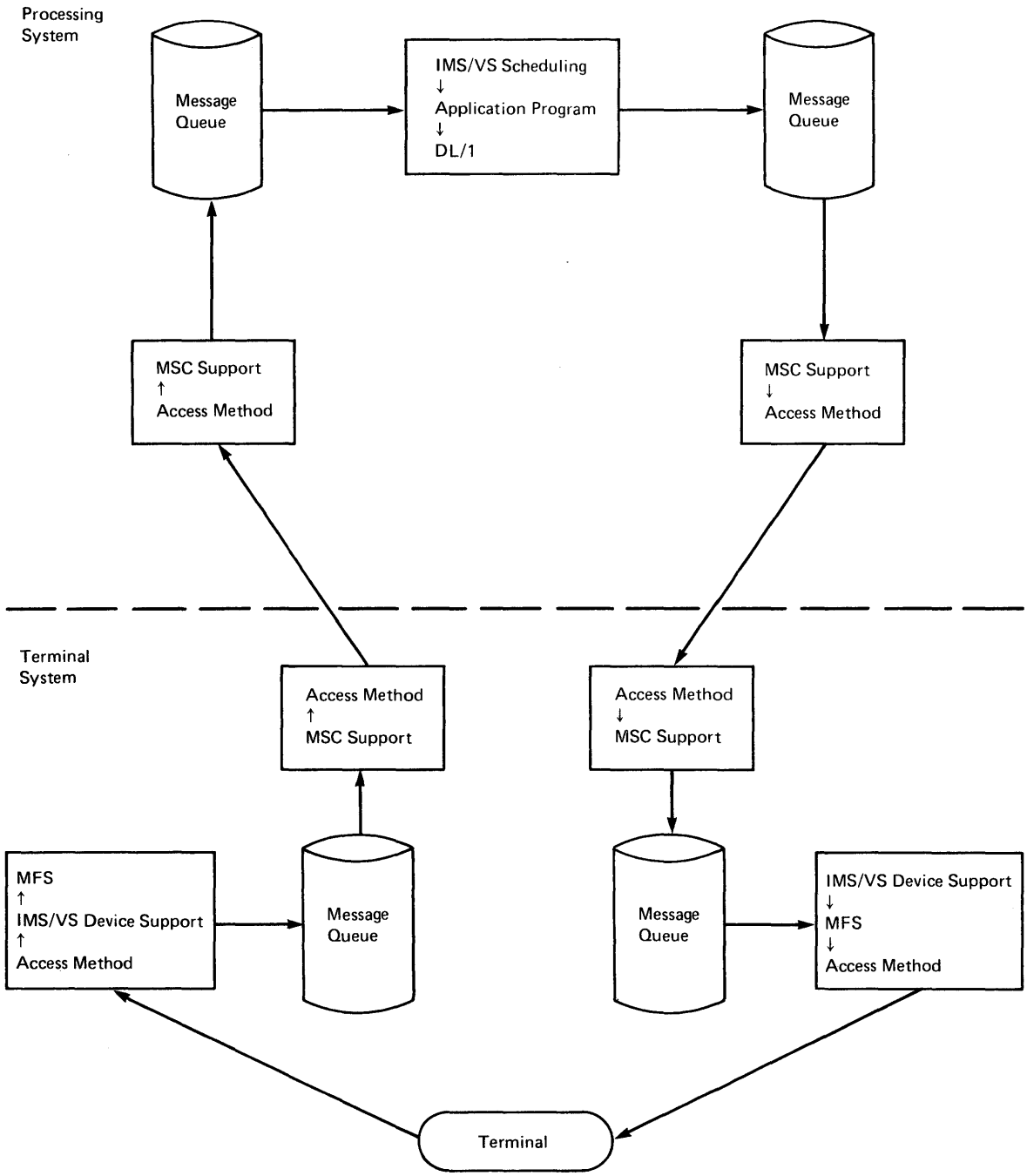


Figure 5-2. Multiple DB/DC Systems Transaction Flow

OVERVIEW OF THE MSC FEATURE

This section presents a general description of the MSC feature and introduces MSC terminology. To do this, an example is used of a multisystem environment consisting of three systems -- System A, System B, and System C. Figure 5-3 shows the sample environment.

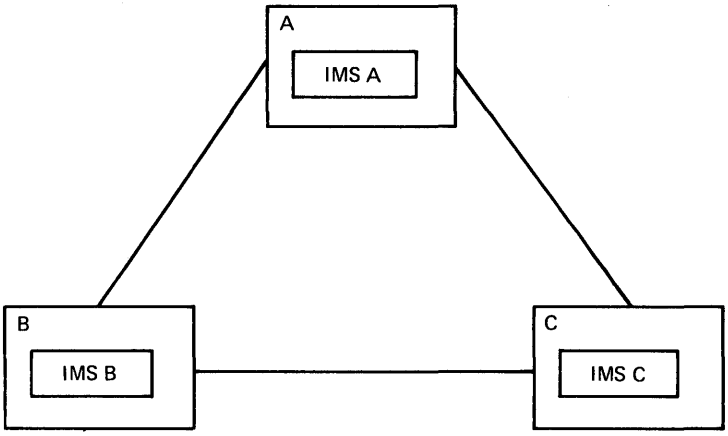


Figure 5-3. A Sample Configuration of Three Systems

In a multisystem environment, the term local system is used to identify a specific system within the multiple configuration. All other systems are considered remote systems. For example, if we were considering the activities required by System B when it receives a message, System B is the local system and Systems A and C are remote systems.

When the multisystem environment is defined, the items defined for each local system include:

- All transactions received by and/or processed by that system
- All logical terminals attached to this system and all logical terminals in remote systems that are referenced by transactions processed in this system or by terminal operators
- The physical and logical connections between this system and the remote systems that share in the processing of the specified transactions

LINKS

The connection between two systems is called a link. All links must be defined during the IMS/VS system definitions for each IMS/VS system. There are two types of links, physical link and logical link. A physical link is the actual hardware connection. A logical link is the mechanism through which a physical link is related to the transactions and terminals that make use of that physical link. The assignment of a logical link to a physical link can be specified during system definition, or made dynamically during system execution.

Physical Link

A physical link is defined by the MSPLINK macro instruction. Three types of physical links are allowed by the MSC feature:

- Binary synchronous communication (BSC) line
- Channel-to-channel (CTC) adapter
- Main storage-to-main storage (MTM)

Only the BSC line and CTC adapter represent actual hardware links. The MTM link is a software link between IMS/VS systems running in the same System/370, and is intended primarily for backup and testing purposes. Physical link buffer sizes must be equal and if BSC is chosen for the physical link, CONTROL=YES must be specified for one physical link and CONTRCL=NO must be specified for the other physical link. Figure 5-4 summarizes very simply the three types of physical links.

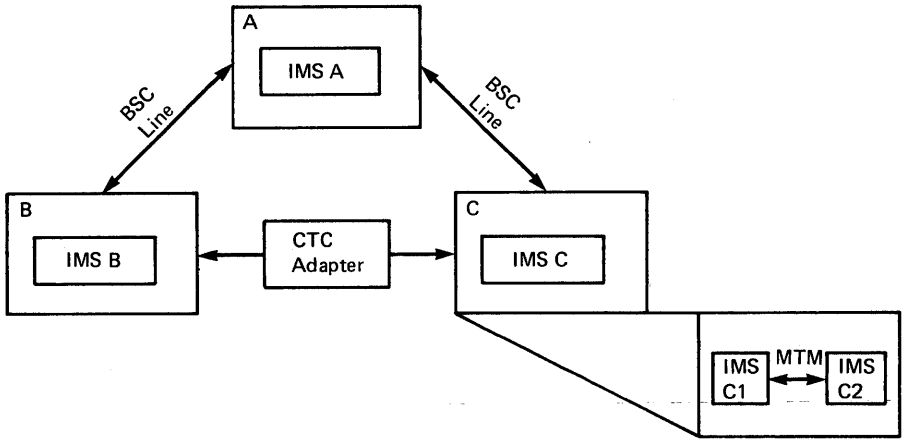


Figure 5-4. Summary of Physical Link Types

One System/370 CPU may be linked to another CPU by one or more of each physical link type, as shown in Figure 5-5. The MTM link is recommended when two or more IMS/VS system reside in one System/370.

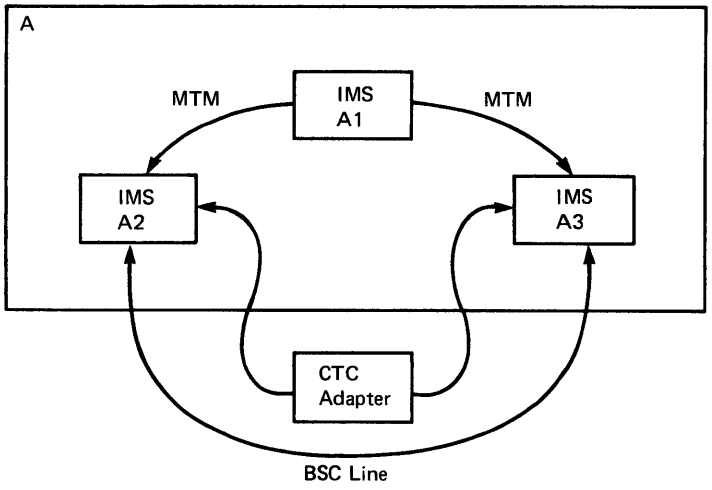


Figure 5-5. Multiple Physical Links in One System/370 CPU

Or, multiple links may exist between CPUs. See Figure 5-6.

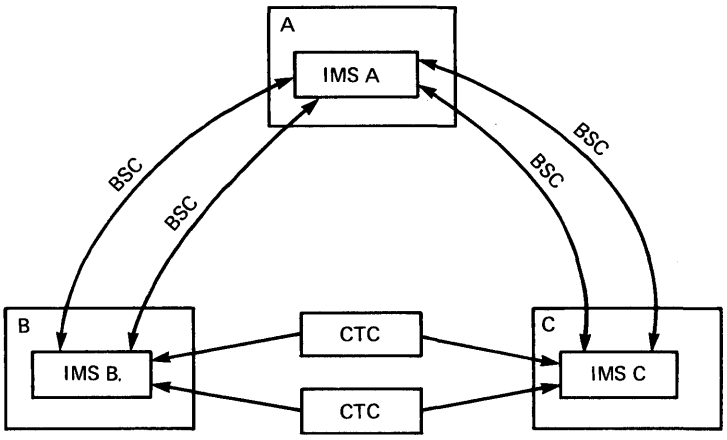


Figure 5-6. Multiple Physical Links in Multiple System/370 CPUs

Logical Link

A logical link is defined by the MSLINK macro-instruction. A logical link relates a physical link to the transactions and terminals that can use that physical link. Each system in a multisystem configuration has one or more defined logical links. Two IMS/VS systems defined to communicate with each other, each through a specific logical link, are called partners. To establish connection between two IMS/VS systems, each partner must have a logical link definition. The two logical link definitions must specify the same partner identifications and be assigned to the same physical link. IMS/VS system definition assigns a number to each defined logical link. Logical link numbers are assigned sequentially, beginning with 1, in the order in which the links are defined. A logical link can be reassigned to a different physical link but the two systems must always communicate through a logical link partnership.

IMS/VS system definition does not require that a physical link be specified for each logical link. The assignment of the physical link to the logical link can alternatively be made online using the IMS/VS /MSASSIGN command. There can be no communication between partners until the assignment is made.

A logical link definition must include one or more logical link paths. Logical link paths are described in this chapter under the heading "Logical Link Path."

If any logical terminals in a remote system are referenced by messages originating in the local system, the logical link definition must also include NAME macros to identify those remote logical terminals.

Figure 5-7 summarizes the relationships of one physical link to one logical link to multiple logical link paths. Although more than one logical link can be defined with each physical link, only one logical link to physical link relationship can be assigned at any one time.

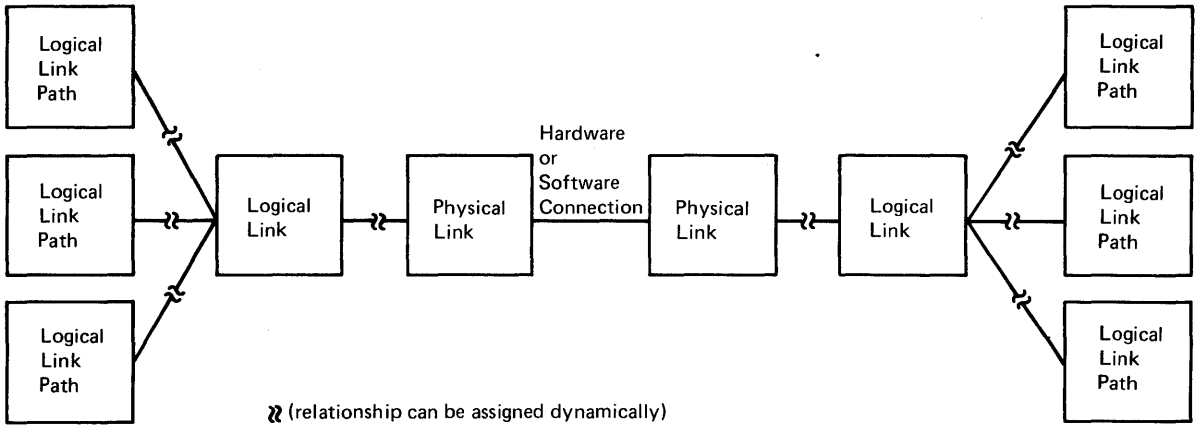


Figure 5-7. Relationship of Physical Link to Logical Link to Logical Link Path

MESSAGE ROUTING

The message routing function of the MSC feature supports transaction processing by more than one system, transaction processing by more than one application program in the same or different systems (by program-to-program switches), and message switches between terminals in the same or different systems. Conversational processing is available to any system in a multisystem configuration to the same extent as in a single system.

Routing Path

The route through which IMS/VS passes a message from its origination through processing is called a routing path. One or more systems may be included in a routing path. The routing path defined depends on the multisystem configuration and the functions assigned to each system.

Logical Link Path

The path between any two systems is called a logical link path. One or more logical link paths must be defined for each logical link. A logical link path is defined in the MSNAME macro and specifies a system identification for the system where messages using this path are to be processed and specifies a system identification for the system being defined.

Each system in a multisystem configuration has one or more unique system identifications (SYSID) ranging from 1 to 255. SYSID assignments are implicit based on the logical link paths defined by MSNAME macros. For example, here are two MSNAME definitions:

- MSNAME SYSID=(2,1)
- MSNAME SYSID=(3,1)

The first definition above says that messages using this logical link path are processed in the remote system whose local SYSID is 2. The second definition above says that messages using this logical link path are processed in the remote system whose local SYSID is 3. By way of these definitions, IMS/VS system definition assigns SYSID 1 to the system being defined and recognizes two remote systems with SYSIDs

of 2 and 3. If a third path were defined with SYSID=(5,4), IMS/VS would also assign SYSID 4 to the local system.

Each system maintains a SYSID table containing all logical link paths defined in that system. The size of this table is determined by the highest SYSID defined and it will contain gaps for SYSIDS that are not defined.

Transactions are assigned to logical link paths in the APPLCTN macro definition. Consider the following application definitions, each with one transaction code defined:

```
APPLCTN     PSB=A
  TRANSACT  CODE=A
APPLCTN     PSB=B,SYSID=(2,1)
  TRANSACT  CODE=B
APPLCTN     PSB=C,SYSID=(3,1)
  TRANSACT  CODE=C
```

The SYSID keyword identifies the logical link path to be used for the transactions associated with the application. Transaction A is considered a local transaction since the absence of the SYSID keyword indicates transaction A is totally processed by the system being defined. Transactions B and C are remote transactions. Relating the application definitions to the previous MSNAME definitions, IMS/VS would return via SYSID 1, responses from transactions B and C to the system being defined unless the application program specified an alternate destination for the response.

Since inconsistencies among SYSID definitions may exist between various system definitions, IMS/VS provides offline and online methods of verifying SYSID assignments. The Multiple Systems Verification utility is provided for offline execution to verify the consistency of definitions prior to attempting online system execution. Using this offline utility, consistencies among all systems in the multisystem configuration can be verified in one execution. You should use this utility to verify the MSC configuration each time an individual system is redefined. The /MSVERIFY command is available for online use to identify and display inconsistencies between two systems.

Logical Destinations

Message routing is accomplished by logical destinations, as it is in a single system environment. A destination is either a logical terminal or a transaction code. It is considered a local destination if it resides in the local system and a remote destination if it resides in a remote system. In a multisystem environment, each system knows (by way of system definition) all local destinations and all remote destinations that may be referenced in this system. IMS/VS system definition requires that all local and referenced remote destinations be defined with unique names.

Message routing is automatic according to the defined scheme unless routing exit routines are employed for dynamic routing control. Routing exit routines are described later in this section.

Input and Destination Systems

To introduce the terms used to describe message routing, let's look again at the sample configuration. Assume a message with a transaction destination is entered into Terminal X attached to System A. The transaction is defined to be processed by System B with its reply to be returned to the originating terminal.

Refer to Figure 5-8. On input, System A is considered the input system because the input terminal (Terminal X) is attached. System B is considered the destination system. The message is considered a primary request until processed. If the application program inserts a message with a transaction code destination during processing, this message is considered a secondary request. A message sent to the input terminal by an application program is called a response.

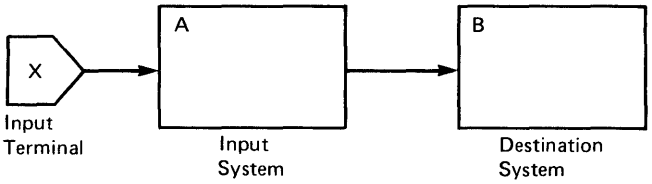


Figure 5-8. Input Terminal and Input System on Input

Refer to Figure 5-9. On output, System A is considered the destination system and Terminal X is considered the destination terminal.

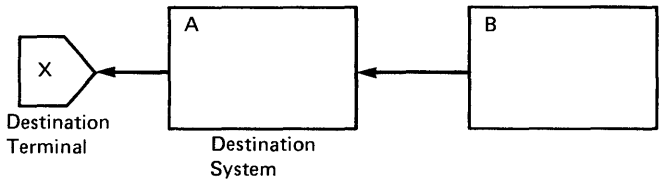


Figure 5-9. Destination Terminal and Destination System on Output

Refer to Figure 5-10. Assume the transaction were defined to originate in System A but be processed by System B with its output being sent to Terminal Y attached to System B. In terms of message routing, this example is the same as the previous one for input. For output, however, System B is considered the destination system and Terminal Y is considered the destination terminal.

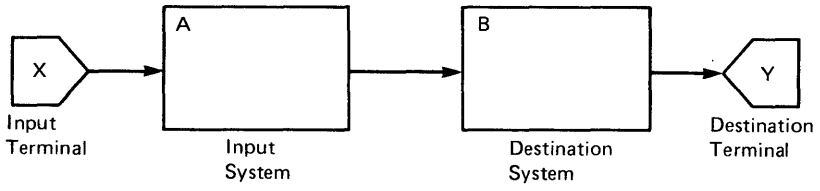


Figure 5-10. Input from and Output to Different Terminals

Intermediate System

Another term related to message routing is intermediate system. Four systems linked as shown in Figure 5-11 illustrate an intermediate system.

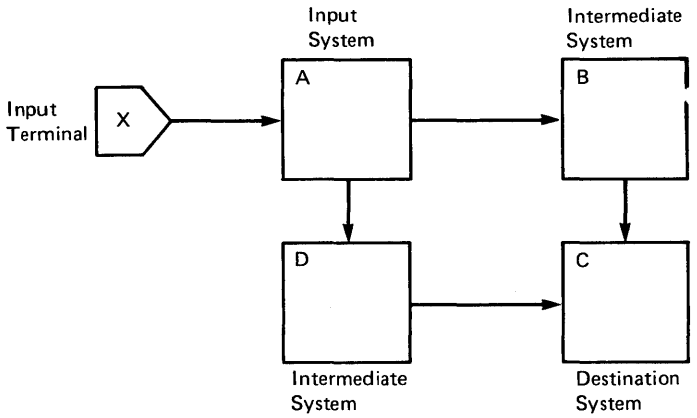


Figure 5-11. An Intermediate System

Assume that a message originating in System A is defined to be processed and replied to in System C. To reach System C (destination system) from System A (input system), the message must be routed through either System B or System D as defined by system definition. In this instance, either System B or System D is considered the intermediate system. By referencing the SYSID table, the intermediate system routes the message to the next link toward the destination system. Whenever there is no direct link between the input and destination systems, there is always at least one intermediate system. If the configuration had just three systems but the link between Systems A and C was unavailable, a message could be routed through System B as the intermediate system by reassigning the appropriate links.

Remote Transaction Priorities

The definition of each transaction code identifies the priority used to send the transaction to the processing system and the response back to the input system. Based on the specified priorities, three priority groups are considered during processing:

- High priority is assigned to primary requests in the input system whose specified priority is 8 through 14.
- Medium priority is assigned to secondary requests, responses and primary requests in an intermediate system, and primary requests in the input system, whose specified priority is 7.
- Low priority is assigned to any requests in the input system whose specified priority is 0 through 6.

In each priority group, message priority is based on the current transaction priority specified in the input system for primary requests and in the most recent processing system for secondary requests and responses.

Stopped Transactions

If a destination transaction code is stopped for queueing, the action taken by the destination system varies based on the type of request:

- For a primary request that is not conversational or that starts a conversation, IMS/VS sends an error message to the input terminal and cancels the message.

- For a primary request that continues a conversation or a secondary request, IMS/VS queues the message. If the request is the first one received for that stopped transaction, IMS/VS also sends a message to the master terminal at that transaction's local system.

Routing Exit Routines

Message routing is automatic according to the defined scheme unless a routing exit routine is provided by the user. The routing exit routine is called before the message destination is verified. There are two types of routing exit routines: terminal routing and program routing.

The terminal routing exit routine executes in the input system and is called on terminal input. This routine can inspect the destination specified and, if desired, change it to any local or remote destination. This routine may examine the first segment of the message to determine what the destination of the message should be. If this routine does not change the destination, the originally specified destination is used for routing. IMS/VS will not call a terminal routing exit routine for commands, for any message received from a terminal in preset destination mode, or from a terminal that is continuing a conversation.

In a configuration with horizontally-partitioned applications, the terminal routing exit routine could be used to screen all input messages and route them to the appropriate processing system based on information in the first segment of the input message. If transactions and links are appropriately defined, this routine might also be used to compare the load on the link associated with the specified transaction with other links. The message could be routed to a less-busy link.

The program routing exit routine is called when an application program issues a change call while processing a transaction that originated in a remote system. This routine can request that an output message be returned to the input system for destination verification. This allows the application program to reply to a remote logical terminal without requiring the processing system to know the logical terminal name, thereby minimizing the number of remote logical terminal definitions per system. If two systems use the same logical terminal names for the master terminal, the program routing exit routine can also be used to send a message to the master terminal of either the local system or the input system. It should not be used for program-to-program switches since the routing for transaction processing is specified during IMS/VS system definition. If the program routing exit routine is not used, the destination specified in the change call must be defined in the processing system. Conversational programs cannot use the program routing exit routine.

Remote Destination Verification

To maintain system integrity and prevent erroneous operation, an IMS/VS system in a multisystem configuration verifies all specified destinations. Remote destination verification occurs on input from a terminal or upon receipt of an application program reply if a remote destination is specified for the message. The routing exit routine, if available, has completed.

Destination verification occurs as follows:

Destination Type	Verified For
Logical terminal	• Destination type: The original destination must have been a logical terminal.
Transaction	• Destination type: The original destination must have been a transaction. • Transaction attributes: The following attributes must be consistent in the transaction definitions in the input and destination systems: - inquiry/noninquiry - single segment/multiple segment - recoverable/nonrecoverable - conversational/nonconversational Conversational transactions must have fixed-length SPAs; SPA size must be the same for all transactions that participate in a conversation.

When an invalid destination is recognized, IMS/VS cancels the message, sends an error message to the input terminal and to the master terminal of the local system, and logs an invalid request. If the message is conversational, the conversation abnormal termination exit routine is called in the input system and the conversation is terminated.

Application Program Abnormal Termination

When an application program abnormally termiantes, and the abnormal termination is not the result of a deadlock situation, a DFS554 message is sent to the master terminal of the system where the abnormal termination occurred. If the input message is still available, an error message that includes the first portion of the input message is sent to the input terminal. When the error message to the input terminal is sent, the DFS554 message includes the logical terminal name of the input terminal.

CONVERSATIONAL PROCESSING

Conversational processing is available to terminals attached to any system in a multisystem configuration to the same extent as in a single system. All transactions used in a conversation must be defined as conversational in each system of the multisystem environment. The input system controls the conversational resources for the duration of the conversation. When the input system receives a conversational transaction, it inserts the scratchpad area (SPA) as the first message segment and routes the message to the destination application program.

Each conversation step can be processed by any system of the multisystem configuration. Program-to-program switches can be routed from system to system. SPAs used in multisystem conversations must be defined as fixed-length to allow IMS/VS to avoid SPA data set updates from a remote system for program-to-program switches. The SPA size

specification must be the same for all transactions that participate in the conversation.

For the most part, multisystem conversations are transparent to terminal operators and application programs. One exception is if a conversational program inserts a message to a response alternate PCB in a remote system. By implication, this destination is in the input system and will, therefore, be verified by the input system. Destination verification in this instance includes assuring that the specified logical terminal is still assigned to the inputting physical terminal. If the logical terminal has been reassigned, the input system invokes the conversation abnormal termination exit routine and terminates the conversation. No status code is returned to the application program.

The other exception is if an application program that does not execute in the input system uses the SPA to specify a transaction code and thereby pass conversation control to another program. If the specified transaction code is invalid, the input system invokes the conversation abnormal termination exit routine and terminates the conversation. No status code is returned to the application program.

Routing Exit Routines

A terminal routing exit routine may be used for the input message that starts a conversation. It is not applicable at any other conversational step since the application program, not the input terminal, provides the destination for continuation of the conversation.

A program routing exit routine cannot be used during conversations.

Remote Destination Verification

Destination verification for program-to-program switches occurs in the system where the program requesting the switch executes. If valid, that system sends the SPA and the message directly to the destination transaction. If invalid, that system returns a status code to the application program.

Destination verification for a message to the input terminal is performed by the input system. The logical terminal specified must still be assigned to the inputting physical terminal. The input system also verifies the next transaction specified in the SPA. If the destination is invalid, the input system invokes the conversation abnormal termination exit routine and terminates the conversation. No status code is returned to the application program.

Normal Conversation Termination

A conversation can be terminated by either the application program or by the terminal operator. An application program normally terminates a conversation by inserting a message to the input terminal with a SPA that contains either no transaction code or the transaction code of a non-conversational transaction. The terminal operator terminates a conversation by entering either the /EXIT or /START LINE command.

The /EXIT command can be used any time during the conversation but the conversation is not terminated until the current conversational step has replied to the input terminal. This means that all data base processing for the current step is accomplished before a conversation ends.

If the input system is shut down and subsequently cold started, all the conversations that it controls are lost. It cancels any conversational messages it receives for input terminals previously involved in active or held conversations.

As in a single system environment, if the input system is shut down and subsequently warm started, all the conversations that it controls are lost if /START LINE is used to start the communication lines. The /RSTART LINE must be used to retain the previously active or held conversations.

If a remote system is shut down when a conversational step is processing or is queued in it, and is subsequently cold started, all references to the conversation are lost. A conversation lost in this way must be specifically canceled in the input system by the /EXIT command.

Abnormal Conversation Termination

A conversation is abnormally terminated if any one of the following occurs:

- The conversational program abnormally terminates.
- An invalid destination in the SPA, or for a conversational response, is recognized in the input system.
- A conversational message is inserted to a terminal whose conversation was terminated.
- Destination verification fails for a conversational message.
- No output was generated in the application program.

The conversation's SPA is passed to the conversation abnormal termination exit routine in the input system along with an indication of the cause of the termination. MULTISYSTEM OPERATIONS

Each system in a multisystem configuration is operationally an independent unit. It exclusively owns its own communication resources, which are controlled by its own master terminal.

MULTISYSTEM COMMUNICATION INITIALIZATION

Communication between two IMS/VS systems does not begin until the /RSTART LINK command is issued in each system. The normal procedure would be for the master terminal operator to issue this command when a system is started up. This procedure does not require coordination between master terminal operators. Communication is allowed only if the characteristics of the specified links are compatible. If a required link is not successfully started, messages will wait until the links have been reassigned.

If a system that has messages queued in it is cold started, these messages are lost. Since the messages that were lost can be from or to terminals and programs in other systems, the impact of a cold start is not limited to the cold started system.

MULTISYSTEM COMMUNICATION TERMINATION

A /PSTOP LINK command from either of two linked systems terminates transmission on the specified link. When transmission is terminated

on one side, its partner in the other system terminates its own transmission and notifies the master terminal operator.

LOGICAL LINK ASSIGNMENTS

Initial logical link assignments (logical link to physical link) are normally made during IMS/VS system definition. The /MSASSIGN command can be used to dynamically make or change a logical link assignment. This approach is used primarily for unscheduled reassignments resulting from failing physical connections or systems.

Since a logical link must always communicate with its partner, the master terminal operators of the two systems must coordinate their assignments to a corresponding physical link. Any type of physical link may be replaced by any other type of physical link.

If a restart is pending on a logical link due to physical link failure, both systems should use the following procedure to reestablish communication through an alternate physical link:

- Reassign the logical link to the alternate physical link.
- Use /RSTART LINK to start the logical link.

SECURITY

Security maintenance in a multisystem environment is performed independently for each system. Password security is verified on terminal input after execution of the terminal routing exit routine. Terminal security is verified on terminal input, and after an application program change call if the call is issued in the input system.

When a destination system receives a message to process, it verifies security based on the input terminal's association with the logical link path. This prevents transactions sent by unauthorized systems from being processed.

RECOVERY

Each system in the multisystem configuration uses the full recovery capabilities of IMS/VS. These capabilities assure that messages are never lost or duplicated within the single system as long as no cold start or emergency restart BUILDQ from an earlier checkpoint is performed, or no log records are lost.

In addition, the MSC feature assures that messages are never lost or duplicated across a multisystem link as long as no system in the configuration is cold started or no log records are lost. This is accomplished by logging information about a transmission in both the sending and receiving systems. This information is restored during restart and exchanged between the systems once the link is started. The sending system can then dequeue a message that was received by the receiving system but for which the acknowledgment was lost due to a link or system failure. The sending system can also resend a message that was sent but never enqueued by the receiving system due to a failure in the receiving system. If a system in the multisystem configuration fails to recover, the messages for which it has recovery responsibility are lost.

Since IMS/VS provides commands to dynamically change link assignments, an inoperable System/370 can be backed up by an alternate

System/370. The IMS/VS system that resides in the inoperable CPU can be executed in the alternate CPU once all involved links are properly reassigned by the master terminal operators.

COMPATIBILITY

IMS/VS system definition has been expanded to include new macros for multisystem facilities. Current terminal and transaction definitions can be used for remote destination definitions since macro operands that do not apply to remote destinations are ignored in remote definitions. The use of Message Format Service (MFS) is the same in a multisystem environment as in a single system environment. If a message is created in one system for a terminal attached to another system, the required message and format descriptions must be available to the system to which the terminal is attached, and definitions with the same name must be defined identically in each system.

PERFORMANCE CONSIDERATIONS FOR MSC

The design and tuning recommendations that apply to a single IMS/VS system are applicable to each IMS/VS system in a MSC environment. There are, however, additional considerations, related to resource consumption and demand, that must be taken into account when defining systems that are part of a MSC configuration.

For IMS/VS transactions that are processed in a local system, the transaction uses essentially the same hardward and software resources that it would in a non-MSC environment. For transactions that are processed in a remote system, additional resources are required. These resources are used to transmit the transaction over physical links to the remote CPU and to receive the response back from the remote CPU. Performance considerations are directly related to minimizing the resources consumed by remote processing and balancing the resource demand between several CPUs in a MSC configuration.

MINIMIZING RESOURCE CONSUMPTION

To minimize resource consumption, you should:

- Design the environment so that as many transactions as possible are processed locally.
- Provide physical links that go directly from local to remote systems; no intermediate systems should be involved in the transaction routing process. Transactions that must be routed through intermediate systems require additional CPU activity, message queue activity, and logging activity.
- Design the Message Queue Buffer Pool in each CPU to eliminate unnecessary Message Queue I/O activity.
- Use CTC links, if possible, rather than BSC links; the CPU requirements to support a CTC link are lower per message than those for BSC links.
- Define the physical link buffer sizes large enough to hold the message prefix plus all the segments of most messages.

BALANCING RESOURCE DEMAND

In a MSC environment with two or more CPUs, you should distribute the workload in a way that avoids excessively high utilization of any one CPU. You do this by distributing IMS/VS applications and their associated transactions and terminals between the available CPUs in a way that, dependent, on the complexity of the application and the capability of the CPU, avoids overloading each CPU.

If the current design of the data bases is such that the data bases and their associated applications cannot be distributed across the available CPUs (vertical partitioning), you can:

- Duplicate or share inquiry-only data bases; this allows the data to be referenced by more than one system.
- Split the data bases into several component data bases (horizontal partitioning). The component data bases must be completely independent for distribution among the available CPUs. For example, it may be possible to divide a data base by key range intervals. The new data bases and their associated applications could then be distributed among the existing IMS/VS systems and the Terminal Routing Exit could be used to route incoming transactions to the correct IMS/VS system. Another possibility is to divide the data base by geographic area. Each IMS/VS system could process the transactions that refer to the data bases for its own geographic area and route transactions that refer to a remote geographic area.

In addition to balancing the workload across CPUs, you may also have to balance the workload on physical links. This occurs when a physical link between two systems is of the BSC type and multiple physical links have been installed. You can balance the workload on physical links by:

- Specifying, during IMS/VS system definition, proper logical link paths and logical links for each remote application.
- Using a user-written Terminal Routing Exit to examine the load on each of the alternative physical links and choose the least busy link for routing.

MSC EXAMPLES

Figures 5-12 and 5-13 illustrate, respectively, horizontal and vertical partitioning.

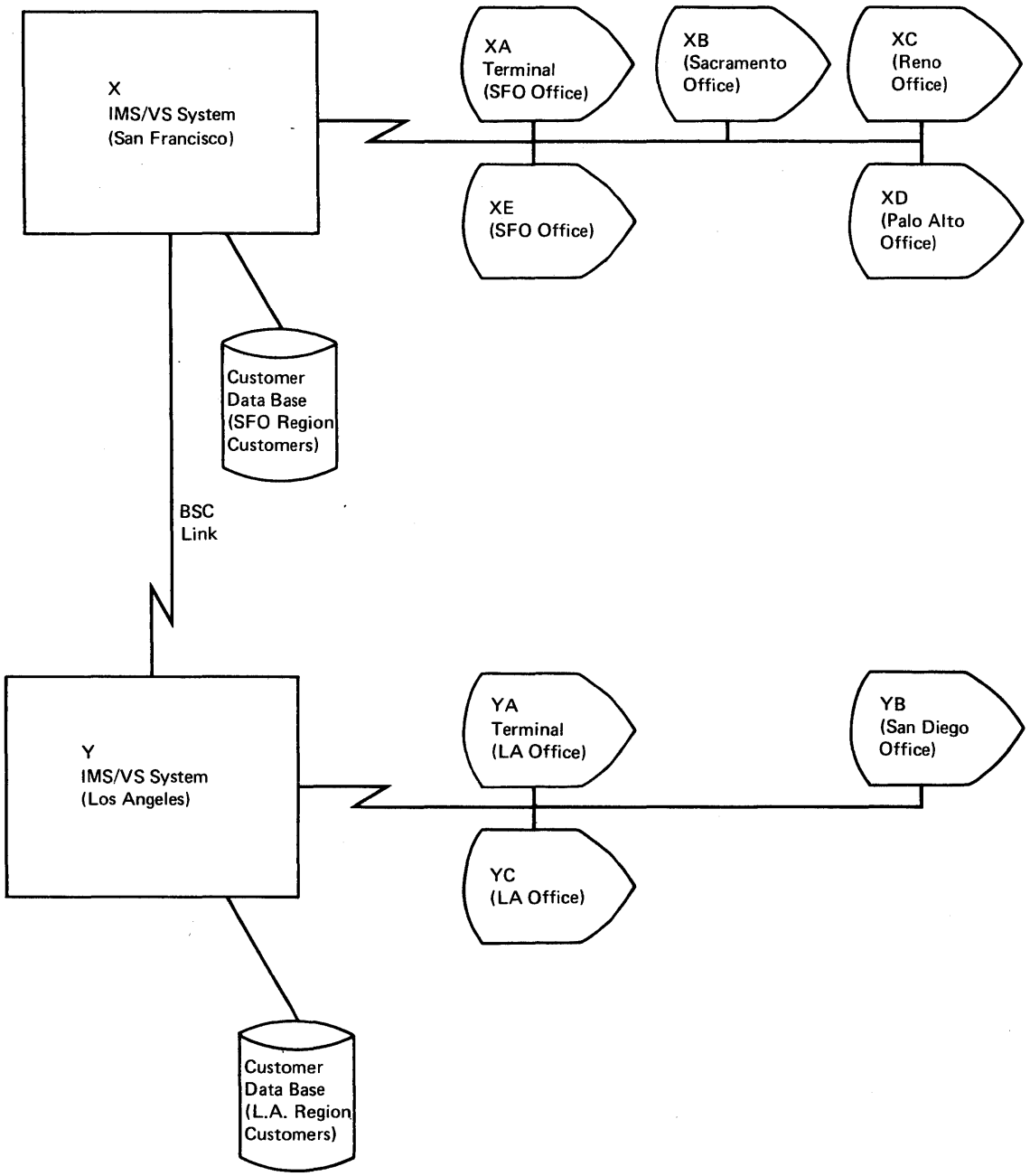


Figure 5-12. Hcrizcntial Partitioning

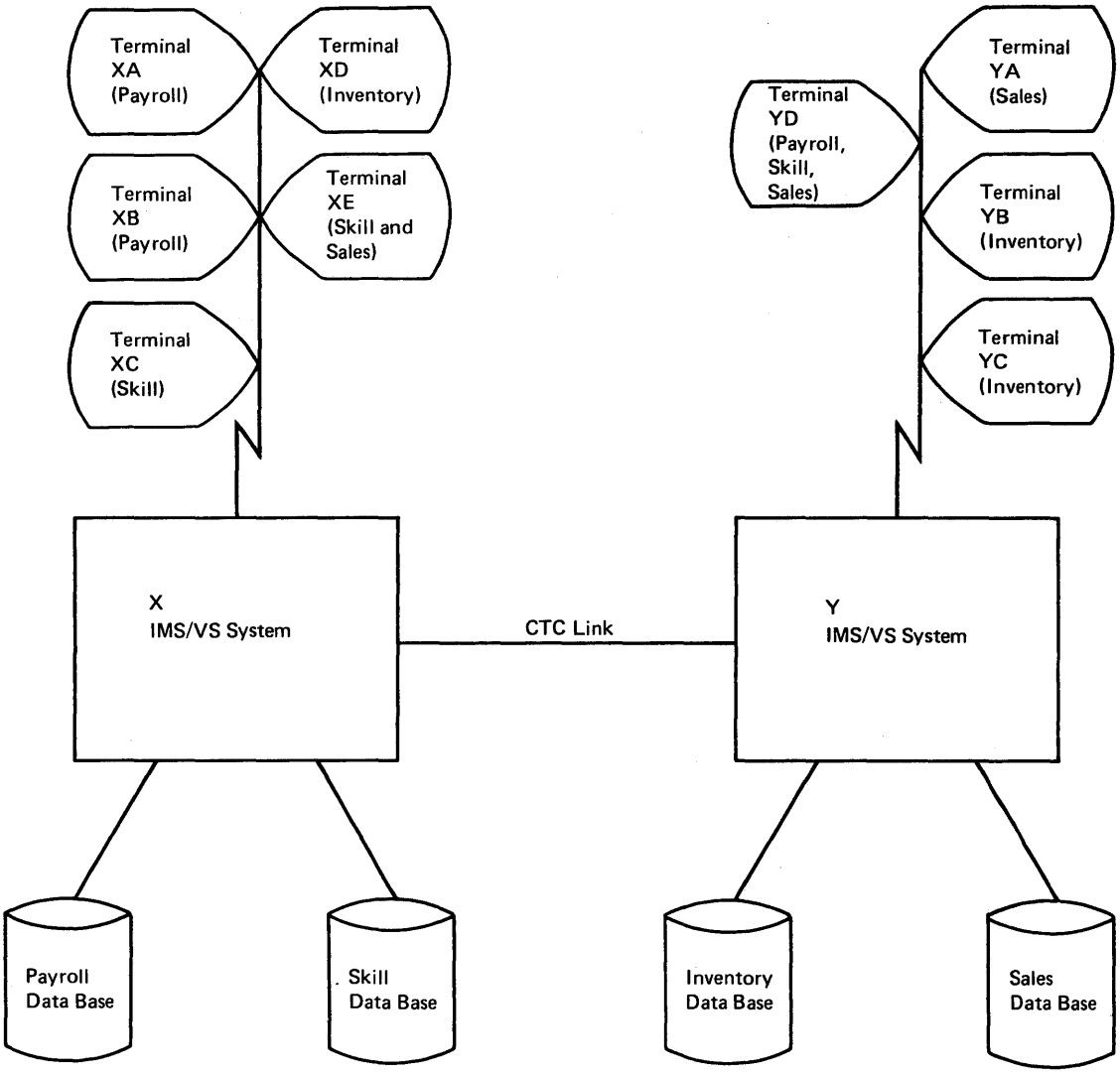


Figure 5-13. Vertical Partitioning

CHAPTER 6. DESIGN CONSIDERATIONS FOR THE FAST PATH FEATURE

This chapter describes the Fast Path feature and contains design considerations for its use.

The Fast Path feature provides improved performance for applications that require large transaction rates and use only simple data structures.

The Fast Path feature shares the IMS/VS Data Communication network. The functions of the Data Base system and the Data Communication feature are not reduced when the Fast Path feature is installed. The Fast Path feature complements the existing system and does not replace it. A system with the Multiple Systems Coupling (MSC) feature installed can also have the Fast Path feature installed; however, the Fast Path feature cannot use the MSC intersystem links.

The Fast Path feature uses two new data bases: Main Storage Data Base (MSDB) and Data Entry Data Base (DEDB).

FAST PATH DATA BASES

The Fast Path data bases are simple in structure and can only be processed by Fast Path transactions; Fast Path transactions cannot process non-Fast Path data bases.

MAIN STORAGE DATA BASE (MSDB)

An MSDB is designed for efficient processing of the most frequently used data in an installation. The MSDB resides in main storage, which reduces I/O activity.

The amount of data that the MSDBs can hold is limited by the amount of available storage.

An MSDB is a root-only data base; it has no dependent segments. Only fixed-length segments are allowed in an MSDB.

There are two types of MSDBs: terminal-related and nonterminal-related.

Terminal-related MSDBs are either fixed or dynamic. Inserts and deletions are not allowed in a fixed terminal-related MSDB but are allowed in a dynamic terminal-related MSDB. Both dynamic and fixed terminal-related MSDBs have the following characteristics:

- Each record in the MSDB is assigned to and owned by an LTERM.
- The name of the LTERM that owns a record is the key of the record. The key doesn't reside in the record. Because Fast Path does not allow more than one message to be processed at a time from a logical terminal, there is no possibility of update conflicts occurring for a segment of a terminal-related MSDB.
- The record can only be updated by messages issued from the LTERM that owns the record. However, the record can be read as a result of messages from LTERMs other than the owner's.

Although an MSDB segment can be owned by one logical terminal, a given logical terminal can own multiple MSDB segments, in a single MSDB or in several MSDBs.

A typical use for a fixed terminal-related MSDB would be in a banking application where each segment is associated with a teller. The segment would hold information such as the teller's cash balance. In this type of application it would be possible for a batch application to read the segment (possibly for general reporting purposes), but update would be impossible.

A dynamic terminal-related MSDB might be used as a 'scratch pad' to contain error information encountered by a transaction while processing data from another MSDB. Once the error has been noted and handled, the dynamic MSDB segment can be deleted.

Nonterminal-related MSDBs have the following characteristics:

- There is no ownership of records in a nonterminal-related MSDB.
- No insert, delete, or replace calls are allowed against a nonterminal-related MSDB. However, fields in MSDB records can be modified with a new DL/I call (FLD) described in a subsequent section.
- The key of nonterminal-related MSDB records can be an LTERM name or a field in the record. If the key is an LTERM name, it doesn't reside in the record, as with a terminal-related MSDB. If the key isn't an LTERM name, it does reside in the sequence field of the record. If the key resides in the record, the records must be loaded in key sequence because, when a qualified SSA is issued on the key field, a binary search is initiated.

A nonterminal-related MSDB with terminal-related keys might be used in a banking teller cash counter application that includes a supervisory function for the transfer of cash between tellers and the updating of their cash records. In this application, the individual tellers no longer own their own records, but these can only be updated by using the LTERM name associated with each MSDB segment.

The nonterminal-related MSDB without terminal-related keys would be used in any application where a large number of people need access to and update capability for data in a high transaction rate situation, for example, a real time inventory control application, where reduction of inventory could be noted from many cash registers.

Defining an MSDB

An MSDB is defined with DBDGEN in the same way as any other IMS/VS data base. The data base is specified as an MSDB by coding ACCESS=MSDB in the DBD statement. The REL keyword in the DATA SET statement is used to specify whether the data base is terminal-related or nonterminal-related. The definition of an MSDB data base is covered fully in the IMS/VS Utilities Reference Manual.

MSDB DL/I Calls

All DL/I calls, except those that specify "within parent," are valid with one or more types of MSDBs. Because an MSDB is a root-only data base, a "within parent" call is meaningless. Additionally, there is a DL/I call, FLD, specifically applicable to MSDBs. The FLD call allows

an application program to check and modify a single field within an MSDB record.

The following shows which DL/I calls can be issued against the different MSDB types.

CALL	MSDB TYPE		
	non-terminal-related	terminal-related fixed	terminal-related dynamic
GU	X	X	X
GN	X	X	X
GHU		X	X
REPL		X	X
ISRT			X
DLET			X
FLD	X	X	X

The FLD Call

The FLD DL/I call is unique to Fast Path. It allows for the operation on a record field rather than on an entire segment. Additionally, it allows conditional operation on a field.

Modification is done with the CHANGE form of the FLD call. The value of a field can be tested with the VERIFY form of the FLD call. These forms of the call allow the application program to test a field value before doing the processing to make a change. If a VERIFY fails, all CHANGE requests in the same FLD call are denied. This call is described fully in the IMS/VS Application Programming Reference Manual.

MSDB Buffer Allocation

Fast Path data base buffers are allocated when a call to update an MSDB is issued by an application program. The buffers hold the update information until a synchronization point is reached. The maximum number of buffers that the application program can use is dependent on the number of normal page-fixed buffers (NBA) and the number of additional page-fixed buffers (OBA) specified in the EXEC PARM field.

DATA ENTRY DATA BASE (DEDB)

A DEDB is an HD-type data base containing a maximum of two segment types, a root and, optionally, one dependent segment type as shown in Figure 6-1. The information in the dependent segments should be looked at as a history file. The dependent segment inserts are kept in time-of-entry seguence to provide a chronological history of transactions against the data base.

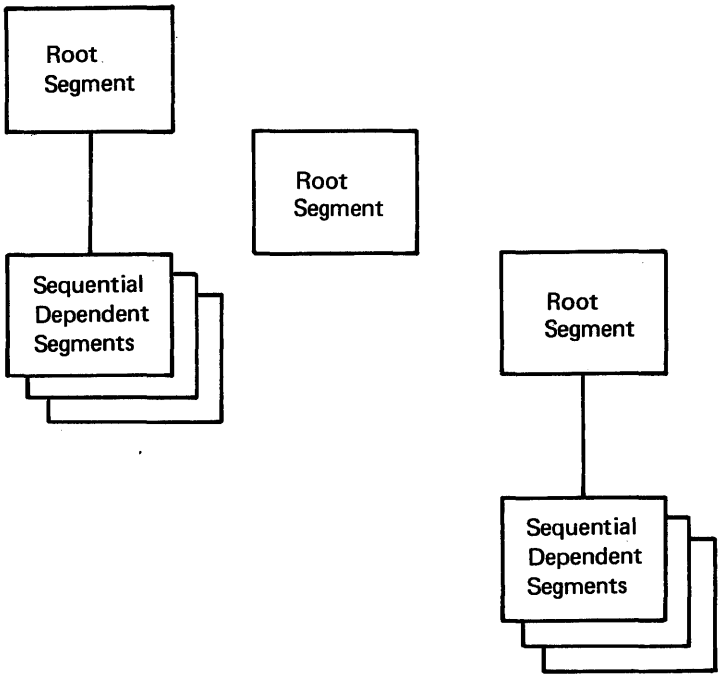


Figure 6-1. DEDB Structure

Root segments in a DEDB have a single key field. They can be accessed directly by this key. The dependent segments can have scan fields by which they can be identified in a sequential scan. Scan fields do not have to be unique. Because the dependent segments are stored in a time-of-entry sequence, there is no guarantee that they will be maintained in scan field sequence.

Both segments in a DEDB are variable in length. Although the lengths are variable, they cannot be changed after the initial insert.

Access to a DEDB is by a subset of DL/I calls that are compatible with the standard IMS/VS DL/I calls. Get, replace, delete, and insert calls are provided for root segments. Get and insert are the only calls allowed against a dependent segment.

A DEDB can be stopped by an operator with the STOP Data Base command. This command does not affect programs that are currently scheduled against the DEDB but does prevent the scheduling of any new programs needing access to the data base in question.

The characteristics of a DEDB are:

- The dependent segments are stored in chronological order, regardless of the root on which they are dependent.
- A data entry data base can be divided into areas. An area is a data set, except that an area holds entire data base records as shown in Figure 6-2. A data set group in IMS/VS holds a part of the logical structure. Each area within a DEDB data base contains root and dependent segments.
- Areas are further divided into units-of-work. Resource allocation is done on the basis of units-of-work rather than on physical records as in IMS/VS.

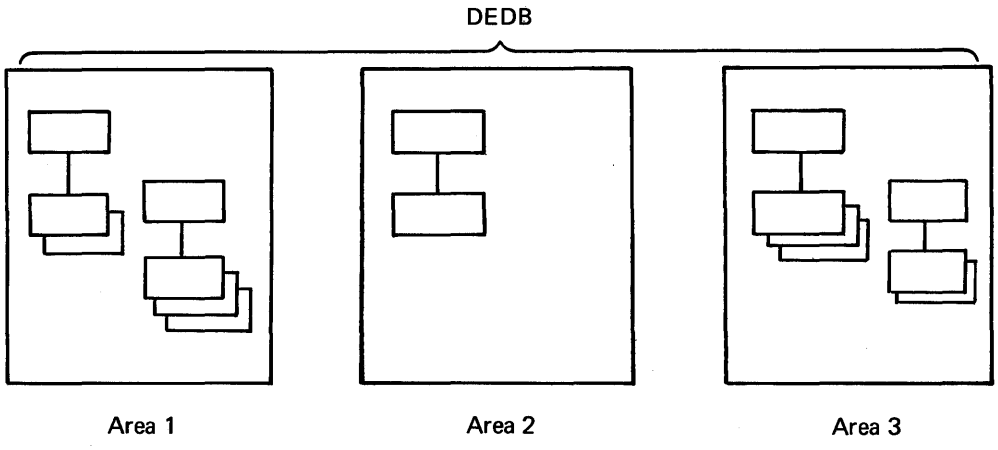


Figure 6-2. DEDB Areas

A DEDB area is divided into three parts: the root addressable part, the independent overflow part, and the sequential dependent part. Figure 6-3 shows how a DEDB area is divided.

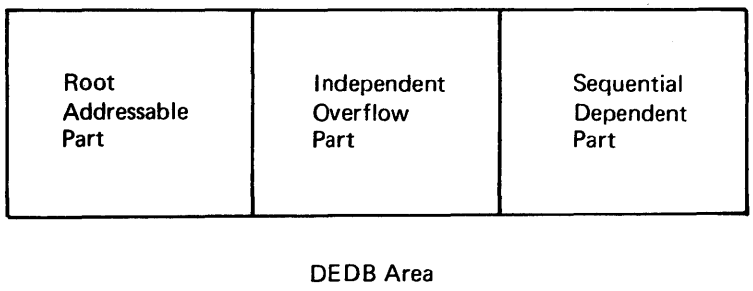


Figure 6-3. DEDB Area Division

Each area in a DEDB is a VSAM ICIP (improved control interval processing) data set. An area is opened by the first call to it from a program that is eligible to access it. A single area in a DEDB can be stopped by the operator with the STOP Area command. Likewise, a single area will be stopped when a write error is detected in the area. An application program accessing an area can encounter errors when the area is filled. Both the root addressable part and the sequential dependent part are subject to out-of-space conditions.

The root addressable part of an area is the only part that is actually divided into units-of-work, as shown in Figure 6-4. A unit-of-work consists of a user-specified number of contiguous control intervals. At any moment, a unit-of-work can only belong to a single processing program. Each unit-of-work is further divided into a base section and an overflow section. The base section contains control intervals that are addressed by a randomizing routine. The overflow section contains logical extensions of any control interval in the unit-of-work.

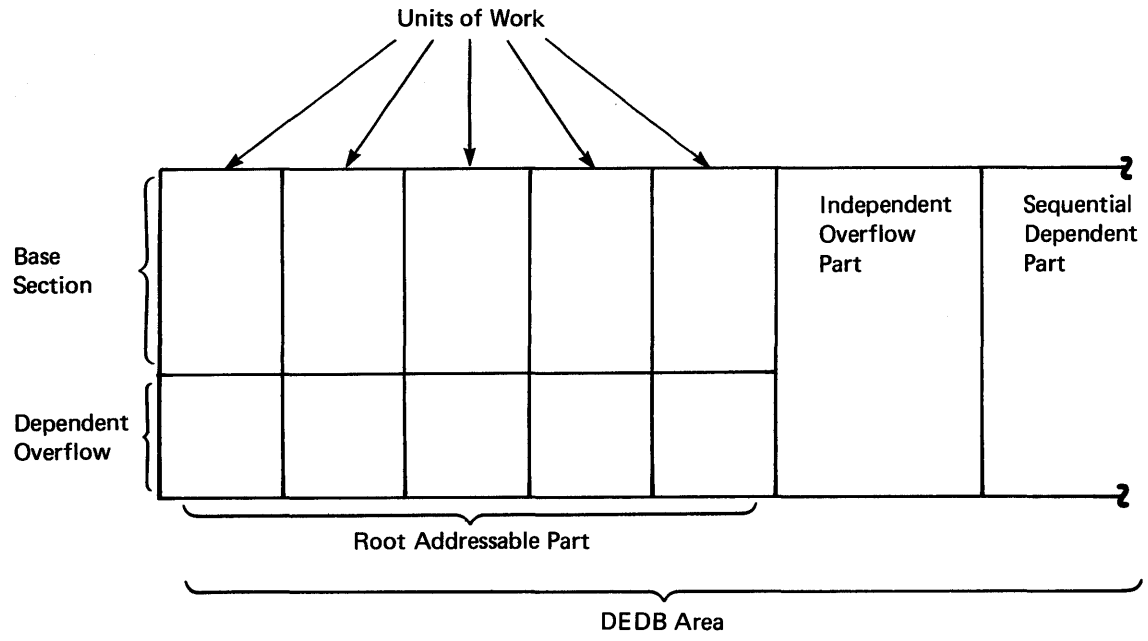


Figure 6-4. DEDB Units-of-Work

The independent overflow part contains empty control intervals that can be used by any unit-of-work in the area. Once a unit-of-work obtains a control interval from the independent overflow part, the control interval can only be used by that unit-of-work. A control interval in the independent overflow part can be considered an extension of the overflow section in the root addressable part as soon as it is allocated to a unit-of-work.

The sequential dependent part holds dependent segments from roots in all units-of-work in the area (see Figure 6-5). The dependent segments are stored in chronological order without regard to the root or unit-of-work that contains the root. When the sequential dependent part is full, it is reused from the beginning. However, before the sequential dependent part can be reused, the user must use the DEDB utility DBFUMDL0 to delete a contiguous portion or all of the dependent segments in the sequential dependent part.

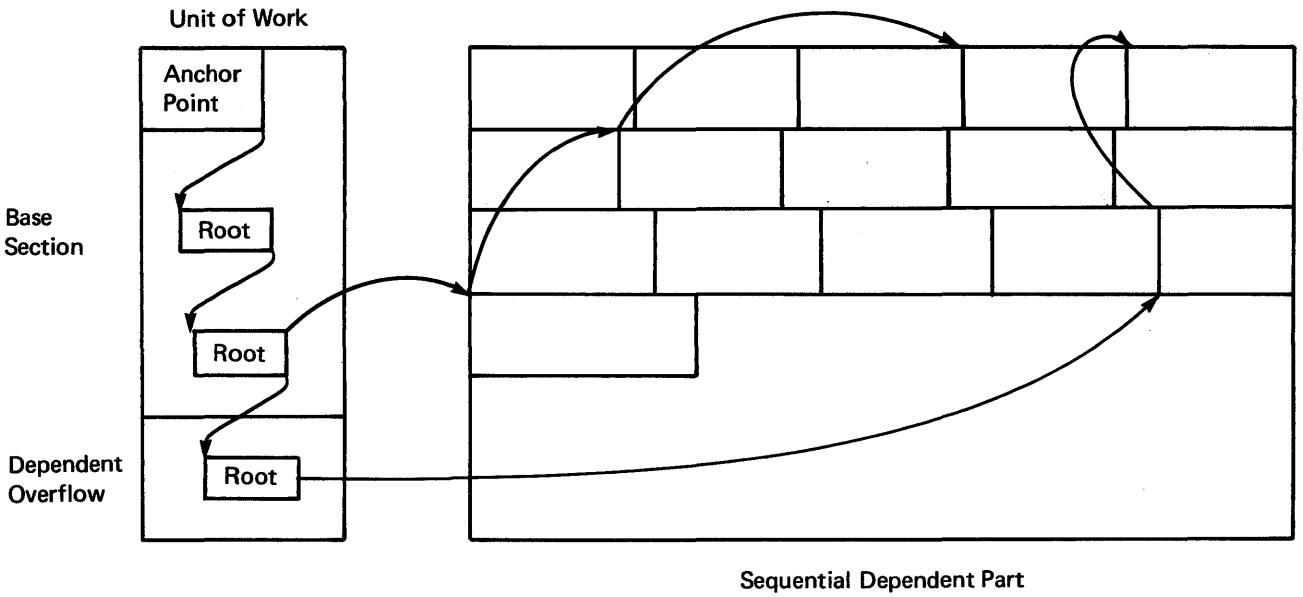


Figure 6-5. Storage of DEDB Dependent Segments in an Area

You can specify the size of the unit-of-work, the base section, the overflow section, and the number of units-of-work in the root addressable part and in the independent overflow part. This gives flexibility in controlling resource and space management.

Each area in a DEDB data base has its own space management parameters. These may be chosen according to the message volume that can vary from area to area. Initialization, reorganization, and recovery are done on an area basis.

DEDB Synchronization Processing

The physical update of a DEDB root is held in abeyance until a synchronization point has been reached and synchronization processing is completed successfully. This sequence eliminates the need for backout processing in case of a user program failure. DEDBs are physically updated after the associated log records are written.

DEDB dependent segments are gathered at synchronization processing time but are not written into the physical data base until a control interval is filled. Logging takes place during synchronization processing, and changes are discarded if a failure occurs.

DEDB Resource Management

Two resource allocations are necessary before a DEDB can be processed. First, the unit-of-work is allocated when a program requests a segment. After the unit-of-work is obtained, buffers are allocated to the program. A common buffer pool is used for all programs and data bases. The allocated buffers and units-of-work are made available following synchronization point processing.

The maximum number of buffers an application can use depends on the number of normal page-fixed buffers (NBA) and the number of additional page-fixed buffers (OBA) specified in the EXEC PARM field. When the limit is reached, any buffers that are allocated but have not been changed will be reused.

Sequential Processing of DEDB Data

DEDB dependents are stored in the dependent part of an area in the order of entry. DEDB root segments are stored as prescribed by the randomizing routine, chained in order of ascending key from each anchor point. Each control interval in the base section of a unit-of-work within an area has a single anchor point. These methods of storage are not well suited to sequential processing.

There is no easy way to process DEDB root segments in key order unless your randomizing routine assigns units-of-work and areas by ascending key. If you are willing to process the root segments in a somewhat arbitrary sequence, you can use Get Next calls. In this case, you will process the roots in order of:

1. ascending area number
2. ascending units-of-work
3. ascending key within each anchor point chain

Dependents chained from different roots within an area are intermixed in the sequential part of an area without regard to which roots are their parents. Usually there is no way to process them efficiently online in any sequential order. If all the sequential dependents were chained from a single root segment, processing with Get Next Within Parent calls would result in a backward sequential order. (Some applications may be able to use this method.) Normally, dependent segments are processed sequentially only by using the sequential dependent scan utility which is described in the IMS/VS Utilities Reference Manual.

Defining DEDB Data Bases

A DEDB is defined through the DBDGEN process as are all other IMS data bases. To specify that a data base is to be a DEDB, ACCESS=DEDB is specified in the DBD statement. Further information on generating a DEDB is contained in the IMS/VS Utilities Reference Manual.

MESSAGE HANDLING

With the Fast Path feature installed, all input messages from Fast Path eligible terminals are directed to a Fast Path routine, with a user exit routine, that determines if the message is for Fast Path or IMS/VS.

Messages from a non-Fast Path eligible terminal are routed directly to IMS processing without resorting to the user exit routine. Messages that meet the Fast Path criteria defined by the user exit routine are routed to the Fast Path message handling routines.

Fast Path input and output messages can only be a single segment and must not exceed a fixed maximum length.

The Fast Path online application programs operate in a wait-for-input mode, and must be prescheduled before a transaction can be entered through Fast Path terminals. Parallel scheduling is supported through IMS/VS system definition.

INPUT MESSAGES

Every Fast Path transaction is defined to the system as a Fast Path potential or Fast Path exclusive transaction. Potential transactions can be executed under IMS/VS processing or Fast Path processing. A user-written exit is required to analyze the input message to determine if it should be routed to IMS/VS or Fast Path.

A Fast Path exclusive transaction can only be processed by a Fast Path application program.

All input messages that are to be processed under Fast Path must be issued from a Fast Path eligible terminal. A Fast Path potential transaction that is to be processed by normal IMS/VS facilities can be issued from a terminal that is not Fast Path eligible.

OUTPUT MESSAGES

Fast Path output messages are limited to a single segment. Only one insert call can be issued against the PCB. The output segment cannot exceed a prespecified segment length defined at system generation.

The output message and the input message are not logged until a Get Unique is issued to the PCB to obtain the next input message. If a failure occurs before this synchronization point is reached, both the input and output message are lost. This prevents the necessity of backing out of a transaction that did not complete because of a failure.

FAST PATH PROGRAM TYPES

Fast Path executes in Fast Path IFP region. The IFPs are handled differently depending on the type of program that is running in the region. There are three uses for the regions in which Fast Path processing is done:

- Applications for processing Fast Path messages
- Applications for processing input external to Fast Path
- Utilities initiated against the data bases

Regions executing message-initiated applications operate in a wait-for-input mode like IMS/VS MPPs. These are waiting-for-input messages from Fast Path terminals. Programs operating in a message-driven region may only use DEDB or MSDB data bases.

Regions executing applications are used in a manner similar to an IMS/VS BMP. This type of region can access and update DEDB and MSDB data bases and can also access OS/VS data sets through OS/VS data management. This region type might be used to update a checking account data base with input from a sequential file of processed checks.

Regions executing online data base utilities execute concurrently with Fast Path processing. This region type is similar to a nonmessage-driven application program but no user-written application program is needed.

Message-driven application programs cannot terminate normally unless a QC status code is posted in the I/O PCB. Nonmessage-driven application programs cannot terminate normally without releasing the buffers. A SYNC or ROLB call must be issued to release the buffers.

SYNCHRONIZATION POINT PROCESSING

Synchronization point processing is performed after a message get unique call or a SYNC call from an application program. The philosophy of Fast Path processing is to hold all application program updates in main storage until a synchronization point is reached. All logging on the IMS/VS log tape is performed during synchronization point processing but before the application of data base updates.

If the application program uses the verify function in MSDB calls, it will be re-executed during synchronization point processing. If the conditions are met, the synchronization point processing completes as expected. If the conditions are not met, the synchronization point processing purges all update information and gives the same input message to the application for reprocessing.

INDEX

IMS/VS Version 1
System/Application Design Guide
SH20-9025-5

Reader's Comment Form

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM shall have the nonexclusive right, in its discretion, to use and distribute all submitted information, in any form, for any and all purposes, without obligation of any kind to the submitter. Your interest is appreciated.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

Fold on two lines, tape, 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.

Reader's Comment Form

Fold and Tape

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 Tape

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

IMS/VS Version 1 System/Application Design Guide Printed in U.S.A. SH20-9025-5

IMS/VS Version 1 System/Application Design Guide Printed in U.S.A. SH20-9025-5

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601