SH20-9145-0

Program Product

IMS/VS Version 1 Primer

Program Number 5740-XX2

Release 1.5

FIRST EDITION (SEPTEMBER 1978)

This edition is a revised edition of the IMS/VS Primer World Trade System Center Bulletin (S320-5767-2) dated September 1977.

This edition addresses IMS/VS Data Base Facilities and IMS/VS Data Communication Facilities. This edition applies to IMS/VS Version 1 Release 1.5, program number 5740-XX2, under OS/VS1 or OS/VS2 Release 2 (MVS), using BTAM or VTAM and the IBM 3270 Information Display System.

Information in this publication is subject to significant change. Any such changes will be published in new editions or technical newsletters. Before using this 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 this 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.

ABOUT THIS MANUAL

This publication is intended for first-time users of the Information Management System/Virtual Storage (IMS/VS). It provides system analysts, data base specialists, system programmers, and application programmers with the information necessary for the design, installation and operation of their initial applications, using a subset of the data base or data base/data communication facilities of IMS/VS.

The IMS/VS Primer Function comprises five separately orderable documents. One is this document (SH20-9145). The second is the _IMS/VS Primer Sample Listings_ (SH20-9149), containing a complete IMS/VS sample application including generation input, source program examples, data base sample data and execution output. The third is the _IMS/VS Primer Master Terminal Operator's Guide -- BTAM_ (SH20-9146), containing a sample operating guide for the master terminal operator of IMS/VS using the Basic Telecommunication Access Method (BTAM). The fourth is the _IMS/VS Primer Master Terminal Operator's Guide -- VTAM_ (SH20-9147), containing a sample operating guide for the master terminal operator of IMS/VS using the Virtual Telecommunication Access Method (VTAM). The fifth is the _IMS/VS Primer Remote Terminal Operator's Guide_ (SH20-9148), containing a sample operating guide for the IMS/VS end-user/terminal operator. The manuals are designed to be used together, i.e., the IMS/VS Primer and the Operating Guides extensively reference the samples in the IMS/VS Primer Sample Listings.

Objectives

The primary objective of the IMS/VS Primer Function is to provide the first-time user of IMS/VS a single document containing all of the information the user would ordinarily need to:

- Plan for IMS/VS use
- Design DL/I data bases
- Design, write, and test IMS/VS programs
- Install the IMS/VS program product (5740-XX2)
- Operate IMS/VS
- Maintain IMS/VS

The only other IMS/VS publications the user of the subset would normally have to refer to are the _IMS/VS General Information Manual_ and the _IMS/VS Messages and Codes Reference Manual_.

While the IMS/VS Primer is designed for the new IMS/VS user, it is applicable to other customers, such as:

- The currently installed IMS/VS user who has a continuing training requirement, and
- The currently installed IMS/VS user who is implementing new applications for departments having no experience using IMS/VS.

By using the approach suggested in the IMS/VS Primer, users can avoid much of the complexity usually associated with IMS/VS. Many of the steps required to install IMS/VS can be shortened, simplified, and/or accomplished in a more orderly manner.

The IMS/VS Primer _is_ _not_ intended to eliminate the need on the part of the user for careful planning, close coordination, and guidance by experienced systems personnel, detailed study of the application requirements, rigorous program testing, proper operating procedures, etc. It _is_ intended to be a learning guide, a source of field-proven techniques and advice, a tested sample system, a subset reference manual, and an operator's guide. By following this manual, users should progress quickly and confidently through the steps required for implementation of a simple, initial IMS/VS application.

Scope of the Manual

Each user has the responsibility to assess the applicability of the IMS/VS Primer Function to his requirements. If desired, users can ask for guidance and counsel from an IBM representative or system engineer. The assessment must be made with a full understanding of the scope and intention of the IMS/VS Primer Function.

Only a subset of the full facilities of IMS/VS is addressed. Although the subset is rich in function, a customer's application might require additional IMS/VS functions.

If a user requires facilities not included in the subset, he should reconsider, if necessary, any recommendations given here.

Summary of Contents

This manual is organized into nine chapters.

- Chapter 1, "Introduction," introduces the IMS/VS data base and data base communication facilities and a sample application used throughout the manual. The chapter is divided into a DB facilities section and a DC facilities section. It also provides a brief overview of our IMS/VS subset.

- Chapter 2, "Data Base Design," provides the data base specialist and system analyst with information and guidelines for data base design. This chapter is applicable to both the DB-only user and the DB/DC user.

- Chapter 3, "Data Communication Design," contains a detailed description of the IMS/VS data communication facility. It provides guidelines for the design and implementation of data communication applications using these facilities. This chapter can be disregarded by the DB-only user.

- Chapter 4, "Data Base Processing," guides the application programmer in the design, coding and testing of DL/I batch and IMS/VS message processing programs. Only the first part of the chapter is applicable to the DB-only user.

- Chapter 5, "Data Base Reorganization/Load Processing," describes when and how data bases should be reorganized.

- Chapter 6, "Data Base Recovery," guides the data base specialist and operations staff in the implementation of data base recovery procedures.

- Chapter 7, "Installing IMS/VS," guides the system programmer through the installation of a subset of IMS/VS data base and data base/data communication system. It also addresses the installation of IMS/VS in the Systems Network Architecture (SNA) environment.

- Chapter 8, "Operations," contains guidelines for the design of operating procedures for the IMS/VS online system. It shows how to adapt the sample master terminal and remote terminal operator guides to your own environment. This chapter can be disregarded by the DB-only user.

- Chapter 9, "Optimization," describes how to monitor and optimize a running application.

Every chapter except the second, third and eighth is divided into two parts. The first part of each chapter deals with the data base management portion of IMS/VS. The second addresses IMS/VS data communication. For your convenience, the following table defines those parts of this manual of interest to each functional area in your organization.

CHAPTER	MANAGEMENT	DB/DC ADMINISTRATOR	DATA BASE SPECIALIST	DATA COMMUNICATION SPECIALIST	SYSTEM ANALYST	SYSTEM PROGRAMMER	APPLICATION PROGRAMMER	OPERATION
1. INTRODUCTION	*	***	***	***	***	***	**	
2. DB DESIGN		***	***	**	**	**	*	*
3. DC DESIGN		***	**	***	**	**	*	*
4. DB PROCESSING		**	***	***	**	**	***	*
5. DB REORGANIZATION		**	**		*	***	*	**
6. DB RECOVERY		**	***	**	*	**		***
7. INSTALLATION		**	**	**		***		**
8. OPERATION		**	*	*	*	***		***
9. OPTIMIZATION		**	***	***	*	***	*	**

LEGEND:

* Reader should be familiar with contents.
** Reader should know specific parts in detail.
*** Reader should have complete detailed knowledge.

Prerequisites

Before using this manual, you should be familiar with the IBM Operating System for Virtual Storage (OS/VS1 or OS/VS2). This manual's design is such that the new IMS/VS user will need to make few, if any, references to other IMS/VS publications, except for the *General Information Manual* (GH20-1260) and *Messages and Codes Reference Manual* (SH20-9030). The more advanced user, however, will find additional information in the listed associated publications.

The reader should be familiar with the information presented in: *IMS/VS General Information Manual* (GH20-1260) (especially Chapter 1, "Introduction to IMS/VS," and Chapter 4, "System Configuration").

Associated Publications

The following IMS/VS Publications should be used if you have a need for more IMS/VS information beyond the scope of our subset:

- *IMS/VS System/Application Design Guide*, SH20-9025
- *IMS/VS Application Programming Reference Manual*, SH20-9026
- *IMS/VS System Programming Reference Manual*, SH20-9027
- *IMS/VS Operator's Reference Manual*, SH20-9028
- *IMS/VS Utilities Reference Manual*, SH20-9029
- *IMS/VS Message Format Service User's Guide*, SH20-9053
- *IMS/VS Advanced Function for Communications*, SH20-9054
- *IMS/VS Installation Guide*, SH20-9081
- *IMS/VS Low Level Code/Continuity Check in Data Language/I Program Reference and Operation Manual*, SH20-9047
- *IMS/VS Program Logic Manual*, LY20-8069
- *IMS/VS Failure Analysis Structure Tables (FAST) for Dump Analysis*, LY20-8050
- *IMS/VS Diagnostic Aids*, LY20-8063

CONTENTS

FIGURES

CHAPTER 1. INTRODUCTION

WHAT IS IMS/VS?

IMS/VS is an IBM program product developed to improve the computer user's ability to implement data base/data communication (DB/DC) applications. It relies on and extends the facilities and functions of Operating System/Virtual Storage (OS/VS) into the DB/DC environment. IMS/VS also makes these data base applications, to a large extent, hardware and software independent.

IMS/VS may be installed in either of two ways:

- A data base management system for batch-only operations
- A data base/data communication system for concurrent online and batch operation

This manual addresses a subset of both versions. It covers the installation and use of the data base system, the data base/data communication system, and the migration of the data base system to the data base/data communication system.

The data base management facility of IMS/VS is also referred to as the Data Language/I facility or DL/I. The functions supported by DL/I are data base definition, creation, access and maintenance. The data base capabilities of DL/I can be used in either the IMS/VS data base system (IMS/VS DB), or the IMS/VS data base/data communication system (IMS/VS DB/DC).

WHY DATA BASES?

Traditionally, data files were designed to serve individual applications, such as inventory control, payroll, accounts receivable, or purchasing. Each data file was specifically designed for its own application and stored separately on tape or on disk. Quite often, the data files of different applications contained common data elements. This redundant data caused an extra problem for the user because it became very difficult to keep it consistent.

Furthermore, the same data in different files often had different formats. This variance in the format of common data meant that application programs were tailored to specific data organizations and even specific physical devices. When new applications, data management techniques, or devices were introduced, the application programs normally had to be changed. As a result, application programs were often in an almost perpetual state of change adding appreciably to the overall cost of data processing.

These undesirable attributes of data files have been largely eliminated by the use of the "*data base*." A *data base* is a collection of interrelated data elements processable by one or more applications.

A data base provides for the integration, sharing, and control of common data. As an example, a manufacturing/distribution company may first integrate the data for an application dealing with parts control and purchase orders (Figure 1-1). Subsequently, application data for customer order processing and accounts receivable may be integrated. The data and the programs of already implemented applications need not change when the data of subsequent applications is integrated.

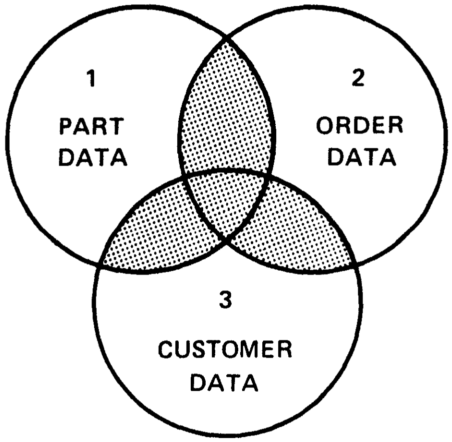


Figure 1-1. Application Data Integration -- Data Base Concepts

A data base provides flexibility of data organization. It allows the addition of data to an existing data base without modification of existing application programs. In Figure 1-1, the accounts receivable data may be added, when it is ready to be integrated, to the parts and orders data base. This independence is achieved by avoiding the direct association between the application program and the physical storage of data.

Thus, the advantages of a data base are:

- Control of data redundancy and reduction of resulting duplicate maintenance.
- Consistency through the use of the same data by all parts of the company.
- Application program independence from physical storage organizations and access methods.
- Reduction in overall application costs.
- Data designs usable for both batch and online processing.
- A system-provided focal point for the control of data.

OUR SAMPLE ENVIRONMENT

IMS/VS is not itself an application. It is a framework within which to construct data base/data communication applications. To make this manual more usable, we will define a sample application. This sample will then be used throughout this manual as a base for all the examples. It will be used to guide you in a natural way throughout all the subsequent steps for a successful implementation of an application using IMS/VS.

The sample application chosen is Parts Control and Order Processing. In even more general terms, it could be called "ITEM control and TRANSACTION processing," where the "ITEM" could be a part, an account, a citizen, or a policy. The "TRANSACTION" could be an order, an invoice, a customer inquiry, etc. The fact that we will use this particular application in the manual does not preclude the use of IMS/VS for other applications. On the contrary, the basic data structure and processing shown in this sample are easy to adjust to other applications.

OUR SAMPLE COMPANY'S REQUIREMENTS

The sample uses a fictitious company that offers a wide variety of building, construction, and engineering parts and materials. The parts and materials are purchased from manufacturers and sold to customers. Most customer orders arrive by telephone. Due to the growth in numbers of orders and varieties of items, an upgrade of the existing parts control and customer order applications was deemed necessary. It was decided to build a new system which integrated these applications.

Some objectives for the new application system were:

- Implement the system in the following order:

    1. Parts control with its associated purchase order processing

    2. Customer order processing

- Provide central control of parts, purchase orders, and customer orders

- Provide accurate status information on parts in stock, on order, and delivered

- Provide accurate entry of both purchase and customer orders, with respect to parts in stock.

- Provide an interface with the existing accounts receivable application, which currently maintains the central customer file. This application and its files will not be converted at this stage.

- Provide a base for the online processing of orders and inquiries at a later stage.

The implementation of the above system will be the common thread throughout the examples used in the manual. We will distinguish three major implementation phases:

1. The Parts Control application, consisting of a central Parts data base and Inventory Report and Purchase Order programs.

2. The Customer Order application which requires an additional Customer Orders data base, to be integrated with the existing Parts data base. A Customer Order program is added.

3. Addition of requirements to the Purchase Order program.

In the manual, the three steps above coincide with the three basic functional expansions of a typical DL/I environment. We shall refer to those as <u>phases</u>.

<u>Note</u>: Phase 1 should be studied and exercised first. Phases 2 and 3 are somewhat independent. The actual data base design of your application could well be initiated on either the phase 2 or the phase 3 functional level.

For each level, we will consider:

- Data base creation

- Data base processing

- Data base reorganization
- Data base recovery

We will also consider the migration aspects of moving from one level to the next.

THE PHASE 1 ENVIRONMENT

Phase 1 of our sample limits itself to the Parts Control application.

The Parts Data Base

Information about parts is managed by the inventory control department. All data will be stored in a Parts data base.

It consists of one record for each part which the company stocks. Within the record we can identify:

- Standard information for the part.
- Stock information for each part.
- Purchase information for each part.

The Parts Inventory Reports

The Parts Inventory Report program provides information about stock delivery and order position of each part the company stocks.

Purchase Order Processing

The Purchase Order program handles the purchase orders issued by the purchasing department. It checks the input, and prints, changes, and deletes orders.

A more detailed description of the phase 1 data base and application can be found in Chapter 2, "Data Base Design," under the topic: "Sample Data Base Requirements for Phase 1."

THE PHASE 2 ENVIRONMENT

Phase 2 of our sample environment considers the addition of a Customer Orders data base and its associated order processing programs.

The Customer Orders Data Base

Information about customer orders is managed by the sales department. All order data will be stored in a Customer Orders data base. It consists of one record for each customer order. Within the record we can identify:

- Standard information for this order and customer.
- Order detail information for each ordered part.
- Shipment information for this order.

A link is required with the parts data base because it is necessary to know which parts are on order by each customer and which customer ordered a given part.

Customer Order Processing

The Customer Orders program inserts, changes and deletes customer orders in the Customer Orders data base. It also checks and updates the part stock information before the order is accepted. This is planned for online processing in the sales department in the near future.

This application also needs access to the already existing central customer file. This central customer file is a key sequenced data set (KSDS) under the Virtual Storage Access Method (VSAM) of OS/VS.

THE PHASE 3 ENVIRONMENT

In phase 3 we consider a change in purchase order processing. The additional requirement is to provide direct access to individual purchase orders, both by part number and by purchase order number.

THE IMS/VS DATA BASE SYSTEM

The IMS/VS data base system contains three major components:

- A system definition facility to allow tailoring of the system to a particular OS/VS environment.
- The DL/I facility through which users meet the data requirements of their own applications.
- Utility programs which assist in the reorganization and recovery of data bases, and monitoring of data base usage.

In the following we will introduce these components and their functions which are of interest to the first-time user.

SYSTEM DEFINITION

Based on user specifications and type of operating system, IMS/VS system definition creates a library with DL/I processing modules, a procedure library and some modules for inclusion in the operating system. We will cover this process in Chapter 7, "Installing IMS/VS."

DATA LANGUAGE/I FACILITY

DL/I Concepts

DL/I allows application programs to be independent of access methods, physical storage organizations, and characteristics of the devices on which the application data is stored. This independence is provided by a common symbolic program linkage and by data base descriptions external to the application programs. The section entitled "Data Base User Interface" defines this interface.

The majority of the data utilized by any company has many interrelationships that can cause significant redundant storage of data when conventional organizations and access methods are used. The storage organizations and access methods of DL/I make it possible to integrate data and control the amount of data redundancy. Processing of data in more than one sequence can be achieved. All data need not be placed in a single common data base. DL/I allows you to physically store the data in more than one data base while maintaining centralized control over all the data.

The concept of data sensitivity allows you to control the use of the data base by each application program. Each program can be limited to (that is, be sensitive to) a predetermined subset of the data. This further enhances data independence. In addition, any application program can be restricted to making only specified types of data base requests against the data to which it is sensitive.

Environment Definitions

Within the DL/I environment, the following definitions apply:

- Segment. A data element of defined length, containing one or more related data fields. It is the basic unit of data transfer between the application program and DL/I.
- A DL/I data base record. A set of related segment occurrences of one or more segment types. Each segment type may have a unique format.
- A DL/I data base. The major unit of DL/I data storage. A set of data base records stored using one of the DL/I organizations and accessible by one or more of the DL/I access methods. A data base is typically composed of one or more common OS/VS or virtual storage access method (VSAM) datasets. DL/I relates its data base records and data bases to a physical storage organization and access method.

Data Independence

DL/I's data base concept allows user's data and programs to be independent of the access methods and storage organizations chosen by the data base designer.

The application program interface to the data in the data base is a common symbolic language. In fact, the application program is unaware of the particular storage organization, storage device, and access method chosen for any data base. Nor is the program aware of any pointers which might be used in the physical storage organization.

Application Data Structures

Application programs written to use DL/I deal with application data structures. This refers to the manner in which the application program "sees" the data. A DL/I application data structure consists of one or more hierarchical data structures programs written to process these data structures can be independent of the physical data structure. Physical refers to the manner in which the data is stored on a direct access storage device. A DL/I application program never deals directly with a physical data structure.

The traditional manner of representing data can be seen in Figure 1-2.

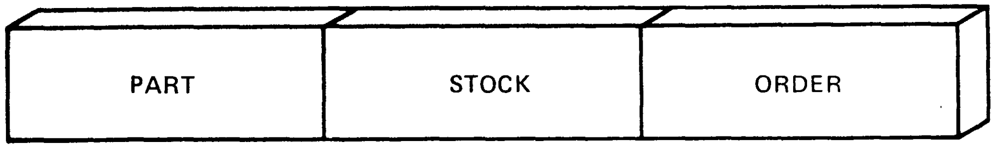


Figure 1-2. Traditional Record Layout

This picture describes:

1. The physical structure of the record as it appears on tape or a direct access storage device.

2. The logical structure for the application. Notice there is no difference between the physical (as stored) and logical (as used) data structure.

Each of the three divisions (PART, STOCK, ORDER) usually contains several data elements, or fields. For example, one of the data elements in STOCK might be stock location. In addition, the record might actually contain multiple STOCK and ORDER divisions for a single PART.

This same record appears in Figure 1-3 as a DL/I logical data structure. The PART, STOCK, and ORDER divisions are now considered <u>segments</u> of data. Each segment is made up of several fields. Stock location is a field within the STOCK segment.

The logical data structure in Figure 1-3 is called a <u>hierarchical data structure</u>.

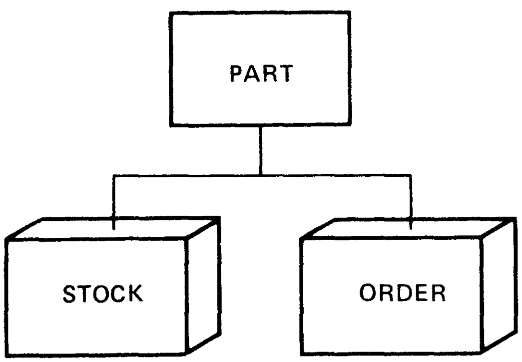


Figure 1-3. Hierarchical Data Structure

Hierarchical Data Structure

The hierarchical data structure in Figure 1-3 describes the data as seen by the application program. It does not represent the physical storage of the data. The physical storage is of no concern to the application program.

The basic building element of a hierarchical data structure is the <u>parent/child relationship</u> between segments of data. See Figure 1-4.

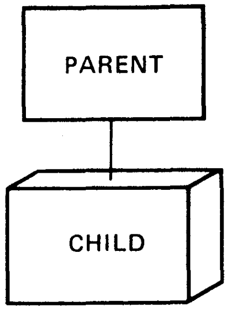


Figure 1-4. The Parent/Child Relationship of DL/I

Each _occurrence_ (or instance) of a _parent segment_ has associated with it 0, 1, 2, or more _occurrences_ of a _child segment_. Each child segment occurrence has associated with it one occurrence of a parent segment.

Sometimes it is necessary to distinguish between a _segment type_, that is, the kind of segment, and the _segment occurrence_, that is, particular instance of its contents and location.

As shown in Figure 1-3, a parent can have several child segment types. Also, a child segment can, at the same time, be a parent segment, that is, have children itself. The segment with no parent segment, that is, the one at the top, is called the _root segment_.

All the parent/child occurrences, for a given root segment, are grouped together in a DL/I data base record. The collection of all these like data base records is a DL/I data base.

Figure 1-5 shows these relations between the segment, the data base record, and the data base.

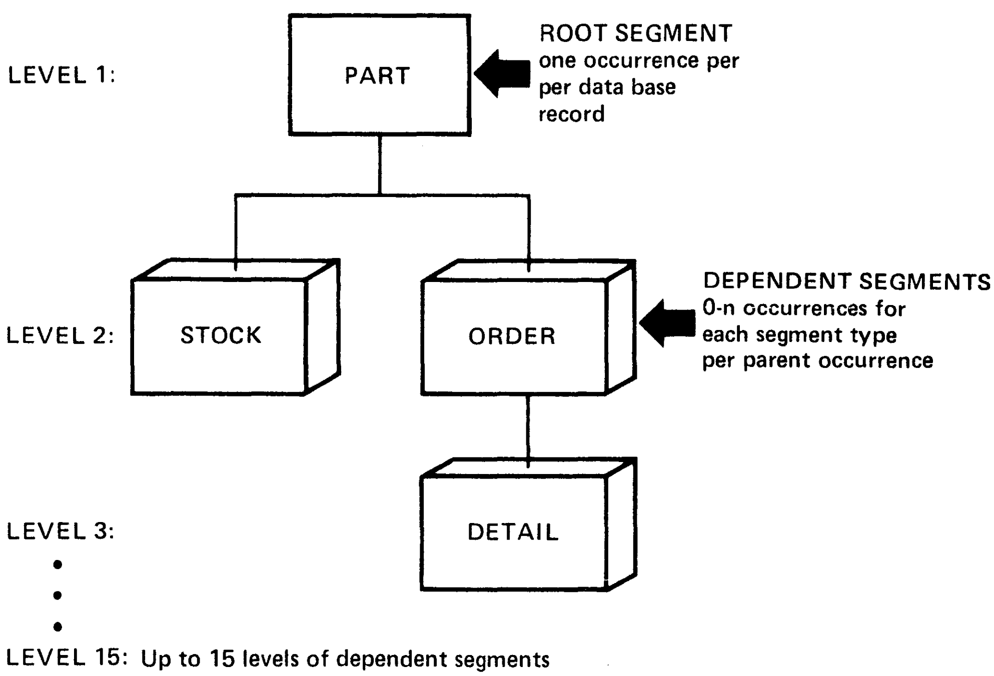


Figure 1-5. Relations between Segment, Data Base Record, and Data Base

Only one segment can appear at the first level in the hierarchy, but multiple segments can appear at lower levels in the hierarchy. For example, multiple STOCK and ORDER segments can exist for one PART segment. Since each dependent segment in the hierarchy has only one parent, or immediate superior segment, the hierarchical data structure is sometimes called a tree structure. Each branch of the tree is called a hierarchical path. A hierarchical path to a segment contains all consecutive segments from the top of the structure down to that segment.

In Figure 1-5, each PART segment with its dependent STOCK, ORDER, and DETAIL segments constitutes a data base record. The collection of all these records for all PARTs is called a data base, that is, the PARTS data base.

Through the concept of program sensitivity, DL/I allows a program to be restricted to "seeing" only those segments of information that are relevant to the processing being performed. For example, an inventory program could be written to see only the PART and STOCK segments of the data base record shown in Figure 1-5. The program need not be aware of the existence of the ORDER segment.

DL/I allows a wide variety of data structures. The maximum number of different segment types is 255 per hierarchical data structure. A maximum of 15 segment levels can be defined in a hierarchical data structure. There is no restriction on the number of occurrences of each segment type, except as imposed by physical access method limits.

Basic Segment Types In A Hierarchical Data Structure

Following is a detailed description of the several segment types and their interrelations within a hierarchical data structure. Figure 1-6 should be referred to when reading this description.

- The segment on top of the structure is the root segment. Each root segment normally has a key field which serves as the unique identifier of that root segment, and as such, of that particular data base record (for example, the part number).
- A dependent segment relies on some higher-level segment for its full meaning and identification.
- A parent/child relationship exists between a segment and its immediate dependents.
- Different occurrences of a particular segment type under the same parent segment are twin segments.
- Segment occurrences of different types under the same parent are sibling segments.

Sequence Fields and Access Paths

To identify and to provide access to a particular data base record and its segments, DL/I uses sequence fields. Each segment normally has one field denoted as the sequence field. The sequence fields in our subset should be unique in value for each occurrence of a segment type below its parent occurrence. However, not every segment type need have a sequence field defined. Particularly important is the sequence field for the root segment, since it serves as the identification for the data base record. Normally, DL/I provides a fast, direct access path to the root segment of the data base record based on this sequence field. This direct access is extended to lower level segments if the sequence fields of the segments along the hierarchical path are specified, too.

Note: The sequence field is often referred to as the keyfield, or simply, the key.

Figure 1-6 shows, as a dotted line, an example of an access path. It must always start with the root segment. This is the access path as used by DL/I. The application program, however, can directly request a particular DETAIL segment of a given ORDER of a given PART in one single DL/I request, by specifying a sequence field value for each of the three segment levels.

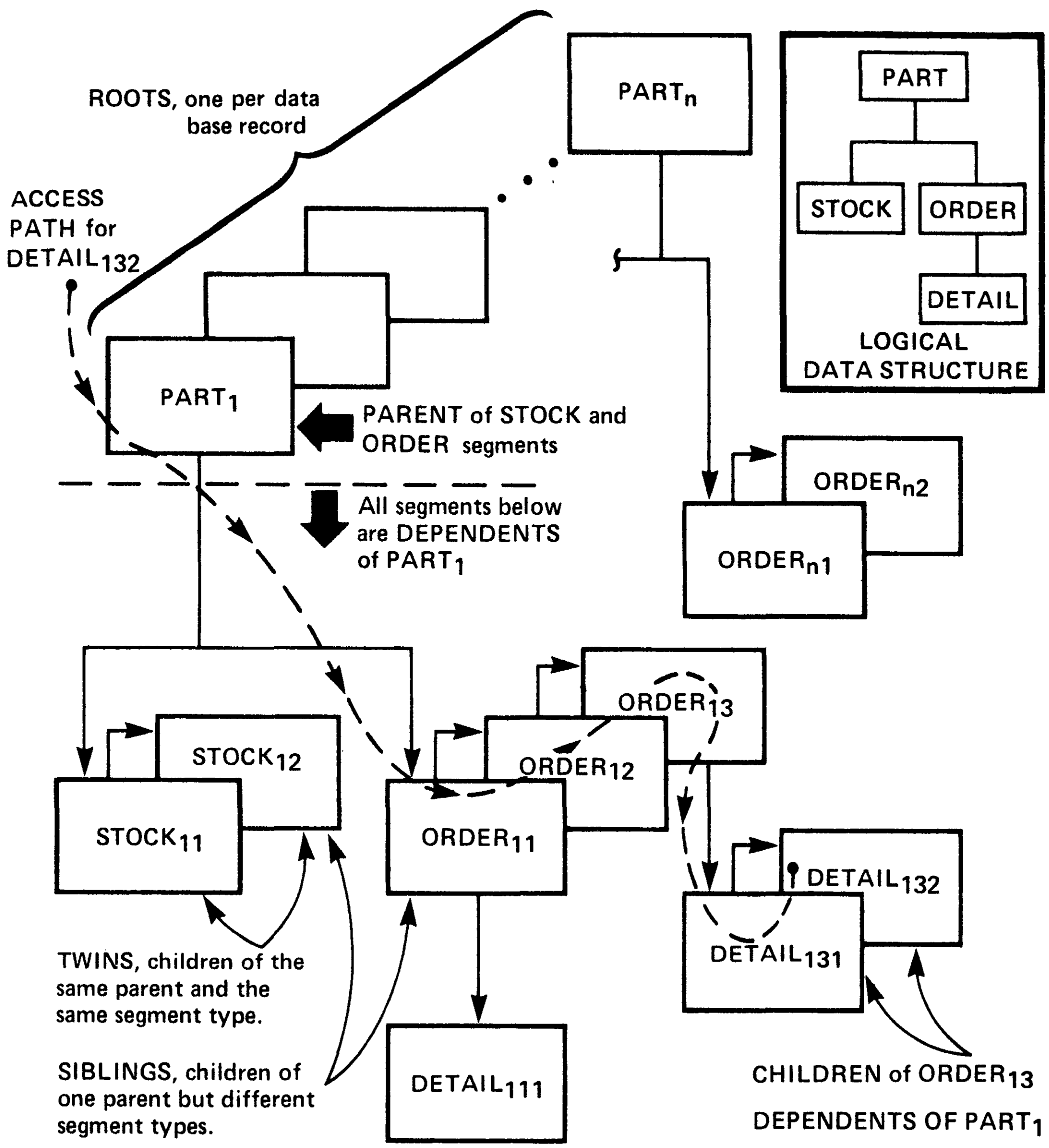


Figure 1-6. Segment Types and Their Relations in a Hierarchical Data Structure.

Logical Relationships

In addition to the basic DL/I facilities discussed so far, DL/I provides a facility to interrelate segments from different hierarchies. In doing so, new hierarchical structures are defined which provide additional access capabilities to the segments involved. These segments can belong to the same data base or to different data bases. A new data base can be defined called a logical data base. This logical data base allows presentation of a new hierarchical structure to the application program. Notice that although the connected physical data bases could constitute a network data structure, the application data structure still consists of one or more hierarchical data structures. This again extends the data independence concept.

The basic mechanism used to build a logical relation is to specify a dependent segment as a logical child, by relating it to a second parent, the logical parent.

In Figure 1-7, the logical child segment DETAIL exists only once, yet participates in two hierarchical structures. It has a physical parent, ORDER, and a logical parent, PART. The data in the logical child segment and in its dependents, if any, are called intersection data.

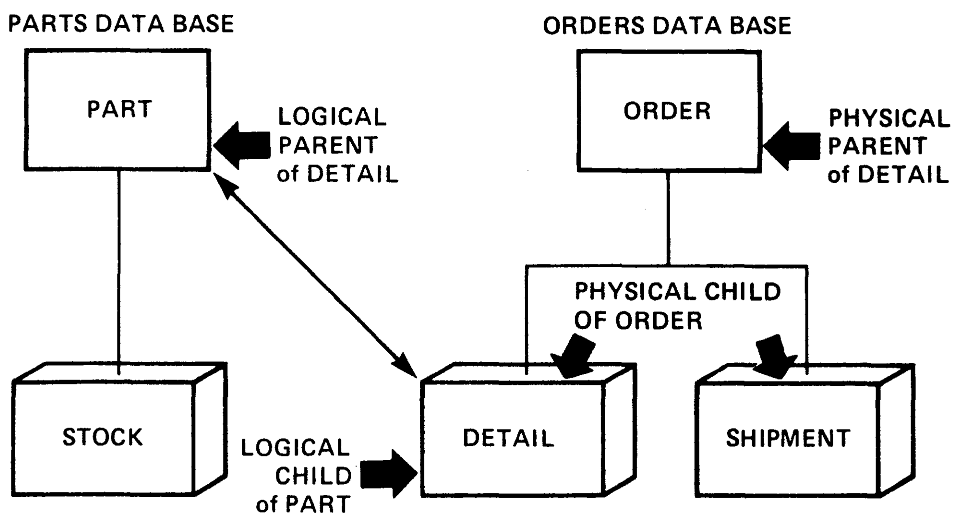
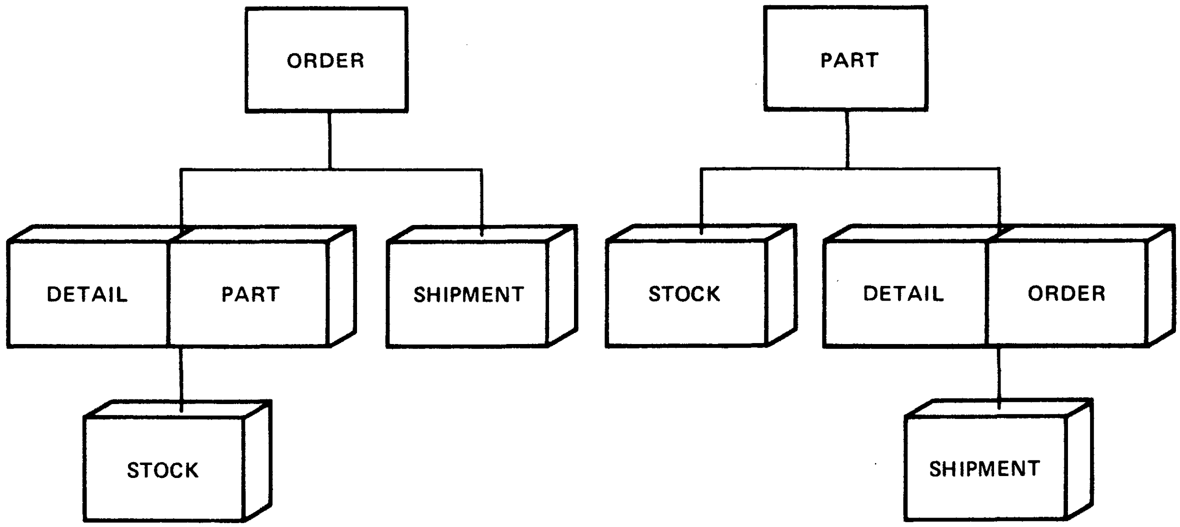


Figure 1-7. Two Logically Related Data Bases, PARTS and ORDERS

By defining two additional logical data bases, two new logical data structures shown in Figure 1-8 can be made available for application program processing, even within one single program.



A. New logical data structure ORDERPART B. New logical data structure PARTORDER

Figure 1-8. The Logical Data Bases After Relating PARTS and ORDER Data Bases.

The DETAIL/PART segment in Figure 1-8A, is a concatenated segment. It consists of the logical child segment plus the logical parent segment. The DETAIL/ORDER segment in Figure 1-8B is also a concatenated segment, but it consists of the logical child segment plus the physical parent segment. Logical children with the same logical parent are called logical twins, for example, all DETAIL segments for a given PART segment. As can be seen in Figure 1-7, the logical child has two access paths. One via its physical parent, the physical access path, and one via its logical parent, the logical access path. Both access paths are maintained by DL/I and can be concurrently available to one program.

Because the DL/I logical relationship function may not be required for your first IMS/VS application we will deal with it separately in this manual. To show the use of the DL/I logical relationship function we will use the phase 2 sample environment.

Secondary Indexing

DL/I provides additional access flexibility with secondary index data bases. Each secondary index represents a different access path to the data base record other than via the root key. The additional access paths can result in faster retrieval of data. For example, the PART and ORDER segments in Figure 1-9 could be retrieved based on the order number in the ORDER segment, if an index were defined for that field. Once an index is defined, DL/I will automatically maintain the index if the data on which the index relies changes, even if the program causing that change is not aware of the index.

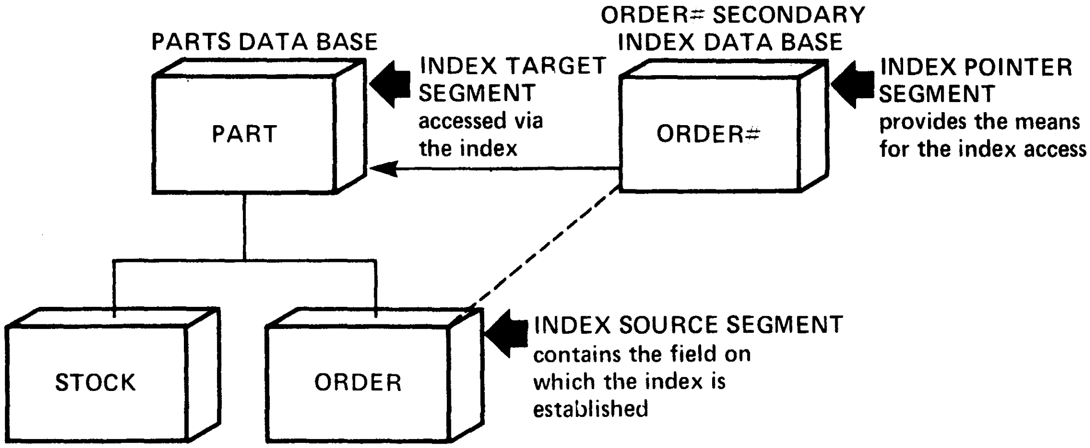


Figure 1-9. A Data Base and Its Secondary Index.

The segments involved in a secondary index are depicted in Figure 1-9:

- The index source segment contains the source field(s) on which the index is constructed, for example, ORDER#.

- The index pointer segment is the segment in the index data base that points to the index target segment. The index pointer segments are ordered and accessed based on the field(s) contents of the index source segment, for example, the order number. This is the secondary processing sequence of the indexed PARTS data base. There is, in general, one index pointer segment for each index source segment, but multiple index pointer segments can point to the same index target segment.

- The index target segment is the segment which becomes initially accessible via the secondary index. It is in the same hierarchical record as the index source segment and is pointed to by the index pointer segment in the index data base. Quite often, but not necessarily, it is the root segment.

- The index source and index target segment may be the same, or the index source segment may be a dependent of the index target segment as shown in Figure 1-9.

In our subset we will always choose the root segment as the target segment. With this approach, it is (for the application program) as if the index search field replaces the original root keyfield. At the same time, however, the original structure is still available to the same application program.

Because you might not need the secondary index function of DL/I, we separate its discussion throughout the manual. The use of this function is shown in the phase 3 sample environment.

DATA BASE DEFINITION

The data base definition language of DL/I provides two levels of data base definitions. Both are generated and maintained independently of your application program(s), thus providing the basis for data independence.

Data Base Description

The first level is the data base description (DBD). Each data base description is created from statements you provide. The statements define the hierarchical data structure and physical organization of the data base. These statements are input to a DL/I utility program. The output of the utility program is a data base description. It is stored in a DBD library. This data base description provides DL/I with the mapping from the application data structure of the data base used in the application program to the physical organization of the data used by the operating system data management access methods. The data structure can be remapped into a different physical organization without program modification. Other application data can also be added to this data base and not require a change to the original application programs. The concept of the data base description reduces application program maintenance caused by changes in the data requirements of the application. There are three types of DBDs:

- The physical DBD provides the definition of a single hierarchical structure. It can be used, in this form, by application programs. If logical relationships exist, the physical DBD contains a definition of those relationships with the other hierarchical structure. These relationships can be within the same DBD or with another DBD. Multiple logical relationships can exist within a single physical DBD.

- The logical DBD provides the redefinition of two or more related hierarchical structures into a new hierarchical structure. These hierarchical structures can be from the same DBD or from different DBDs. The logical DBD relies on the logical relationships which were defined in the physical DBD(s).

- The secondary index DBD allows definition of a secondary access path into a physical or logical DBD.

The process of generating a DBD is referred to as data base description generation (DBDGEN).

Program Specification Block

The second level of data base definition defines the application data structure for each application program.
A program specification block (PSB) is created from statements you provide for each of your application programs. It defines the application data structure required by that application program. A PSB contains one or more program communication blocks (PCBs), one for each hierarchical data structure the program intends to use. Each PCB defines the hierarchical (sub)structure the program "sees" from the physical or logical data base. It specifies for each segment the kinds of access allowed by the program, that is, read only, update, insert, and/or delete. The PSB is created, like the DBD, by a DL/I utility program. It is stored in a PSB library. The process of generating a PSB is referred to as program specification block generation (PSBGEN).

APPLICATION PROGRAM INTERFACE

IMS/VS provides a common data manipulation language, called the DL/I language interface, for the application program. Through this interface, the application program can request that DL/I:

- Retrieve a unique segment (GET UNIQUE)

- Retrieve the next sequential segment (GET NEXT)

- Replace the data in an existing segment (REPLACE)
- Delete an existing segment (DELETE)
- Insert a new segment (INSERT)

Such a request is often referred to as a DL/I call or call. A DL/I call may deal with one or more segments in a hierarchical path. Segment retrieval is based upon either or both of the following:

- Position in the data base, as set by previous calls
- Comparisons between fields within the segments in the specified path, and values supplied with the DL/I call.

The IMS/VS data manipulation language can be used in COBOL, PL/I or Assembler language programs. The data manipulation language is independent of data base organization and access methods. Only a small interface module is link edited to your application program.

LOGGING AND CHECKPOINT/RESTART FACILITY

DL/I provides a logging facility. If selected, images of data in the data base before and after modification are written to a system log data set. This log data set, together with a previously made image copy of the data base, can be used for data base reconstruction should an application or system failure occur. You may also include DL/I checkpoint calls in your batch application programs. This enables you to restart a job from the last checkpoint in the event of program or system failure.

DATA SECURITY

IMS/VS DB provides two mechanisms for data security. The first is the program specification block which controls the data base access of each application program at the segment level. For maximum benefit of this security provision, the library containing the PSBs should be password protected.

The second mechanism is the extended security support of IMS/VS which provides an interface between IMS/VS and the Resource Access Control Facility (RACF) program product in the OS/VS2 MVS environment. This extended security support is not included in our subset. For more information you should refer to the IMS/VS General Information Manual and the IMS/VS System/Application Design Guide.

UTILITY PROGRAMS

The IMS/VS DB system includes a comprehensive set of utilities. These utilities are used in our subset to:

- Implement logical relationships and/or secondary indexes at initial load time of the data base(s).
- Recover data bases in the event of program or system failure.

- Reorganize data bases, if needed, to:
    - Optimize direct access storage.
    - Change the storage organization or access method.
    - Change logical/physical data structure.
- Monitor the performance of programs to aid in optimization.

IMS/VS BATCH SYSTEM FLOW

The Data Language/I facility of IMS/VS is used in a batch-only data base environment as shown in Figure 1-10.

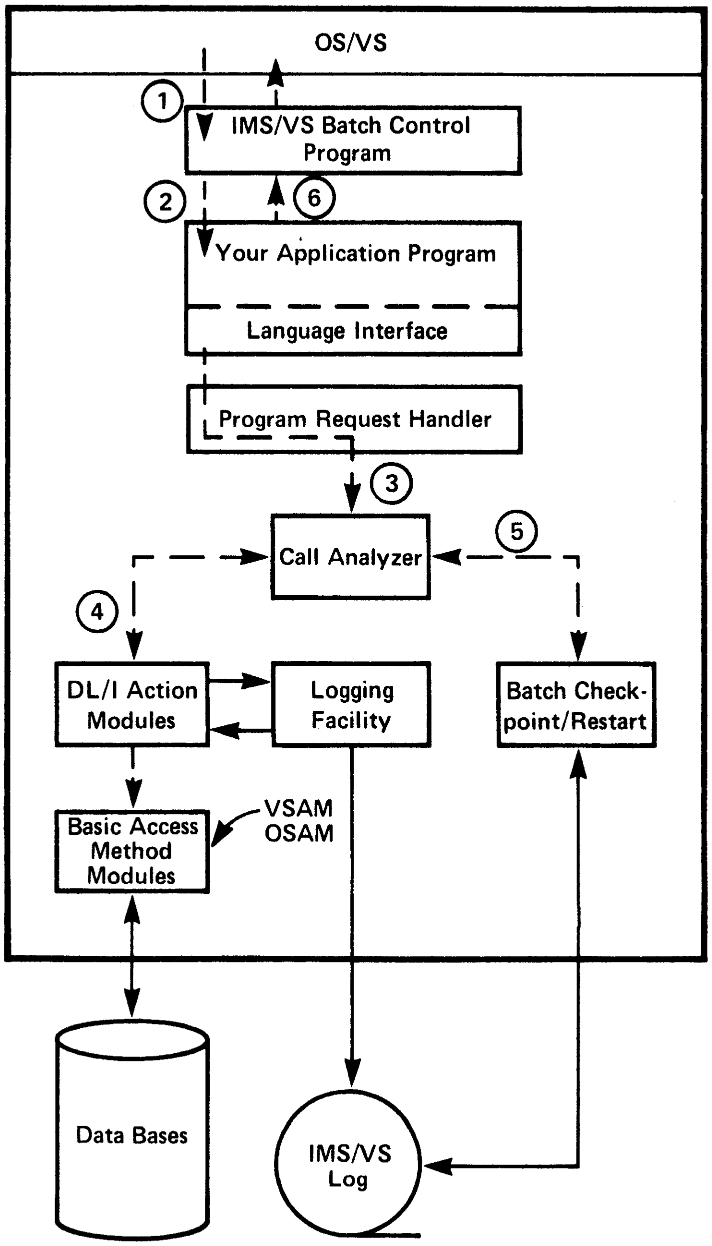


Figure 1-10. IMS/VS Batch Processing Region System Flow

The following notes relate to the circled numbers in Figure 1-10:

1. The IMS/VS batch control program is invoked by OS/VS task management. It supervises the loading of required IMS/VS modules and initializes the batch operating environment.

2. It links to your batch application program, which has been link-edited with the language interface.

3. When your application program issues a DL/I call, control is passed via the language interface to the program request handler. The program request handler provides preliminary checking of the call parameters, and passes control to the DL/I call analyzer.

4. Depending on the function requested, the DL/I call analyzer passes control to the appropriate call processor module. The DL/I action modules request services from the OS/VS data management access method modules and log their activity on the IMS/VS log.

5. Optionally your program can request a checkpoint to establish a restart point. Checkpoints are logged on the IMS/VS log to enable restart if required.

6. When your application program finishes processing, it returns control to the batch control program for termination processing.

Note: In the IMS/VS DB system, a data base can be accessed, for update, by only one application program (one partition/region) at a time.

DATA BASE ADMINISTRATION

The centralization of data and control of access to this data is inherent to a data base management system. One of the advantages of this centralization is the availability of consistent data to more than one application. As a consequence this dictates a tighter control of that data and its usage. Responsibility for an accurate implementation of control lies with the Data Base Administration (DBA) function. Because the actual implementation of the DBA function is largely dependent on a company's organization, we limit ourselves to a discussion of the characteristics of a DBA. Quite often, the DBA function at new IMS/VS installations is performed by an individual or group with experience in both application and system programming.

DBA CHARACTERISTICS

- The DBA provides standards for, and controls the administration of, the data bases and their use.

- The DBA provides guidance, review, and approval of data base design.

- The DBA determines the rules of access to the data bases and monitors their security.

- The DBA controls the data base integrity and availability, monitoring the necessary activities for reorganization and back-up/recovery.

- The DBA is not responsible for the actual contents of the data bases. This is a responsibility of the user. But the DBA enforces procedures for accurate, complete, and timely updates of the data bases.

- The DBA approves the operation of new programs with existing production data bases, based on results of testing with test data bases.

- The DBA is responsible for the maintenance of current information about the data in the data base. Initially, this responsibility might be carried out using a manual approach. But it can be expected to grow to a scope and complexity sufficient to justify, or even necessitate, the use of a data dictionary program.

NAMING CONVENTIONS

Good naming conventions are mandatory in a data processing project, especially, in a multi-application environment. They are a prerequisite for the eventual implementation of a data directory, or dictionary, system. In the following section we will propose a naming convention as an example, and we will use it in all the samples in this manual. You might adapt this convention to your own specific environment. In doing so, you should consider the following guidelines:

- Each entity should have a unique name.

- Each name should contain an entity classification.

- Each name should contain a system, application or project identification.

- Each name should contain a version identification.

Naming Convention For Entities

All entity names to be used in our sample will be coded: tsvmmmmm where:

```
    t = type identifier:
         B is DBD
         P is PSB and/or Program
         S is Segment
         F is Field
         D is DDname
         T is Transaction

    s = system, application or project identifier.  In all
         samples the following are used:

         E is an example
         0 is of general use

    v = version number.  In samples the following codes
         are used:
         0 if of general use
         1 if used in phase 1 and later
         2 if used in phase 2 and later
         3 if used in phase 3 and later

    mmmmm = mnemonic (user's choice)
```

Note: The online IMS/VS system requires the program and PSB name to be the same. Therefore, the programs are renamed during linkage-editing on the sample job.

Sample Job Names

The sample jobs referenced in this manual and listed in the "IMS/VS Primer Sample Listings", have the following naming convention:

- //SAMPInn for the IMS/VS installation jobs, if OS/VS1.
- //SMVSInn for the IMS/VS installation jobs, if OS/VS2 (MVS).
- //SAMPnnn for the sample application jobs.

SAMPLE DISTRIBUTION AND LISTINGS

The samples referenced in this manual are distributed as part of IMS/VS. After IMS/VS installation, as described in Chapter 7, "Installing IMS/VS," two sample libraries are available:

- IMSVS.PRIMESRC, which contains all the sample programs, DBDs, PSBs, data base input data, etc.
- IMSVS.PRIMEJCE, which contains all the sample jobs to install IMS/VS and exercise the sample programs and procedures.

For your convenience, both sample libraries are listed in the "IMS/VS Primer Sample Listings" publication, together with selected sample job output.

THE PROJECT APPROACH

The implementation of IMS/VS-based applications is most successfully done with a project approach. With this approach, you assure that adequate planning is done in a timely manner, stating all the necessary steps for the design, test, and installation of the application. For more complex applications, a project team with a definition of the tasks and responsibilities of all parties involved is recommended.

THE PROJECT CYCLE

Like most other data processing projects, an IMS/VS project can generally be divided into the following phases: preliminary investigation, planning, design and implementation, testing, and operation and maintenance. Figure 1-11 shows the relative manpower requirements for each of the phases.

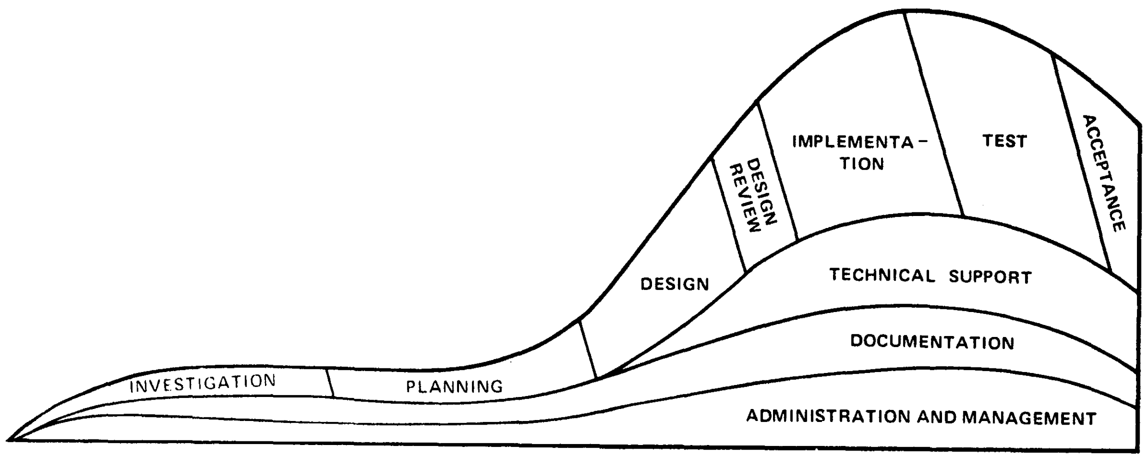


Figure 1-11. The Project Cycle

Following is a brief introduction to each of the phases:

The Idea: Normally, a user requirement or a management decision is the initial starting point of the project.

Preliminary Investigation: This phase concentrates on the definition of the objectives. A feasibility study, with a preliminary cost/benefit analysis, is conducted.

Planning: A project plan is established. A project team is formed, and the tasks and responsibilities of individuals and departments are defined. A budget is established for the project, and resources are allocated. Approval for the implementation is obtained. A change control procedure is implemented to control modifications during implementation.

Design and Implementation: The system is designed, and that design is reviewed. After design approval, detail designs are worked out and approved, coding is done, and a test plan is created.

Test: Both unit tests and integrated system tests are performed. These are followed by an acceptance test.

Operation and Maintenance: Production use of the system is started. Any further changes to the system are controlled via maintenance procedures.

Another important aspect is project administration. The timely and accurate planning for and establishing of standards and guidelines is mandatory for an efficient project implementation and later maintenance. Most organizations already have standards which should be extended into the data base environment. At a minimum, standards should be available for:

- Naming of data base items such as DBDs, PSBs, segments, and fields.
- Documentation of data structures, programs, and procedures (production, reorganization, recovery).
- Administration of data sets, data bases, back-up copies and log tapes and their interrelationships.

All of this should be under the control of a data base administration (DBA) function.

The following sample project plan should be adapted to your specific environment. Typical additional activities might be data clean-up, and conversion of existing programs and data.

Gross PERT Chart

Figure 1-12 shows a gross PERT chart for the implementation of an IMS/VS DB project. The necessary system-oriented activities, such as hardware and operating system installation, and system maintenance, are not included since these are largely dependent upon the installation environment. The following descriptions apply to the activities shown in the PERT chart (Figure 1-12).

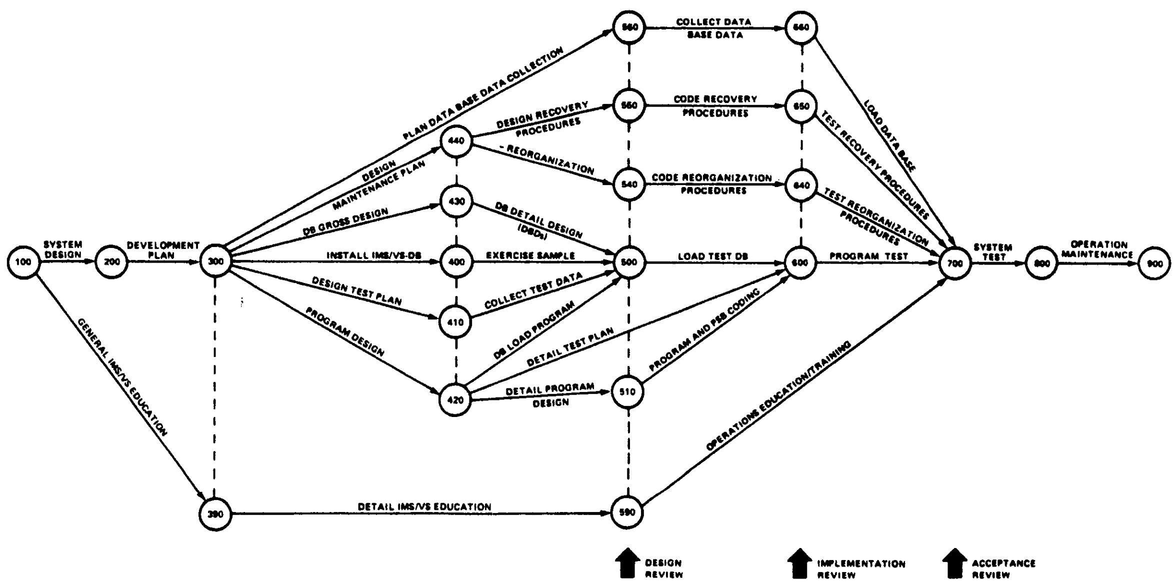


Figure 1-12. IMS/VS-DB Installation Plan PERT Chart.

System Planning (000-100): The sample PERT chart is adapted to your project. Manpower and machine time estimates are compiled. External interfaces are defined. Elapsed time calculations are performed, and the chart is extended with the proper timefram. The critical path is calculated. A Gantt chart can be constructed showing the duration and people involved for each activity. Figure 1-13 contains an example of such a Gantt chart. The Gantt chart should clearly state the actual days/months to be spent by each individual.

System Design (100-200): The overall system design is made. All components and their interfaces are defined. The user interface is detailed and reviewed for acceptance.

Development Plan (200-300): A detailed plan for the development of data bases and programs is devised. All single activities and their dependencies are determined.

Education (100-390-590-700): Together with the development plan, the education of all parties involved should be arranged.

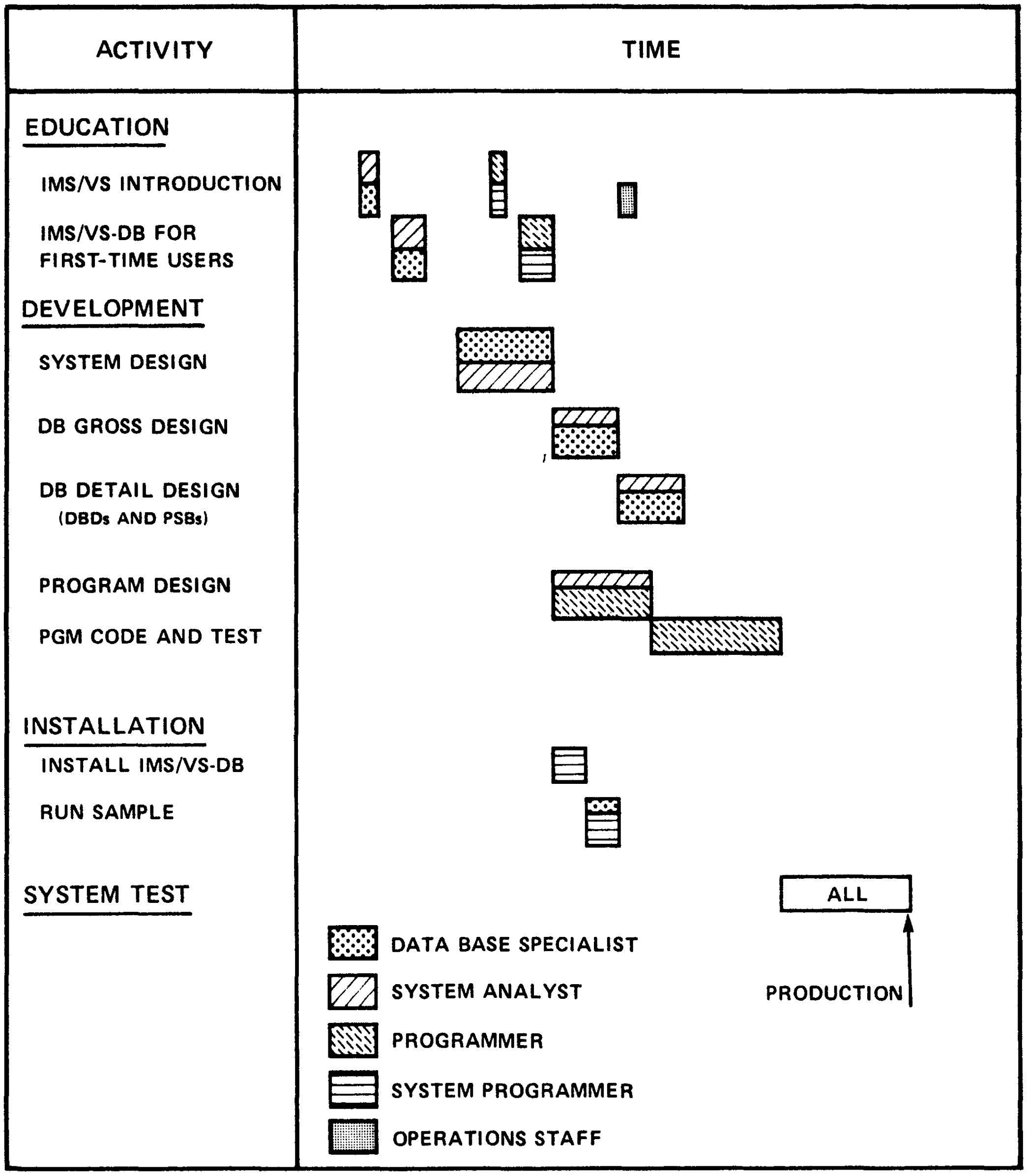


Figure 1-13. Sample Gantt Chart.

Data Base Gross Design (300-430): An overall data base design, specifying the logical data structures and the basic physical implementation, is created.

Program Design (300-420): The individual programs are defined and their input, processing, output and data base accesses are defined. Common guidelines and routines are established. Often more than 50% of the data base processing programs are reports. Using COBOL or PL/I report writer features or a report writer/query language such as GIS/VS can help to minimize the manpower required for program design.

Collect_Data_(300-410-500;300-560-660): Both test data and live data are collected, or procedures/programs are established for the conversion of existing data files.

Recovery_and_Reorganization_(300-440-550-650-700): A timely plan for recovering and reorganization can avoid later redesign and reprogramming. These procedures, although rarely needed, are vital to the data base integrity and availability. Therefore a thorough test plan must be made and carried out before production starts. The production staff should be carefully trained in problem determination and the secure and accurate execution of such procedures. An incomplete treatment of this topic is the most common source of problems when implementing a data base management system.

Install_IMS/VS_DB_and_Run_Sample_(300-400-500): The system programmer installs the IMS/VS data base system. The sample application provided with the system is exercised to get practical experience with the system. Conventions and procedures for system maintenance are established.

Data_Base_Detail_Design_(430-500): The detailed logical and physical data base structures are defined. Access methods are selected, and the DBDs are coded and tested.

Program_Detail_Design_(420-510): Detail flowcharts, decision tables, pseudocode, or other design documents, are established for each individual program. The data base call sequences are defined in a standard fashion.

Test_Plan_(420-600): A detail test plan is made. Procedures for unit test and system test are established.

Develop_Load_Programs_and_Load_Test_Data_Bases_(420-500-600): Load program(s) are designed, coded, and tested with the test data, resulting in test data bases for program and recovery/reorganization tests.

Design_Review_(500): The basic aim of the design review is to assure that the specified requirements are met. Major review topics are:

- Are the applications really what the users want?
- Is the performance expectation still valid?
- Are there any pitfalls in the data base and program design?

Program_and_PSB_Coding_and_Test_(510-600-700): Each individual program is coded and tested, using the test data bases and the test procedures.

Load_"Live"_Data_Bases_(660-700): The data bases are loaded with actual data. This process at times exposes inconsistencies in data. You may need to include extra time to resolve these inconsistencies. Back-up copies are made immediately after initial load to provide a full back base for system test.

System_Test_(700-800): Integrated tests are executed on the live data bases. Reorganization and back-up/recovery procedures are tested on those data bases.

Operation_and_Maintenance_(800-900): Production use of the system starts. The established monitoring and maintenance procedures are enforced. Feed-back is given to development for future projects. It is strongly recommended that the test environment be maintained in addition to the production environment. This will be of benefit for future trouble shooting, application modification, and application extensions.

THE IMS/VS DATA COMMUNICATION FEATURE

The IMS/VS Data Communication feature provides a symbolic program linkage between data communication terminals and the remainder of IMS/VS. This is in addition to the previously discussed Data Language/I facility, which is an integral part of the full IMS/VS DB/DC system.

In our subset we will mainly consider the operation of IMS/VS-DC in the Systems Network Architecture (SNA) environment, utilizing the Virtual Telecommunication Access Method (VTAM) and the Network Control Program/Virtual Storage (NCP/VS). However, Chapter 7, "Installing IMS/VS," also covers the use of the Basic Telecommunication Access Method (BTAM). Those not planning to use VTAM with IMS/VS should skip to the section "IMS/VS Data Communication Concepts."

We will similarly limit ourselves to the following hardware components:

- 3705 Local Communications Controller.
- IBM 3270 Information Display System, local and/or remote (leased lines only).

Figure 1-14 depicts the relations between these system components.

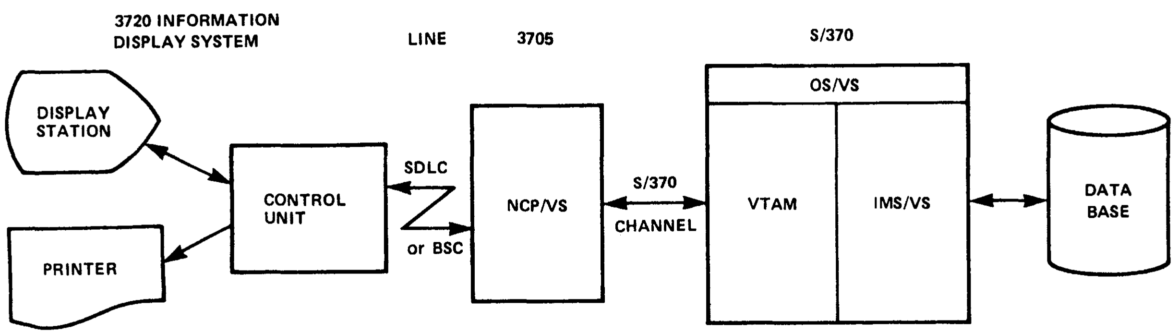


Figure 1-14. IMS/VS in the SNA Environment

SOME BASIC SNA CONCEPTS

Systems Network Architecture was designed as an architectual base for the development of a data communication network and its components, such as:

- Terminal subsystems -- IBM 3270 Information Display System
- Line protocols -- Synchronous Data Link Control (SDLC)
- Communication controllers -- 3705
- Network control programs -- NCP/VS
- Telecommunication access methods -- VTAM

SNA formally defines the functional responsibilities of communication system components. In an SNA structure, all nodes (linked elements) adhere to these definitions. The scope of SNA definitions ranges from bit-level message header formats to the protocol of message sequences and to the classification of network nodes according to function.

Separation of Functions into Logical Layers

A key concept of SNA is the division of the communication system functions into a set of well-defined layers. The major functional layers defined by SNA are:

- Transmission subsystem layer
- Function management layer
- Application layer

SNA is structured into these layers for two basic reasons:

1. To permit changes to be made in one layer without affecting other layers.

2. To allow interactions between functionally paired layers in different units. This pairing is required to support the distribution function.

The Transmission Subsystem Layer: The transmission subsystem is concerned with the routing and movement of data units between origins and destinations. The transmission subsystem does not examine, use, or change the contents of these data units. This separation, where the routing of a data unit is independent of the contents of the data unit, means that a change in the method of transmission between nodes requires no change in the data unit itself. Therefore, the support provided by the function management layer can be used across a variety of physical connections.

Paths through the network may be shared by many applications. The paths may consist of several physical components with interconnecting data links. The transmission subsystem provides the control necessary to manage these shared resources.

The Function Management Layer: The application layer employs a set of requests to invoke the services of the function management (FM) layer. The function management layer presents information from one application layer to another application layer. Separation of the function management layer from the application layer and from the transmission subsystem layer allows device-specific transformations to be distributed out of the main processor.

The Application Layer: The application layer is concerned only with application functions. This layer performs the user's application processing and need not be involved in the protocol or procedures for controlling a communication line or routing data units through the network.

Note: In SNA terminology, the whole of IMS/VS is called one application program. This use should not be confused with that pertaining to IMS/VS application programs written by the user.

End Users, Nodes and Sessions

End users are the ultimate sources and destinations of information. End users include programs (that is, IMS/VS) and operators (that is, terminal users). The structure of SNA allows end users to be independent of, and unaffected by, the specific services and facilities used for information exchange. End users are represented by nodes. So a 3270 display unit is a node. So is IMS/VS itself. Notice that a 3705 Communications Controller is also a node, an intermediate one. To allow information exchange between two nodes, these two nodes must be engaged in a session. Sessions are generally initiated (logon) and terminated (logoff) by one of the nodes.

VTAM Role in SNA

The *Virtual Telecommunication Access Method* (VTAM) is actually the implementation of SNA in CS/VS. VTAM manages the activities of a data communication system. It allccates resources and manages the flow of data between the nodes in the system. To accomplish this, VTAM provides the following functions:

Starting and stopping the network: VTAM enables an installation to define the data communication system and some of its characteristics. Once the system is defined, VTAM can be started and the system initialized. VTAM can alsc be used to shut down the system in an orderly fashicn.

Changing the configuraticn dynamically: VTAM enables the network operator at an OS/VS system console to monitor the use of the resources within the data ccmmunication system and to alter the network as necessary.

Allocation: VTAM controls the allocation of network resources. By owning and controlling all resources, VTAM provides a focal point within the system for controlling the netwcrk.

I/O processing: VTAM manages the transmission of data between applicaticn prcgrams (that is, IMS/VS) and terminals. It enables applicaticn prcgrams and terminals to communicate with each cther independently of how the terminals are ccnnected to the central prccessing unit. VTAM also relies upon the distributed function throughout the netwcrk (such as in ccmmunications controllers and prcgrammable terminals) to reduce the processing requirements in the central processing unit.

Reliability, availability, and serviceability (RAS): VTAM offers a design and facilities that reduce the incidence of problems in the data communication system, reduce the impact of errors that do occur, and assist in maintaining the data communication system.

NCP/VS and the 3705 Communicaticns Controller

The Netwcrk Control Frogram/VS active in the 3705 Communicaticns Controller, prcvides the following basic functions:

- Sending and receiving data to and from VTAM in the central processing unit (CPU) via a S/370 channel.
- Sending and receiving data to and from terminal control units via communication lines. Bcth binary synchronous communications (BSC) and SDLC line disciplines can be used with the 3270 Infcrmation Display System.

IMS/VS DATA COMMUNICATION CONCEPTS

The fcllowing sections give an overview of the concepts and facilities of our subset of the IMS/VS Data Communication feature.

Physical Terminals

Physical terminals are the hardware devices used to enter or reccrd messages being sent cr received over ccmmunication lines. Within the IMS/VS environment, physical terminals may be permanently attached to leased communicaticn lines cr cperate on a switched communication line (remote attachment), or be attached directly to the CPU channel (local attachment).

Although IMS/VS supports a wide variety of terminals and terminal subsystems, in this manual we will only consider the IBM 3270 Information Display System (also referred to as 3270) attached locally or via leased lines. In addition, we will limit ourselves to the following 3270 control units and their attached display and printer stations:

- 3271 Model 1, 2, 11 or 12
- 3272 Model 1 or 2
- 3274 Model 1B or 1C (BSC line protocol only)
- 3275 Model 1 or 2
- 3276 Model 1, 2, 3, or 4 (BSC line protocol only)

<u>3270 Device Compatibility</u>: The 3270 hardware provides for the display of a small size screen format on a larger size screen display unit. A 12x40 screen format for a 3277/3275 Model 1 will be displayed in the top left part of a 12x80 display unit (a 3276/3278 Model 1 or 11). A 24x80 screen format (for a 3277/3275/3276/3278 Model 2), will be displayed in the top part of a 32x80 display unit (3276/3278 Model 3) or a 43x80 display unit (a 3276/3278 Model 4).

Logical Terminals

A logical terminal is a name that is related to a physical terminal, that is, a node. One physical terminal can have one or more logical terminals associated with it. The user of IMS/VS refers to the logical terminal in the construction and transmission of messages. The user is never concerned about such things as physical terminal addresses. If a physical terminal becomes inoperative, the logical terminal(s) associated with that physical terminal can be dynamically reassigned to another physical terminal, thereby reassigning output queues of messages to another physical destination.

Master Terminal

The master terminal is a logical terminal that acts as the operational hub of IMS/VS. The master terminal operator has complete control of IMS/VS communication facilities, message scheduling, and data base operations. This facility is used for checkpointing and restarting the system, for continuous monitoring of the system, and for dynamically altering the operation of the system. In case of master terminal failure, the operating system console can be used as an alternate master terminal. Since the master terminal is a logical terminal, it may be dynamically reassigned to another physical terminal. In our subset, the master terminal must be a 3270 display unit with a screen size of 24x80 (1920 characters) in combination with a 3270 printer.

Input Messages

IMS/VS processes three basic types of input messages. The first one to eight characters of the first message segment determine the message type and identify the destination of the message text that follows.

- If the input message identifier is a transaction code, the message is a transaction, and its destination is the application program defined to process the transaction.

- If the input message starts with the name of a logical terminal, the destination is a terminal. This message type is known as a terminal-to-terminal message switch.

- If the first character of the input message is a slash (/), the message is an IMS/VS command. The command code immediately follows the slash. IMS/VS commands are entered by IMS/VS terminal operator to direct IMS/VS to display or alter the status of one or more IMS/VS system resources.

Output Messages

Output messages to IMS/VS terminals originate from application programs in response to terminal input, from IMS/VS itself, or from other terminals (message switches). An application program can send output messages to logical terminals other than the one generating the input message.

Message Format Service

IMS/VS provides a comprehensive editing facility for the IBM 3270 Information Display System and other terminals. It is called message format service (MFS).

MFS allows application programs to deal with logical messages instead of device-dependent data, thus simplifying application development. The presentation of data on the device or operator input may be changed without application program changes. Full paging capability is provided for display devices, thus allowing the application program to output a large amount of data to be divided into multiple screens for display on a terminal. The terminal operator can page through subsequent screens within the message. At the end he can return to the first page or skip to the next output message. Input can be accepted from any screen if so defined in MFS format definition statements.

The basic concept of MFS is that the application designer describes to the IMS/VS MFS language utility:

- The input message format as it will appear from the device

- The input message format as it is to be presented to IMS/VS and the application program

- The output message format that the program will present to IMS/VS

- The output message format as it is to appear on the device

Based on above descriptions, IMS/VS formats the data coming from the device going to the application program, and vice versa.

Message Queuing

All input and output messages, except command input, are queued in main storage, with direct-access storage backup as required. In this way, messages can be received by the system although the resources necessary to process them might not be immediately available. For improved performance, long and short messages are queued on separate direct-access data sets. Space in a message queue data set is reused when it is no longer required for a previous message.

Conversational Processing

Conversational processing lets the user retain message continuity from a given terminal even though the program that processes the conversation is not retained in main storage throughout that conversation. Whenever a transaction code is defined as conversational, the application program can interrelate messages from a given terminal using a scratchpad area (SPA). A unique SPA is created for each physical terminal from which a conversational transaction is entered.

Typical contents of the scratchpad area are data from the terminal and from data bases to be saved between interaction passes of the conversation. One scratchpad area is used for each terminal operating in conversational mode. IMS/VS automatically compresses and expands scratchpad contents to reduce data movement and I/O requirements.

Any subsequent data entry from a terminal already operating in conversational mode causes the message processing program processing the transaction to receive both the contents of the scratchpad area and the input terminal data. Each input message is considered as an individual unit of work for the program.

A terminal command is available to enable the terminal operator to end a conversation prior to its normal completion. Commands to temporarily suspend and save an incomplete conversation, and to resume that conversation at a later time are also available to the terminal operator.

Security

IMS/VS enforces several types of user-defined security requirements. In our subsets, two types of security verification may be designated: terminal security and password security. Terminal security ensures that a transaction or command may be entered only from specific, designated logical and/or physical terminals. Password security ensures that a transaction or a command message will not be processed unless a user-defined password is appended to the transaction code or to the command verb.

Security violations are recorded on the IMS/VS master terminal and system log after a specified threshold count. Access to IMS/VS data bases by non-IMS/VS applications or operations must be secured by the user's own operational policy and procedural controls. The extended security support of IMS/VS provides an interface between IMS/VS and the Resource Access Control Facility (RACF) program product OS/VS2 MVS only or a user written exit. This extended security support is not included in our subset. For more information you should refer to the IMS/VS General Information Manual and the IMS/VS System/Application Design Guide.

Terminal Command Language

The IMS/VS terminal command language is used by IMS/VS terminal operators to display and alter system resources. The major command functions are described below. Most commands can be specified to operate on one or more occurrences of a particular resource type. Most commands for dynamically interrogating or altering the processing functions of IMS/VS are limited to the master terminal. The major functions available to the master terminal operator through commands are:

- Starting, stopping, or otherwise modifying the system functions of message receiving, queuing, scheduling, and sending.

- Allowing the IMS/VS system to purge its message queues prior to shutdown.
- Temporarily halting transaction processing, message processing, program scheduling and execution, and data base usage.
- Starting and stopping message processing regions/partitions.
- Initiating and controlling IMS/VS checkpoints and restarts.
- Modifying logical terminal to physical terminal assignments.
- Displaying the status of various resources, such as transaction types, programs, data bases, message queues, and communications facilities.
- Displaying main storage buffer pool and control block pool utilization.

The major functions available to the remote terminal operator (nonmaster terminal operators) through commands are:

- Terminating, saving, or releasing a conversation.
- Sending a message to a selected logical terminal.
- Formatting a 3270 display screen for data input.
- Displaying the identification of the master terminal.

Transaction Response Mode

Response mode is an option that causes interactions between the terminal operator and the application program to be synchronized. When IMS/VS receives an input transaction that causes response mode to be used, IMS/VS accepts no more communication from that terminal until the application program response has been transmitted. This will be the typical mode of operation in our subset.

MESSAGE SCHEDULING

Separate operating system regions or partitions are used for message processing. These regions or partitions are initiated through the normal operating system job management routines during IMS/VS initialization or by an IMS/VS master terminal command during IMS/VS execution.

All messages acceptable to the system are predefined and verified through a 1- to 8-character code in the first segment of a message. When a valid message is completely received and queued, its presence is made known to message scheduling. When the required resources for message scheduling are available, processing is initiated.

LOGGING AND CHECKPOINT/RESTART

This facility supports logging, checkpointing, shutting down, and restarting IMS/VS executions. The online checkpoint and restart functions are dependent upon queuing all messages on direct-access storage and recording of all messages and data base modifications on the system log.

Logging

In the IMS/VS DB/DC system, all message and data base modifications are recorded on a central system log data set. This log data set is compatible with the DL/I batch log data sets. The data base changes of both types of log data sets can be accumulated into one change accumulation data set. This provides a consistent recovery mechanism for data bases used in online and batch operations. In our subset, this log data set must be on a magnetic tape.

Checkpoints

Periodic checkpoints of IMS/VS are used to provide the ability to restart after loss of main storage, direct access storage message queues, or data bases. The master terminal operator can enter commands to take a checkpoint and IMS/VS itself automatically takes a checkpoint periodically. The following checkpoints are distinguished:

- System-scheduled checkpoints based upon log activity.
- After a master terminal request for orderly termination of the system. Unprocessed input messages may be retained on direct access storage queues or recorded on the IMS/VS system log for subsequent processing.

Restarts

IMS/VS can be stopped and restarted daily or at explicit intervals. Restart reconstructs the system after a controlled stop, an emergency stop, or a data base destruction. To start the IMS/VS system, the operator instructs the operating system to start IMS/VS. Once the IMS/VS control program is operative, one or more jobs, which become IMS/VS processing regions/partitions, may be initiated. Remaining regions or partitions are used for batch processing. Upon initiation of the IMS/VS control program, a message is transmitted to the master terminal requesting an indication of the type of restart for IMS/VS. The operator's response causes control to pass to the restart facility, which optionally reads the old system log. This log contains input messages received but not processed, or output messages generated but not transmitted, on the previous execution of IMS/VS.

Any other information required to restart the system is also carried on the log. Messages on this log are put back into the same message queues in which they were left at the previous system stop. After completion of the restart processing, the master terminal operator may enter commands to initiate communication line operation, message processing, and data base use.

Restart without a system log is equivalent to an initial start (cold start) for all message transmission and processing.

When IMS/VS is restarted after an abend, the restart capabilities of IMS/VS provide the following information to the master terminal:

- The name of the message processing program that was executing in each message processing address space at the time of abend.
- The input messages that caused the message processing programs to be scheduled.

Data base modifications are logged. This information is used at restart. The data base modifications caused by programs in process at time of failure are automatically backed out and the original input messages are reprocessed in their entirety.

In addition to system restart, facilities are provided to reconstruct data bases using image copies and the system log.

UTILITY PROGRAMS

In addition to the utility programs available with DL/I, the IMS/VS Data Communication feature provides several utility programs. The following utility programs are introduced in our subset:

- Application control block maintenance. Uses the output of program specification block and data base description generations to create and maintain the control blocks in a form directly usable by the IMS/VS online system.

- Security maintenance. Creates control blocks that describe the terminal and transaction security requirements. IMS/VS uses a scheme of passwords for terminal and transaction access.

- System log analysis. Produces statistical reports having usage of message types and terminals.

- MFS language. Creates control blocks that describe message and device formats for devices using message format service (MFS).

- A DC Monitor report program which provides information regarding the performance of the system. This is based on the output collected by an optionally activated monitor in IMS/VS.

- System log recovery and termination programs to recover or terminate the system log in case of machine errors.

IMS/VS DATA BASE/DATA COMMUNICATION SYSTEM FLOW

The following three kinds of regions, address spaces, or partitions under OS/VS are distinguished in an IMS/VS-DB/DC system.

- The control (CTL) region contains the IMS/VS control program. It controls the terminals and data bases.

- The message processing program (MPP) region hosts the application programs for message driven processing of the data bases. The MPP region is controlled by and relies upon the CTL region.

- The batch message processing (BMP) region contains an application program for batch processing of the data bases managed by the CTL region.

Once the IMS/VS control region or partition and one or more message processing regions or partitions have been initialized by the operating system job management facility, the following system flow occurs. See Figure 1-15.

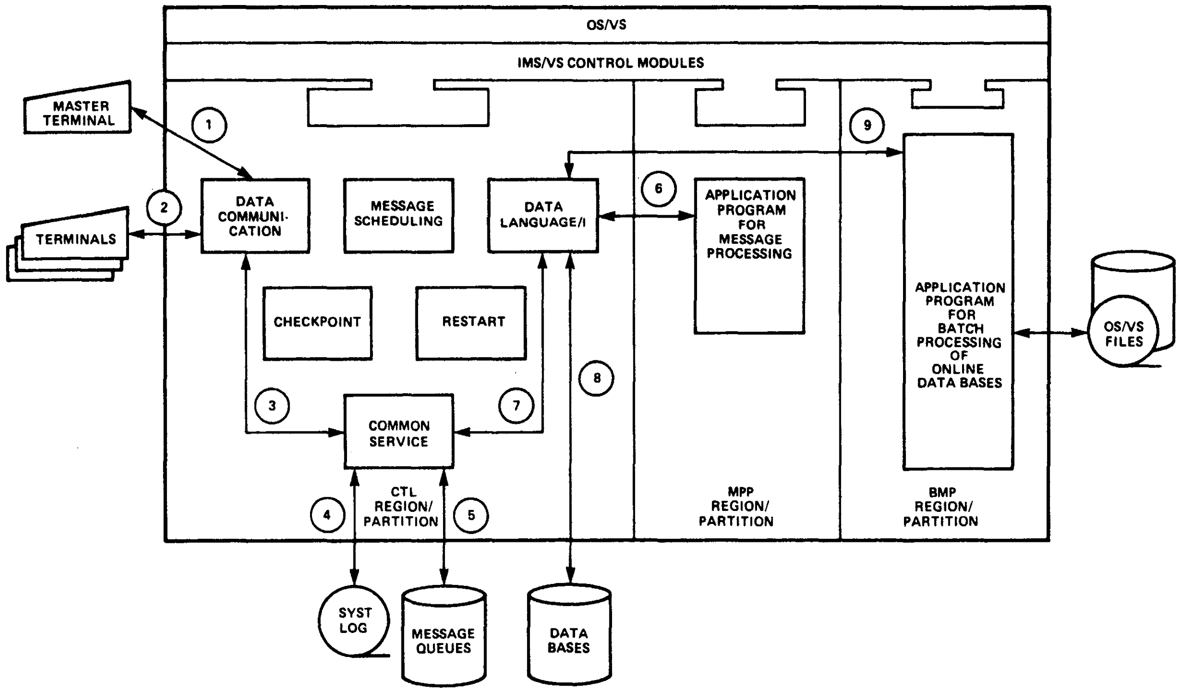


Figure 1-15. IMS/VS Data Base/Data Communications System Flow

The following notes relate to the circled numbers in Figure 1-15.

1. The data communication facility (event 1) requests restart instructions from the master terminal. After the completion of restart, the master terminal enables communication from all user terminals (event 2).

2. When an input message or message segment is received (event 2), data communication calls the common service (event 3), and the input message is logged (event 4) and queued (event 5).

3. When there are input messages queued and waiting for processing, and a message processing region or partition becomes available, control is passed to scheduling to determine the application message processing program to be scheduled. The application program is loaded (if needed) into a region/partition and given control.

4. The application program subsequently makes requests for the input message and/or data base reference (event 6). Control passes to DL/I for either message reference (event 7) or data base reference (event 8). The message reference is accomplished through common service.

5. While the application program is executing, modifications can be made to the data base (event 8) and/or output messages may be queued (events 5 and 7).

6. When the application program terminates or requests another input message, all its queued output messages are transmitted to the designated output terminal(s) (events 3 and 2) in our subset.

BATCH PROCESSING OF ONLINE DATA BASES

Once the IMS/VS control region or partition has been initiated by the operating system, a _batch message_ processing (BMP) region or partition can be initiated. The application program in the BMP region or partition is scheduled by operating system job management. Its execution, however, is controlled by the IMS/VS control region. This BMP region or partition may contain an application program for batch processing of online data bases. DL/I is used for data base reference and update (Figure 1-15). Any data reference is initiated by the batch message processing program (event 9).

DATA COMMUNICATION ADMINISTRATION

The data base administration function as introduced in the first part of this chapter is extended and complemented with a data communication administration function in the IMS/VS DB/DC environment.

DCA CHARACTERISTICS

- DCA provides standards for and controls the administration of the online system and its data base use.
- DCA provides standards and guidelines for message format service usage and enforces the administration of device and message formats.
- DCA is responsible for the transaction and logical terminal security control. Passwords should be regularly changed and the security maintenance utility should be used in a controlled manner.
- DCA maintains the logical terminal, physical terminal, and mode or physical line assignments. DCA interfaces with network control or provides that function itself.
- DCA is the central contact function for a user liaison group or implements that function itself.

SAMPLE IMS/VS DB/DC PROJECT PLAN

The sample IMS/VS DB project plan as discussed earlier in this chapter (see Figure 1-12) can easily be extended to the IMS/VS DB/DC environment. Figure 1-16 shows a gross PERT chart for such a project.

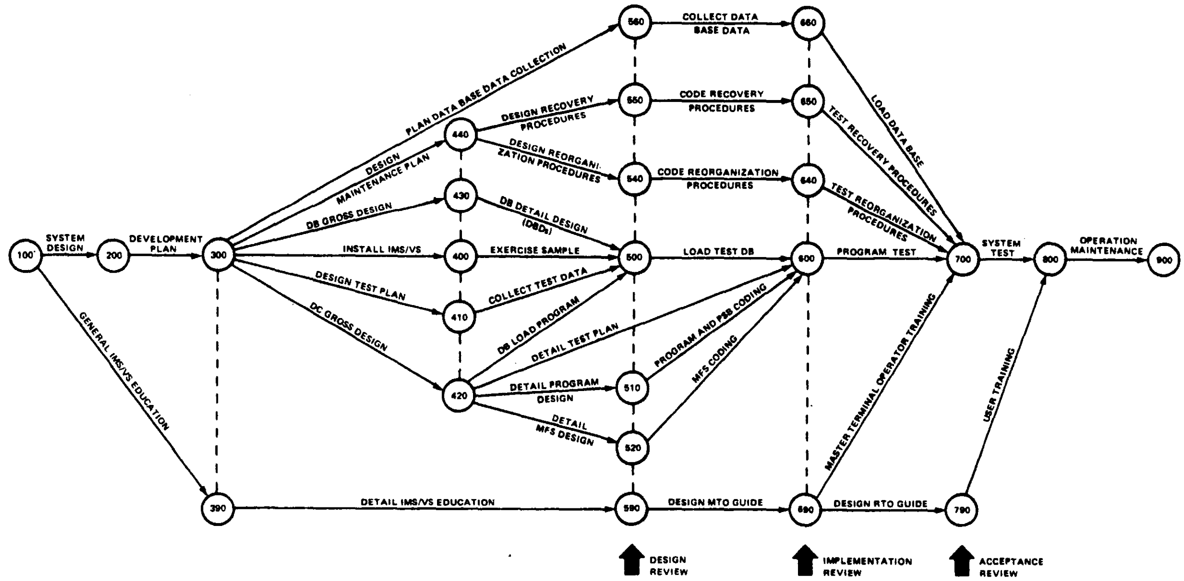


Figure 1-16. IMS/VS-DE/DC Installation Plan PERT Chart.

The following activities should be extended or added to the DB-only version.

Install VTAM, NCP and IMS/VS (300-400): The system programmer installs VTAM, NCP (if remote network) and the IMS/VS DB/DC system.

DC Gross Design (300-420): The transactions, programs, device and message formats, and their interrelations are defined.

Detail MFS Design and Coding (420-520): The formats are developed in a standard way. They are tested with their corresponding message processing programs.

Design MTO Guide and MTO Training (590-700): The IMS/VS Primer Master Terminal Operator's Guide should be adapted to your environment and used in the training of the MTO.

Design RTO Guide and User Training (690-800): The IMS/VS Primer Remote Terminal Operator's Guide should be adapted to your environment and used in the training of the remote terminal operators. The end user departments should be educated in a timely manner in the use of an online system.

IMS/VS PRIMER FUNCTION SUBSET OVERVIEW

The IMS/VS Primer Function is an implicit, open-ended subset of standard IMS/VS functions. The subset selected is aimed at the first time IMS/VS user, developing his first and simple IMS/VS application. Following is a brief overview of the IMS/VS Primer Function subset.

This overview is mainly aimed at the existing IMS/VS user. It should be used to identify the usability and/or limitations of the Primer Function in your environment.

Data Base Subset

DL/I Storage Organization and Access Methods:

- Only HDAM, HIDAM, SHISAM, and GSAM (BSAM)
- VSAM, OSAM, and BSAM (for GSAM)
- No ISAM or OSAM for HIDAM
- Single data set groups
- Single volume OSAM data set
- No variable length segments
- No segment compression
- No DL/I exits, except HDAM randomizing modules
- No hierarchic pointers
- Basic rules/recommendations for pointer selections
- SCAN=3 (default) in DBD
- Only HIDAM free space distribution parameter (FRSPC)
- No BLOCK or RECORD parameter in DBD; mandatory SIZE parameter
- No 3850 support

Logical Relationships:

- Only bi-directional virtual pairing
- No uni-directional or bi-directional physical pairing
- Mandatory RULES=VVV for logical child segment
- Mandatory RULES=PLV for physical and logical parent segment
- Mandatory physical storage of logical parent concatenated key
- Mandatory sequence field in logical path to the logical child
- Basic rules/recommendations for pointer selection

Secondary Indexes:

- Root segment is always target segment (no inverted structures)
- No overflow data set (ESDS); mandatory /SX field if non-unique keys in index pointer segment
- No shared indexes in a secondary index data base

DL/I Call Functions:

- GU, GN, GHU, GHN, ISRT, REPL, and DLET calls
- XRST and CHKP calls (only extended checkpoint/restart function)
- No Boolean qualification statements in segment search arguments

- D, N, F, L and - command codes (path call included)
- No multiple positioning (multiple PCBs will be used)

Data Base Design:

- Simple data base design technology based on the transaction/data element matrix
- Basic rules/guidelines for logical and physical design, including organization, access method, and pointer attribute selection

Data Base Reorganization:

- All logical related data bases are reorganized at the same time (data base scan utility not used)
- No utility control facility (UCF)
- Simple guidelines for monitoring reorganization requirements
- No partial reorganization

Data Base Recovery:

- Recovery with and without DL/I log tape
- Mandatory log data set change accumulation
- Mandatory write ahead log tape
- Log tape recovery
- No utility control facility (UCF)
- Simple procedural guidance for data base error detection, classification, and recovery
- Basic guidelines for image copy and log tape administration, and data set retention periods
- No online image copy

Installation and Operation:

- Only OS/VS1 and OS/VS2 (MVS) support
- VSAM is mandatory
- The IMS/VS DB installation process is described separately from the DB/DC installation process
- No ACBLIB for data base only system
- Simple interpretation guidelines for DB Monitor output and data base buffer pool statistics
- No support for power warning feature
- Sample application programs written in both ANS COBOL (compiler used: OS/VS COBOL, 5740-CB1) and PL/I (optimizer compiler used: 5734-PL3)

Data Communication Subset

Device Support:

- IBM 3270 Information Display System, the following control units and attached terminals, via BTAM or VTAM, locally or remotely attached:

  - 3271 Model 1, 2, 11 or 12

  - 3272 Model 1 or 2

  - 3274 Model 1B or 1C (BSC line protocol only)

  - 3275 Model 1 or 2

  - 3276 Model 1, 2, 3, or 4 (BSC line protocol only)

DL/I Message Call Functions:

- GU, GN, ISRT, and CHNG calls
- XRST/CHKP call combination for BMPs

Message Format Service:

- Only message formatting option 2
- Dynamic cursor positioning via attribute byte only
- Single segment input
- No multiple page input
- One output segment equals one logical page, equals one physical page
- Logical paging, no physical paging
- No device type mixing in MFS
- No program function keys (PFKs)
- No message field and segment edit routines
- No prompt facility
- No operator control tables
- No selector pen
- No operator identification card reader

Message Processing:

- Only single segment input messages
- Multi segment output message
- Maximum length output segment of 1388
- Only response mode transactions
- Non recoverable inquiry only transactions
- Recoverable update transactions

- Conversational transactions, with fixed size main storage scratch pad area (SPA) of 1300 bytes

Data Communications Design:

- Concepts of online transaction design, based on application, terminal user, and system characteristics
- Basic MPP structure for simple inquiry, update and conversational programs
- Basic guidelines for screen design
- On-line data base design considerations

Installation and Operation:

- Long message queue record length of 1500 bytes
- Short message queue record length of 250 bytes
- VTAM or BTAM; no mixture
- Forced terminal and password security
- No exit routines in IMS/VS except HDAM randomizing modules
- One MPP and one BMP region
- No message queue access via BMP
- Mandatory IMS/VS shutdown for data base recovery or reorganization
- Single mode transaction scheduling
- Single transaction scheduling class and priority
- No program-to-program switching
- Sample set of VTAM Level 2 definition statements
- Sample set of NCP/VS definition statements to be used with IMS/VS and VTAM Level 2 sample definitions
- Single log tape only
- No online DUMPQ
- No disk logging and enhanced restart
- No automated operator interface support
- No resource access control facility (RACF) support
- Mandatory hardcopy of all eligible master terminal commands and responses
- Only /ASSIGN, /BROADCAST, /CHECKPOINT, /CLSDST (VTAM only), /DEQUMP, /DISPLAY, /ERESTART, /EXIT, /FORMAT, /HOLD, /IDLE (BTAM only), /NRESTART, /OPNDST (VTAM only), /PSTOP, /PURGE, /RCLSDST (VTAM only), /RELEASE, /RSTART, /START, /STOP, and /TRACE commands for the Master Terminal Operator
- Only /EXIT, /FORMAT, /HOLD, /RCLSDST, and /RELEASE commands for the Remote Terminal Operator

CHAPTER 2. DATA BASE DESIGN

As in almost any system implementation, the design is the most challenging task to be performed. The best optimization or tuning effort which you can perform is a sound initial design. On the other hand, a designer is often bound to a time limit and does not know all future requirements. To cope with these problems, a designer needs a good plan and proper techniques.

The most crucial topic in the design of applications with data base management systems is the data base design. In this chapter we will introduce data base design with DL/I. We will also provide guidance in selecting those DL/I functions which will result in an open-ended design. Our major objective is a good overall design resulting in good overall performance rather than a design which maximizes the performance of a single application program.

Should you have a specific performance objective for a particular application, then you are advised to study Chapter 9, "Optimization," in detail after reading this chapter, and before starting your actual data base design.

ABOUT THIS CHAPTER

This chapter consists of three parts.

1. Introduces the sample application in detail. It sets the requirements and the environment for the actual data base design process. It is meant to give the background for the examples used in the two following parts.

2. Introduces the functions of DL/I, available to the data base designer. It also contains the specification of the DL/I data base definition language. This part will be the major reference area after the initial study of this chapter.

3. Introduces the concepts, techniques, and guidelines for the designing of data bases with DL/I. It is aimed at those individuals who are designing their first data bases with DL/I. As such, it is more oriented towards learning than referencing.

Each of the above three parts is constructed along the three phases of data base implementation:

- Phase 1: Basic data bases
- Phase 2: Data bases with logical relationships
- Phase 3: Data bases with secondary indexes

With this gradual approach you should be able to design simple data structures with a minimal amount of effort and still be able, when the need arises, to exploit the full DL/I function. Once again, you should realize that data base design is not just a matter of creative imagination. Most of it is systematic labor. The intent of this chapter is to help you with this, by providing techniques for an efficient accomplishment of this challenging task.

SAMPLE DATA BASE REQUIREMENTS

PHASE 1 SAMPLE REQUIREMENTS

PARTS Data Base Contents

Following is a list of all the data elements to be stored in the PARTS data base together with their system names. The system name follow the naming convention described in Chapter 1.

PARTS Data Elements:

Name	Description	Length
PE1PGDSC	Part name, full description	50
PE1PGSNM	Part name, short description	13
PE1PGPNR	Part number code	8
PE1PGUNT	Unit of measure for quantities	8
PE1PGPRI	Part base price	8
PE1PGDIM	Unit dimensions	8
PE1PSLCC	Stock physical location code	12
PE1PSCNT	Stock physical count quantity (tally)	6
PE1PSDAT	Date of last physical stock count	6
PE1PSISS	Total issued from stock in current period	6
PE1PSREC	Total receipts to stock in current period	6
PE1PPOSU	Supplier's name	20
PE1PPQCD	Quantity ordered	6
PE1PPQRD	Quantity received	6
PE1PPODT	Purchase order date (MMDDYY)	6
PE1PPDDT	Delivery date (MMDDYY)	6
PE1PPONR	Purchase order number	8
PE1PGNEW	New (superseding) part number	8
PE1PGOLD	Old (superseded) part number	8
PE1PGEQV	Equivalent part number	8

Inventory Report Processing

Every week a report is made of all the parts in stock with a listing of:

- Part number
- Part name, short
- Part name, long (optional)
- Quantity in stock
- Quantity issued from stock in current period
- Quantity received in current period
- Quantity on order

We will refer to this application function as transaction TE1INVPP. On demand (averaging twice a day), the same information is needed for specific parts, normally 1 to 10. This transaction, TE1INVQU should be designed with the idea that it will be done online at a later stage.

Purchase Order Processing

Daily, an average of 100 orders are processed, each containing an average of 2 parts and a maximum of 6. The purchase order forms, delivered by the purchase department, are keypunched and sorted in purchase order/part number sequence. This application is also planned to go online in the near future, with video terminals installed at the purchase order department.

Note: An order signal list could be produced in the same program which generates the weekly parts inventory report but this will not be addressed in our sample.

The functions performed by this application are:

- Entry of new orders, transaction TE1PONEW.
- Change of existing orders, transaction TE1POCNG.
- Deletion of orders after delivery, transaction TE1PODEL.

PHASE 2 SAMPLE REQUIREMENTS

Sample Data Bases for Phase 2

In the phase 2 environment we will add the Customer Order Processing Application. This application requires information from the:

- Existing Parts data base
- Existing Central Customer file
- New Customer Orders data base

The data elements required from each of these are described below.

Parts Data Elements: Primarily the same data elements as in phase 1 are required, although some are not used in this application.

Central Customer Data Elements:

Name	Description	Length
FE2PCNUM	Customer Number	6
FE2PCNAM	Customer Name	20
FE2PCADR	Customer Address	20
FE2PCCTY	Customer City	20
FE2PCPCD	Postal Code	6

Customer Order Data Elements:

Name	Description	Length
PE2OGREF	Order Number	6
PE2OGSTA	Order Status Code	2
PE2OGCNR	Customer Number	6
PE2OGCDT	Order Entered Date	6
PE2OGDDT	Order Due Date for Delivery	6
PE2OGDWK	Order Due Week for Delivery	2
PE2OGSPC	Special Delivery Instructions	20
PE2OGORI	Order Origin Code	2
PE2ODPNR	Part Number This Orderline	8
PE2ODQTY	Part Quantity Ordered	6
PE2ODPRI	Part Base Selling Price	8
PE2ODTAX	Part Sales Tax Category	1
PE2OSNR	Shipment Number	8
PE2OSDAT	Shipment Date MMDDYY	6
PE2OSMET	Shipment Method	20
PE2ODBOR	Backorder Flag	1

Sample Application for Phase 2

The sample application for Phase 2 is Customer Order Processing. This consists of three basic transactions:

- TE2CONEW -- adds a new customer order to the Customer Orders data base
- TE2COCNG -- changes data in an existing customer order
- TE2CODEL -- deletes a customer order from the data base

The customer order characteristics are:

- An average of 500 orders per day, maximum of 1000
- Each order contains a maximum of 8 order lines, one order line for each part type ordered, an average of 3 order lines per order.
- The average delivery time of an order is two weeks.
- After delivery, the order is deleted from the data base.
- Access to the order information is required via both the order number and the part number (the latter, for changing the order whenever changes to the status of a part occur).

Additionally, another application requires access to the customer orders for each part. So we must link the part and customer order information. The customer name and address are also needed during customer order processing. This information is maintained in the Accounts Receivable application which will not be converted at this time. This information is stored in a VSAM KSDS. The key of this KSDS is the customer number, which will be stored in the Customer Order data base for reference. This KSDS will be defined and accessed as a root only DL/I data base.

The advantages to this approach are:

- The current KSDS is still available for the non-DL/I Accounts Receivable application.
- The same KSDS can be processed as a DL/I data base, thus allowing the new customer application full DL/I function.

- This root only data base can easily be extended with additional segments when the Accounts Receivable application is converted to DL/I. This conversion can be done with minimal impact on the Customer Order application.

PHASE 3 SAMPLE REQUIREMENTS

For the phase 3 sample application, we incorporated one additional requirement for the purchase order application. This requirement provides fast purchase order information for one or more purchase orders, based on the purchase order number. To implement this new transaction, IE3PCINQ, we will utilize the secondary index function of DL/I.

THE DL/I DATA BASE FACILITY

This part of Chapter 2 provides an introduction to the DL/I functions and their use. It is the main source of reference for the data base designer and/or data base administrator. This part is subdivided into:

- A discussion of the DL/I data base organizations
- A presentation of the DL/I data base definition language

The first part provides the insight into DL/I necessary for the data base design. The second part provides details for the implementation of the data base(s). Each part has three sections. These sections cover the following main data base facilities:

- Physical data bases and storage organizations
- Logical relationships
- Secondary indexes

PHYSICAL DATA BASE AND STORAGE ORGANIZATIONS

To support a wide variety of data base requirements, DL/I provides several data base storage organizations. However, your application programs will be typically independent of the particular organization chosen for a given data base.

In our subset, we will limit ourselves to the following data base storage organizations and their associated data base types.

Organization	Data Base Type
Hierarchical Direct Access Method	HDAM
Hierarchical Index Direct Access Method	HIDAM
Simple Hierarchical Index Sequential Access Method	SHISAM
Generalized Sequential Access Method	GSAM

The data base type, its organization, and structure are defined in the <u>data base description</u> (DBD). To use a data base in an application program, you must provide a <u>program specification block</u> (PSB). The PSB specifies the data base(s) to be used and the kind of usage required. DBDs and PSBs are created during <u>data base description generation</u>

(DBDGEN) and *program specification block generation* (PSBGEN), respectively. This is discussed in detail later in this chapter.

Before discussing each of the above organizations in detail, we will first elaborate some more on some basic DL/I concepts which were introduced in Chapter 1.

The DL/I Data Base Record

As introduced in Chapter 1, a DL/I data base record as shown in Figure 2-1 consists of one root segment and a number of dependent segments. Each dependent segment can have a variable number of occurrences below its parent occurrence.

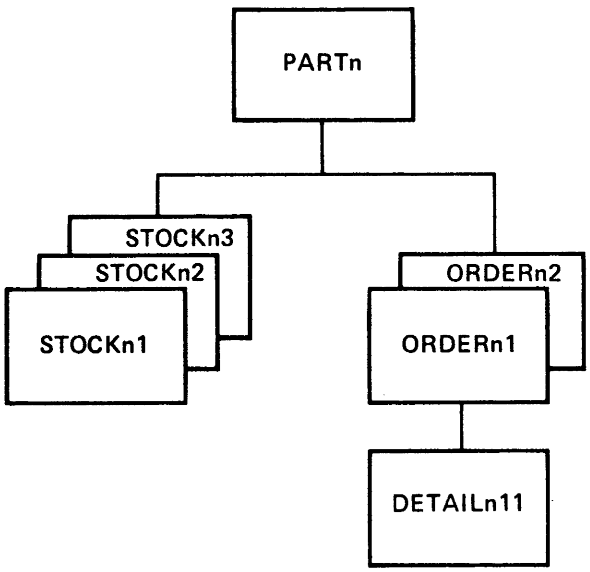


Figure 2-1. A DL/I Data Base Record

In its most elementary form, this record could be stored in one or more physical records. In principal, the segments would be stored in their hierarchical sequence, as shown in Figure 2-2.

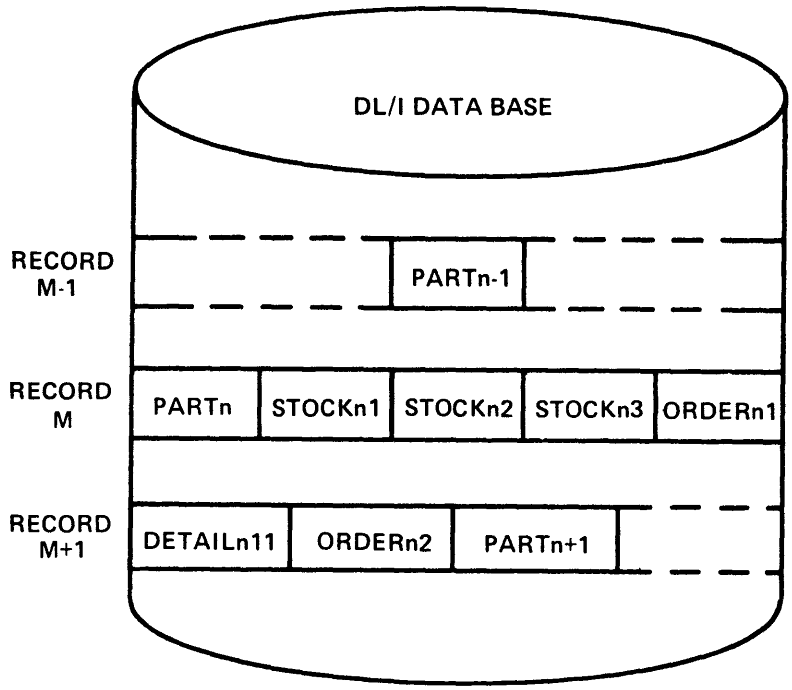


Figure 2-2. A DL/I Data Base Record in Physical Storage

It should be noted that the above figure is a simplification. In reality DL/I uses more elaborate storage organizations to allow for efficient replacement, insertion, and deletion of segment occurrences. Generally available functions include, for example:

- Space re-use of deleted segments
- Chaining of segments to be added later in the right hierarchical sequence
- Direct or key-sequenced access for the root segment based on the root segment sequence field (=key field).

This will be discussed in more detail for each of the data base organization methods.

Segment Format

A segment in a DL/I data base record consists of a prefix and a data portion. The prefix contains the system data used by DL/I and is not presented to application programs. The data portion contains the user data as seen by the application program. The prefix of a segment contains a segment code, a delete byte, and optional pointers.

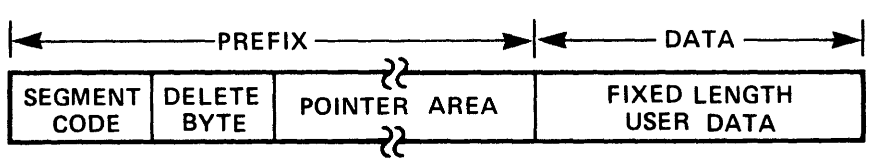


Figure 2-3. Segment Format

The one-byte *segment code* is used to identify the segment. It is the first byte of the prefix. The second byte is the *delete byte*. It is used to maintain the status of a segment within the data base.

Note: SHISAM and GSAM data bases can contain only one segment type. These data base organizations do not contain segment prefixes.

Pointers are used in HDAM and HIDAM data bases for linking the segments within one data base record in their hierarchical order. Pointers are also used to link segments involved in logical relationships, and to implement index pointing. The segment types in each data base are coded in hierarchical sequence from 1, the root segment, up to 255, as shown in Figure 2-4.

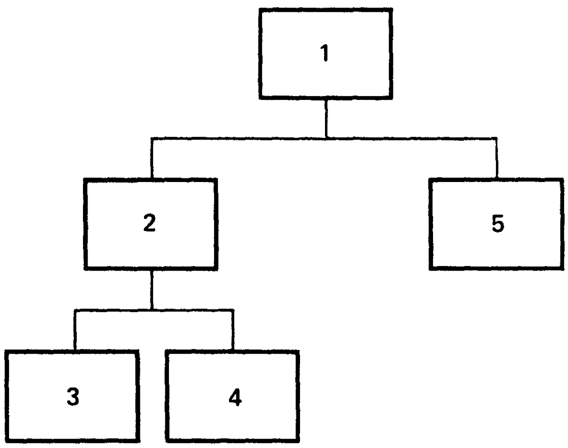


Figure 2-4. Segment Types Numbered in Hierarchical Sequence

Note that each occurrence in a data base of a given segment type contains the same segment code. Each segment occurrence is normally identified by its concatenated key.

The Concatenated Key

The *concatenated key* of a segment consists of all keys from the root down the hierarchical path to and including the key of the segment itself, as shown in Figure 2-5.

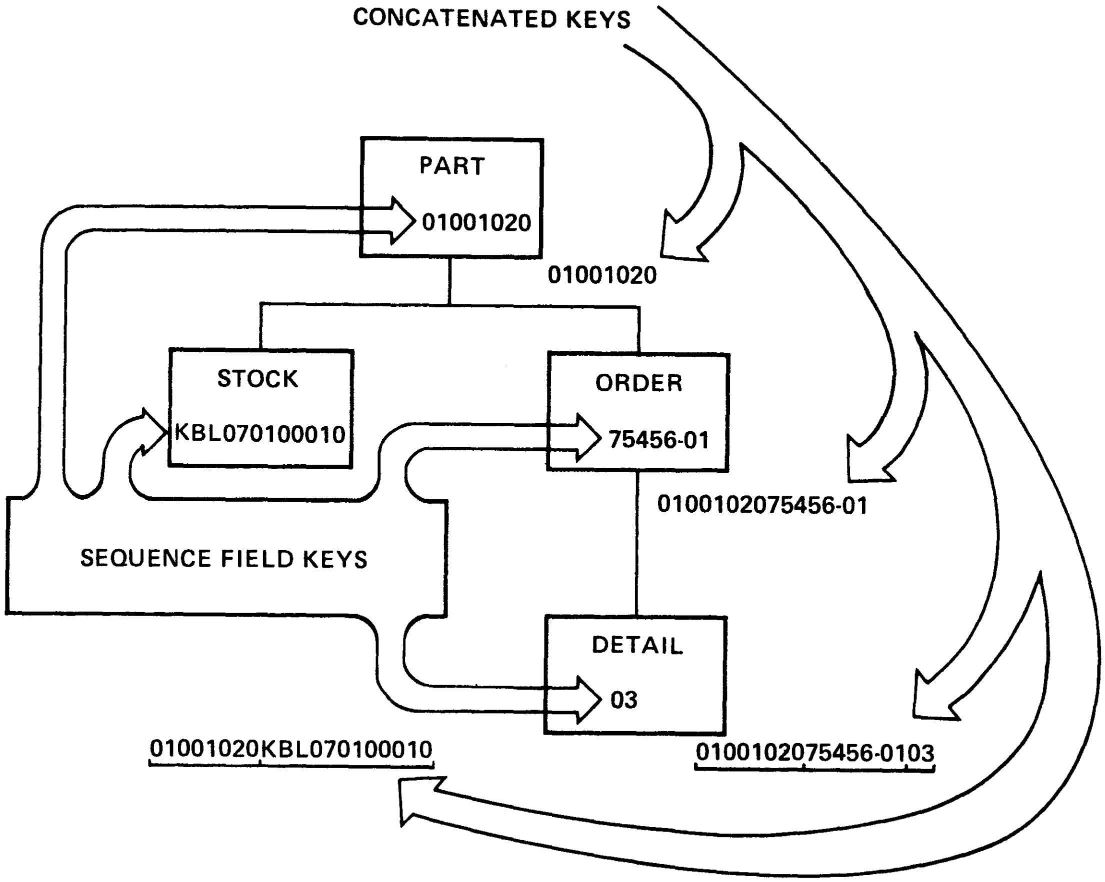


Figure 2-5. Concatenated Keys

A unique concatenated key is not required for every segment. However, a unique key is required for the root segment, except for HDAM.

Calls and Data Base Positioning

For a better understanding of each particular data base organization, we include now a basic description of the DL/I calls used to process segments in a data base.

The segments in a DL/I 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. A call references a parameter list which includes all data required by DL/I to complete the call. Included in the list are a function code and, optionally, one or more SSAs (segment search arguments). The function code states the call to be performed, and the SSAs define the segments along the hierarchial path down to, and including, the segment to be processed. A call is _unqualified_ when no SSA is included with the call, and is _qualified_ when one or more SSAs are included. 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 Chapter 4, "Data Base Processing."

The basic direction of movement in a DL/I data base is "top to bottom, left to right." Position in a data base is the segment or segments from which the search for another segment starts. Normally DL/I retains position at each level of the hierarchical path down to the last retrieved segment.

Get Unique: The GU (get unique) call is used to retrieve a specific segment or path of segments from a data base. At the same time it 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 the 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.

Hold Form of Get Calls: A GHU (get hold unique), or GHN (get hold next), indicates the intent of the user to issue a subsequent delete or replace call. A get hold call must be issued to retrieve the segment 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 data bases, and to add segments in existing data bases.

To control where occurrences of a segment type are inserted into a data base, the user normally defines a unique sequence field in each segment. When a unique sequence field is defined in a root segment type, the sequence field of each occurrence of the root 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. If no sequence field is defined, a new occurrence is inserted after the last existing one.

Delete: The DLET (delete) call is used to delete a segment from a data base. When a segment is deleted from a DL/I data base, its dependents, if any, are also deleted.

Replace: The REPL (replace) call is used to replace the data in the data portion of a segment or path of segments in a data base. Sequence fields cannot be changed with a replace call.

SSA (Segment Search Argument): An SSA specifies the conditions which a segment must meet to satisfy the call. An SSA can contain three parts. As a minimum, it contains the name of the segment type. Optionally, an SSA can also contain command codes and/or qualification statements. Commands codes, when used, specify a functional variation of a call, such as: retrieve last occurrence of the segment under its parent. Qualification statements identify, through field values, the segment occurrence of the specified segment type. A qualification statement contains a field name, relational operator, and comparative value. When occurrences of the segment type are searched by DL/I, the specified field is compared to the comparative value as the relational operator specifies. If only the name of the segment type is specified, the first encountered occurrence of that type will satisfy the call.

OS/VS Access Methods Used by DL/I

For each data base organization, DL/I uses one or more OS/VS access methods for the actual storage and retrieval of the data base records. Commonly used access methods are:

- The key sequenced data set (KSDS) and entry sequenced data set (ESDS) of the virtual storage access method (VSAM) of OS/VS.

- Overflow sequential access method (OSAM). This is a special physical access method supplied with DL/I. As far as OS/VS is concerned, an OSAM data set is described as a physical sequential data set (DSORG=PS).

HDAM AND HIDAM STORAGE ORGANIZATIONS

Both of these data base organizations are implemented with the hierarchical direct method of segment storage. In the hierarchical direct method, the segment occurrences in a hierarchy are connected in storage via four byte direct address pointers in the segment prefixes. A description of the types of pointers used in HDAM and HIDAM data bases can be found at the end of this section.

HDAM and HIDAM Access Characteristics

Two of the primary advantages of HDAM and HIDAM data bases are space reuse and the ability to directly access segments within the data base.

The segment storage organization used for HDAM and HIDAM data bases is essentially the same. The primary difference, at the access method level, between HDAM and HIDAM data bases is that access to occurrences 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 DL/I. To access the same root in a HIDAM data base, an index must be searched by DL/I 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. On the other hand, sequential processing of data base records is not necessarily in root key sequence, with HDAM.

__HDAM__: Refer to Figure 2-6 for the following discussion. An HDAM data base consists basically of one ESDS or OSAM data set. To access the data in an HDAM data base, DL/I uses a randomizing module. The randomizing module is used by DL/I to compute the address for the root segment in the data base. This address consists of the __control interval__ (CI), if VSAM, or __block__, if OSAM, number, and an __anchor point__ number. Anchor point(s) are located at the beginning of the CI/blocks. They are used for the chaining of root segments which randomize to that CI/block. A general randomizing module is supplied with the system. See the section "HDAM Randomizing Modules" in Chapter 7, which also contains guidelines to help you write your own randomizing module if required.

The ESDS or OSAM data set is divided into two areas:

- The root addressable area. This is the first n control intervals/blocks in the data set. You define n in your DBD.

- The overflow area is the remaining portion of the data set.

The root addressable area is used as the primary storage area for segments in each data base record. The overflow area is used for overflow storage. Since data base records vary in length, a parameter (in the DBD) is used to control the amount of space used for each data base record in the root addressable area. This parameter, "bytes" in the RMNAME= keyword, 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 specified number of bytes.

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 overflow area. It should be noted that the "bytes" value only controls segments consecutively inserted in one data base record. Consecutive inserts are inserts to one data base record without an intervening call to process a segment in a different data base record.

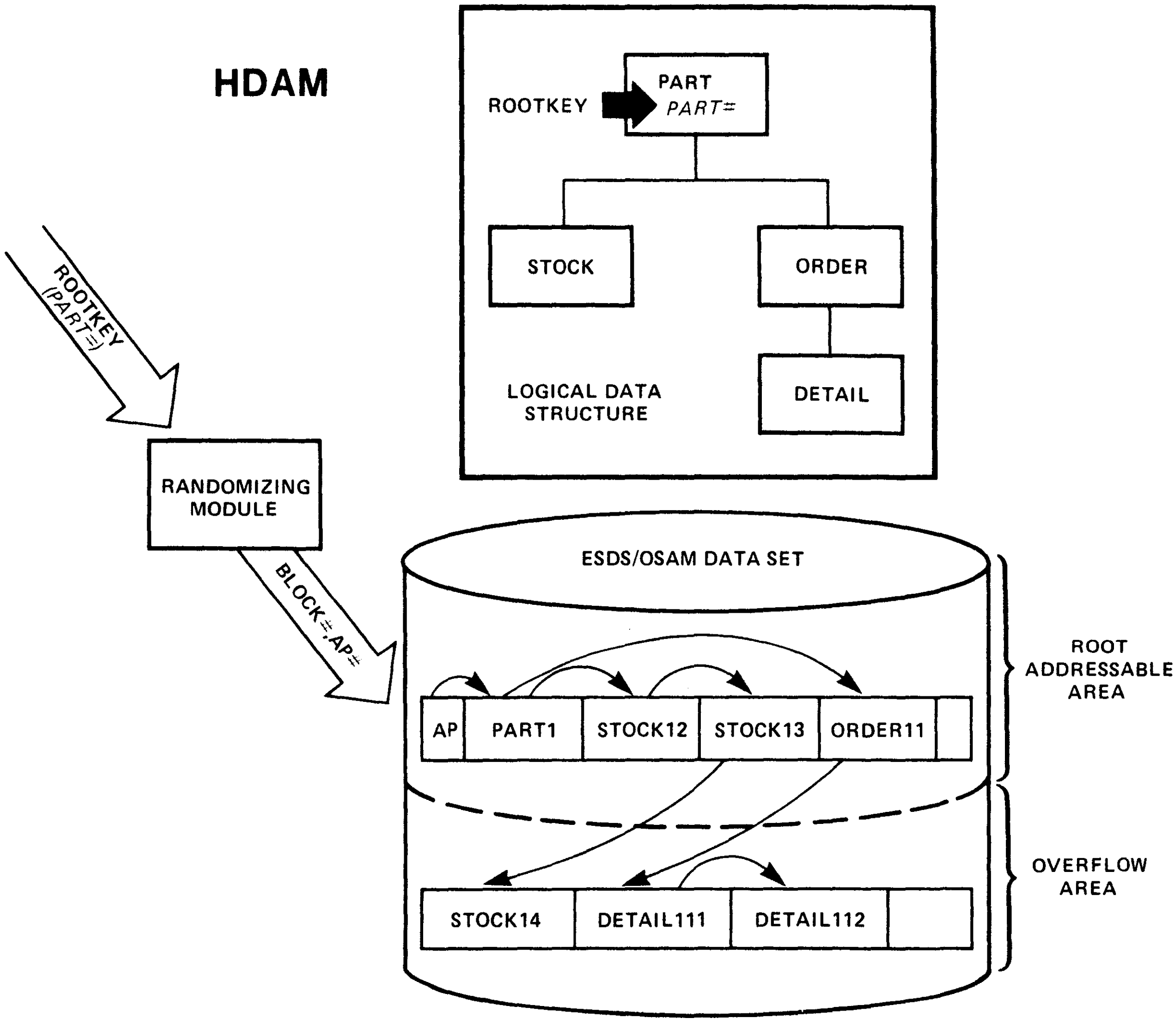


Figure 2-6. HDAM Data Base in Physical Storage

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 HIDAM primary index data base and the other is defined as the main HIDAM data base. In the following discussion the term "HIDAM data base" refers to the main HIDAM data base defined through DBDGEN.

The HIDAM primary index data base is used to locate 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 for the root segment type. The value of this sequence field is used by DL/I to create an index segment for each root segment. This index segment in the HIDAM primary index data base contains, in its prefix, a pointer to the root segment in the main HIDAM data base.

The HIDAM primary index data base consists of a KSDS; its only data (and key) is the sequence field of the root segment. In our subset, the main HIDAM data base consists of one ESDS. The segment storage organization in this ESDS is comparable to the one in the HDAM ESDS. Figure 2-7 shows the layout of the HIDAM data base.

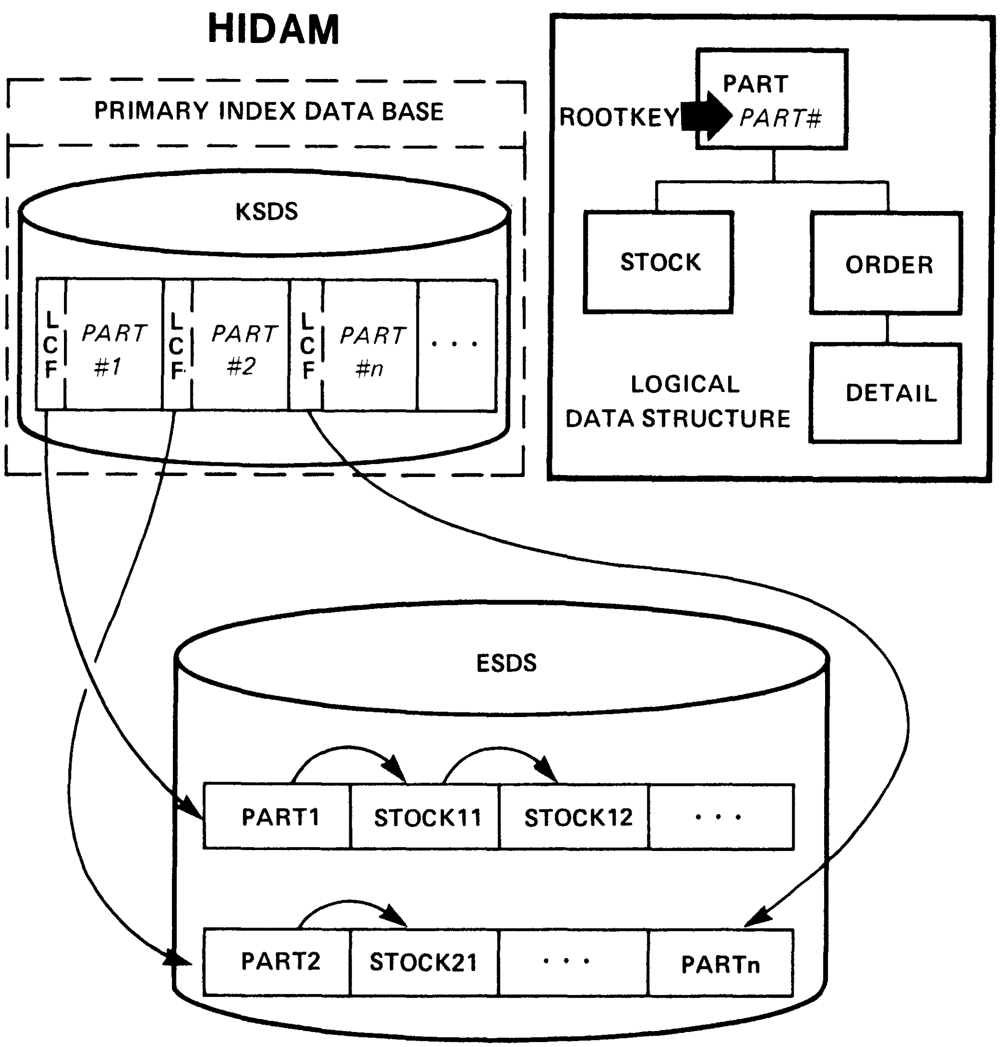


Figure 2-7. HIDAM Data Base in Physical Storage

Inserts and Deletes in HDAM and HIDAM

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. These system fields are used by DL/I to find space when inserting a segment, or to record free space when a segment is deleted. Normally, the space a segment occupies is immediately freed after the deletion of the segment. You only need to be aware of these system-maintained fields when doing CI/blocksize calculations because they are allocated within your selected CI/blocksize. We will cover this when providing guidelines for such calculations later in this chapter.

Also, with HIDAM, you can specify free space at data base load time (initial load or reload during reorganization). This is specified in the DBD for the ESDS. For the primary index KSDS, free space can be assigned with the VSAM access method services DEFINE command. In theory, you can also specify free space in the DBD for an HDAM data base. This is, however, not recommended because it might conflict with the randomizing module algorithm.

Pointers in HDAM and HIDAM

To link each segment in an HDAM or HIDAM data base to its related segment, direct address pointers are used. The pointers are four bytes long, and are placed by DL/I 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.

Note: The following discussion of pointers is included for those of you who are interested in the internal DL/I storage organization. A complete comprehension is not required for basic data base design, because we will give detail guidelines for the necessary pointer selection in the implementation part.

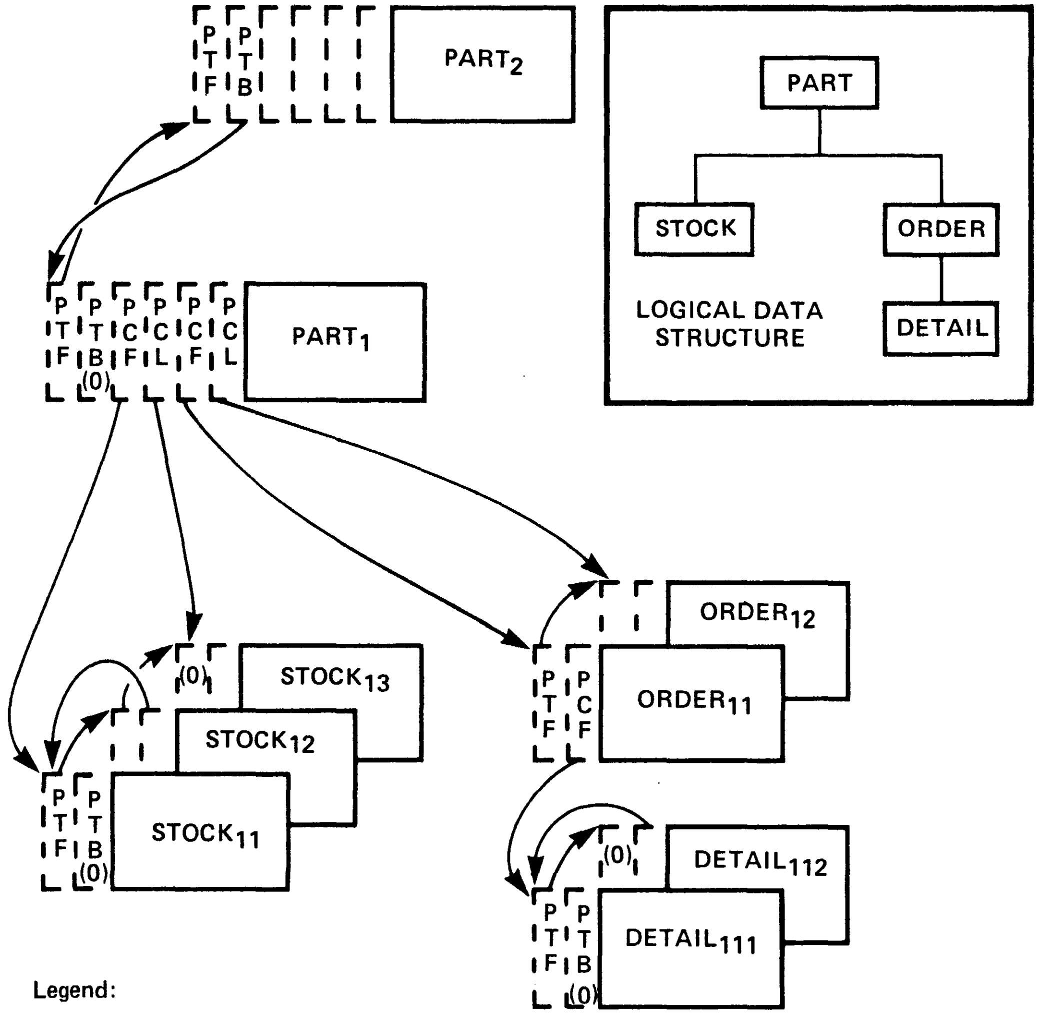
The most common method is a combination of physical child/physical twin pointing. Figure 2-8 should be referred to when reading the following description of pointers.

Physical Child/Physical Twin Pointers: Every parent segment in the data base has a pointer to the first occurrence of each of its child segment types. This is the physical child (first) pointer. Optionally, per child segment type, there is also a pointer to the last occurrence of that child segment type, the physical child last pointer. This physical child last pointer will improve segment insert performance of that child if that segment has no sequence field defined. It also improves the performance of a get call which, via a command code, explicitly requests the last segment occurrence.

Usually, every segment in a HIDAM or HDAM data base has a pointer in its prefix which points to the next (based on sequence field) occurrence of this segment under the same parent. (If it is the last occurrence under the parent, this pointer is zero.) This pointer is named the physical twin (forward) pointer. If it is the root segment, the physical twin pointer points to the next root if HIDAM. In HDAM, the physical twin pointer is used to chain the root segment(s) of the anchor point. If there is never more than one occurrence of a segment for a given parent, then you should omit this pointer.

Optionally, you can also select a pointer in each segment prefix which points to the previous segment occurrence under the same parent. This is the physical twin backward pointer. This pointer will improve delete performance if the segment to be deleted is a logical child or is located via the physical child last pointer (that is, command code last).

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. In our subset, we will always select physical twin forward and backward pointers for the root of a HIDAM data base.



Legend:

PTF: Physical twin forward pointer
PTB: Physical twin backward pointer
PCF: Physical child first pointer
PCL: Physical child last pointer

Note that PTB and PCL are optional.

Figure 2-8. Direct Address Pointers in HDAM and HIDAM

SHISAM STORAGE ORGANIZATION

The data structure of a SHISAM data base consists of only one segment type, the root segment, with a unique sequence field. Because of this, there is no segment prefix needed. The physical storage organization is a single VSAM KSDS (Key Sequenced Data Set). This makes it possible to process a non DL/I KSDS as a DL/I data base with full DL/I function. The main use of the SHISAM organization is as a migration tool to DL/I for existing KSDS or ISAM files. It is not recommended for new data bases. (See also the phase 2 sample environment earlier in this chapter.)

<u>Note</u>: The logical record length of the KSDS must be an even number for SHISAM.

FUNCTIONS AND USE OF GSAM

An OS/VS sequential file can be defined to DL/I as a GSAM data base. However, the normal concepts of hierarchical structures do not apply to GSAM.

When using GSAM for sequential input and output files, DL/I will control the physical access and position of those files. This is necessary for the repositioning of such files in case of program restart. When using GSAM, DL/I will, at restart time, reposition the GSAM files in synchronization with the data base contents and your application program's working storage. To control this, the application program should use the restart (XRST) and checkpoint (CHKP) calls. These calls will be discussed in Chapter 4, "Data Base Processing."

When to Use GSAM

Whenever you want your program to be restartable, you should use GSAM for its sequential input and output files. There are two reasons why you should want to do this. The first is to save time if a program rerun is required in case of program or system failure. This is normally only done for long-running update programs (one or more hours). The other reason stems from a planned online usage of the data bases.

To be able to run a batch program in parallel with the online system, using the same data bases, that program must be executed as a batch message processing (BMP) program. A BMP runs as a batch job, but uses the online control region of IMS/VS for the access of DL/I data bases. In that way, IMS/VS will provide complete data integrity across the batch and online use of the data. To do so, however, the IMS/VS data base/data communication system will isolate the data base updates of a particular program until program termination. By using the checkpoint call, the batch program can free those updated data base segments for immediate access by other batch and/or online programs.

Supported Data Sets

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

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

GSAM supports the Basic Sequential Access Method (BSAM), on DASD, unit record, and tape devices and ESDS on DASD devices. In our subset, we will only consider BSAM fixed and variable length record formats.

The terms segment, segment type, hierarchical, parent, child, etc., are not applicable to GSAM data sets, nor do the concepts of either key or field apply.

When program restart is required, you should not use temporary files, that is, for SYSIN/SYSOUT spooling. They may be deleted by OS/VS after program or system failure.

A GSAM data base may also be a data set previously created by use of OS/VS BSAM, or QSAM. Conversely, a GSAM data base may be accessed later by other programs using these OS/VS access methods.

DL/I LOGICAL RELATIONSHIPS

WHY LOGICAL RELATIONSHIPS

We have so far addressed only single hierarchical data structures. Quite often, especially with different applications, several DL/I data bases are needed. In addition, there is often a requirement to access the same data in different hierarchical structures and different data bases. This can create problems of:

- Consistency -- if data is stored more than once, how to update all occurrences at the same time.
- Data Redundancy -- if large data elements are stored many times, this may consume excessive external storage.
- Access of Data -- if data is stored more than once, which access path should be used to access the appropriate copy of the data.

The above problems can be solved by storing the data only once and providing a linkage mechanism between hierarchical structures. With this linkage a new access path is provided to data in data base A, based on data in data base B, and, if desired, vice versa.

DL/I's logical relationships provide this function. The basic linkage is always between two segments. However, the linkage can extend to several data bases. On the other hand, the resulting compound data structure will always be presented as a hierarchical structure to a particular application. The basic mechanism of the DL/I logical relationship is the connection of a segment to two parents in two different hierarchical structures. Normally, any segment has only one parent. By giving a segment two parents, that segment (and its dependents) belong to two different hierarchical structures. This enables the definition of a new hierarchical structure which contains segments from both related structures. Such a definition is called a logical data base.

BUILDING LOGICAL RELATIONSHIPS

Segment Types Involved in Logical Relationships

The following segment types are needed to establish a logical relationship. All three must be present for any logical relationship. You should refer to Figure 2-9 when reading the following discussion.

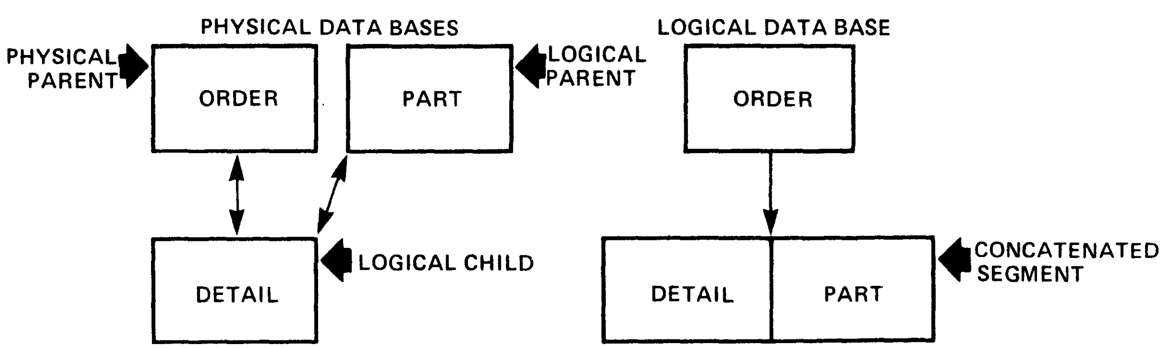


Figure 2-9. Segment Types Involved in Logical Relationships

Logical Child Segment: This segment has two parents. A logical parent and a physical parent. The logical child segment and its dependents, if any, are accessable via both parents. The access path via its physical parent is called physical access path. The access path via its logical parent is called the logical access path. By definition, a logical child segment contains the concatenated key of the logical parent followed by user data, if any. The remainder of the user data in the logical child is called intersection data. It is present at the intersection of the two parents. The logical parent concatenated key (LPCK) is always presented together with the intersection data, whenever the logical child is accessed via its physical path (see Figure 2-10).

```
┌- - - - - - - -------------------------------------------------┐
|             |              |                                  |
|    PREFIX   |     LPCK     |        INTERSECTION DATA         |
|             |              |                                  |
└- - - - - - - -------------------------------------------------┘
              |                                                 |
              |<---------TO/FROM USER'S I/O AREA--------------->|
```

Figure 2-10. Logical Child Segment Format

Whenever you insert a logical child segment in its physical data base, you must present the LPCK. It identifies the logical parent.

Logical Parent Segment: This segment may reside in the same or a different data base as the logical child.

Physical Parent Segment: This is the normal parent segment of the logical child in its physical data base as defined earlier.

The most common method for implementing logical relationsips between HDAM and HIDAM data bases is based on direct address pointers, which are all 4-byte relative byte address pointers similar to other pointers in HDAM and HIDAM.

The Virtual Logical Child Segment (VLC): To be able to define the view of the logical parent on its logical children and their occurrence sequencing, DL/I introduces a special segment type. It is named the virtual logical child and is defined as a dependent of the logical parent segment. It does not exist in physical storage itself. Its only role is to provide a mechanism to define the logical parent's view of the data in the logical child. It controls the access from the logical parent to the logical child. It is used to define the sequencing of the logical child segment when that logical child segment is accessed via its logical parent. The virtual logical child is said to be paired with the real logical child. See Figure 2-11.

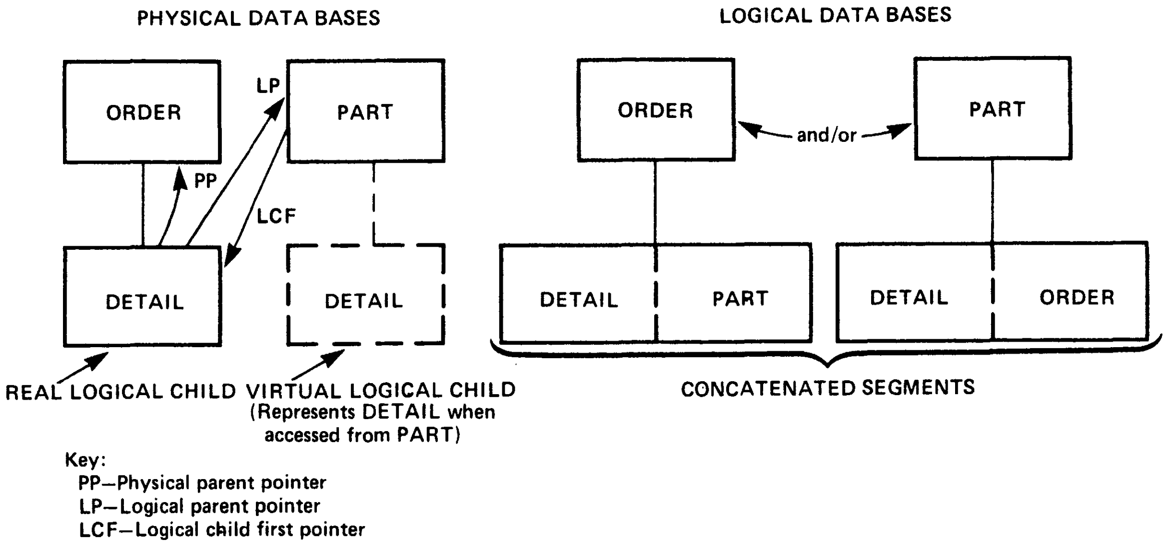


Figure 2-11. Virtual Paired Bidirectional Logical Relationship

When accessed, the virtual logical child contains the concatenated key of the physical parent of the real logical child, plus the intersection data of the real, logical child. So the virtual logical child DETAIL' in Figure 2-11 contains the key of the ORDER segment plus the user data of the real DETAIL segment.

__The Destination Parent__: With bidirectional pairing, DL/I refers to the parent which is other than the one used to access the logical child as the __destination parent__. As a consequence, the logical child always starts with the __destination parent concatenated key__ (DPCK).

__Logical and Physical Data Bases__

The __physical data bases__ used to implement a logical relationship must be HDAM or HIDAM data bases. Figure 2-12 shows the physical data bases of our Phase 2 sample environment. The order line segment in the Customer Orders data base is the logical child of the part segment in the Parts data base. Notice that the virtual logical child is not shown, although it will appear in the DBD as discussed later.

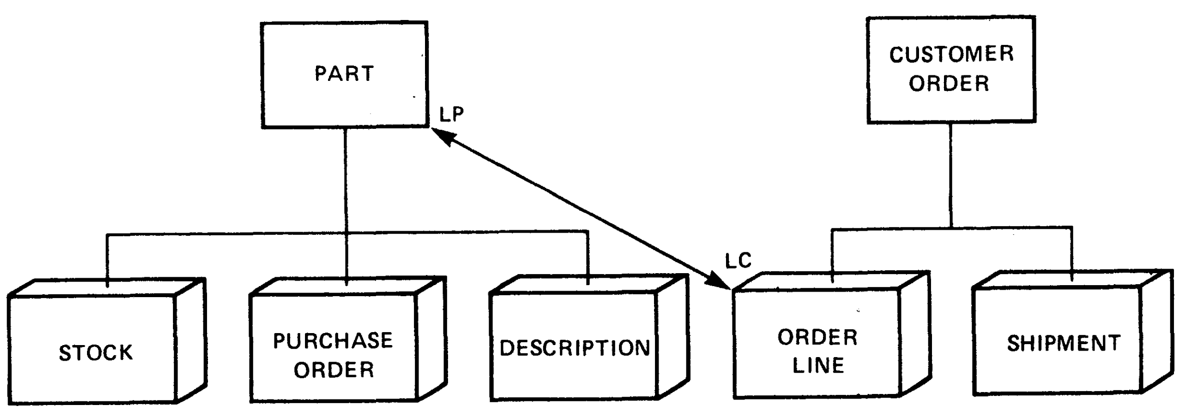


Figure 2-12. The Phase 2 Physical Data Bases

A discussion on how this structure is derived can be found in the last part of this chapter. A logical data base is a redefinition of one or more physical data bases which contain logical relationships. It yields a new hierarchical structure which is composed of structures from both related structures. The new structure can be processed by application programs as if it were physically present. The logical data base can only be defined if the proper logical relationships are defined in the physical data bases.

The Concatenated Segment: All segments in the logical data base stem from one segment in one of the physical data bases, except when the logical child is accessed. Whenever the logical child is accessed in a logical data base, it is optionally concatenated with the destination parent segment. See Figure 2-13. The destination parent is the parent of the LCHILD other than the one from which you came.

LOGICAL CHILD		DESTINATION PARENT
DPCK	INTERSECTION DATA	

Figure 2-13. Concatenated Segment Format

Notice that the concatenated segment is different for the two paths:

- When accessing the real logical child below its physical parent, the concatenated segment will consist of:
    1. The real logical child, which consists of:
        a. The concatenated key of the logical parent
        b. The data of the real logical child segment, if any
    2. Optionally, the logical parent segment itself.
- When accessing the virtual logical child below the logical parent of the real logical child, the concatenated segment will consist of:
    1. The virtual logical child, which consists of:
        a. The concatenated key of the physical parent
        b. The data of the real logical child segment, if any
    2. Optionally, the physical parent segment itself.

Note: The concatenated segment only exists in a logical data base.

Because of the bidirectional virtual pairing, you can always define two logical data bases with one logical relationship.

Figure 2-14 shows the two logical data bases which can be defined using the related physical data bases of Figure 2-12.

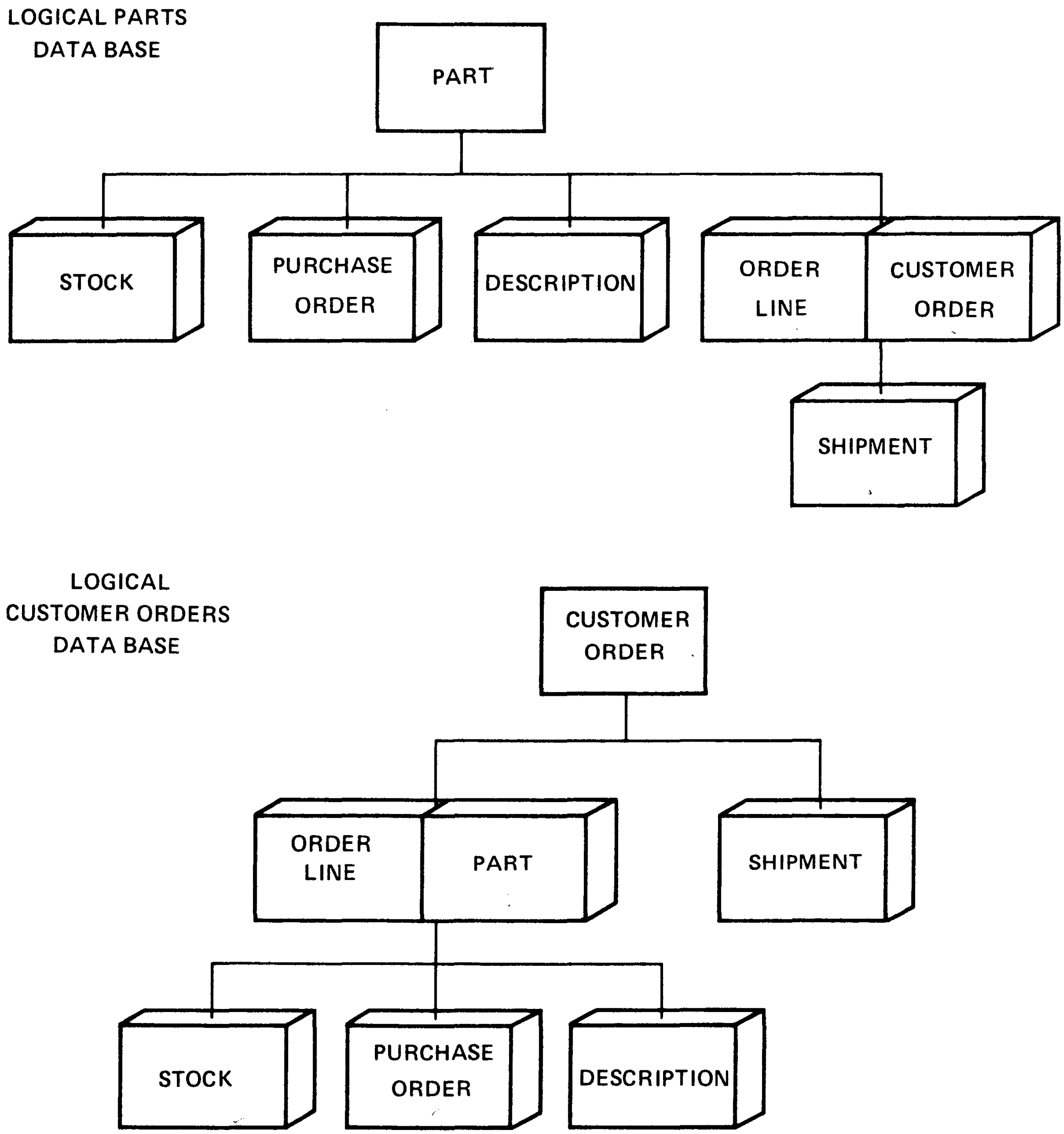


Figure 2-14. Phase 2 Logical Data Bases

The above logical data bases will be used by our sample Phase 2 application programs.

The exact rules for defining and processing logical data bases will be discussed in the following section.

LOGICAL RELATIONSHIP DESIGN RULES

In constructing logical relationships with DL/I, two sets of rules must be observed. One set for constructing the physical data bases and the second set for constructing logical data bases. It should be clear that a logical data base can be defined only if the underlying physical data bases are properly defined.

If necessary, multiple logical data bases can be defined for a given set of logically related physical data bases. However, good practice is to generate one logical data base for each physical root segment which contains only the segments needed in your applications.

Rules for Defining Logical Relationships in Physical Data Bases

Logical Child:

1. A logical child segment must have one and only one physical parent segment and one and only one logical parent segment.

2. A logical child segment is defined as a physical child segment in the physical data base of its physical parent.

3. In its physical data base, a logical child segment cannot have another logical child as its immediate dependent.

Logical Parent:

1. A logical parent segment can be defined at any level of a physical data base including the root level.

2. A logical parent segment can have one or multiple logical child segment types.

3. A segment in a physical data base cannot be defined as both a logical parent and a logical child.

4. A logical parent segment can be defined in the same or a different physical data base as its logical child segment.

Physical Parent:

1. A physical parent segment of a logical child cannot also be a logical child. This is the same as rule 3 for the logical child.

Multiple logical relationships can be established within a single data base or between two or more data bases, as long as the above rules are obeyed.

Rules for Defining Logical Data Bases

1. The logical data base itself is always a single hierarchical structure.

2. It must start with the root of a physical data base and can contain only segments defined in physical data bases.

3. In following a hierarchical path, no segments may be skipped.

4. The logical child plus the destination parent is always presented as one concatenated segment.

5. The dependents of a concatenated segment are:

    - The dependents of the logical child

    - The logical or physical dependents of the destination parent

    The above dependents should not be intermixed, nor should their relative order be changed. But you can start with either of them.

    - The physical parents up to the root of the destination parent in destination parent to root order

6. If physical parents of a destination parent are included, then you can also include their logical or physical dependents in their normal order.

7. Any number of logical relationships can be used in a single hierarchical path in the logical data base up to the maximum of 15 segment levels.

__Notes:__

1. Because of the virtual logical child concept, paths are bidirectional and can be intermixed and/or repeated in a single logical data base.

2. All segments of related data bases are available as long as you follow the above rules. The same physical segment type could appear in several different paths if needed.

Figure 2-15 shows some examples of logically related physical data bases and their associated logical data bases. It illustrates most of the above rules. This example is not representative for a typical DL/I application; it merely shows the different possible combinations.

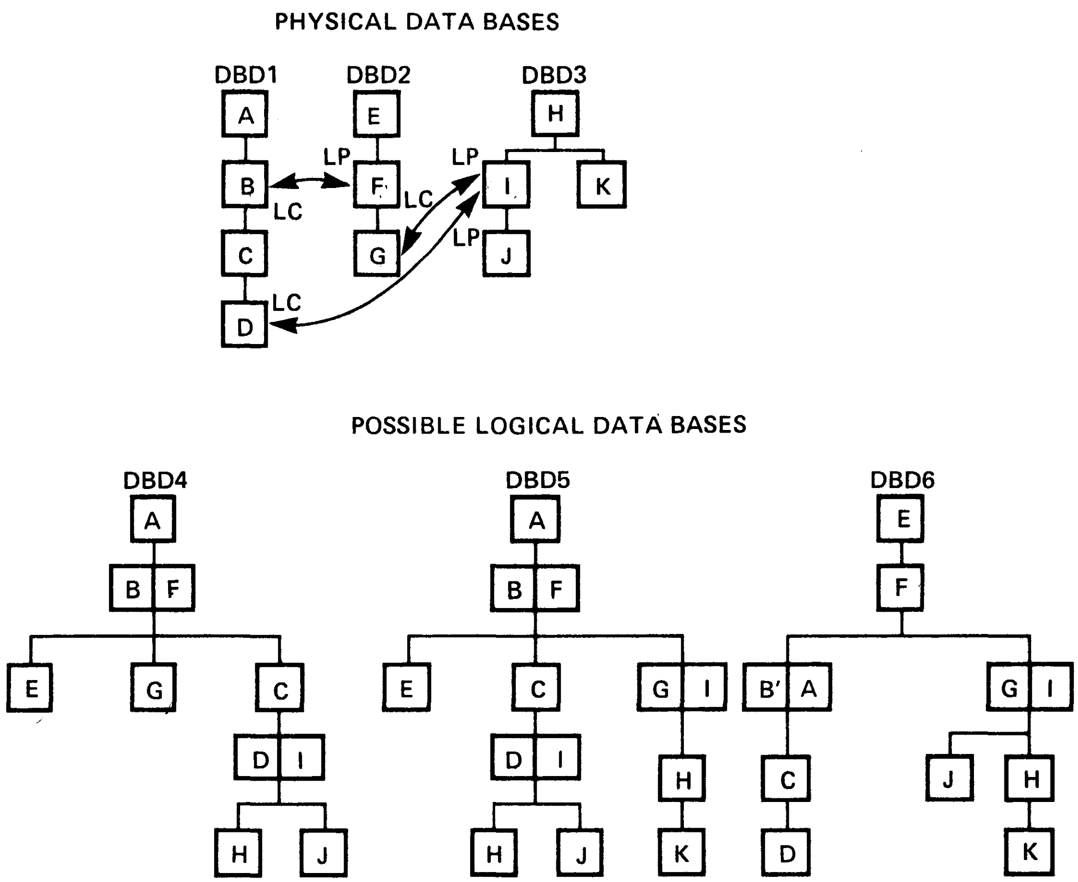


Figure 2-15. Using Multiple Logical Relationships

PROCESSING LOGICALLY RELATED SEGMENTS

Deleting Logically Related Segments

Logical Child: The logical child can be deleted via its physical parent path or its logical parent path. If a logical child is deleted in either way, then all its dependents in the physical data base are deleted. If a concatenated segment is deleted in a logical data base, then only the logical child segment is deleted with its physical children. The destination parent is not deleted. In our subset, the logical child will be automatically deleted if either its physical or logical parent is deleted.

Logical Parent: The logical parent can only be deleted via its physical parent path. If the logical parent is deleted then all its children will be deleted including logical children.

Physical Parent: The physical parent can only be deleted via its physical parent path. If the physical parent is deleted, then all its children are deleted including logical children.

Inserting Logically Related Segments

Logical/Physical Parent: Either parent type can only be inserted via its physical parent path.

Logical Child: The logical child can be inserted via either path, but the destination parent must already exist.

Replacing Logically Related Segments

After a get hold call of the concatenated segment, fields in both the logical child and the destination parent can be changed before the replace call, except sequence fields, see Figure 2-16.

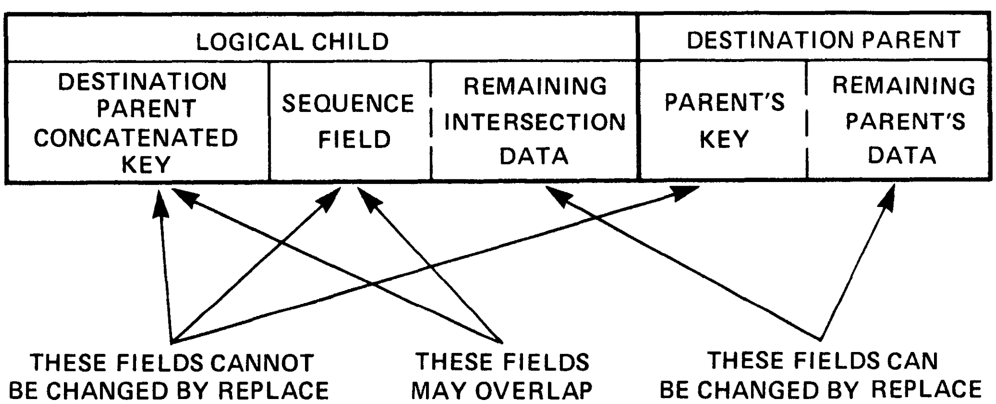


Figure 2-16. Replacing Fields in a Concatenated Segment

LOGICAL RELATIONSHIPS IMPLEMENTATION TECHNIQUE

The following pointers are used by DL/I, in our subset, to implement logical relationships. These pointers are maintained in the segment prefix in the same way as the previously discussed physical child and physical twin pointers. Again, a detailed comprehension of those pointers is not required at the moment, as we will give detailed guidelines for their selection in the implementation part of this chapter.

Pointers Used for Logical Relationships in HDAM/HIDAM

Logical Parent Pointer (LP): The logical parent pointer is within the prefix of the logical child segment and points to the logical parent occurrence of that logical child. This pointer is always present and is never zero. Each logical child must have one and only one logical parent just as it has only one physical parent.

Logical Child First Pointer (LCF): The logical child first pointer is within the prefix of the logical parent and points to the first occurrence of its logical child segment. If a segment has several logical segment types, it contains one LCF pointer for each segment type. If a logical parent has no logical child occurrences, the corresponding LCF pointer is zero. The logical child first pointer is required.

Logical Child Last Pointer (LCL): The logical child last pointer is within the prefix of the logical parent and points to the last occurrence of its logical child. There is one LCL for each defined logical child segment type. The LCL pointer is optional. Its only use is to improve the performance of the logical child insert if no sequence field is defined for the logical chain. See "Role of the Virtual Logical Child" earlier in this chapter.

Logical Twin Forward Pointer (LTF): The logical twin forward pointer is within the prefix of the logical child segment and links all logical child occurrences of a particular logical parent. This pointer is required if any logical parent occurrence has more than one logical child occurrence.

Logical Twin Backward Pointer (LTB): The logical twin backward pointer links logical twins but in the reverse order of the LTF. This pointer serves a complementary performance role as the physical twin backward pointer in deleting logical children. It should always be used -- together with the LCL -- if there are multiple occurrences of a logical child for any logical parent occurrence.

Physical Parent Pointer (PP): DL/I uses a physical parent pointer in the prefix of the logical child to locate that physical parent if the access was via the logical parent. This PP pointer is repeated up through the hierarchy to the root. A physical parent pointer is also present in the logical parent if this is not a root segment. It then points to the physical parent of the logical parent, etc. You never need to specify the inclusion of this pointer in the DBD. DL/I will include it automatically if needed.

DL/I SECONDARY INDEXES

The secondary indexing capability of DL/I allows additional access paths to a data base record. Secondary indexes provide:

- A secondary processing sequence, enabling direct and/or sequential processing of data base records on non-root-key field values. These search fields can be located in the root segment or a dependent segment.

- Automatic updating of the secondary index is always done, even if the program causing the change is not sensitive to the secondary index.

WHEN TO USE SECONDARY INDEXES

Secondary indexes should be mainly used when frequent, direct access to the data base record is required on non-root-key fields. It should be realized that a secondary index incurs additional system cost in CPU and I/O time. If the information on which the secondary index is established is changed, then DL/I has to change the index entry.

Especially for batch processing, you should compare the costs of full or partial data base scan plus a subsequent sort of the output versus the cost of using secondary indexes. For online data base processing, the choice is easier. Terminal user's response requirements normally do not allow for full data base scans.

SEGMENT TYPES INVOLVED IN SECONDARY INDEXES

The segment types and associated terms involved in secondary indexes are (see Figure 2-17):

- 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 data base.

- Index Pointer Segment

  A segment defined in a secondary index data base that contains the data and pointers used to index the "index target segment." It controls the secondary processing sequence.

- Index Target Segment

  The segment that is pointed to by an index pointer segment. In our subset, it will always be a root segment. In that case, it is as if the search field "replaces" the original root segment sequence field.

- Index Source Segment Type

  A segment that is the source from which a secondary index is created.

- Secondary Processing Sequence

  The sequential order in which occurrences of an index target segment type are accessed through a secondary index. It is the order of the index pointer segment.

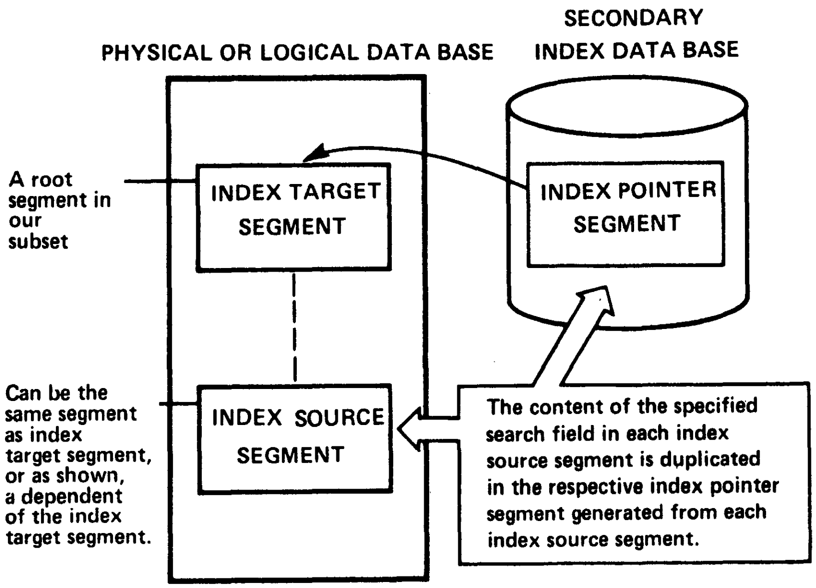


Figure 2-17. Segment Types Associated with a Secondary Index

Although a secondary index can be used in programs which use only logical data bases, their implementation is strictly on the physical data base level. Figure 2-18 shows the physical data bases of our phase 3 sample environment. The only difference from phase 2 is the Purchase Order Number secondary index data base. By utilizing this secondary index data base, an application program can process the physical and/or logical Parts data base directly by purchase order number.

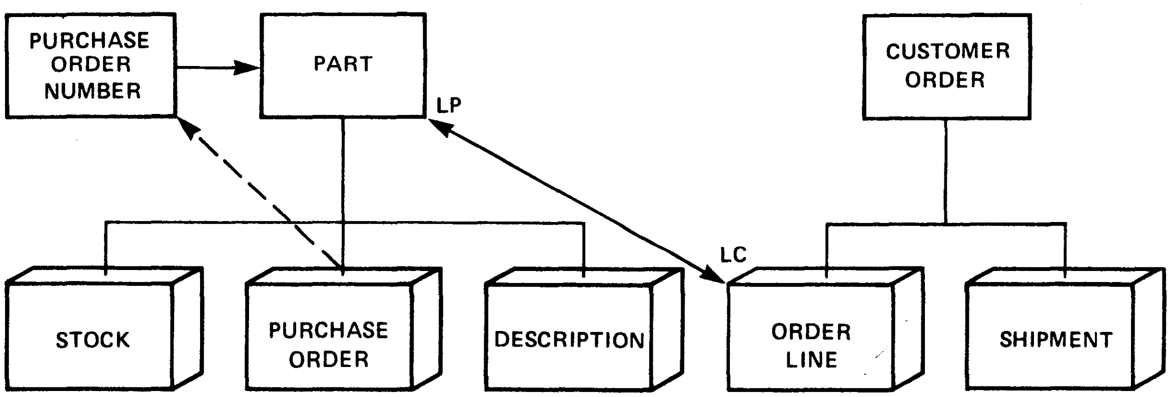


Figure 2-18. Phase 3 Physical Data Bases

DESIGN RULES FOR SECONDARY INDEXING

Several rules should be observed when designing basic secondary indexes:

1. The index target segment should be a root segment in our subset.

2. The index source segment and the index target segment must be defined in the same physical DBD. They can be the same segment.

3. A logical child segment cannot be used as an index source segment. However, a dependent of a logical child can be used as an index source segment.

4. A secondary index can be used with a logical DBD, but the index target segment should be the root segment. Nothing additional need be specified in the logical DBD.

IMPLEMENTATION TECHNIQUE

In discussing secondary indexes we have to distinguish between two different data base types. The first is the indexed data base. This data base contains the index source and index target segments. It is an HDAM or HIDAM data base. The second is the secondary index data base. This data base contains the index pointer segments which contain pointers in their prefix to the index target segments. An INDEX data base consists of a single KSDS. Figure 2-19 shows the physical format of the KSDS logical record for the INDEX data base.

SEGMENT = VSAM logical record			
PREFIX		DATA = KSDS KEY	
Delete flag	Direct address index target segment pointer	Search field	Subsequence field (Optional)
1	4	N	4

Figure 2-19. Logical Record Format for the Index Pointer Segment

Index Pointer Segment Format

The index pointer segment contains:

- Delete flag (1 byte) controls the delete status of the index pointer segment.

- Pointer to the index target segment (4 bytes).

- Search field (N bytes) contains a duplication of one to five index source segment fields which together define the secondary sequence.

- Subsequence field (4 bytes), optional. It is required in our subset if the search fields in the index pointer segments are non-unique. If specified, it contains the relative byte address of the index source segment. It is never used to access the index source segment. Its sole use is to provide a unique key for the KSDS logical record. In the DBDs, its field name must start with the three characters.

CREATING A SECONDARY INDEX

Secondary indexes are created with the standard DL/I data base reorganization utilities, see Chapter 5. They can be created at initial data base load time or later. No user programming is needed to create a secondary index. Also existing programs need not be changed unless they want to use the secondary index.

DATA BASE DESCRIPTION GENERATION

After you finish the design of your data bases you must specify them to DL/I. This section gives the guidelines for the use of the DL/I data base definition language: the data base description generation (DBDGEN). Again this section is divided into three subjects in concurrence with the three phases:

1. Basic DBDGEN for physical data bases

2. DBDGEN for logical relationships

3. DBDGEN for secondary indexes

For each data base to be used with DL/I, a data base description (DBD) must be generated. A DBD consists of a set of DL/I-supplied macro instructions, coded by you to specify the data base characteristics you need. The DBD is processed by an OS/VS assembler and the generated load module is stored by the linkage editor in the IMSVS.DBDLIB library for subsequent processing of the data base. See Figure 2-20.

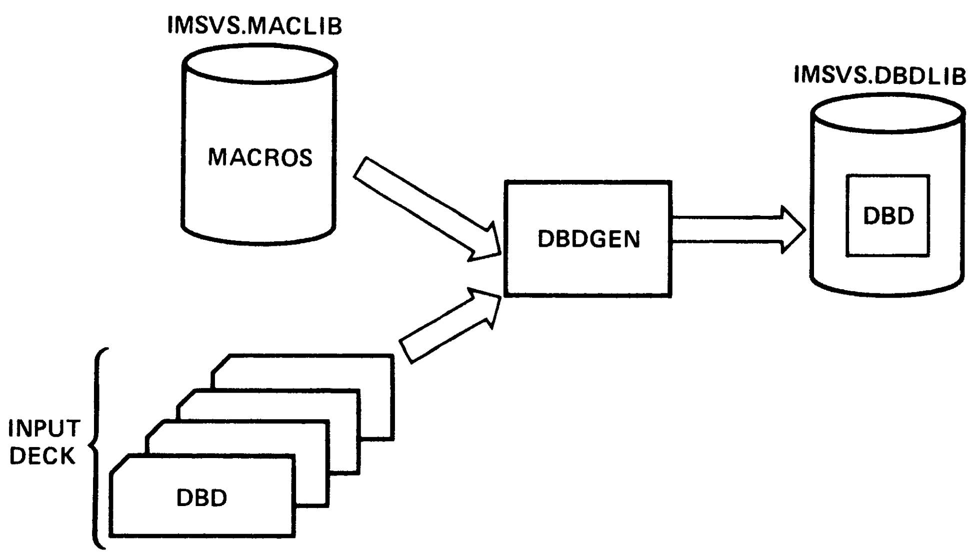


Figure 2-20. Data Base Description Generation (DBDGEN)

Figure 2-21 shows the sequence of the macro statements in the DBD input deck. The DBDGEN is executed by invoking a JCL cataloged procedure named DBDGEN, which is available in IMSVS.PROCLIB.

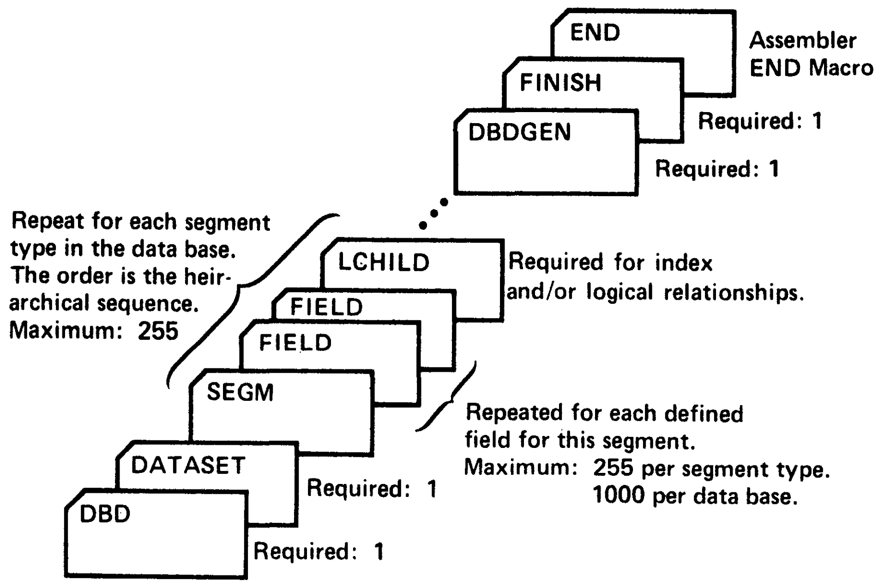


Figure 2-21. DBDGEN Input Deck Structure

DBDGEN CODING CONVENTIONS

DBDGEN statements are Assembler language macro instructions and therefore, are subject to the rules contained in the publication OS/VS-DOS/VS-VM/370 Assembler Language, GC33-4010.

In the generalized format shown in the following descriptions of the control statements, these syntax conventions apply:

a. Words written in all capital letters must appear exactly as written.

b. Words written in lowercase letters are to be replaced by a user-specified value. Valid user-specified values are numeric values or one- to eight-character alphameric names.

c. The control cards are free form. Operation codes must begin after column one. Operands must follow an operation code or prior operand. The first operand must be separated from the operation code by at least one blank column. Each operand should be separated from the previous operand by a comma. Operands may be continued on subsequent cards, but must start in card column sixteen on the continuation card. A nonblank character must be coded in column 72 if a continuation card follows.

   [ ] indicates optional operands. The operand enclosed in the brackets (for example, [VL]) may or may not be present, depending on whether or not the associated option is desired. If more than one item is enclosed in brackets one or none may be coded.

   { } indicates that a choice of an operand parameter must be made. One of the operand parameters from the vertical stack within braces must be coded.

   ,... indicates that more than one set of parameters may be designated in the same operand.

Example:

```
|<- Column 1        |<- Operands - Column 16
|          |<- Operation - Column 4            Column 72 ->|

   /--------------------------------------------------------------.
  /         |          |                                           |
 |          |DBD       |NAME=BE1PARTS,                        *    |
 |          |          |ACCESS=HIDAM                               |
 |          |          |                                           |
 '-----------------------------------------------------------------'
```

BASIC DBDGEN CONTROL STATEMENTS FORMATS

DBD Statement

This statement names the data base being described and specifies the organization used. There is only one in the input to DBDGEN. The format of the DBD macro instruction is:

```
   /--------------------------------------------------------------.
  /         |          |                                           |
 |          |DBD       |NAME=dbname1                               |
 |          |          |                                           |
 |          |          |          ( SHISAM            )            |
 |          |          |          (HDAM    [,CSAM])   )            |
 |          |          |,ACCESS= <         [ VSAM ]    >           |
 |          |          |          (                   )            |
 |          |          |          ( HIDAM             )            |
 |          |          |          ( INDEX             )            |
 |          |          |                                           |
 |          |          |[,RMNAME=(mod,anch,rbn,bytes)]             |
 |          |          |                                           |
 |          |          |[,PASSWD={YES}]                            |
 |          |          |         {NO }                             |
 |          |          |                                           |
 '-----------------------------------------------------------------'
```

Legend:

DBD

identifies this statement as the DBD control statement

NAME=dbname1

dbname1 is the name of the DBD for this data base. This name can be from one to eight alphameric characters. The first one should be an alphabetic character. It should be unique for each DBD in your installation's DL/I environment.

ACCESS=

specifies the DL/I access method and the operating system access method to be used for this data base. The value of the operand has the following meanings.

SHISAM

specifies a SHISAM data base with only a root segment with no prefix. It is a single VSAM KSDS.

HDAM

specifies a HDAM data base. OSAM or VSAM can be selected as the operating system access method. VSAM is the default.

HIDAM

specifies the HIDAM main data base. VSAM ESDS is used as the operating system access method in our subset.

INDEX

specifies the INDEX data base of a HIDAM data base. VSAM KSDS is used as the operating system access method in our subset.

<u>Note</u>:

- Guidelines for selecting the best access methods for a particular data base are provided under the topic "Selecting Data Base Organization and OS/VS Access Methods" later in this chapter.
- When VSAM is used, guidelines for the VSAM Access Method Services DEFINE command is produced in the DBDGEN output listing. These guidelines should be taken into account when defining the VSAM data set cluster.

RMNAME=(mod,anch,rbn,bytes)

should be specified only if ACCESS=(HDAM,...)

mod

specifies the load module name of the randomizing module to be used for this data base. For more details on randomizing modules see "HDAM Randomizing Modules" in Chapter 7.

anch

specifies the number of root anchor points desired in each control interval or block in the root addressable area of an HDAM data base. The default value of this parameter is one. "anch" must be an unsigned decimal integer and must not exceed 255.

When a user randomizing routine produces an anchor point number in excess of the number specified for this parameter, the anchor point used is the highest number in the control interval or block. When a randomizing routine produces an anchor point number of zero, DL/I uses anchor point one in the control interval or block.

rbn

specifies the maximum relative block number value that the user wishes to allow a randomizing module to produce for this data base. This value determines the number of control intervals or blocks in the root addressable area of an HDAM data base. "rbn" must be an unsigned decimal integer whose value does not exceed $2^{24}-1$. If the randomizing module produces an rbn greater than this parameter, the highest control interval or block in the root addressable area is used by DL/I. If the randomizing module produces a block number of zero, control interval or block one is used by DL/I.

bytes

specifies the maximum number of bytes of a data base record that can be stored into the root addressable area in a series of inserts unbroken by a call to another data base record. If this parameter is omitted, no limit is placed on the maximum number of bytes of a data base record that can be inserted into this data base's root segment addressable area. "bytes" must be an unsigned decimal integer whose value does not exceed $2^{24}-1$.

PASSWD=YES

causes DL/I Open to use the NAME=operand for this DBD as the VSAM password when opening any data set for this data base. This parameter is only valid for DBDs that use VSAM as the Operating System access method. The default is NO.

When the user defines the VSAM data set(s) for this data base using the DEFINE statement of OS/VS Access Method Services, the control level (CONTROLPW) or master level (MASTERPW) password must be the same as the NAME for this DBD. All data sets associated with this DBD must use the same password. For a description of the use and format of passwords for VSAM, see <u>OS/VS Access Method Services</u>.

For the IMS/VS DB/DC (online) system, all VSAM OPENs will bypass password checking and thus avoid operator password prompting. For the IMS/VS DB (batch) system, VSAM password checking is performed. In the batch environment, operator password prompting will occur if PASSWD=NO is specified and the data set is password-protected with passwords not equal to DBDNAME.

The intended use of this facility is to allow you to prevent accidental access of IMS/VS data bases by non-IMS/VS programs.

<u>DATASET Statement</u>:

This statement provides the link with the OS/VS data set and defines additional physical data set attributes. There is one for each DBD. The format of the DATASET macro instruction is:

	DATASET	DD1=ddname1
		$,DEVICE=\left\{\begin{matrix}2314\\2305\\2319\\3330\\3340\\3350\end{matrix}\right\}$
		$,MODEL=\left\{\begin{matrix}1\\2\\11\end{matrix}\right\}$
		,SIZE=size
		[,FRSPC=(fbff,fspf)]

Legend:

DATASET

identifies this statement as the DATASET control statement.

DD1=ddname1

identifies the ddname used in the JCL to execute DL/I application programs using the data base. It should be unique throughout the DL/I environment of your installation.

DEVICE=

specifies the device type used for storage of this data set.

MODEL=

specifies the model of the above device type. The valid combinations are:

For 2305: 1 or 2 (2 is the default)

For 3330: 1 or 11 (1 is the default)

SIZE=

specifies control interval size for VSAM data sets or blocksize for OSAM data sets. For VSAM data sets the size must be:

1. A multiple of 512 bytes
2. If larger than 8192, a multiple of 2048
3. Not larger than 30720

For OSAM data sets the size must be an even number, not exceeding 32K bytes, and must not exceed the maximum non-keyed blocksize per track of the direct access storage device used.

Notes:

1. As part of the CI/blocksize you specify, DL/I and VSAM allocate space for system fields. These are:
   - Free space anchor point — 4 bytes
   - Anchor points (HDAM only) — 4 bytes for each anchor point
   - VSAM control fields (ESDS) — 7 bytes
2. There is a free space element of eight bytes for each free space of 8 bytes or more.
3. Guidelines for selecting CI/blocksize , the bytes, anch and rbn parameters are provided later in this chapter.

FRSPC=(fbff,fspf)

specifies how free space is to be distributed in an HDAM or HIDAM data base. The fbff is the free block frequency factor, and it specifies that every nth control interval or block in this data set 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 fspf is the free space percentage factor. It specifies the minimum percentage of each control interval or block that is to be left as free space in this data set. The range of fspf is from 0 to 99. The default value for fbff and fspf is 0.

Notes:

1. If the total of the percentage of free space specified and any segment size exceed the control interval or block size, a warning message is issued by DBDGEN that flags oversized segments. When loading oversized segments, the "fspf" specification is ignored and one control interval or block is used to load each oversized segment.

2. In general, it is not advantageous to use the FRSPC=parameter for HDAM. In most cases, you can better control the free space in HDAM with the size of the root addressable area (rbn in the RMNAME=parameter of the DBD statement). This will be addressed later in this chapter under the topic "Design the Physical Data Structures."

SEGM Statement

This statement is used once for each segment to be defined in the DBD. Its basic format is:

```
|          | SEGM     | NAME=segname1
|          |          | [,PARENT=((segname2 [,SNGL])) ]
|          |          |                     [ DBLE]
|          |          | ,BYTES=bytes
|          |          |       { T  }
|          |          | ,PTR= { TB }
|          |          |       { NT }
```

Legend:

SEGM

identifies this statement as a SEGM control statement

NAME=segname1

specifies the name of the segment as used by DL/I and the application program. It is one to eight alphameric characters and each segment name should be unique in the DL/I environment of your installation.

PARENT=

specifies the name of the physical parent of this segment. This keyword should be omitted for the root segment. The second parameter controls the physical child pointer(s) in the physical parent of this segment.

SNGL specifies only a physical child first pointer is used in this segments parent.

DBLE specifies both a physical child first and physical child last pointer are used in this segment's parent.

Recommendation: DBLE should be specified if:

1. Average twin chain is more than 3 to 5 and frequent retrieve lasts, and/or

2. Segment has no sequence field and frequent inserts are expected.

BYTES=

specifies the length of the data portion of the segment in bytes. This length does not include the prefix, which is established solely by DL/I. This length cannot exceed the maximum logical record length or control interval/block size of the data set minus the space occupied by system fields. See the SIZE parameter of the DATASET statement. It should be an even number.

PTR=

controls the physical twin pointer options. Specify:

PTR=NT (no twin pointer) if never more than one occurrence of this segment under its parent. No sequence field may be defined for the segment if PTR=NT is specified.

PTR=TB (twin forward and backward pointers) if:

- No sequence field is defined and frequent inserts are expected.
- Retrieve last plus subsequent delete is frequently used.
- The segment is a logical child. See phase 2.
- It is the root segment of a HIDAM data base.

PTR=T (only twin forward pointer) in all other cases.

FIELD Statement

This statement is used once for each field to be defined in the DBD. The FIELD statements follow the SEGM statement of the segment in which these fields belong. This statement is required for all sequence fields and fields which are to be used in SSAs. The basic format is:

```
 /---------------------------------------------------------------
/         |          |                                          |
|         |FIELD     |NAME=(fldname1[,,SSEEQ])                  |
|         |          |                                          |
|         |          |,BYTES=bytes                              |
|         |          |,START=startpos                           |
|         |          |       (X)                                |
|         |          |,TYPE= {P}                                |
|         |          |       (C)                                |
|         |          |                                          |
-----------------------------------------------------------------
```

Legend:

FIELD

identifies this statement as a FIELD control statement.

NAME=

fldname1

specifies the name of the field being defined within a segment type. The name specified can be referred to by an application program in a DL/I call SSA. Duplicate field names must not be defined for the same segment type. fldname1 must be a one to eight character alphameric value.

SEQ

the presence of the keyword SEQ as a parameter of this operand identifies this field as a sequence field in the segment type. FIELD statements containing the keyword SEQ must be the first FIELD statements following a SEGM statement in a DBD generation input deck. As a general rule, a segment can have only one sequence field. If a sequence field is specified, then its value must be unique in our subset for all segment occurrences under a given parent.

A unique sequence field is optional for all dependent segment types. It must be provided for the root segment of SHISAM, HIDAM, and primary HIDAM INDEX data bases.

When no sequence field is defined for a segment, new occurrences of the segment will be inserted at the end of the physical twin chain. It is required, in our subset, that all parent segments which participate in logical relationships have unique sequence fields (except if PTR=NT is specified). This includes the physical and the logical parent and their parent segments up to their roots.

BYTES=

specifies the length of the field being defined in bytes. The maximum allowed is 255.

START=

specifies the starting position of the field being defined in terms of bytes relative to the beginning of the segment. Maximum allowed value is 30720. startpos for the first byte of a segment is one. Overlapping fields are permitted.

TYPE=

specifies the type of data that is to be contained in this field. The value of the parameter specified for this operand indicates that one of the following types of data will be contained in this field:

X = hexadecimal data

P = packed decimal data

C = alphameric data or a combination of types of data.

It should be noted that all DL/I calls perform field comparisons on a byte-by-byte binary basis. No check is made by DL/I to ensure that the data contained within a field is of the type specified by this operand, except when the defined field is indexed.

LCHILD Statement

This statement is used once for each index or logical relation a segment has. It immediately follows the SEGM statement or FIELD statements of the segment involved. At this point we will only discuss its use in defining the primary index of a HIDAM data base. The basic format is:

```
|          | LCHILD   | NAME=(segname,dbname)
|          |          |
|          |          | [,PTR=INDX]
|          |          |
|          |          | [,INDEX=fldname]
```

The LCHILD statement is coded both in the INDEX data base and in the HIDAM main data base. For the INDEX data base, code:

NAME=(segname,dbname)

segname is the name of the HIDAM root segment and dbname is the name of the HIDAM data base as coded in the DBD statement.

INDEX=fldname

fldname is the name of the sequence field of the HIDAM root segment.

For the HIDAM main data base, code:

NAME=(segname,dbname)

segname is the name of the only segment in the primary INDEX data base for this data base, and dbname is the name of that INDEX data base.

PTR=INDX

must be coded as shown. It provides for the linkage with the INDEX data base.

DBDGEN Statement

This statement must be included. It indicates the end of DBD generation control cards to define the DBD. The format is:

```
   /-----------------------------------------------------------------┐
  /        |          |                                              |
 |         |DBDGEN    |                                              |
 |         |          |                                              |
 └-------------------------------------------------------------------┘
```

FINISH Statement

This statement must be included. It sets a non-zero condition code for link-edit if there are DBDGEN errors. The format is:

```
   /-----------------------------------------------------------------┐
  /        |          |                                              |
 |         |FINISH    |                                              |
 |         |          |                                              |
 └-------------------------------------------------------------------┘
```

END Statement

This statement must be included. It indicates the end of the input statements to the OS/VS assembler.

```
   /-----------------------------------------------------------------┐
  /        |          |                                              |
 |         |END       |                                              |
 |         |          |                                              |
 └-------------------------------------------------------------------┘
```

Execution of DBDGEN (JCL)

DBDGEN must be run as a normal operating system job after IMS/VS System definition. System definition causes the DBDGEN procedure to be placed in the IMSVS.PROCLIB library. To process a request for a DBDGEN, the DBD generation control cards must be created and appended to the following JCL (which invokes the DBDGEN procedure):

```
//DBDGEN  JOB  MSGLEVEL=1
//        EXEC DBDGEN,MBR=
//C.SYSIN DD   *
          DBD
          DATASET
          SEGM
          FIELD           DBD generation
          LCHILD          control cards
          DBDGEN
          FINISH
          END
/*
```

where keyword operand MBR=

is the name of the DBD to be generated. This name should be the same as the first name specified for the NAME= keyword on the DBD statement. When a data base PCB (see PSBGEN later in this chapter) relates to this DBD generation, this operand value must be the name used in the DBDNAME= operand on the data base PCB statement within a PSB generation.

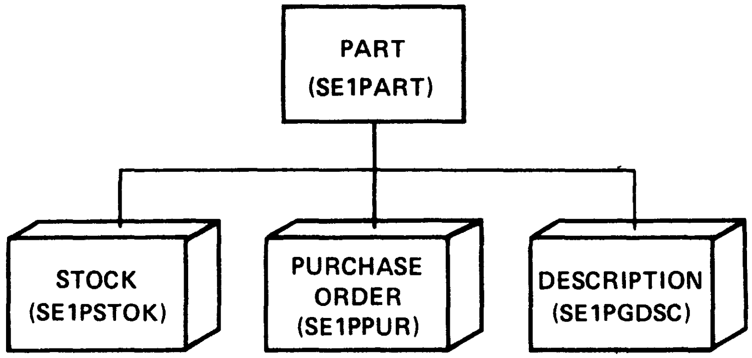
Note: If the defined DBD is for the primary INDEX data base of an HIDAM data base, only one each of the SEGM, FIELD and LCHILD statements is allowed.

Examples Of Physical DBDs

Figure 2-22 shows a sample HDAM data base which uses OSAM. This is our sample, BE1PARTS, included in IMSVS.PRIMESRC, Phase 1 PARTS data base. Job //SAMP110 in IMSVS.PRIMEJCL can be used for its DBDGEN.

Figure 2-23 shows the HIDAM version of the same PARTS data base. As can be seen, two DBDs are required, one for the index data base and one for the main data base.

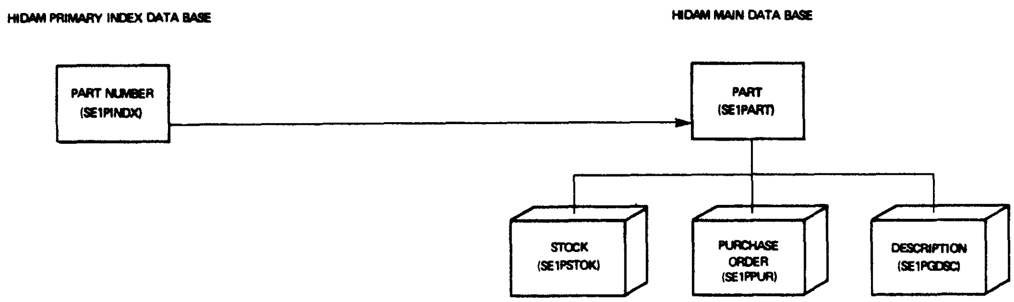
Notice that the HIDAM data bases use only VSAM. The DBDs of Figure 2-23 are not provided in IMSVS.PRIMESRC. However, they can be easily established if you were interested in using HIDAM for the PARTS data base.




```
*                                  DESCRIPTION OF PARTS DATABASE
*                                  FOR PRIMER SAMPLE PROJECT PHASE 1
*                                  ---------------------------------------
*
         DBD      NAME=BE1PARTS,ACCESS=(HDAM,OSAM),
                  RMNAME=(DFSHDC40,4,80,500)
*
         DATASET DD1=DE1PARTS,DEVICE=3330,MODEL=1,SIZE=2048
*
*                                  PARTS- GENERAL INFORMATION (ROOT)
*                                  ---------------------------------------
         SEGM     NAME=SE1PART,BYTES=80,PTR=T
         FIELD    START=01,BYTES=08,TYPE=C,NAME=(FE1PGPNR,SEQ)
         FIELD    START=09,BYTES=13,TYPE=C,NAME=(FE1PGSNM)
         FIELD    START=22,BYTES=08,TYPE=C,NAME=(FE1PGNEW)
         FIELD    START=30,BYTES=08,TYPE=C,NAME=(FE1PGOLD)
         FIELD    START=38,BYTES=08,TYPE=C,NAME=(FE1PGEQV)
         FIELD    START=46,BYTES=08,TYPE=C,NAME=(FE1PGUNT)
         FIELD    START=54,BYTES=08,TYPE=C,NAME=(FE1PGPRI)
         FIELD    START=62,BYTES=08,TYPE=C,NAME=(FE1PGDIM)
*
*                                  PARTS- STOCK INFORMATION
*                                  ---------------------------------------
        SEGM      NAME=SE1PSTOK,BYTES=40,PARENT=((SE1PART,SNGL)),PTR=T
        FIELD     START=01,BYTES=12,TYPE=C,NAME=(FE1PSLOC,SEQ)
        FIELD     START=13,BYTES=06,TYPE=C,NAME=(FE1PSDAT)
        FIELD     START=19,BYTES=06,TYPE=C,NAME=(FE1PSCNT)
        FIELD     START=25,BYTES=06,TYPE=C,NAME=(FE1PSISS)
        FIELD     START=31,BYTES=06,TYPE=C,NAME=(FE1PSREC)
*
*                                  PARTS- PURCHASE ORDER INFORMATION
*                                  ---------------------------------------
         SEGM     NAME=SE1PPUR,BYTES=60,PARENT=((SE1PART,SNGL)),PTR=T
         FIELD    START'01,BYTES=08,TYPE=C,NAME=(FE1PPONR,SEQ)
         FIELD    START=09,BYTES=06,TYPE=C,NAME=(FE1PPODT)
         FIELD    START=15,BYTES=20,TYPE=C,NAME=(FE1PPOSU)
         FIELD    START=35,BYTES=06,TYPE=C,NAME=(FE1PPQOD)
         FIELD    START=41,BYTES=06,TYPE=C,NAME=(FE1PPQRD)
         FIELD    START=47,BYTES=06,TYPE=C,NAME=(FE1PPDDT)
*
*                                  PARTS- GENERAL DESCRIPTION
*                                  ---------------------------------------
         SEGM     NAME=SE1PGDSC,BYTES=80,PARENT=((SE1PART,SNGL)),PTR=NT
         FIELD    START=01,BYTES=50,TYPE=C,NAME=(FE1PGLNM)
*
         DBDGEN
         FINISH
           END
```

Figure 2-22. Phase 1 HDAM PARTS DBD, BE1PARTS

HIDAM PARTS DBDs




```
*                            DESCRIPTION OF PRIMARY INDEX        *
*                            INTO THE PARTS HIDAM DATABASE       *
*                            ----------------------------        *
*                                                                *
    DBD      NAME=BE1PINDX,ACCESS=(INDEX,VSAM)
*                                                                *
    DATASET DD1=DE1PINDX,DEVICE=3330,MODEL=1,SIZE=2048
*                                                                *
    SEGM     NAME=SE1PINDX,BYTES=8                               *
    LCHILD   NAME=(SE1PART,BE1PARTS),INDEX=(FE1PGPNR)            *
    FIELD    NAME=(FE1INDX,SEQ),BYTES=8,START=1
*
    DBDGEN
    FINISH
    END
```

```
*                            DESCRIPTION OF PARTS DATABASE
*                            FOR PRIMER SAMPLE PROJECT PHASE 1
*                            -------------------------------------
*
    DBD      NAME=BE1PARTS,ACCESS=HIDAM
*
    DATASET DD1=DE1PARTS,DEVICE=3330,MODEL=1,SIZE=2048
*
*                            PARTS- GENERAL INFORMATION (ROOT)
*                            -------------------------------------
    SEGM     NAME=SE1PART,BYTES=80,PTR=TB
    LCHILD   NAME=(SE1PINDX,BE1PINDX),PTR=INDX
    FIELD    START=01,BYTES=08,TYPE=C,NAME=(FE1PGPNR,SEQ)
     .
     .
     .
    DBDGEN
    FINISH
     END
```

Figure 2-23. Sample DBDs for a HIDAM Data Base

DBDGEN FOR GSAM

A GSAM DBD contains the following statements:

```
--------------------------------------------------------------
 /        |         |                                         |
/         |         |                                         |
|         |DBD      |NAME=dbname,ACCESS=(GSAM,BSAM)           |
|         |DATASET  |DD1=ddname,RECFM=recfm                   |
|         |         |[,RECORD=lrecl]                          |
|         |         |[,SIZE=blksize]                          |
|         |DBDGEN   |                                         |
|         |FINISH   |                                         |
|         |END      |                                         |
|         |         |                                         |
--------------------------------------------------------------
```

NAME=dbname

specifies the name of this data base.

DD1=ddname

specifies the name of the DD statement used in the JCL when accessing this data base.

RECFM=recfm

specifies the format of the records in the dataset. The record format is specified using the characters defined below:

F -- the records are of fixed length.

FB -- the records are of fixed length and are blocked.

V -- the records are of variable length.

VB -- the records are variable length and are blocked.

RECORD=lrecl

specifies the size of a logical record for a fixed length record and the maximum logical record length for a variable length record.

SIZE=blksize

specifies the blocksize of the GSAM dataset for fixed length records or the maximum blocksize for variable length records.

The record and size parameters can also be specified via the JCL. Two sample GSAM DBDs, B00INP01 and B00OUT01 are included in IMSVS.PRIMESRC. Their DBDGENs can be executed with job //SAMP010 in IMSVS.PRIMEJOB. Furthermore, these DBDs can be used by your own application programs if the file attributes are the same.

DBDGEN FOR LOGICAL RELATIONSHIPS

To support the logical relationship function, DBDGEN is extended in two ways:

- Additional control statements and parameters can be specified in the physical DBD.
- A different type of DBD is created for the definition of the logical data base. However, this is done with an extension of the existing control statements.

The DBDGEN process itself is unchanged.

Coding a Logical Relationship in a Physical DBD

The following control statements are unchanged:

```
DBD
FIELD
DBDGEN
FINISH
END
```

Note: Additional restrictions exist for the length of a sequence field of a segment involved in a logical relationship. See the section "Restrictions" for the Data Base Prefix Resolution Utility in Chapter 5, "Data Base Reorganization/Load Processing."

The following statements are extended:

SEGM
LCHILD

<u>Logical Child</u>: For each defined logical child, you need to code two SEGM statements. One within its physical parent's DBD and one within its logical parent's DBD. The format under the physical parent DBD, that is, for the real logical child is:

```
|          | SEGM     | NAME=segname1
|          |          | ,PARENT=
|          |          |     ((segname2, SNGL),(segname3,P,dbname2))
|          |          |                 DBLE
|          |          | ,BYTES=bytes
|          |          |          { T }
|          |          | ,PTR= (LP,{ TB }[,{ LT  }])
|          |          |          { NT }   { LTB }
|          |          |
|          |          | ,RULES=VVV
```

NAME=segname1

segname1 is the name of the logical child segment.

PARENT=

segname2 is the name of the physical parent segment of this logical child.

SNGL and DBLE have the same meaning as before.

segname3 is the name of the logical parent of this logical child. P should be specified as shown in our subset, it defines the logical parent concatenated key to be stored with the segment in physical storage. dbname2 is the DBD name of the logical parent's data base.

BYTES=bytes

has the same meaning as before. Notice however that the logical child always contains the logical parent's concatenated key in the first n bytes, and its length must be included here.

PTR=

LP must be specified as shown in our subset. It provides for a pointer to the logical parent in the prefix of the LCHILD.

T the same considerations as before apply.

TB it is highly recommended that you specify TB if there are, on the average, more than 3 to 5 logical child occurrences per physical parent.

NT should be specified if never more than one occurrence of this segment per parent

LT if specified, only a logical twin forward pointer is used for the logical twin chain.

LTB if specified, both a logical twin forward and backward pointer are used for the logical twin chain. This should be selected whenever there are, on the average, more than 2 to 3 logical child occurrences for a logical parent.

RULES = VVV

should be specified as shown for our subset.

The format under the logical parent, that is, for the virtual logical child is:

```
 /-----------------------------------------------------------------
/       |         |
|       |SEGM     |NAME=virtchild                                 |
|       |         |,PARENT=segname2                               |
|       |         |,SOURCE=((segname3,D,dbname1))                 |
|       |         |,PTR=PAIRED                                    |
|       |         |                                               |
-------------------------------------------------------------------
```

Legend:

NAME=virtchild

virtchild is the name of the virtual logical child. Remember that the virtual logical child does not actually exist. Its only purpose is to define the logical child as seen from the logical path. It can be followed by a sequence field which controls the sequence of the logical child segment when accessed via its logical path, that is, the logical twin chain sequence.

PARENT=segname2

segname2 is the name of the logical parent, that is, the physical parent of the virtual logical child.

SOURCE=((segname3,D,dbname1))

segname3 is the name of the real logical child and dbname1 is the DBD name of the data base which contains that logical child. D should be specified in our subset, it defines that both key and data of the segment are accessible by the PSB.

PTR=PAIRED

Should be specified as shown. It defines this segment as a virtual logical child.

Physical and Logical Parent: One additional parameter must be specified in the SEGM statement of both the physical and the logical parent:

SEGM NAME=.....,RULES=PLV

For each logical child segment type, an LCHILD statement must be added immediately following the SEGM and/or FIELD statement of the __logical__ parent. Its basic format is:

```
 /---------------------------------------------------------------
/       |        |                                              |
|       |LCHILD  |NAME=(segname1,dbname)                        |
|       |        |                                              |
|       |        | [,PTR= {SNGL} ]                              |
|       |        |        {DBLE}                                |
|       |        |                                              |
|       |        |,PAIR=virtchild                               |
|       |        |                                              |
----------------------------------------------------------------
```

__Legend__:

NAME=(segname1,dbname)

segname1 is the segment name of the logical child in the DBD whose name is dbname.

PTR= __SNGL__
DBLE

SNGL specifies that there will be only a logical child first pointer in the prefix of the logical parent. DBLE specifies that both a logical child first and last pointer will appear in the logical parent.

Recommendations:

Specify SNGL if a sequence field is defined for the virtual logical child and command code L (retrieve last) is rarely or never used to access the logical child.

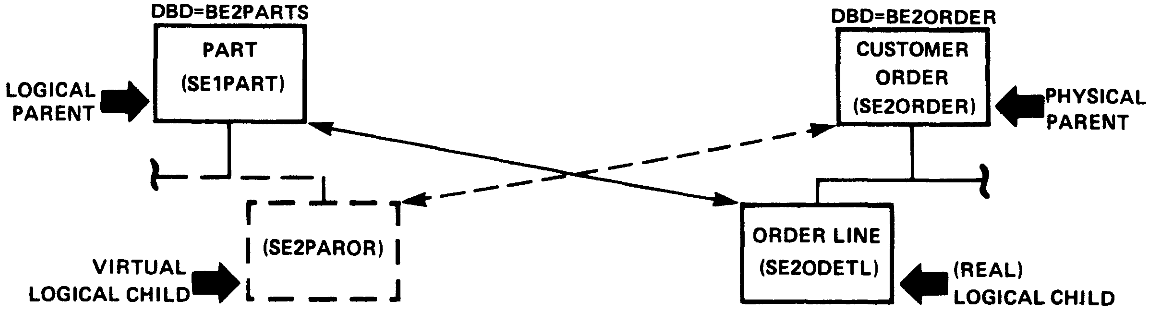
Specify DBLE if no sequence field is defined for the virtual logical child and/or command code L is heavily used and there are, on the average, more than three occurrences of virtual children within a logical parent.

PAIR=virtchild

virtchild specifies the name of the virtual logical child which must be defined in the same DBD (see previous SEGM statement).

__Examples of Physical DBDs With Logical Relationships__

Figure 2-24 shows the two logically related physical DBDs of our Phase 2 sample environment. Only those DBD statements are shown which are essential to the logical relationship function. Compare these DBDs with the ones of Figure 2-22 and 2-23. The DBDs of Figure 2-24 are also included in IMSVS.PRIMESRC. Their DBDGENs can be performed with job //SAMP210 in IMSVS.PRIMEJOB.




```
*                         DESCRIPTION OF PARTS DATABASE
*                         FOR PRIMER SAMPLE PROJECT PHASE 2
*                         ----------------------------------------
*
   DBD      NAME=BE2PARTS,ACCESS=(HDAM,VSAM),                          *
            RMNAME=(DFSHDC40,4,80,500)
*
   DATASET DD1=DE2PARTS,DEVICE=3330,MODEL=1,SIZE=2048
*
*                         PARTS- GENERAL INFORMATION (ROOT)
*                         ----------------------------------------
   SEGM     NAME=SE1PART,BYTES=80,PTR=T,RULES=PLV
*
   LCHILD   NAME=(SE2ODETL,BE2ORDER),PAIR=SE2PAROR,PTR=DBLE
   FIELD    START=01,BYTES=08,TYPE=C,NAME=(FE1PGFNR,SEQ)
     •
     •
     •
*                         VIRTUAL LOGICAL CHILD
*                         (CONNECTION FROM CUSTOMER-ORDER DB)
*                         ----------------------------------------
   SEGM     NAME=SE2PAROR,PARENT=SE1PART,PTR=PAIRED,                   *
            SOURCE=((SE2ODETL,D,BE2ORDER))
   FIELD    START=01,BYTES=06,NAME=(FE2ODRNR,SEQ)
*           DEFINES SEQUENCE OF LT-CHAIN
*
     •
     •
     •
   DBDGEN
   FINISH
   END
```

```
*                         DESCRIPTION OF ORDER DATABASE
*                         FOR PRIMER SAMPLE PROJECT PHASE 2
*                         ----------------------------------------
*
   DBD      NAME=BE2ORDER,ACCESS=HIDAM
*
   DATASET DD1=DE2ORDER,DEVICE=3330,MODEL=1,SIZE=2048
*
*                         CUSTOMER-ORDER GENERAL INFORMATION
*                         ----------------------------------------
   SEGM     NAME=SE2ORDER,BYTES=60,PTR=TB,RULES=PLV
   LCHILD   NAME=(SE2ORDRX,BE2ORDRX),PTR=INDX
*
   FIELD    START=01,BYTES=06,TYPE=C,NAME=(FE2OGPEF,SEQ)
     •
     •
     •
     •
     •
*                         CUSTOMER-ORDER DETAIL INFORMATION
*                         ----------------------------------------
   SEGM     NAME=SE2ODETL,BYTES=30,                                    *
            PARENT=((SE2ORDER,DBLE),(SE1PART,P,BE2PARTS)),             *
            PTR=(TB,LTB,LP),RULES=VVV
   FIELD    START=09,BYTES=03,TYPE=C,NAME=(FE2ODLNR,SEQ)
     •
     •
     •
   DBDGEN
   FINISH
   END
```

Figure 2-24. Phase 2 Physical DBDs

Coding A Logical DBD

A logical DBD, based on existing physical DBDs, defines a new view of logically related data bases. This view is always a hierarchical data structure. Following are the control statements used and their formats:

DBD Statement:

```
/-----------------------------------------------------------------------
|          |DBD        |NAME=dbdname1,ACCESS=LOGICAL                    |
|          |           |                                                |
L------------------------------------------------------------------------
```

NAME=dbdname1

dbdname1 is the name of this logical DBD. It must be unique in your installation and the same name as specified in the MBR operand of DBDGEN.

ACCESS=LOGICAL

defines this DBD as a logical DBD

DATASET Statement:

```
   /-------------------------------------------------------------┐
 /         |          |                                           |
|          |DATASET   |LOGICAL                                    |
|          |          |                                           |
└-------------------------------------------------------------┘
```

This statement must be coded as shown.

SEGM Statement:

The segments in a logical DBD must be coded in hierarchical sequence following the rules for defining logical data bases as presented earlier in this chapter.

```
   /-------------------------------------------------------------┐
 /         |          |                                           |
|          |SEGM      |NAME=segname1                              |
|          |          |[,PARENT=segname2]                         |
|          |          |,SOURCE=((segname3,D,dbname1)              |
|          |          |    [,(segname4,D,dbname2)])               |
|          |          |                                           |
└-------------------------------------------------------------┘
```

NAME=segname1

segname1 is the name of this segment.

PARENT=segname2

segname2 specifies the name of the parent of this segment. segname2 must be defined previously in this DBD. This parameter should be omitted for the root segment.

SOURCE=((segname3,D,dbname1)[,(segname4,D,dbname2)])

This parameter specifies the source(s) of the defined segment. The long form is only applicable to concatenated segments.

Non-concatenated segments:

segname3 defines the source segment. The source segment must be defined in a physical DBD whose name is dbname1.

Concatenated segments:

- segname3 defines the logical child as defined in the physical DBD. If the preceding parent segment is the physical parent, then the name of the logical child must be coded. If the preceding parent is the logical parent, then the name of the virtual logical child must be coded.
- dbname1 defines the physical DBD in which segname3 is defined.
- segname4 defines the destination parent.
- dbname2 defines the physical DBD name of the destination parent.

Note: The destination parent (segname4) should be included in the concatenated segment only if your application has a real need for it. If it is not specified, DL/I does not need to access the destination parent except for insert and delete calls.

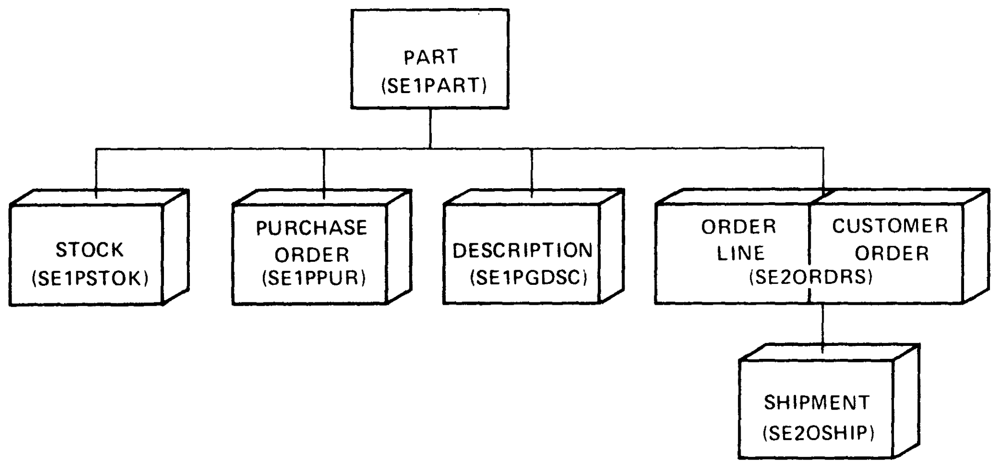
DBDGEN, FINISH And END Statements

These should be coded as before.

Note that no LCHILD or FIELD statements are allowed in a logical DBD.

Example Of Logical DBDs

Figure 2-25 shows the logical DBD for our Phase 2 PARTS data base, BE2LPART.

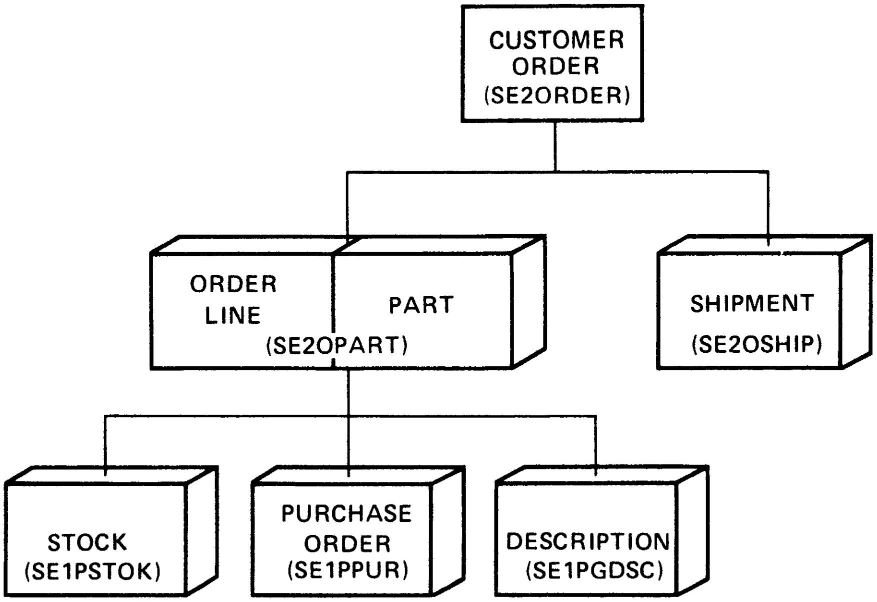



```
                        DATABASE DESCRIPTION OF THE COMBINED
                        PARTS/ORDER DATABASE (LOGICAL)
DBD     NAME=BE2LPART,ACCESS=LOGICAL
DATASET LOGICAL

SEGM    NAME=SE1PART,SOURCE=((SE1PART,D,BE2PARTS))
SEGM    NAME=SE1PSTOK,SOURCE=((SE1PSTOK,D,BE2PARTS)),                  *
        PARENT=SE1PART
SEGM    NAME=SE1PPUR,SOURCE=((SE1PPUR,D,BE2PARTS)),                    *
        PARENT=SE1PART
SEGM    NAME=SE1PGDSC,SOURCE=((SE1PGDSC,D,BE2PARTS)),                  *
        PARENT=SE1PART
SEGM    NAME=SE2ORDRS,                                                 *
        SOURCE=((SE2PAROR,D,BE2PARTS),(SE2ORDER,D,BE2ORDER)),          *
        PARENT=SE1PART
SEGM    NAME=SE2OSHIP,SOURCE=((SE2OSHIP,D,BE2ORDER)),                  *
        PARENT=SE2ORDRS
DBDGEN
FINISH
END
```

Figure 2-25. Phase 2 Logical DBD for the PARTS Data Base

Figure 2-26 shows the logical DBD for our Phase 2 CUSTOMER ORDERS data base, BE2LORDR.




```
                     DATABASE DESCRIPTION OF THE COMBINED
                     ORDER/PARTS DATABASE (LOGICAL)
                     ---------------------------------------
DBD     NAME=BE2LORDR,ACCESS=LOGICAL
DATASET LOGICAL

SEGM    NAME=SE2ORDER,SOURCE=((SE2ORDER,D,BE2ORDER))
SEGM    NAME=SE2OPART,                                              *
        SOURCE=((SE2ODETL,D,BE2ORDER),(SE1PART,D,BE2PARTS)),        *
        PARENT=SE2ORDER
SEGM    NAME=SE1PSTOK,SOURCE=((SE1PSTOK,D,BE2PARTS)),               *
        PARENT=SE2OPART
SEGM    NAME=SE1PPUR,SOURCE=((SE1PPUR,D,BE2PARTS)),                 *
        PARENT=SE2OPART
SEGM    NAME=SE1PGDSC,SOURCE=((SE1PGDSC,D,BE2PARTS)),               *
        PARENT=SE2OPART
SEGM    NAME=SE2OSHIP,SOURCE=((SE2OSHIP,D,BE2ORDER)),               *
        PARENT=SE2ORDER
DBDGEN
FINISH
END
```

Figure 2-26. Phase 2 Logical DBD for the CUSTOMER ORDERS Data Base

The logical DBDs of Figure 2-25 and 2-26 are included in IMSVS.PRIMESRC. Their DBDGENs can be performed with job //SAMP210 in IMSVS.PRIMEJCB.

DBDGENS FOR SECONDARY INDEXES

To support the secondary index function, the DBDGEN process is extended. We differentiate between the index target DBD and the index pointer DBD.

CODING AN INDEX TARGET DATA BASE

The control statements extended for the secondary index function are:

SEGM
FIELD
LCHILD

A new control statement is added:

XDFLD

The following control statements are unchanged:

DBD
DATASET
SEGM
DBDGEN
FINISH
END

Coding the Index Target Segment

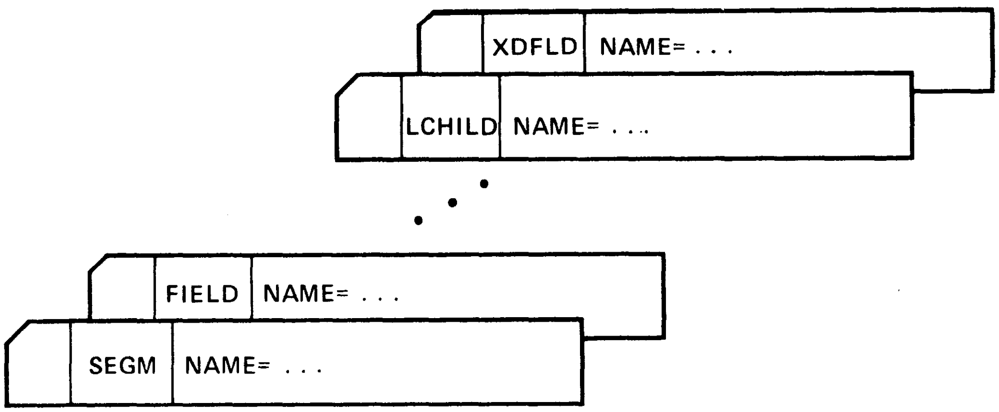


Figure 2-27. DBD Statements for Index Target Segment

SEGM Statement

SEGM

is a standard SEGM statement for the root segment. It has no additional parameter for secondary indexes. It is recognized as an index target segment because of the following LCHILD and XDFLD statements.

LCHILD Statement

```
  /------------------------------------------------------------------.
 /         |             |                                            |
|          |LCHILD       |NAME=(segname1,dbname),PTR=INDX             |
|          |             |                                            |
'---------------------------------------------------------------------'
```

LCHILD

This statement provides the link to the index data base.

NAME=(segname1,dbname)

segname1 is the name of the index pointer segment as defined in the INDEX data base. dbname is the name of the DBD for the INDEX data base.

PTR=INDX

identifies the LCHILD statement as an index type.

Note: There are three types of LCHILD statements; one for the primary index of an HIDAM data base, one for the definition of a logical child under its logical parent, and one for the definition of the index target segment. All three types could occur below the root segment of a HIDAM data base. There could be multiple occurrences of LCHILD statements for both logical relationships and secondary indexes. The relative order of the LCHILD statements should be as described above. If multiple secondary indexes are to be defined for one segment, the XDFLD statement must immediately follow its corresponding LCHILD statement.

XDFLD Statement

```
   /------------------------------------------------------------------,
 /        |          |                                                |
|         |XDFLD     |NAME=fldname                                    |
|         |          |,SEGMENT=segname                                |
|         |          |,SRCH=list1                                     |
|         |          |[,SUBSEQ=/SXname]                               |
|         |          |                                                |
L---------------------------------------------------------------------J
```

XDFLD

This statement defines the index source fields, that is, the fields used for the secondary index access. It defines the source data for the index search field in the INDEX data base.

NAME=fldname

specifies the name of the secondary index field. fldname is a normal field name which can be used in the SSA for the call which requests secondary index access. It must be unique among all field names specified for the above index target segment.

SEGMENT=segname

specifies the index source segment for this secondary index relationship. segname must be the name of a subsequently defined segment type, which is hierarchically below the index target segment type or it can be the name of the index target segment type itself. The segment name specified must not be a logical child segment. If this operand is omitted, the index target segment type is assumed to be the index source segment.

SRCH=list1

specifies which field or fields of the index source segment are to be used as the search field of a secondary index. list1 must be a list of one to five field names defined in the index source segment type by FIELD statements. If two or more names are included, they must be separated by commas and enclosed in parentheses. The sequence of names in the list is the sequence in which the field values will be concatenated in the index pointer segment search field. The sum of the lengths of the participating fields forms the length of this XDFLD as used in SSAs.

SUBSEQ=/SXname

This parameter must be coded if duplicate index pointer segments may occur. /SXname must be the same as coded in the corresponding field statement of the index source segment. (See the next section, "Coding the Index Source Segment.")

Coding the Index Source Segment

```
                  ----------------------------------------┐
                 /         |           |                   |
                |          |FIELD      |NAME=/SXname,...   |
            ----|-------------------|-------------------┐  |
            /          |         |                      |  |
           |           |FIELD    |NAME=...              |---┘
      ----------------------------------------------┐   |
     /         |          |                         |   |
    |          |SEGM      |NAME=...                 |---┘
    |          |          |                         |
    └-----------------------------------------------┘
```

Figure 2-28. DBD Statements for Index Source Segment

SEGM Statement

SEGM

This is a standard SEGM statement with no additional parameters. It is recognized as an index source segment because it is defined in a preceding XDFLD statement under the index target segment. It must not be a logical child.

FIELD Statement

```
  /------------------------------------------------------------┐
 /          |          |                                       |
|           |FIELD     |NAME=/SXname,BYTES=4,START=1           |
|           |          |                                       |
└--------------------------------------------------------------┘
```

FIELD

In addition to the normal FIELD statements for the segment, one extra FIELD statement can be added. Its name must start with /SX. This field is required whenever duplicate XDFLDs may occur in the data base. Although the values of BYTES and START are disregarded, they must be coded. Note that the /SXname field is called a "system related field." It provides control information to DL/I and it is completely transparent to the application program. Example: In our purchase order, secondary index, there may well occur multiple index pointer segments with the same purchase order number (that is, for the different parts ordered in one purchase order). Therefore, this function is required in that data base, otherwise duplicate KSDS keys would occur.

CODING A SECONDARY INDEX DBD

The following statements are used in a secondary index DBD:

```
DBD
DATASET
SEGM
LCHILD
FIELD
DBDGEN
FINISH
END
```

DBD Statement

```
  /--------------------------------------------------------------------
 /          |          |                                               |
|           |DBD       |NAME=dbname1                                   |
|           |          |,ACCESS=(INDEX[,DOSCOMP])                      |
|           |          |                                               |
 ----------------------------------------------------------------------
```

NAME=dbname1

specifies the name of the secondary index data base. It should be the name specified by the MBR keyword of DBDGEN.

ACCESS=(INDEX[,DOSCOMP])

INDEX identifies this as an index data base. DOSCOMP is an optional parameter and should be specified if this data base was created with DOS DL/I.

DATASET Statement

```
  /--------------------------------------------------------------------
 /          |          |                                               |
|           |DATASET   |DD1=ddname1,DEVICE=device,MODEL=model          |
|           |          |,SIZE=size                                     |
|           |          |                                               |
 ----------------------------------------------------------------------
```

The values specified for the DD1, DEVICE and MODEL parameters are exactly the same as discussed under "Basic DBDGEN Control Statements Formats."

SIZE=size

specifies the control interval size of the KSDS for the INDEX data base. This value must conform to the rules specified under "Basic DBDGEN Control Statements Formats." See also Selecting CI/block sizes later in this chapter.

SEGM Statement

```
  /--------------------------------------------------------------------
 /          |          |                                               |
|           |SEGM      |NAME=segname1,BYTES=bytes                      |
|           |          |                                               |
 ----------------------------------------------------------------------
```

Only one SEGM statement with its associated LCHILD and FIELD statements is required for the secondary index data base.

NAME=segname1

specifies the name of the segment being defined. Although not used by application programs in the subset, it should be unique among the segment names in your installation.

BYTES=bytes

specifies the length of the data portion of the index pointer segment. If a /SXname field is defined in the SUBSEQ parameter of the corresponding XDFLD statement, then its length (4 bytes) must be included here.

LCHILD Statement

```
 /-----------------------------------------------------------------┐
/          |          |                                            |
|          |LCHILD    |NAME=(segname1,dbname)                      |
|          |          |,PTR=SNGL                                   |
|          |          |,INDEX=fldname                              |
|          |          |                                            |
└------------------------------------------------------------------┘
```

NAME=(segname1,dbname)

specifies the segment name of the index target segment and the name of the DBD in which it is defined.

PTR=SNGL

specifies that a 4-byte direct byte address pointer in the prefix of the index pointer segment will be used. It will point to the index target segment.

INDEX=fldname

specifies the fieldname of the indexed field. This fldname must be specified as the name of an XDFLD below the index target segment.

FIELD Statement

```
 /-----------------------------------------------------------------┐
/          |          |                                            |
|          |FIELD     |NAME=(fldname1,SEQ)                         |
|          |          |,BYTES=bytes                                |
|          |          |,START=1                                    |
|          |          |                                            |
└------------------------------------------------------------------┘
```

Only one FIELD statement should be coded for each SEGM statement.

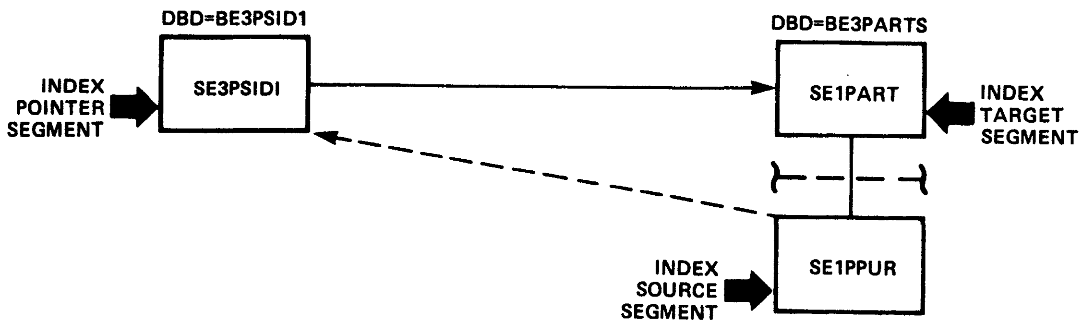
NAME=(fldname1,SEQ)

fldname1 is the name of this field. It is not used by the application program in our subset. However, it should be specified following the rules of other fieldnames. SEQ defines this as a unique sequence field and must be specified as shown in our subset.

BYTES=bytes

specifies the length of the field. This is the length of the search field as defined in the XDFLD statement, plus four if the /SX field is included. It also is the length of the key for the KSDS. In our subset, it is equal to the length of the preceding segment.

The DBDGEN, FINISH and END statements should be coded as before. Figure 2-29 shows the physical PARTS DBD (BE3PARTS) and its associated PURCHASE ORDER secondary index DBD (BE3PSID1) for our Phase 3 sample environment. These DBDs, together with the Phase 3 CUSTOMER ORDERS DBD (BE3ORDER) are included in IMSVS.PRIMESRC. Their DBDGENs can be performed with job //SAMP310 in IMSVS.PRIMEJOB.




```
*
*                             DESCRIPTION OF SECONDARY INDEX
*                             INTO THE PARTS DATABASE
*                             FOR PRIMER SAMPLE PROJECT PHASE 3
*                             ---------------------------------------
*
    DBD      NAME=BE3PSID1,ACCESS=INDEX
*
    DATASET DD1=DE3PSID1,DEVICE=3330,MODEL=1,SIZE=2048
*
    SEGM     NAME=SE3PSID1,BYTES=12
    LCHILD   NAME=(SE1PART,BE3PARTS),INDEX=FE3PSID1,PTR=SNGL
    FIELD    NAME=(FE3PSXD1,SEQ),BYTES=12,START=01
*
    DBDGEN
    FINISH
    END
```

```
*
*                             DESCRIPTION OF PARTS DATABASE
*                             FOR PRIMER SAMPLE PROJECT PHASE 3
*                             ---------------------------------------
*
    DBD      NAME=BE3PARTS,ACCESS=(HDAM,VSAM),                        *
             RMNAME=(DFSHDC40,4,80,500)
*
    DATASET DD1=DE3PARTS,DEVICE=3330,MODEL=1,SIZE=2048
*
*                             PARTS- GENERAL INFORMATION (ROOT)
*                             ---------------------------------------
    SEGM     NAME=SE1PART,BYTES=80,PTR=T,RULES=PLV
*
    FIELD    START=01,BYTES=08,TYPE=C,NAME=(FE1PGPNR,SEQ)
      •
      •
      •
*                             CONNECTION TO CUSTOMER-ORDER DB.:
    LCHILD   NAME=(SE2ODETL,BE3ORDER),PAIR=SE2PAROR,PTR=DBLE
*                             CONNECTION WITH 2ND INDEX :
    LCHILD   NAME=(SE3PSID1,BE3PSID1),PTR=INDX
    XDFLD    NAME=FE3PSID1,SEGMENT=SE1PPUR,SRCH=FE1PPONR,             *
             SUBSEQ=/SXPPUR
      •
      •
      •
*                             PARTS- PURCHASE ORDER INFORMATION
*                             ---------------------------------------
    SEGM     NAME=SE1PPUR,BYTES=60,PARENT=((SE1PART,SNGL)),PTR=T
    FIELD    START=01,BYTES=08,TYPE=C,NAME=(FE1PPONR,SEQ)
      •
      •
      •
    FIELD    START=01,BYTES=04,TYPE=C,NAME=/SXPPUR
      •
      •
      •
    DBDGEN
    FINISH
    END
```

Figure 2-29. Phase 3 Physical DBDs

PROGRAM SPECIFICATION BLOCK GENERATION (PSBGEN)

For each program which uses a DL/I data base, a program specification block (PSB) is needed. Although one PSB can serve different batch application programs, it is recommended, for integrity purposes, that each program have its own PSB. In the online IMS/VS system, a Separate PSB is required for each online program. Each PSB consists of one or more program communication blocks (PCBs), one for each data base the program uses.

The PSB is generated, as shown in Figure 2-30, in a similar manner to the DBD using the OS/VS assembler and linkage editor. The generated load module is stored in IMSVS.PSBLIB.

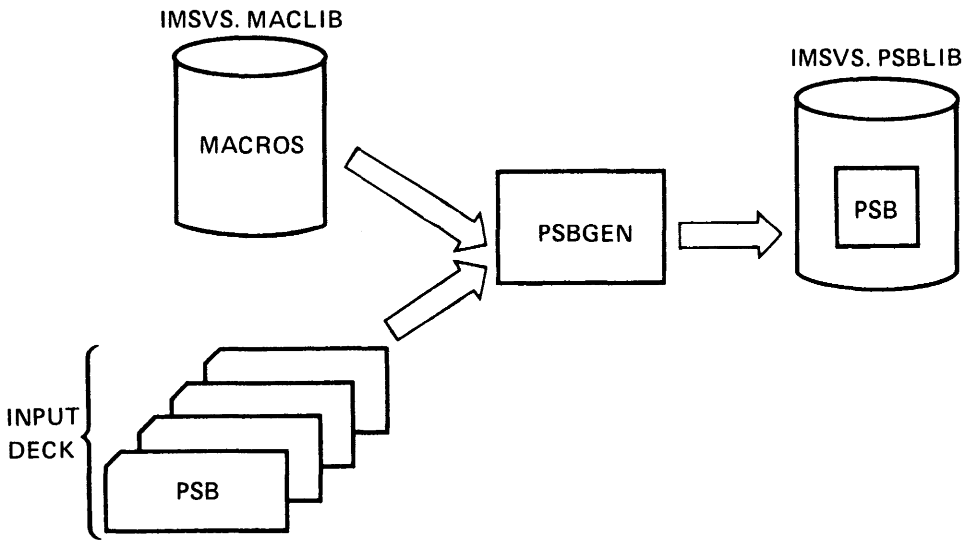


Figure 2-30. Program Specification Block Generation (PSBGEN)

Figure 2-31 shows the sequence of the macro statements in the PSBGEN input deck.

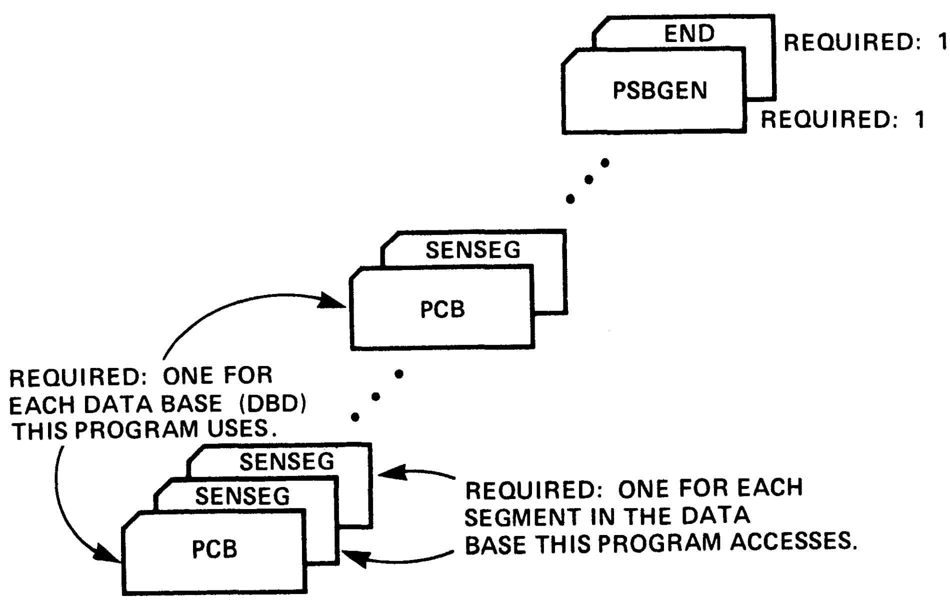


Figure 2-31. PSBGEN Input Deck Structure

The PSBGEN is executed by invoking a JCL cataloged procedure named PSBGEN, which is available in the IMSVS.PROCLIB.

The coding conventions for the PSB are exactly the same as for the DBD.

BASIC PSB CODING

Following are the basic PSB control statement formats.

PCB Statement

This statement is coded once for each data base the program intends to use. The format is:

```
|          |PCB     |TYPE=DB
|          |        |,DBDNAME=dbdname
|          |        |
|          |        |           (       A       )  [P]
|          |        |,PROCOPT=  <[G][I][R][D]   >  [P]
|          |        |           (       L       )  [S]
|          |        |
|          |        |,KEYLEN=value
|          |        |
```

Legend:

TYPE=DB

is a required keyword parameter for all data base PCBs.

DBDNAME=

specifies the name of the DBD which is accessed via this PCB. It can be a physical or logical DBD.

PROCOPT=

specifies the processing options on sensitive segments declared in this PCB that may be used in an associated application program. Specifying superfluous processing options is undesirable from a data base security point of view and can result in unnecessary additional data base processing. This operand allows a maximum of four characters. The letters in the operand have the following meanings:

G - Get function.

I - Insert function.

R - Replace function.

D - Delete function.

Note: The functions above can be coded in any combination of three; if all four are required, code "A".

A - All, includes the above four functions.

P - Required if command code D (path call) is to be used on get type calls or insert calls. Determines maximum length of the I/O area. P cannot be coded with L.

L - Load function for data base loading (except HIDAM).

LS- Segments loaded in ascending sequence only (HIDAM, HDAM). This load option is required for HIDAM.

KEYLEN=value

is the value specified in bytes of the longest concatenated key for a hierarchical path of sensitive segments used by the application program in the hierarchical data structure.

GSAM PCB: The format for the GSAM data base PCB statement is:

```
 /-------------------------------------------------------------------
/          |          |                                              |
|          |PCB       |TYPE=GSAM                                     |
|          |          |,DBDNAME=name,PROCOPT= {G[ S]}                |
|          |          |                       {L[ S]}                |
|          |          |                                              |
---------------------------------------------------------------------
```

where:

TYPE=GSAM

is a required keyword parameter for all GSAM data base PCBs.

DBDNAME=name

is a required keyword parameter for the name that specifies the GSAM DBD to be used as the primary source of data set description. SENSEG statements must not follow this PCB statement.

PROCOPT=

is a required parameter for the processing options on the data set declared in this PCB that can be used in an associated application program The operand is specified using the characters defined below:

G - Get function
L - Load function
S - Large scale sequential activity. If specified, GSAM will use multiple-buffering. This is recommended for heavy sequential processing.

Note: The GSAM PCB statements must follow the PCB statements with TYPE=TP or DB if any exist in the PSB generation. The convention is:

TP PCBs - first
DB PCBs - second
GSAM PCBs - last

SENSEG Statement

This statement is coded once for each segment the program is sensitive to in the DBD defined in the preceding PCB. The SENSEG statements should appear in the same hierarchical sequence as in the DBD. However only those segments should be included to which the program needs access. All segments should be specified in the hierarchical path to any required segment. No SENSEG statements should be coded for a GSAM PCB. The basic format of the SENSEG statement is:

```
 /----------------------------------------------------------------------
/        | SENSEG    | NAME=segname1
|        |           |
|        |           | ,PARENT=segname2
|        |           |
|        |           |          (        A      )   [P]
|        |           | [,PROCOPT=([G][I][R][D])   [P]]
|        |           |
-------------------------------------------------------------------------
```

Legend:

NAME=segname1

is the name of the segment type as defined through a SEGM statement during DBD generation. The field is from 1 to 8 alphameric characters.

PARENT=segname2

is the name of the segment type that is the parent of the segment type whose name is specified in the NAME operand. If this SENSEG statement defines a root segment type, this operand must equal zero. For all dependent segment types, this operand must specify the name of the dependent's parent.

PROCOPT=

specifies the processing options allowable on this sensitive segment by an associated application program. This operand has the same meaning as the PROCOPT operand on the PCB statement. If this PROCOPT operand is not specified, the PCB PROCOPT operand is used as default. If there is a difference in the processing options specified on the PCB and SENSEG statements, SENSEG PROCOPT overrides the PCB PROCOPT. When loading a data base, you should specify a PROCOPT only in the PCB statement.

PSBGEN Statement

This statement specifies the end of the PSB and provides interface parameters for the application program. It is the last statement before the END statement. The basic format is:

```
 /----------------------------------------------------------------------
/        |           |
|        |           |      (COBOL)
|        | PSBGEN    | LANG=( PL/I )
|        |           |      (ASSEM)
|        |           | ,CMPAT=YES
|        |           | ,PSBNAME=psbname
|        |           | ,IOEROPN=(451,WTOR)
|        |           |
-------------------------------------------------------------------------
```

LANG=

specifies the language in which the application program is written. It must be either COBOL, PL/I, or ASSEM, with no trailing blanks.

CMPAT=YES

should be selected, except for initial load programs. It provides an extra dummy PCB in the PSB. This benefits migration to online processing at negligible cost.

PSBNAME=psbname

is the parameter keyword for the alphameric name of this PSB. The name value for the PSBNAME must be eight characters or less in length. This name becomes the load module name for the PSB in the library IMSVS.PSBLIB. This name must be the same as the program load module name in the program library. No special characters may be used in the name. It must be the name in the MBR= operand of PSBGEN.

IOEROPN=(451,WTOR)

Should be coded as shown to concur with the recovery procedures of our subset. Whenever a read or write data base I/O error would occur during batch processing, the OS/VS system console operator will be notified (message DFS0451A).

The reply should be 'ABEND'; DL/I will then abend with a U451 abend code. The data set in error should then be recovered. See Chapter 6, "Data Base Recovery," for details. This parameter can be omitted when initially loading the data base.

Note: Before above reply is given, the operator should take proper actions to prevent the execution of any other DL/I jobs against the affected data bases. See Chapter 6, "Data Base Recovery," for details.

END Statement

This statement is required at the end of the PSB deck. It indicates the end of the input for the OS/VS assembler.

```
  /---------------------------------------------------------------------┐
 /         |          |                                                  |
|          |END       |                                                  |
|          |          |                                                  |
└------------------------------------------------------------------------┘
```

Sample Basic PSBs

Figure 2-32 shows two PSBs for the Phase 1 sample environment. The top one (PE1PARTS) is the PSB for loading the Phase 1 PARTS data base. This PSB can be generated with job //SAMP100 in IMSVS.PRIMEJOB. The second one (PE1CPPUR) is the one for the purchase order program. It also contains GSAM PCBs and it can be generated with job //SAMP101 (COBOL) or //SAMP102 (PL/I) in IMSVS.PRIMEJOB.

```
*                                PROGRAM SPECIFICATION FOR LOADING
*                                THE PHASE 1 PARTS DATABASE
         PCB     TYPE=DB,PROCOPT=L,
                 DBDNAME=BE1PARTS,KEYLEN=20
*
         SENSEG NAME=SE1PART
         SENSEG NAME=SE1PSTOK,PARENT=SE1PART
         SENSEG NAME=SE1PPUR,PARENT=SE1PART
         SENSEG NAME=SE1PGDSC,PAPENT=SE1PART
         PSBGEN LANG=ASSEM,PSBNAME=PE1PARTS
          END
```

```
*                                PROGRAM SPECIFICATION FOR
*                                PURCHASE-ORDER UPDATING OF
*                                THE PHASE 1 PARTS DATABASE
         PCB     TYPE=DB,PROCOPT=AP,
                 DBDNAME=BE1PARTS,KEYLEN=20
*
       SENSEG NAME=SE1PART,PROCOPT=GP
       SENSEG NAME=SE1PPUR,PROCOPT=AP,PARENT=SE1PART
*
          PCB    TYPE=GSAM,PROCOPT=G,
                 DBDNAME=B00INP01
          PCB    TYPE=GSAM,PPOCOPT=L,
                 DBDNAME=B000UT01
         PSBGEN LANG=COBOL,CMPAT=YES,PSBNAME=PE1CPPUR,IOEROPN=(451,WTOR)
          END
```

Figure 2-32. Sample PSBs for Phase 1

EXECUTION OF PSBGEN -- JCL

PSBGEN is run as a normal Operating System job after IMS/VS system definition. IMS/VS system definition causes the procedure named PSBGEN to be placed in the IMSVS.PROCLIB procedure library. The following JCL cards are used to invoke the PSBGEN procedure.

```
//PSBGEN   JOB  MSGLEVEL=1
//        EXEC  PSBGEN,MBR=
//C.SYSIN  DD   *

     PCB
     SENSEG    The control cards
     PSBGEN    for PSB generation.
     END

  *
```

where keyword operand MBR=

is the name of the PSB to be generated. This name must be the same as the name specified on the PSBNAME= operand of the PSBGEN statement.

CODING PSBs FOR LOGICAL DATA BASES

When a physical DBD contains logical relationships, the PCB and the application program can still refer to the physical DBD. However, this should be restricted to initial data base load programs. Remember also, the logical child always contains the logical parent's concatenated key. This should not be forgotten when inserting a logical child in a physical DBD. You can never access a virtual logical child in a physical data base, since it does not exist.

To use a logical data base, the program needs a separate PCB. This PCB is coded in the same manner as a PCB for a physical DBD. The only difference is that it refers to the DBD name and SEGMENT names of a logical DBD. You should only code SENSEG statements for the segments the program actually needs and the segments in the hierarchical path to those segments. All of this is based on the logical DBD, so the hierarchical path may well include physical and logical paths. Figure 2-33 shows the PSB for the Phase 2 processing program PE2CORDR, containing a PCB for both the logical data bases in addition to a PCB for the SHISAM data base. This PSB is listed in IMSVS.PRIMESRC, its PSBGEN can be performed with job //SAMP201 (COBOL) or //SAMP202 (PL/I) in IMSVS.PRIMEJOB.

```
*                          PROGRAM SPECIFICATION BLOCK FOR PHASE 2
*                          ORDER UPDATE PROGRAM PE2CORDR.
*
*                      CUSTOMER DATABASE VIEW
*
        PCB    TYPE=DB,DBDNAME=BE2PCUST,PROCOPT=G,KEYLEN=6
       SENSEG NAME=SE2PCUST
*
*                      ORDER DATABASE VIEW
*
        PCB    TYPE=DB,DBDNAME=BE2LORDR,KEYLEN=14
       SENSEG NAME=SE2ORDER,PROCOPT=AP
       SENSEG NAME=SE2OPART,PARENT=SE2ORDER,PROCOPT=A
       SENSEG NAME=SE2OSHIP,PARENT=SE2ORDER,PROCOPT=GI
*
*                      PARTS DATABASE VIEW
*
        PCB    TYPE=DB,DBDNAME=BE2LPART,KEYLEN=20
       SENSEG NAME=SE1PART,PROCOPT=GRP
       SENSEG NAME=SE1PSTOK,PARENT=SE1PART,PROCOPT=GR
*
       PSBGEN LANG=COBOL,CMPAT=YES,PSBNAME=PE2CORDR,IOEROPN=(451,WTOR)
        END
```

Figure 2-33. Sample PSB for Phase 2

CODING PSBs FOR SECONDARY INDEXES

To use a secondary index, an application program should use a PCB with the following additional parameter in the PCB statement.

<u>The PCB Statement</u>

```
   /----------------------------------------------------------------
  /         |          |                                           |
 |          |PCB       |TYPE=DB,...    ,PROCSEQ=indxdbname         |
 |          |          |                                           |
 ------------------------------------------------------------------
```

PROCSEQ=indxdbname

specifies the name of the secondary index used to process the data base named in the DBDNAME operand through a secondary processing sequence. The operand is invalid if PROCOPT=L or LS.

<u>Notes</u>:

1. The DBD specified in the PCB for the secondary processing sequence can be a logical DBD. No provisions are necessary in the logical DBD, but its root segment must be the target segment of the physical DBD.

2. If non-unique index fields are used, you must specify of the /SX field in our subset. As a consequence, the sequence of root segments with the same index field value, when sequentially retrieved via the secondary index, will be unpredictable. This sequence will also vary across reorganization of the target data base.

Figure 2-34 shows the PSB for the Phase 3 processing program, PE3CPPUR. This PSB contains a PCB for the normal processing sequences and a PCB for the secondary processing sequence.

```
*                           PROGPAM SPECIFICATION FOR
*                           PURCHASE-ORDER UPDATING OF
*                           THE PHASE 3 PARTS DATABASE
*
*                          PRIMARY INDEX VIEW OF DATABASES
        PCB     TYPE=DB,PROCOPT=AP,                                      *
                DBDNAME=BE3PARTS,KEYLEN=20
*
      SENSEG NAME=SE1PART,PROCOPT=GP
      SENSEG NAME=SE1PPUR,PROCOPT=AP,PARENT=SE1PART
*
*                          SECONDARY INDEX VIEW OF DATABASES
        PCB     TYPE=DB,PROCOPT=GP,DBDNAME=BE3PARTS,KEYLEN=16,           *
                PROCSEQ=BE3PSID1
*
      SENSEG NAME=SE1PART
      SENSEG NAME=SE1PPUR,PARENT=SE1PART
*
         PCB    TYPE=GSAM,PROCOPT=G,                                     *
                DBDNAME=B00INP01
         PCB    TYPE=GSAM,PROCOPT=L,                                     *
                DBDNAME=B00OUT01
        PSBGEN LANG=COBOL,CMPAT=YES,PSBNAME=PE3CPPUR,IOEROPN=(451,WTOR)
         END
```

Figure 2-34. Sample Phase 3 PSB

THE DATA BASE DESIGN PROCESS

The process of data base design in its simplest form can be described as: The structuring of the data elements for the various applications in such an order that:

- Each data element is readily available by the various applications, now and in the foreseeable future.
- The data elements are efficiently stored on secondary storage.
- Controlled access is enforced for those data elements with specific security requirements.

In practice, one is often forced to compromise, based on available resources in manpower, hardware and software.

CONCEPTS OF DATA BASE DESIGN

Because data base design is an area where there has been little formal standardization, there has been no consistent vocabulary for describing the concepts involved. This section presents the concepts and terms used in the following introductory data base design discussion.

Entities

A data base contains information about entities. An *entity* is something that:

- Can be uniquely identified.
- We may now or in the future collect substantial information about.

In practice this definition is limited to the context of the applications under consideration. Examples of entities are: parts, projects, orders, customers, trucks, etc. It should be clear that defining entities is a major step in the data base design process. The information we store in data bases about entities is described by data elements.

Data Elements

A *data element* is a unit of information that specifies a fact about an entity. For example, suppose the entity is a part. Name=Washer, Color=Green, and Weight=143 are three facts about that part. Thus these are three data elements. A data element has a name and a value. A data element *name* tells the kind of fact being recorded; the *value* is the fact itself. In the above example, Name, Color, and Weight are data element names; Washer, Green and 143 are values. A value must be associated with a name to have a meaning.

An *occurrence* is the value of a data element for a particular entity. Figure 2-35 illustrates the concepts of data elements and their occurrences in recording the facts about two entities, parts (Entity A) and orders (Entity B).

ENTITY A: PARTS		
DATA ELEMENT	OCCURRENCES	
Name	Value	Value
Part Number	03003710	03003720
Name	Screw	Washer
Unit Price	$3.00	$1.00
Unit Quantity	100 pieces	100 pieces
Stock Quantity	2000	3000
ENTITY B: ORDERS		
DATA ELEMENT	OCCURRENCES	
Name	Value	Value
Order Number	190F60	190F60
Part Name	Screw	Bolt
Part Number	03003710	03003730
Quantity	500 units	300 units
Supplier Name	Allied Screw	Allied Screw
Order Code	A	X

Figure 2-35. Concepts of Data Elements

Quite often, data elements which add information to an entity are called _attributes_. An attribute is always dependent on an entity. It has no meaning by itself. Depending on its usage, an entity can be described by one single data element or more. Ideally, an entity should be uniquely defined by one single data element, for example, the order number of an order. Such a data element is called _the key_ of the entity. The key serves as the identification of a particular entity occurrence, and is a special attribute of the entity. Keys are not always unique. In such cases, entities with equal key values are called _synonyms_. For instance, the full name of a person is generally not a unique identification. In such cases we have to rely on other attributes such as full address, birthday or an arbitrary sequence number. A more common method is to define a new attribute, which serves as the unique key, for example, employee number.

The Transaction

Data in itself is not the ultimate goal of a data base management system. It is the application function performed on the data which is important. The best way to represent that function is the _transaction_, which is the smallest application unit representing a user interacting with the data base. For example, one single order, one part inventory status.

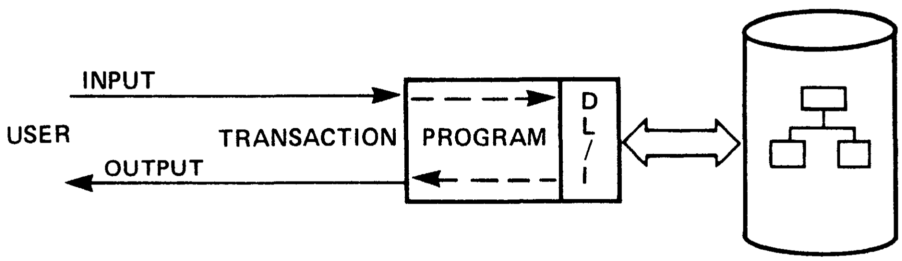


Figure 2-36. The Transaction

Transactions are processed by application programs. In a batch system, large numbers of transactions are accumulated (that is, all orders of a day), then processed against the data base with a single scheduling of the desired application program. Although transactions are always distinguishable, even in batch, some people prefer to talk about programs rather than transactions. But, especially in a DB/DC environment, a clear understanding of transactions is mandatory for good design. The transaction is in some way the individual usage of the application by a particular user. As such, it is the focal point of the DB/DC system.

In this chapter we will utilize the transaction for the data base design. A similar role is set aside for the transaction in program design by adding detailed input, processing and output descriptions to the data element usage.

Access Paths

Each transaction bears in its input some kind of identification with respect to the entities used (for example, the part number when accessing a Parts data base). These are referred to as the _access paths_ of that transaction. In general, transactions require random access, although for performance reasons sequential access is sometimes used. This is particularly true if the transactions are batched and they are numerous, relative to the data base size, or if information is needed from most data base records.

For efficient random access, each access path should utilize the entity's key. With proper data base design, DL/I generally provides fast physical access via a key. Therefore, identification of the transaction access path is essential for a design to yield good performance.

The Transaction/Data Element Matrix

A convenient way to specify the transactions, the data element and their interaction is the *transaction/data element matrix*, Figure 2-37.

ENTITY	DATA ELEMENTS	APPLICATION: PARTS INVENTORY		PURCHASE ORDERS		CUSTOMER ORDERS		
	TRANSACTIONS	PARTS REPORT	PART INQUIRY	NEW ORDER	CHANGE ORDER	NEW ORDER	CHANGE ORDER	DELETE ORDER
PART	PART NAME	R	R	R	R	R	R	
	PART NUMBER	R	$\boxed{R}$	R	R	R	R	
	STOCK LOCATION	R	R	U	U	R	R	
CUSTOMER	CUSTOMER NAME					R		
	CUSTOMER NUMBER					$\boxed{R}$		
	ORDER NUMBER					I		
CUSTOMER ORDER	PART NUMBER					I	U	D
	CUSTOMER NUMBER					I	R	D
	PART QUANTITY					I	U	D
	ORDER NUMBER					$\boxed{I}$	$\boxed{R}$	$\boxed{D}$

Legend: □ DIRECT ACCESS PATH (KEY)
○ SEQUENTIAL ACCESS PATH

Figure 2-37. The Transaction/Data Element Matrix

The transaction/data element matrix specifies, in its simplest form, the processing intent of the application transactions against the data base elements:

- Retrieve; read only R
- Update in place U
- Add, insert I
- Delete D
- All of above A
- Null, not sensitive - or blank

The data elements which are direct access paths for a transaction are denoted by a boxed matrix item. These should be keys. Sequential access is indicated by a circle around the matrix item.

THE DATA BASE DESIGN TASKS

The process of designing a data base (Figure 2-38) can be generally divided into the following tasks:

- Gathering requirements
- Designing application data structures
- Designing physical data structures
- Design evaluation

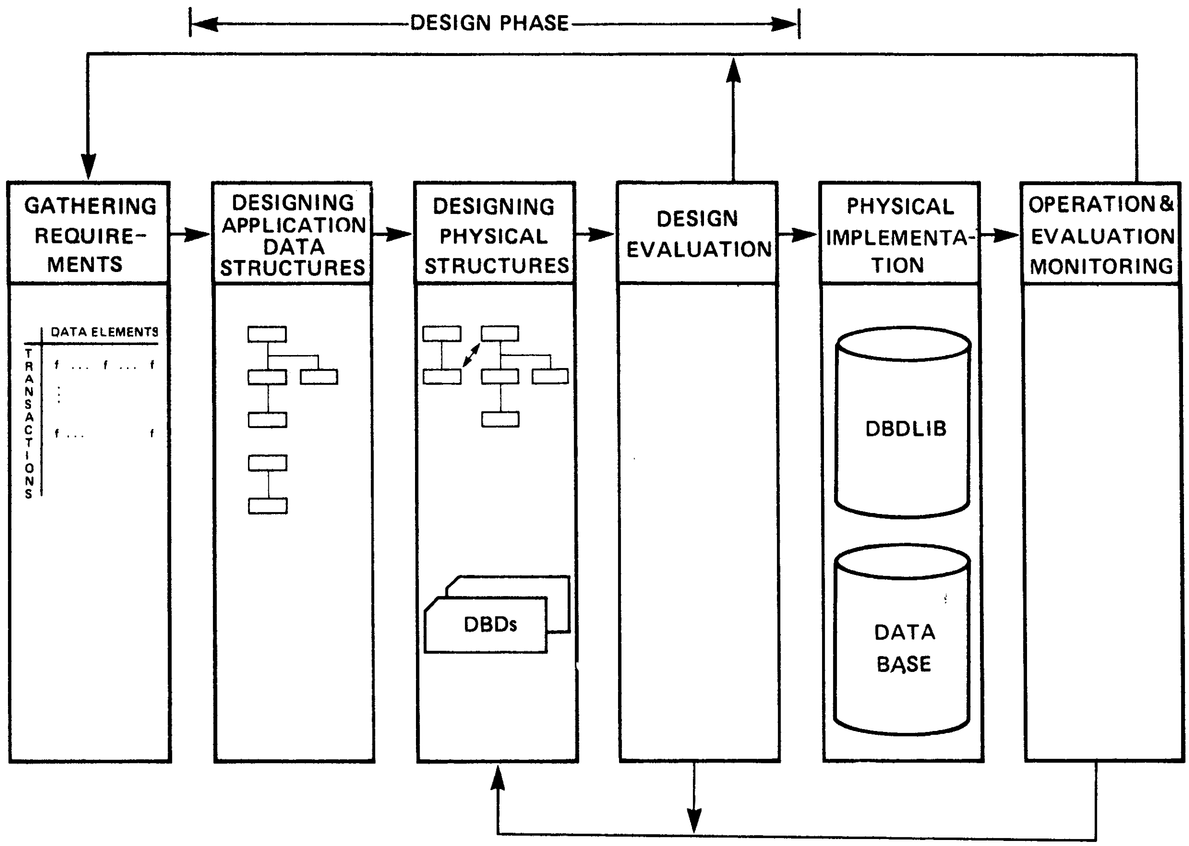


Figure 2-38. The Steps in Data Base Design

Usually the above steps are repeated until the design satisfies the requirements. After this design process, the actual development, implementation (data base load) and production begins. During production, the system is subject to monitoring which can give feedback for the design phase. This will be discussed in Chapter 9, "Optimization".

GATHERING REQUIREMENTS

The first step of the data base design poses many questions: What do the applications need? What inputs are required to drive them? What data outputs will they produce? How are the data elements related to one another? Which elements are identifiers and which elements do they identify? How frequently are they used? Have input sources been specified for all data elements?

During the process of gathering requirements, these and related questions are answered primarily during conversations between a data base designer and an analyst from the department that requests an application. In some organizations, a set of forms appropriately filled in marks the end of the requirements gathering step; in other organizations, less formality is involved. In any case, this first step in data base design ends when the designer collects the data needs of the individual applications that will use the data base being designed.

The requirements for a data base should contain:

- The data being managed, that is, the entities and associated data elements
- The relations between the entities and data elements as needed by the various users
- The functions being performed against the data, that is, the transactions
- The access path as required by the transactions

The first step in gathering the requirements is to determine the entities. This is not a trivial task, because the choice of entities is dependent on the environment.

A data element which, initially, is considered an attribute, could become an entity itself when new applications are added. For instance, the data element color is normally seen as an attribute. But in a paint factory process it might very well be an entity itself. It should be clear that the change of a given data element from attribute to entity could have a significant impact on the data structure. To avoid this as much as possible, one should be very careful in the choice of entities.

To register the functions performed against the data elements, first construct the transaction/data element matrix. Optionally when the matrix becomes too large, one can construct a separate matrix for each major application. Another useful approach is to make a large drawing for display on the wall. This process is most effective if the matrix not only contains the applications of the immediate future, but also as much as possible about future applications and data elements.

Additional columns could be added for miscellaneous information such as:

- Occurrence frequencies of transactions and data elements
- Size and format of data elements
- Priorities and response/turnaround time criteria
- Availability (how long can the function be suspended)
- Security (who may have access to the information made available by this transaction)

- Input/output descriptions per transaction, for application program design

The transaction/data element matrix, together with a detailed description of the data base and its use, constitutes the requirements for the design step. For the detailed description of the data base, its segments and fields, a documentation scheme should be established. As a minimum, forms should be used for a manual registration of the data base, the segment layout, the fields and their attributes. It is very important to register which program uses which data elements. The next step would be to use the Assembler DSECT, COBOL COPY, or PL/I %INCLUDE facility for centralized management of segment descriptions. Ultimately, a data dictionary system might be utilized.

For each phase of our sample environment, we can now construct a transaction/data element matrix.

Phase 1 Transaction/Data Element Matrix

The phase 1 transaction/data element matrix is shown in Figure 2-39. It is clear the main entity is parts. Other possible entities could be purchase order, supplier and stock location. However, we assume no need to gather more information on these in our Phase 1 sample environment.

Notice, the following information is added to the transaction data element matrix:

- For each data element, we list its size and its occurrence per entity. 0 - 4 means that this data element occurs a minimum of zero times, and a maximum of four.

- For each transaction, we list its average frequency in weeks (W) or days (D).

Phase 2 Transaction/Data Element Matrix

In the phase 2 environment, we add the Customer Order Processing application. This extends the phase 1 transaction/data element matrix of Figure 2-39 to the one shown in Figure 2-40. Essential here is that, besides adding new data elements for the customer order processing, this new application also requires the existing PARTS data elements.

Also notice that the part number data element appears beneath both the PARTS and the CUSTOMER ORDERS entities. This constitutes the basic requirement for a linkage or relation between these entities as we will see later.

Phase 3 Transaction/Data Element Matrix

In the phase 3 environment, we added the purchase order inquiry transaction, TE3POINQ. This transaction requires a direct and a sequential access path to the purchase order information based on the purchase order number. This is because we want to be able to list an individual purchase order, or a range of purchase orders in their order number sequence. See Figure 2-41. In practice, this access path could also be used for the purchase order change (TE1POCNG) and delete (TE1PODEL) transactions.

ENTITY	OCCURRENCE	SIZE	APPLICATION / FREQUENCY / TRANSACTIONS / DATA ELEMENTS	PARTS INV		PURCHASE ORDER			
				TE1INVRP 1W	TE1INVQU 2D	TE1PONEW 100D	TE1POCNG 10D	TE1PODEL 100D	
PARTS	0.3	50	FE1PGDSC	R					
	1	13	FE1PGSNM	R	R	R	R		
	1	8	FE1PGPNR	R	$\boxed{R}$	$\boxed{R}$	$\boxed{R}$	$\boxed{R}$	
	1	8	FE1PGUNT	R	R	R	R	R	
	1	8	FE1PGPRI	R	R	R	R	R	
	1	8	FE1PGDIM	R	R	R	R	R	
	1-6	12	FE1PSLOC	R	R				
	1-6	6	FE1PSCNT	R	R				
	1-6	6	FE1PSDAT	R	R				
	1-6	6	FE1PSISS	R	R				
	1-6	6	FE1PSREC	R	R			U	
	0-4	20	FE1PPOSU	R	R	I	U	D	
	0-4	6	FE1PPQOD	R	R	I	U	D	
	0-4	6	FE1PPQRD	R	R	I	U	D	
	0-4	6	FE1PPODT	R	R	I	U	D	
	0-4	6	FE1PPDDT	R	R	I	U	D	
	0-4	8	FE1PPONR	R	R	I	R	D	
	1	8	FE1PGNEW	R	R				
	1	8	FE1PGOLD	R	R				
	1	8	FE1PGEQV	R	R				

Legend: $\boxed{}$ DIRECT ACCESS PATH (KEY)
○ SEQUENTIAL ACCESS PATH

Figure 2-39. Transaction/Data Element Matrix for Phase 1

ENTITY	OCCURRENCE	SIZE	DATA ELEMENTS	PARTS INV: TE1INVRP 1W	PARTS INV: TE1INVQU 2D	PURCHASE ORDER: TE1PONEW 100D	PURCHASE ORDER: TE1POCNG 10D	PURCHASE ORDER: TE1PODEL 100D	CUSTOMER ORDER: TE1CONEW 500D	CUSTOMER ORDER: TE1COCNG 600D	CUSTOMER ORDER: TE1CODEL 500D
PARTS	0.3	50	FE1PGDSC	R							
	1	13	FE1PGSNM	R	R	R	R		R	R	
	1	8	FE1PGPNR	R	$\boxed{R}$	$\boxed{R}$	$\boxed{R}$	$\boxed{R}$	$\boxed{R}$	$\boxed{R}$	
	1	8	FE1PGUNT	R	R	R	R	R	R	R	
	1	8	FE1PGPRI	R	R	R	R	R	R	R	
	1	8	FE1PGDIM	R	R	R	R	R			
	1-6	12	FE1PSLOC	R	R				R	R	
	1-6	6	FE1PSCNT	R	R				R	R	
	1-6	6	FE1PSDAT	R	R						
	1-6	6	FE1PSISS	R	R				U	J	
	1-6	6	FE1PSREC	R	R			U	R	R	
	0-4	20	FE1PPOSU	R	R	I	U	D			
	0-4	6	FE1PPQOD	R	R	I	U	D			
	0-4	6	FE1PPQRD	R	R	I	U	D			
	0-4	6	FE1PPODT	R	R	I	U	D			
	0-4	6	FE1PPDDT	R	R	I	U	D			
	0-4	8	FE1PPONR	R	R	I	R	D			
	1	8	FE1PGNEW	R	R						
	1	8	FE1PGOLD	R	R						
	1	8	FE1PGEQV	R	R				R	R	
CUSTOMER	1	6	FE2PCNUM						$\boxed{R}$	$\boxed{R}$	
	1	20	FE2PCNAM						R	R	
	1	20	FE2PCADR						R	R	
	1	20	FE2PCCTY						R	R	
	1	6	FE2PCPCD						R	R	
CUSTOMER ORDERS	1	6	FE2OGREF						$\boxed{I}$	$\boxed{R}$	$\boxed{D}$
	1	2	FE2OGSTA						I	U	D
	1	6	FE2OGODT						I	U	D
	1	6	FE2OGDDT						I	U	D
	1	2	FE2OGDWK						I	U	D
	0.1	20	FE2OGSPC						I	U	D
	1	2	FE2OGORI						I	U	D
	1-8	6	FE2ODQTY						I	U	D
	1-8	8	FE2ODPRI						I	U	D
	1-8	1	FE2ODTAX						I	U	D
	0.1	8	FE2OSNR						I	U	D
	0.1	6	FE2OSDAT						I	U	D
	0.1	20	FE2OSMET						I	U	D
	1-8	1	FE2ODBOR						I	U	D
	1	6	FE2OGCNR						I	R	D
	1-8	8	FE2ODPNR						I	U	D

Legend: $\boxed{}$ DIRECT ACCESS PATH (KEY)
◯ SEQUENTIAL ACCESS PATH

Figure 2-40. Transaction/Data Element Matrix for Phase 2

ENTITY	OCCURRENCE	SIZE	DATA ELEMENTS	PARTS INV: TE1INVRP (1W)	PARTS INV: TE1INVQU (2D)	PURCHASE ORDER: TE1PONEW (100D)	PURCHASE ORDER: TE1POCNG (10D)	PURCHASE ORDER: TE1PODEL (100D)	PURCHASE ORDER: TE3POINQ	CUSTOMER ORDER: TE1CONEW	CUSTOMER ORDER: TE1COCNG	CUSTOMER ORDER: TE1CODEL
PARTS	0.3	50	FE1PGDSC	R								
	1	13	FE1PGSNM	R	R	R	R		R	R	R	
	1	8	FE1PGPNR	R	$\boxed{R}$	$\boxed{R}$	$\boxed{R}$	$\boxed{R}$	R	$\boxed{R}$	$\boxed{R}$	
	1	8	FE1PGUNT	R	R	R	R	R		R	R	
	1	8	FE1PGPRI	R	R	R	R	R		R	R	
	1	8	FE1PGDIM	R	R	R	R	R				
	1-6	12	FE1PSLOC	R	R					R	R	
	1-6	6	FE1PSCNT	R	R					R	R	
	1-6	6	FE1PSDAT	R	R							
	1-6	6	FE1PSISS	R	R					U	U	
	1-6	6	FE1PSREC	R	R			U		R	R	
	0-4	20	FE1PPOSU	R	R	I	U	D	R			
	0-4	6	FE1PPQOD	R	R	I	U	D	R			
	0-4	6	FE1PPQRD	R	R	I	U	D	R			
	0-4	6	FE1PPODT	R	R	I	U	D	R			
	0-4	6	FE1PPDDT	R	R	I	U	D	R			
	0-4	8	FE1PPONR	R	R	I	R	D	$\boxed{R}$			
	1	8	FE1PGNEW	R	R							
	1	8	FE1PGOLD	R	R							
	1	8	FE1PGEQV	R	R					R	R	
CUSTOMER	1	6	FE2PCNUM							$\boxed{R}$	$\boxed{R}$	$\boxed{D}$
	1	20	FE2PCNAM							R	R	D
	1	20	FE2PCADR							R	R	D
	1	20	FE2PCCTY							R	R	D
	1	6	FE2PCPCD							R	R	D
CUSTOMER ORDERS	1	6	FE2PGREF							$\boxed{I}$	$\boxed{R}$	D
	1	2	FE2OGSTA							I	U	D
	1	6	FE2OGODT							I	U	D
	1	6	FE2OGDDT							I	U	D
	1	2	FE2OGDWK							I	U	D
	0.1	20	FE2OGSPC							I	U	D
	1	2	FE2OGORT							I	U	D
	1	6	FE2ODQTY							I	U	D
	1	8	FE2ODPRI							I	U	D
	1	1	FE2ODTAX							I	U	D
	0.1	8	FE2OSNR							I	U	D
	0.1	6	FE2OSDAT							I	U	D
	0.1	20	FE2OSMET							I	U	D
	1-8	1	FE2ODBOR							I	U	D
	1	6	FE2OGCNR							I	R	D
	1-8	8	FE2ODPNR							I	U	D

Legend: $\boxed{}$ DIRECT ACCESS PATH (KEY)
◯ SEQUENTIAL ACCESS PATH

Figure 2-41. Transaction/Data Element Matrix for Phase 3

DESIGN THE APPLICATION DATA STRUCTURE

The data elements can now be arranged in an application data structure, which consists of one or more hierarchical data structures. We always construct hierarchical data structures based on the transaction/data element matrix; that is, the way the application program views it.

Segment Grouping

For each transaction, we start with the access path of that transaction to the entity, and construct a desired hierarchical view for that transaction. If more than one entity is accessed in one transaction, multiple hierarchical structures are required for that transaction. For each hierarchical structure, we try to group the needed data elements in the same type of segments. Each root segment of such a basic structure contains the key field which is used in the access path. If multiple key fields (for example, part number and stock number) are used in one access path, these may become the sequence fields of a parent/child combination.

The first field in the root segment is the key; the sequence field. To the root segment are added those data elements which are of a general nature, frequently used and/or compact, and occur once (or maximum, perhaps 3 times) per entity.

Next we group those data elements together in segments which belong logically to each other, based on their nature and use. Likely candidates for separate segments are those data elements which have multiple occurrences for a given root. The final result of the logical structure design step is a set of hierarchical data structures. These represent the view of the data by the different application programs, the application data structure.

Phase 1 Application Data Structure

Based on the transaction/data element matrix of Figure 2-39 and above guidelines for designing application data structures, we construct the following structure for the phase 1 Parts data base (Figure 2-42).

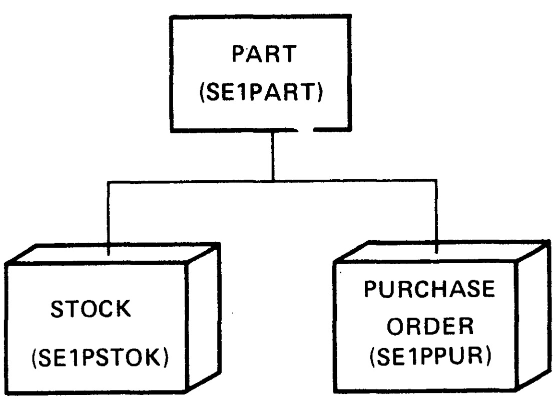


Figure 2-42. Phase 1 Application Data Structure

Access Paths

Sequential access is needed via the part short name, FE1PGSNM and direct access is needed via the part number FE1PGPNR. We can, however, process the TE1INVRP transaction in part number sequence and then sort the output in part short name sequence if needed. Direct access via part number is very important for later online processing.

The Root Segment, SE1PART

The rootkey is the part number FE1PGPNR field. The next step is to add the following fields to the root segment because they are of general use and occur for each part only once:

Name	Description	Length
FE1PGPNR	Part Number Code	8
FE1PGSNM	Part Description - Short Name	13
FE1PGNEW	New (Superseding) Part No.	8
FE1PGOLD	Old (Superseded) Part No.	8
FE1PGEQV	Equivalent part No.	8
FE1PGUNT	Unit of Measure	8
FE1PGPRI	Price	8
FE1PGDIM	Dimensions	8
FE1PGLNM	Part Name (Long Description)	50
	SEGMENT LENGTH	119

We define separate segments for stock and purchase order data elements because each can have multiple occurrences for each part and they are used separately.

The Stock Segment, SE1PSTOK

This segment has 1-6 occurrences for each part:

Name	Description	Length
FE1PSLOC	Stock Physical Location Code	12
FE1PSDAT	Stock Physical Count Date (MMDDYY)	6
FE1PSCNT	Stock Physical Count Quantity (TALLY)	6
FE1PSISS	Stock Total Issues Current Period	6
FE1PSREC	Stock Total Receipts Current Period	6
	SEGMENT LENGTH	36

The Purchase Order Segment, SE1PPUR

Name	Description	Length
FE1PPONR	Purchase Order Number	8
FE1PPODT	Purchase Order Date MMDDYY	6
FE1PPCSD	Supplier's Name	20
FE1PPQOD	Quantity Ordered	6
FE1PPQRD	Quantity Received	6
FE1PPDDT	Delivery Date MMDDYY	6
	SEGMENT LENGTH	52

The above application data structure of the Phase 1 Parts data base, will be input for the physical data base design in the next design step.

Phase 2 Application Data Structure

To support the Phase 2 transaction/data element matrix of Figure 2-40, we need two hierarchical structures in addition to the one shown in Figure 2-42. The result is shown in Figure 2-43. The design of the segments in the new hierarchical structures is done similar to the design of the Phase 1 Parts data structure.

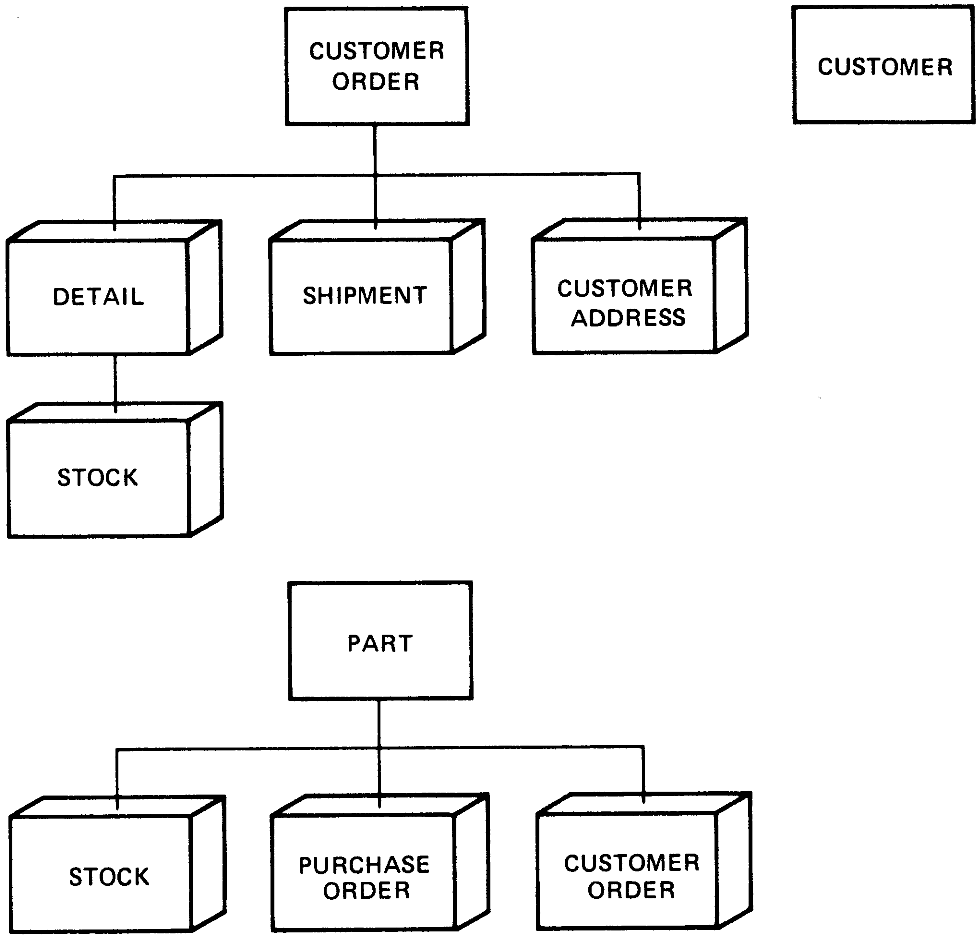


Figure 2-43. Phase 2 Application Data Structure

The following considerations apply:

- The hierarchical data structure PARTS is extended with a CUSTOMER ORDER segment. This provides the customer order per part relation.
- Several segments appear in different structures. They also vary in their data element content. This is essentially data redundancy, which will be addressed in the physical design step. At this time, however, we are mainly interested in the data structure as needed by the transactions.

Note: In your situation, these structures could be far more complex. For instance, the Customer data structure could have separate segments for accounts receivable, marketing statistics, etc. The PARTS structure could have a component and assembly structure. This is not addressed in our sample but can be easily implemented with DL/I.

Phase 3 Application Data Structure

The essential additional requirement of Phase 3 (see its transaction/data element matrix on Figure 2-41) is the need for access to the part and purchase order data elements via the purchase order number. This is reflected in the Phase 3 application data structure in Figure 2-44.

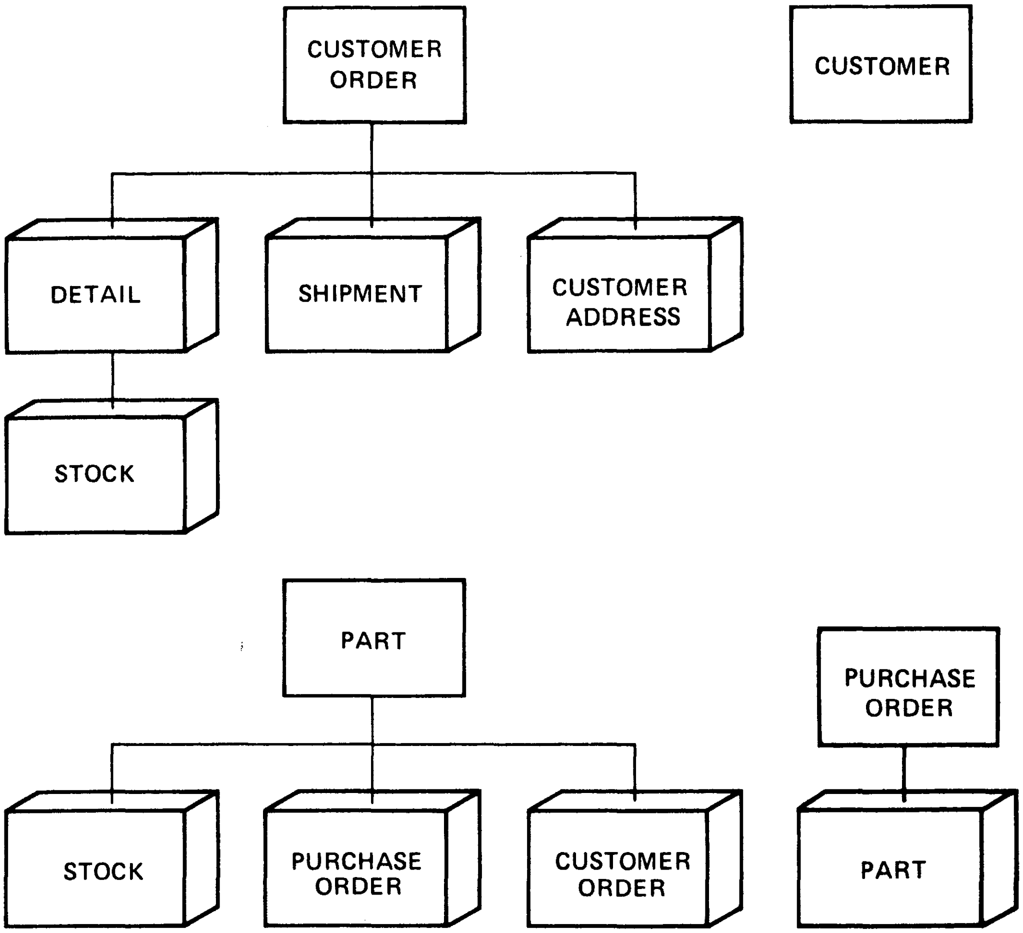


Figure 2-44. Phase 3 Application Data Structure

DESIGN THE PHYSICAL DATA STRUCTURES

In this step, the logical structures are matched against the functions and characteristics of DL/I. Physical data base structures are defined and specified in DBDGEN control statements. The DL/I storage organization and OS/VS access method are selected. Additional considerations may yield changes in the segment design. See Figure 2-45.

GROUP IN ONE SEGMENT <------>	SEPARATE SEGMENTS
Few occurrences (<3)	Multiple occurrences (>10)
Small (<20 bytes)	Large (>100 bytes)
High use (every access to record)	Low use (once a month)
Read-only	Update, Insert, Delete
General use	Secured use
Only dependent upon a single data element	Dependent upon relation of data elements

Figure 2-45. Grouping Data Elements into Physical Segments

The numbers shown in Figure 2-45 are not fixed. They merely provide a basis for your own estimates. Additional considerations:

- Single versus multiple occurrences. If a data element has a high number of occurrences, it is likely to be a segment itself, especially if it is large. If it is small and highly used, then all its occurrences could be stored in the same segment. However, the occurrence control would then be the responsibility of the application program, as DL/I itself does not provide for multiple occurrences of the name field in a segment.

- A very large segment can have a negative impact on DL/I's management of space on direct access devices. So the basic rule is: "Try to keep them more or less the same size".

- If a data element needs special security (that is, only particular applications may have access to it), it can be stored in a separate segment together with other data elements with the same security requirements.

The final result of the physical structure design steps is the data base descriptions (DBDs) and program specification blocks (PSBs) for the data bases and their processing programs.

Phase 1 Physical Data Base Design

We will now match our requirements as specified in the application data structure of Figure 2-42 and the transaction/data element matrix of Figure 2-39 with the available DL/I functions as presented earlier in this chapter. The outcome of this matching is the physical data base design reflected in the DBD and the physical data set attributes.

Selecting Data Base Organization

Access methods can, in general, be changed during reorganization without affecting application programs. Still, because the access method is one of the most critical performance factors, it should be carefully selected. First we will discuss selection of the DL/I access method, HDAM, HIDAM, or SHISAM.

When to Choose HDAM: HDAM is recognized in practice to be the most efficient storage organization of DL/I. It should be your first choice, at least in the online environment. HDAM's prime advantages are:

1. Fast direct access (no index accesses) with few I/O operations

2. Single data set and associated control blocks

3. Small working set in main storage for DL/I

4. Good physical sequential access

Disadvantages of HDAM are:

1. You need a randomizing module.

2. Sequential access in root key order is not possible if the physical sequence of data base records in storage is not the same as the root key sequence. This is dependent on the randomizing module and root key characteristics.

In many cases, the disadvantages for HDAM do not apply or can be circumvented. The effort needed to circumvent should be weighed against the savings in terms of main storage and CPU usage. There is no doubt, however, that an application with only HDAM data bases is the most compact one. Some possible solutions for the above HDAM disadvantages are:

1. The IMS/VS system provides a general randomizing module, DFSHDC40, which can be used for any key range. Furthermore, the section "HDAM Randomizing Modules" in Chapter 7, "Installing IMS/VS," will provide you with guidelines on how to write your own randomizing module.

2. If heavy sequential processing is required and a randomizing module which maintains key sequence cannot be designed, then sort techniques can be used:

   a. If the program is non-input driven, as is the case with many report programs, simple Get Next processing presents all the data base records in physical sequential order. The output could then be sorted in the desired order. Also, in many instances, only certain selected segments are required. So the output file of the extract can be a fairly small file.

   b. If there are input transactions which would normally be sorted in root key sequence they should instead be sorted in physical sequence. This can be readily done with an E61 sort exit routine which passes each root key to the randomizing module for address calculation and then sorts on the generated addresses plus root key instead of the root key itself. An example of such a routine, DFSOASRT is provided in IMSVS.PRIMESRC.

3. A secondary index could be built with the root key as index search argument. The cost of this should be weighed against the cost of sorting as in 2 above. The secondary index provides full generic key search capability, however.

We will select HDAM as the DL/I access method for our initial Parts data base, and will use Technique B above in loading it. (For details see "Loading a HDAM Data Base" in Chapter 5.)

When to Choose HIDAM: If you cannot use HDAM for some reason, then use HIDAM (see above discussion).

When to Choose SHISAM: This access method should only be used as a migration tool. That is, if your organization currently has files based on ISAM or KSDS access methods. It is not recommended for new data bases. With SHISAM, new programs can use the DL/I interface with full recovery function. Existing VSAM programs can access the data base as a regular KSDS and older ISAM-based programs can use the ISAM-VSAM interface.

We will utilize a SHISAM data base in our phase 2 environment.

Which OS/VS Access Method

For HDAM you could choose either ESDS or OSAM as the physical access method. There is not much difference as far as DL/I is concerned, although OSAM requires less main storage for code and control blocks. In general you should select ESDS if your installation already uses VSAM or plans to use it for other data bases.

The real benefits from OSAM are gained when you have an application which uses HDAM/OSAM *exclusively*. In any case, conversion from HDAM/OSAM to HDAM/VSAM is relatively simple once you have gained experience with VSAM itself.

For the phase 1 data base we will select OSAM as the physical access method.

Physical Segment Design

In the final steps of segment design we must look at the physical parameters more closely:

- The segment length
- The number of occurrences per segment per parent
- Location of segments in the hierarchy
- Average data base record size

Performance Aspects: The main measure of access performance is the number of I/O requests necessary to satisfy the calls an application program issues. These are mainly dependent upon the physical data base design and the data base buffer pool size; the latter will be discussed in Chapter 9, "Optimization." Second, the number of required DL/I calls should be weighted.

Basic recommendations (HDAM and HIDAM):

- Try to locate the segments most often used together with the root segment into one control interval/block. The segments are initially physically stored in hierarchical sequence so the most frequently used segments should be on the left of the structure (low segment codes).
- Try to avoid long twin chains, that is, many occurrences of a particular segment under one parent. Chain length should be estimated in terms of blocks needed to store such segments. For example, 100 segments of 20 bytes (including prefix) cause less performance problems than 10 segments of 1500 bytes each if the block was 3000 bytes. See also the discussion of the "bytes" parameter under Basic Recommendations (HDAM) below.

- Inserts after initial load will first check the block of the hierarchically preceding segment for available space. If no space is found, nearby blocks in the buffer pool are examined. If still no space is found, a bit map block is used to search for space within ±3 cylinders in our subset. The bit map block contains one bit for each block in the data set. Bit map blocks are repeated each N blocks; N is number of bits in a block. The bit is set to one if the corresponding block contains enough consecutive free space to hold the largest segment (including prefix) of the DBD. If no space is found, the segment is stored at the end of the data set for HIDAM and in the overflow area for HDAM.

Basic recommendation (HDAM):

- During consecutive inserts (no intervening calls) of segments of a particular data base record, the bytes parameter in the RMNAME keyword in the DBD statement will limit the amount of data stored in the root addressable area. If the limit is reached (bytes includes prefix) consecutive inserts are placed in the overflow area. Using this parameter, especially during initial load and reload, can benefit an equal distribution in the case of a large variation in data base record size. See also, HDAM space calculation later in this chapter.

Physical Data Base Structure for Phase 1

Applying above guidelines to the phase 1 Parts data base gives the final physical data base structure of Figure 2-46.

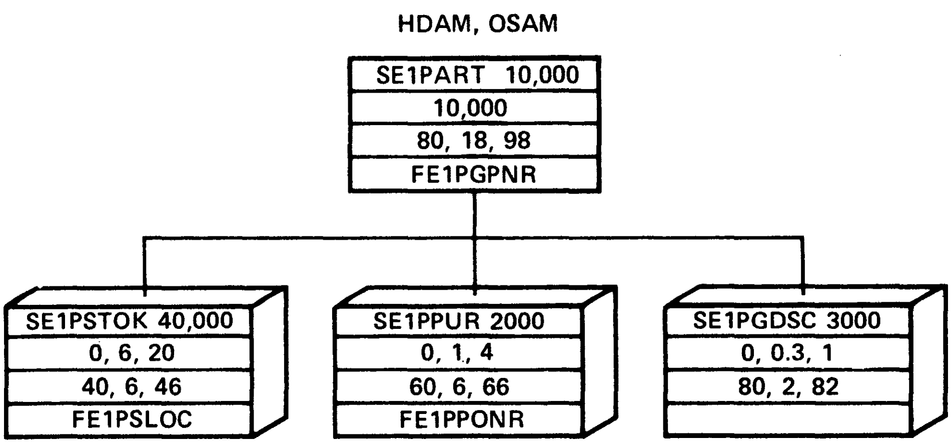


Figure 2-46. Physical Data Base Structure for Phase 1 PARTS Data Base

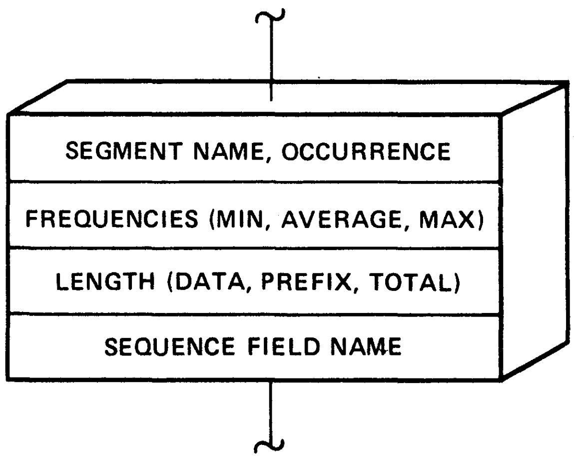
As you will notice, we created a fourth segment, SE1PGDSC which contains the full parts descriptive name, FE1PGLNM, since:

- This information is rarely needed, especially in the foreseen online processing

- By bringing back the root segment from 148 bytes to 98 bytes (including prefix) we improve the segment insert processing of the stock and especially the purchase order segment. This results because the free space bit map is based on the largest physical segment size.

Furthermore, we added a dummy field to the segments. This could be done in practice if you expect the segment to be expanded in the near future. At least you should make all segments an even number of bytes.

We also added to the data base structure in Figure 2-46 the main physical segment attributes which are of most importance for performance considerations. It is recommended that you maintain such structural figures for your data bases. They have proven to be very valuable for performance monitoring and design reviews. A description of those attributes follows in Figure 2-47.



Legend:

- Segment name, occurrence specifies the segment name and the total number of this segment occurrence in the data base.
- Frequencies specifies the minimum, average and maximum number of occurrences for this segment per parent occurrence.
- Length specifies the segment data length, the segment prefix length and the total (=sum of data + prefix) length of the segment.
- Sequence field specifies the name of this segments sequence field, if any.

Figure 2-47. Specification of Physical Segment Attributes

Coding the Phase 1 PARTS DBD,HDAM

We can now code the DBD and discuss the final parameters such as pointer options and CI/blocksize parameters. Some iteration with the preceding section is normally necessary because the pointer options selected influence the size of the segment prefix and, as such, can have an impact on physical segment design. The final DBD of our Phase 1 Parts data base is listed in Figure 2-22 earlier in this chapter under the topic "Basic DBDGEN".

Considerations for Pointer Selection

Because there is no use expected of the physical child last pointer in any segment, code PARENT = ((segname, SNGL)) in all dependents. Because of this (no deletes after retrieve last), only physical twin forward pointers are needed. Code PTR=T in all segments. Because there is never more than one occurrence of the SE1PGDSC segment for any part, the physical twin forward pointer for this segment should be suppressed; code PTR=NT.

Selecting CI/Blocksizes

In choosing the CI/blocksize the following considerations apply:

- Try to fit all highly needed segments of a data base record into one (or more consecutive) CI/blocks.
- One blocksize for all data base OSAM data sets (if any) will limit the amount of subpools. However, using a unique size for a highly used data base allows a dedicated subpool specification for that data base.
- Large blocksizes favor sequential processing and DASD space utilization. On the other hand, if you are primarily processing directly, you should determine the segments needed per data base record per transaction.

Basic recommendations for the practical minimum CI/blocksize for ESDS and OSAM data sets are given for each device type in Figure 2-48. The underlined numbers would be a general "best fit" for OS/VS1. the numbers between parenthesis would be the general "best fit" for OS/VS2.

Device Type	OSAM Blocksize or VSAM ESDS CIsize (blocks/track)			
2314/2319	1536 (4)	(<u>2048</u>) (3)		
3330	1536 (7)	<u>2048</u> (6)		(4096) (3)
3340	1536 (5)		<u>2560</u> (3)	(4096) (2)
3350				(<u>4096</u>) 4

<u>nnnn</u>: OS/VS1 Recommendation
(nnnn): OS/VS2 Recommendation
Blocksizes 1536 and 2560 are only applicable to OSAM

Figure 2-48. Recommended CI/Blocksize Parameters

Additional considerations:

- In case of large data base records (greater than 500 bytes) and/or heavy sequential processing and/or large data bases, you should consider increasing the sizes shown in figure 2-48, especially for OS/VS1.
- For OSAM, the blocksize is limited to the maximum non-keyed blocksize of a track.

For KSDS, for INDEX data bases, you should select a control interval size of 2048 or 4096 for the data component and 1024 for the index component.

ANCH, RBN, BYTES and SIZE Parameters for HDAM

The following basic guidelines apply to above parameters for a HDAM data base:

1. SIZE = 5 X AVRL

2. BYTES = SIZE

3. RBN = 1.25 X NROOTS X AVRL/SIZE

4. ANCH = 1.25 X NROOTS/RBN

Where:

- AVRL is the average data base record length, including segment prefixes.

- SIZE is the net CI/blocksize. Remember that DL/I will allocate some control fields within your selected CI/block. These are:

  - Free space anchor point: 4 bytes

  - For each anchor point: 4 bytes (only in the root addressable area)

  - VSAM control fields (ESDS): 7 bytes

  In addition, there will be a free space element of 8 bytes for each consecutive free space of 8 bytes or more in the CI/block.

- BYTES is the maximum number of bytes of a single data base record, to be inserted by consecutive insert calls against the same PCB.

- ANCH is the number of root anchor points per CI/block (round to next higher).

- RBN is the number of CI/blocks in the root addressable area.

Ideally, 4 to 5 data base records should fit in one CI/block. However, for very large data base records -- one average record per N CI/blocks -- you should consider a randomizing algorithm, which skips every N CI/blocks. The BYTES parameter should then be no less than the average data base record size and the number of anchor points per CI/block should be one.

Example, Our PARTS Data Base:

For our PARTS data base, we calculate (assume 10,000 records):

AVRL =
(10,000X98+40,000X46+2,000X66+3,000X82)/10,000
= 320 bytes

The maximum data base record length is:
98+20X46+4X66+82 = 1364 bytes.

And the minimum data base record length is:
98 bytes.

The data and prefix length of each segment can be found in the DBDGEN macro expansion output listing. The field "SEGMENT LENGTH" contains the data length of the segment in bytes. The field "LENGTH OF SEGMENT PREFIX" contains the length in bytes of the segment prefix.

SIZE : 5 X 320 = 1600, rounded to 2048

BYTES = 2048

Because our maximum data base record size is 1364 bytes, this could be specified as the BYTES limit.

RBN = 1.25 X 10,000 X 320/2048 = 1954

For 3330, this would require 326 tracks or 18 cylinders. An initial space allocation of 20 cylinders (10% for the overflow area) will be appropriate.

ANCH = 1.25 X 10,000/1954 = 6

We now check our net CI/block size in the root addressable area, which is:

2048 - 4 - 4 X 6 = 2020

This is large enough to hold, generally, more than five data base records.

Defining VSAM Data Sets

VSAM data spaces are defined with its Access Method Services. Job //SAMP270 in IMSVS.PRIMEJOB shows how to do this for a HDAM (PARTS) data base and a HIDAM (Customer Order) data base. Note that the DATA and INDEX components are separately named.

Whenever defining a KSDS, you should check the DBDGEN output listing. It gives the proper access method service control statements for the definition of the KSDS (that is, the location of the key in the KSDS record).

Note: Job //SAMP270 defines the VSAM data sets in the VSAM data space defined with job //SAMP007.

OSAM Data Set Allocation

OSAM space can be allocated via normal JCL as an OS/VS sequential data set. No DCB information should be provided in the JCL. OSAM space can be reused but only if the blocksize (SIZE parameter in DBD) has not been changed, that is, the same as indicated in the DSCB on DASD.

Job //SAMP170 in IMSVS.PRIMEJOB, which loads the Phase 1 PARTS data base shows how to allocate the space for the OSAM dataset.

Phase 2 Physical Data Base Design

The Phase 2 application data structure in Figure 2-43 can be easily implemented with the use of the logical relationship function of DL/I. Merely define the orderline segment as a logical child of the part segment as shown in Figure 2-43. In addition the following considerations apply:

- The physical data base design for the Parts and Customer Orders data bases is done in much the same way as for Phase 1.
- The access method for the CUSTOMER ORDER data base is HIDAM/VSAM. This is done to show an example of a HIDAM data base.
- The access method for the PARTS data base is changed to HDAM/VSAM to provide a VSAM only environment.
- As discussed previously, we will use a separate SHISAM data base for the customer name and address instead of duplicating that data in the Customer Order data base. The key (customer number) of this SHISAM data base will be stored in the root segment of the Customer Order data base.

Notes:

1. The real logical child can, in reality, be located either in the Parts or the Customer Orders data base. Their is no difference for the application program as to where it is located (except for the initial load program). Which implementation to choose is purely a performance matter. This will be discussed in Chapter 9, "Optimization," under the topic "Optimizing Physical and Logical Twin Chains."
2. Whenever the accounts receivable application is converted to DL/I, the SHISAM data base could be converted to a full HDAM or HIDAM data base. Additional segments can then be added with minimal impact on the existing DL/I application programs. Also, if necessary, a logical relationship could be implemented between this Customer data base and the Customer Orders data base, much in the same way as between the Parts and the Customer Orders data base.

Two sets of DBDs are needed for the phase 2 applications:

- Physical DBDs with logical relationships, and
- Logical DBDs for the application programs.

The DBDGEN process of these DBDs is described under the topic "DBDGEN for Logical Data Bases" earlier in this chapter. The physical DBDs for the Parts and Customer Orders data bases are shown in Figure 2-24.

Due to expected high activity against the logical child segment all associated pointers are specified forward and backward. This should be done in all cases where there is considerable activity expected with a logical child.

The corresponding logical Parts DBD (BE2LPART) is listed in Figure 2-25. The logical Customer Orders data base is listed in Figure 2-26.

All above DBDs, together with the SHISAM DBD (BE2PCUST) are also included in IMSVS.PRIMESRC. Their DBDGENs can be executed with job //SAMP210 in IMSVS.PRIMEJOB.

Phase 3 Physical Data Base Design

In our Phase 3 sample data base design, we will use the secondary index function of DL/I.

Analyzing our Phase 3 requirements as reflected in its transaction/ data element matrix (Figure 2-41) and application data structure (Figure 2-44), we see the need for the access of the parts data via purchase order number.

Actually, the best way, from a pure data base design point of view, is to implement this via a logical relationship. This logical relationship should then be established between a new Purchase Orders data base and the Parts data base. However, we choose to use the secondary index function for this with the following considerations:

- Exemplify the difference between the logical relationship and secondary index functions.
- Adding of the secondary index to the PARTS data base has the least impact on the existing Phase 1 and Phase 2 application programs.
- If there is no expected extension of the purchase order application, it might also, in a real live situation, be the most economical, solution.

Furthermore, we will select the parts segment as the index target segment. This is according to the limitations of our subset as set before (that is, target=root segment). In this way, the Phase 3 requirements can be very easily implemented, especially by the application programs.

Above discussions are reflected in our Phase 3 DBDS:

- The physical Parts data base BE3PARTS
- The physical Customer Orders data base BE2ORDER and its primary index BE3ORDRX
- The secondary index DBD BE3PSID1

These DBDs are all included in IMSVS.PRIMESRC. Their DBDGENs can be performed with job //SAMP310 in IMSVS.PRIMEJOB. The DBDs for BE3PARTS and BE3PSID1 are shown in Figure 2-29 under the topic "DBDGEN For Secondary Indexes".

Note: The Phase 3 application program BE3CPPUR uses a PSB which references the Phase 3 physical DBDs. Ideally, this PSB should use a Phase 3 logical PARTS data base BE3LPART. This DBD is much the same as for Phase 2. It is suggested that you exercise this change yourself.

DESIGN EVALUATION

Following the physical data base design, a design evaluation should be performed. The two main questions for this evaluation are:

- Does the data base design support the applications functional requirements?
- Does the data base design provide for a satisfactory performance?

The first question is not considered here, because it is too application and installation dependent. The second question's answer is also largely installation dependent. However, Chapter 9, "Optimization," provides you with a simple hand calculation technique to compare alternative designs. In addition, a checklist is included which addresses the most important performance factors for DL/I data bases.

CHAPTER 3. DATA COMMUNICATION DESIGN

This chapter is complementary to the previous chapter on data base design. It provides the data communication designer and system analyst with a detailed description of the IMS/VS data communication functions, and guidelines on how to use these functions.

The three main parts in this chapter are:

- A description of an online application sample. The sample application is used to demonstrate the general requirements of an online system. The sample application is used also in the examples in the remainder of the chapter.
- A more formal description of the IMS/VS data communication functions, including the specification of IMS/VS message format service usage.
- An extension of the data base design process of the previous chapter into the data communication area. Besides considerations for online data base design, this part provides guidelines for online application program design and message format design.

THE PHASE 4 SAMPLE REQUIREMENT

The basic requirement of phase 4 of our sample environment is to run the phase 3 (see Chapter 2) sample applications in an online environment.

PHASE 4 SAMPLE DATA BASES

The phase 4 sample data base requirements are in general the same as for phase 3. The only added requirement is that they should be accessible online. As we will see, this usually will not require any change in the data base design. In the sample online system we will use the phase 3 data bases.

PHASE 4 BATCH PROGRAMS

In phase 4 of our sample system, both the Inventory Report Processing and the Purchase Order Processing will remain batch applications. We will show in the sample system how the pre-phase 4 programs of these applications can be executed with the online data bases without any modifications to the programs.

PHASE 4 ONLINE PROGRAMS

The essential requirement for phase 4 is to perform the Customer Order Processing as an online application, using IMS/VS data base/data communication facilities. All the transactions defined in phase 2 (see "Sample Application for Phase 2" in Chapter 2) should be available via the 3270 Information Display System. In addition a simple online customer name and address inquiry application will be implemented.

IMS/VS DATA COMMUNICATION FACILITIES

In the following sections, we will discuss the IMS/VS data communication facilities within our subset. It is assumed that you have a clear understanding of the concepts and terminology as presented in Chapters 1 and 2.

To explain the IMS/VS data communication facilities, we will follow a message through the system.

THE MESSAGE

The goal of IMS/VS is to perform online transaction processing. This consists of:

1. Receiving a request for work to be done. The request is entered at a remote terminal. It is usually made up of a transaction code which identifies to IMS/VS the kind of work to be done and some data which is to be used in doing the work.

2. Initiating and controlling a specific program which will use the data in the request to do the work the remote operator asked to be done, and to prepare some data for the remote operator in response to the request for work (for example, acknowledgment of work done, answer to a query, etc.).

3. Transmission of the data prepared by the program back to the terminal originally requesting the work.

The above sequence is the simplest form of a transaction. It involves two messages: an input message from the remote operator requesting that work be done, and an output message to the remote operator containing results or acknowledgement of the work done.

Multiple and Single-Segment Messages

A message, in the most general sense, is a sequence of transmitted symbols. In the context of IMS/VS, this is called a transmission. A transmission may have one or more messages, and a message may have one or more segments. A segment is defined by an end-of-segment (EOS) symbol, a message is defined by an end-of-message (EOM) symbol and a transmission is defined by an end-of-data (EOD) symbol. The valid combinations of the conditions represented by EOS, EOM, and EOD are:

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

The relations between transmission, message and segment is shown in Figure 3-1.

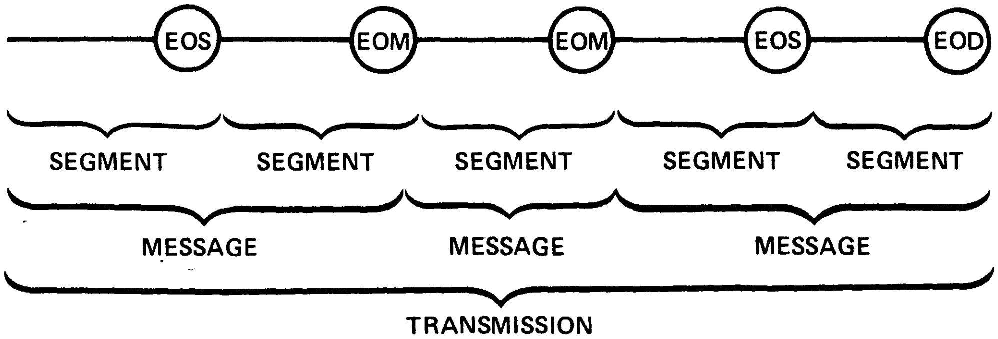


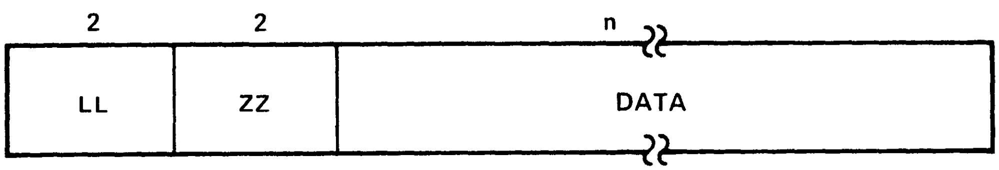
Figure 3-1. Transmission, Message, and Segment Relations

The character values or conditions that represent the end of segment and/or message are dependent on the terminal type.

In our subset, 3270 terminals only, the physical terminal input will always be a single segment message and transmission. The EOS, EOM and EOD condition will all be set after the enter or program function key is pressed and the data is transmitted.

On the output side, a message can be divided into multiple segments. Also an application program can send different messages to different terminals, that is, a message to a printer terminal and a message to the input display terminal. Each segment requires a separate insert call by the application program.

The format of a message segment as presented to or received from an application program is shown in Figure 3-2.



LL	:	total length of segment in bytes including the LL + ZZ fields.
ZZ	:	IMS/VS system fields
DATA	:	application data

Figure 3-2. A Message Segment.

IMS/VS ONLINE OPERATION: OVERVIEW

As introduced in Chapter 1, IMS/VS online processing is done with three different types of regions, address spaces, or partitions under OS/VS:

- The control (CTL) region contains the IMS/VS control program. It controls the terminals and data bases.
- The message processing (MPP) region contains a user program for message-driven processing of the data bases. The MPP region is controlled by and relies upon the CTL region. The application program which execute in the MPP region is called a message processing program or MPP. Different MPPs can be subsequently loaded and activated in a single MPP region.

- The *batch message processing* (*BMP*) *region* contains an application program for batch processing of the data bases managed by the CTL region. Application programs executing in a BMP region are called *batch message processing programs* or *BMPs*.

Figure 3-3 gives an overview of these 3 region types and the control and data flow within them.

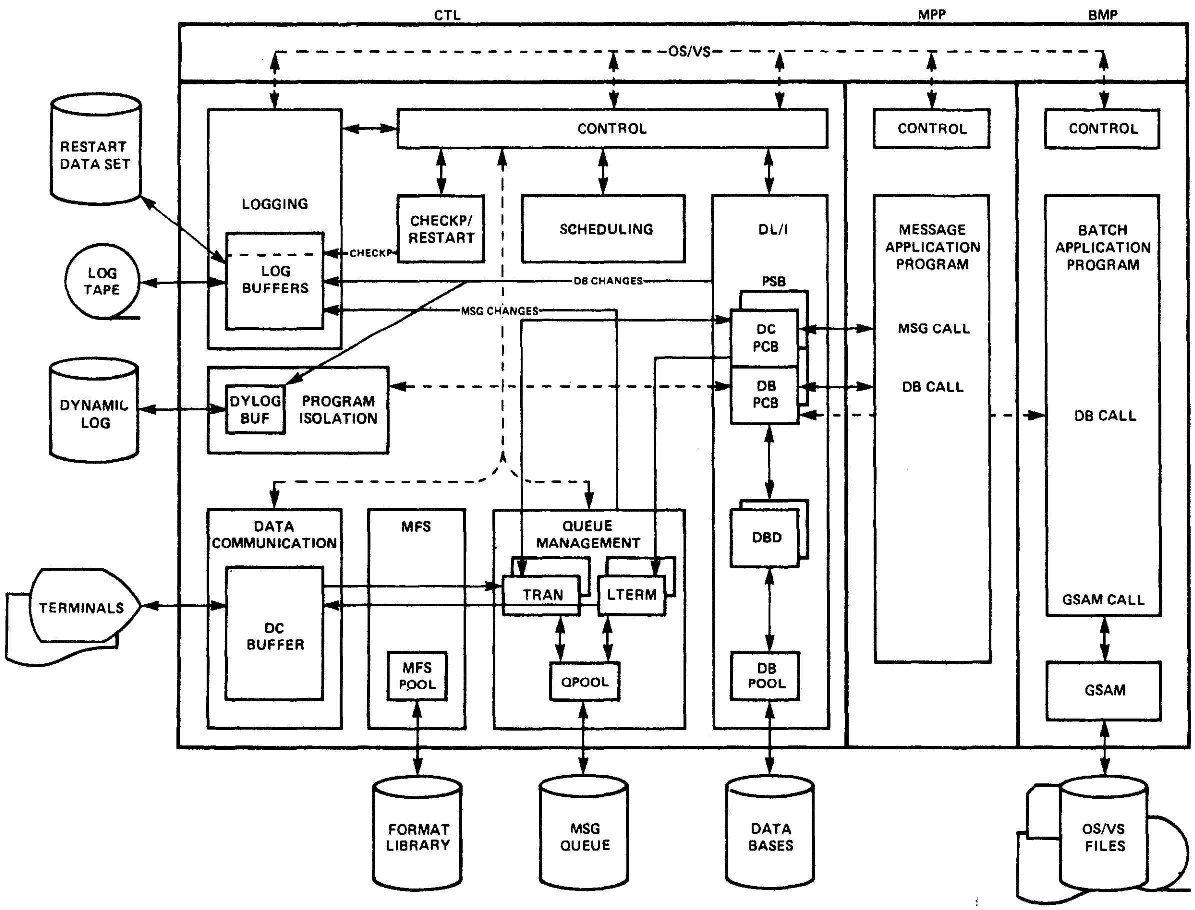


Figure 3-3. The IMS/VS Regions and Their Control/Data Flow.

The CTL Region

The CTL region is initiated through an OS/VS start command. The terminals, data bases, and the log tape are all attached to this region. A type 2 supervisor call routine is used for switching control information, message and data base data to the MPP/BMP region and back.

The CTL region normally runs as a system task and uses OS/VS access methods for terminal and data base access.

Once the CTL region is started, its general data flow is as follows. See Figure 3-3.

1. The input data from the terminal is read by the data communication modules. After editing by message format service (MFS), this input data is put in the input queue (TRAN), which is sequenced by transaction code.

2. The scheduling modules will start the processing of a transaction in an MPP if an MPP is free and other conditions are met.

3. Upon request from an MPP/BMP, the DL/I modules pass a message or data base segment to or from the MPP/BMP.

   Note: In MVS, the DL/I modules, control blocks, and pools reside in the common storage area (CSA) and the CTL region is not needed for most DB processing.

4. The message output from the MPP is put in the output message queue (LTERM), which is sequenced by logical terminal name.

5. The communication modules retrieve the message from the output queue and send it to the output terminal. MFS is used to edit the screen and printer output.

6. All changes to the message queues and the data bases are recorded on the log tape. In addition, checkpoints for system (emergency) restart and statistical information are logged.

   Notes:

   1. The physical logging modules run as a separate task and use OS/VS ESTAE for maximum integrity.

   2. The checkpoint identification and the log tape volume serial numbers are recorded in the restart data set.

7. Program Isolation assures data base integrity when two or more MPPs/BMPs update the same data base. It also backs out (undoes) data base changes made by failing programs. This is done by maintaining a short-term, dynamic log of the old data base element images.

8. The control module provides multi-tasking for the above activities. These separate functions, that is, input processing, queueing, MPP processing, data base call processing, output processing, etc., can be executed asynchronously for multiple transactions. However, they will be executed in sequence for a unique transaction occurrence. The OS/VS EVENT facility is used for the control of the multi-tasking and input/output operations.

The MPP Region

An MPP region is started with an IMS/VS command. The CTL region in turn issues an OS/VS command to initiate a region via OS/VS job management. Message processing application programs (MPPs) are loaded/activated in this region as required. They are scheduled by the control region. If the application issues DL/I calls to access data bases or terminals, these calls are processed in or under supervision of the control region. An MPP region must not use OS/VS data sets because these cannot be repositioned during emergency restart. Also, OPEN/CLOSE processing of these data sets might cause (performance) problems.

The 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 the BMP region can access only IMS/VS data bases that are defined in the IMS/VS control region.

BMPs can access OS/VS data sets. However, if the BMP uses the extended checkpoint/restart facility, these data sets should be defined as GSAM data bases.

RELATIONSHIP OF DB/DC TO DB SYSTEM

In general, all the DL/I data base facilties as presented in Chapter 2 are available in the IMS/VS DB/DC system. The only exception is that GSAM data bases (and other OS/VS files) cannot be used by a message processing program (MPP). They can be used in a batch message processing program (BMP), however.

The DL/I Region

Even with an IMS/VS CTL region and related MPP/BMP regions active, a batch-only DL/I region can be executed. This DL/I region provides the same functions as the batch-only system. However, this DL/I region cannot have access to data bases connected to the CTL region. It should therefore only be used for batch processing when the CTL region is not active, or for processing data bases that are not used by the online system.

In our subset, we will assume that all batch processing while the CTL region is active is done by BMPs.

TERMINAL INPUT DATA PROCESSING

Figure 3-4 should be referred to for the following discussion.

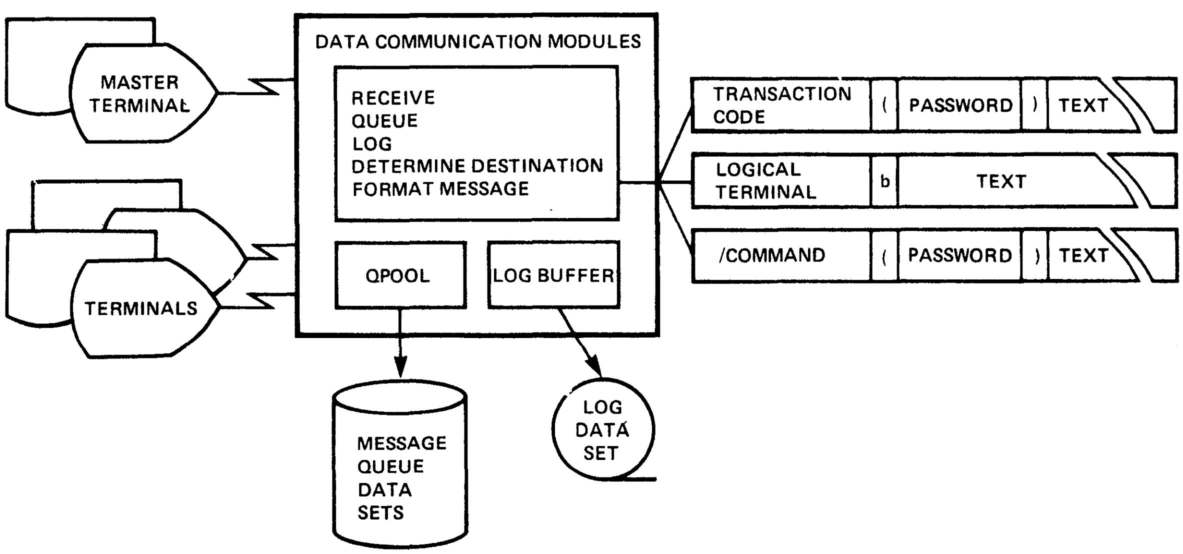


Figure 3-4. Input Message Processing

When IMS/VS reads data from a terminal via the telecommunication access method, it first checks the type of input data.

Input Message Types

As discussed in Chapter 1, the three basic types of terminal input are:

- A command, which starts with a slash (/).
- An input message, to be routed to an application program for processing. The program destination is defined by the 1- to 8- byte transaction code included as the first part of the input.
- A message switch, to be routed to another terminal. The terminal destination is defined by the 1 to 8 byte logical terminal name included as the first part of the input.

Input Message Origin

IMS/VS maintains the origin of an input message. When a message is made available to an application program, this origin is also made available to that program, via its program communication block (PCB). This origin is the logical terminal name (LTERM), which is associated with the inputting physical terminal at the time the input is received.

If more than one LTERM is defined or assigned to a physical terminal, they are maintained in a historical chain; the oldest defined/assigned first. Any input from the physical terminal is considered to have originated at the first logical terminal of the chain. If, for some reason (such as security or a stopped LTERM), the first logical terminal is not allowed to enter the message, all logical terminals on the input chain are interrogated in chain sequence for their ability to enter the message. The first appropriate LTERM found is used as message origin. If no LTERM can be used, the message is rejected with an error message.

Terminal Input Destination

The destination of the terminal input is dependent upon the type of input.

An input command goes directly to the IMS/VS command processor modules. Both the message switch and the transaction input are stored in the message queues. The transaction input from the 3270 displays is first processed by message format service (MFS), except when input is from a previously cleared or unformatted screen.

MFS provides an extensive format service for both input and output messages. It is discussed in detail later in this chapter.

Once the input is enqueued to its destination in the message queue, the input processing is completed.

MESSAGE QUEUEING

All input and output messages in IMS/VS (except command input) are queued in message queues. See Figure 3-5.

With this approach, input processing, output processing, command processing, and application program processing can, to a large extent, be performed asynchronously. This means, for example, that the input processing of message A can be done in parallel with the data base processing for message B and the output processing for message C. A, B, and C can be different occurrences of the same or different message types and/or transaction codes.

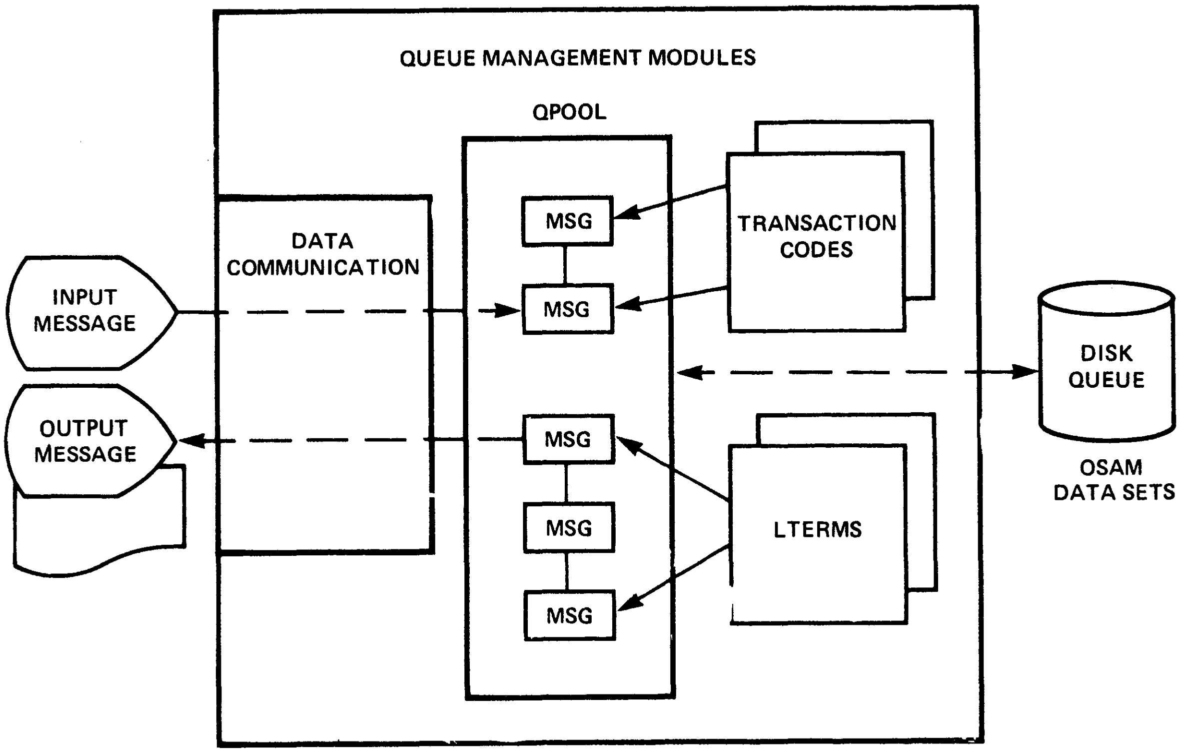


Figure 3-5. Message Queueing.

The message queues are sequenced by destination. A destination can be:

- A message processing program (MPP), that is, for transaction input. Ordering is by transaction code.
- A logical terminal (LTERM), that is, for a message switch, command responses, and output generated by application programs.

The message queues are maintained in main storage (QPOOL), with overflow data sets on direct access storage, the queue data sets. The queue blocks in main storage and on direct access storage are reusable. This helps to minimize the number of I/Os operations required during processing.

Queue Size, Performance Considerations

Because we will consider only 3270 terminals in a mostly interactive environment, message queueing will be primarily in main storage.

Chapter 7 contains detailed guidelines for selecting message queue parameters such as block sizes, QPOOL size, queue data set allocation, etc.

MESSAGE SCHEDULING

Once an input message is available in the message queue, it is eligible for scheduling. Scheduling is the routing of a message in the input queue to its corresponding application program in the message processing partition/region. See Figure 3-6.

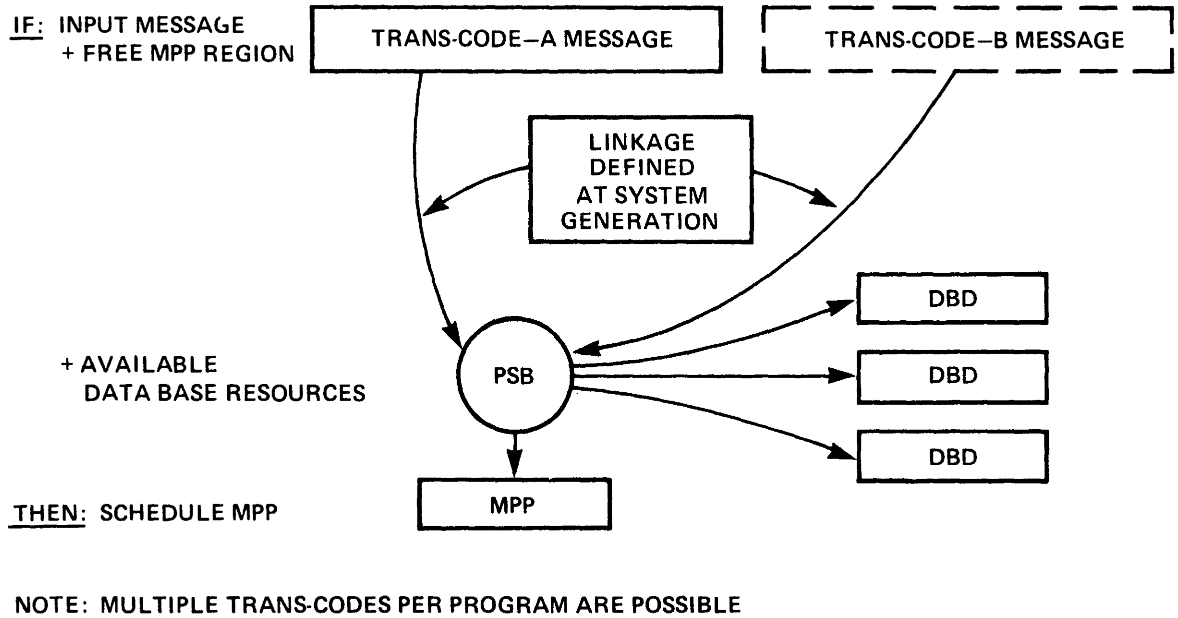


Figure 3-6. Message Scheduling.

The linkage between an input message (defined by its transaction code) and an application program (defined by its name) is established at system definition time. Multiple transaction codes can be linked to a single application program, but only one application program can be linked to a transaction code.

In our subset we will limit outselves to a simple, straightforward scheduling algorithm. In principle, it will be FIFO (first in, first out) scheduling with nc particular priority mechanism.

Note: This scheduling mechanism is a general "best-fit" for an initial IMS/VS installation. This will not prohibit the introduction of more sophisticated algorithms later. To do so would require changes only to IMS/VS parameters and would be transparent to the application programs.

Scheduling Conditions

The following conditions must be met for a successful scheduling:

1. An MPP region must be available. Actually, the termination of an MPP triggers the scheduling process.

2. There must be a transaction input message in the queue.

3. The transaction and its program are not in a stopped state.

4. Enough buffer pool storage is available to load the program specification block (PSB) and the referenced data base control blocks if not already in main storage.

5. The data base processing intent does not conflict with an already active application program (a BMP for instance). Processing intent is discussed in more detail in the following section on data base processing intent.

If the first transaction code with a ready input message does not meet all the above conditions, the next available input transaction is interrogated, and so forth. If no message can be scheduled, the scheduling process is stopped until another input message is enqueued. If the scheduling is successful, the IMS/VS routines in the dependent region load the corresponding MPP and pass control to it.

Scheduling_A_BMP

A BMP is initiated in an OS/VS partition/region via regular OS/VS job management. However, during its initialization the IMS/VS scheduler in the control region is invoked to assure the availability of the data base resources for the BMP.

Data_Base_Processing_Intent

A factor that significantly influences the scheduling process is the intent of an application program 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 active program in a MPP or BMP region, the scheduling process will select another transaction and try again.

The data base intent of a program at scheduling time is determined via the PROCOPT= parameters in the PSB.

With the program isolation feature (see the next section), IMS/VS minimizes possible conflicts during scheduling.

A conflicting situation during scheduling will only occur if a segment type is declared exclusive use (PROCOPT=E) by the program being scheduled and an already active program references the segment in its PSB (any PROCOPT) or vice versa.

Example: If a BMP is executing with a defined PROCOPT=E for the CUSTOMER ORDERS root segment (see Figure 2-12), then no MPP that references the same segment will be scheduled. That is, if the MPP to be scheduled may reference the logical PARTS data base (Figure 2-14) and its PSB contains a SENSEG statement for the concatenated segment, it will not be scheduled before the above mentioned BMP has ended. Note: 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/VS 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.

APPLICATION PROGRAM PROCESSING

Once an application program is scheduled in a dependent region, it is loaded into that region by IMS/VS.

MPP_Processing

After the load of the MPP, it is given control. The normal processing steps of an MPP are described below and in Figure 3-7.

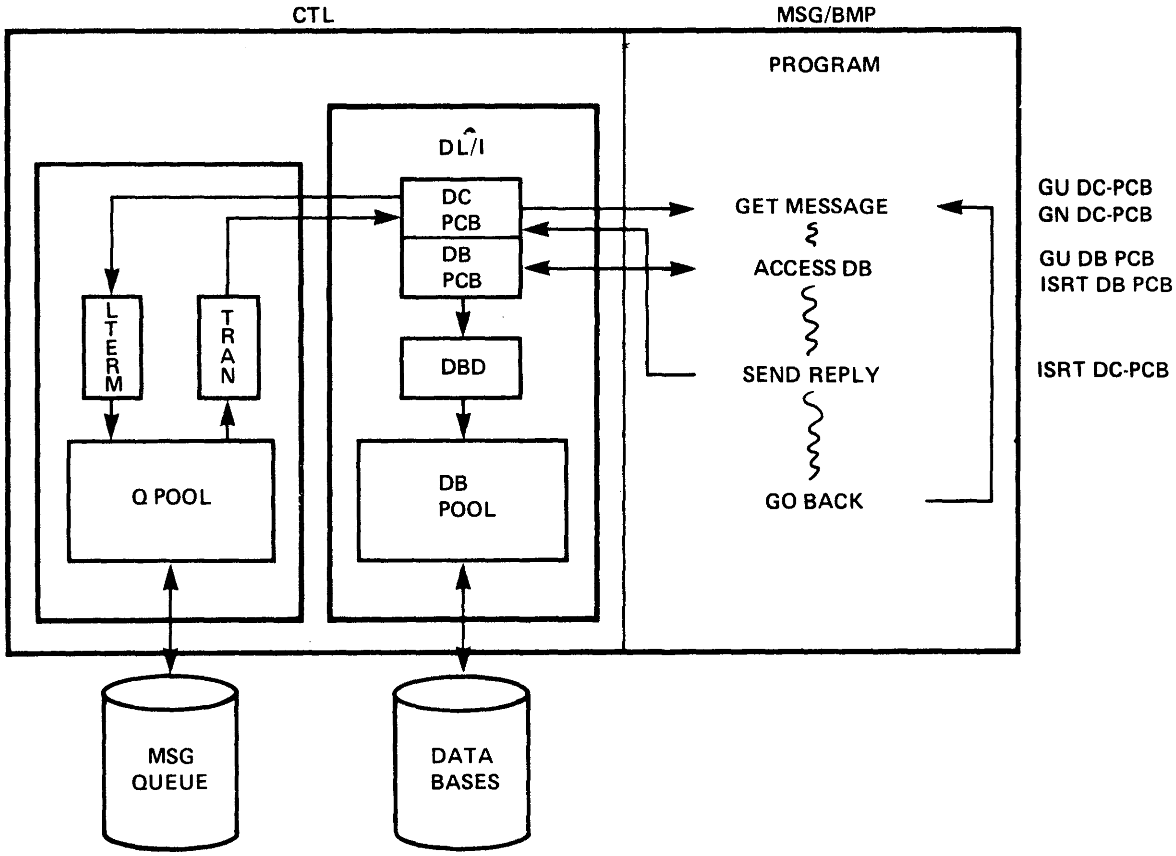


Figure 3-7. Basic MPP Flow

1. Retrieve the input message via a DL/I message call.

2. Check the input message for syntax errors.

3. Process the input message, requesting necessary DL/I data base accesses.

4. Send output to the originating and/or other (for example, printer) logical terminals via DL/I message calls.

5. Retrieve the next input message or terminate.

Role of the PSB

The program specification block (PSB) for an MPP or a BMP contains, besides data base PCBs, one or more PCB(s) for logical terminal linkage. The very first PCB always identifies the originating logical terminal. This PCB must be referenced in the get unique and get next message calls. It must also be used when inserting output messages to that LTERM. In addition, one or more alternate output PCBs can be defined. Their LTERM destinations can be defined in the PCBs or set dynamically with change destination calls.

DL/I Message Calls

The same DL/I language interface which is used for the access of data bases is used to access the message queues.

The principal DL/I message call function codes are:

- GU, get unique. This call must be used to retrieve the first, or only, segment of the input message.

- GN, get next. This call must be used to retrieve second and subsequent message segments.

- ISRT, insert. This call must be used to insert an output message segment into the output message queue.

  Note: These output message(s) will not be sent until the MPP terminates or requests another input message via a get unique.

- CHNG, change destination. This call can be used to set the output destination for subsequent insert calls.

For a detailed description of the DL/I message calls and guidelines for their use, see Chapter 4, "Data Base Processing."

Program Isolation and Dynamic Logging

When processing DL/I data base calls, the IMS/VS program isolation function will ensure data base integrity.

With program isolation, all activity (data base modifications and message creation) of an application program is isolated from any other application program(s) running in the system until that application program commits, by reaching a synchronization point, that the data it has modified or created is valid. A synchronization point in our subset is established with a get unique for a new input message and/or a checkpoint call (BMP only), or program normal termination (GOBACK or RETURN).

Program isolation allows two or more application programs to concurrently execute with common data segment types even when processing intent is segment update, add, or delete.

This is done by a dynamic enqueue/dequeue routine which enqueues the affected data base elements (segments, pointers, free space elements, etc.) between synchronization points.

At the same time, the dynamic log modules log the prior data base record images between those synchronization points.

This makes it possible to dynamically back out the effects 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 of other application program(s) running concurrently in the system.

With program isolation and dynamic backout, 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.

One example of a deadlock occurs in the following sequence of events:

1. Program A updates data base element X.

2. Program B updates data base element Y.

3. Program A requests Y and must wait for the synchronization point of program B.

4. Program B in turn requests X and must wait for the synchronization point of program A.

A deadlock has now occurred: both programs are waiting for each other's synchronization point. The dynamic enqueue/dequeue routines of IMS/VS intercept possible deadlocks during enqueue processing (in above example during enqueue processing of event 4).

Upon detecting a deadlock situation, one of the application programs involved in the deadlock is abnormally terminated (pseudo abend). The activity of the terminated program will 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 transaction that was being processed by the abnormal terminated program will be saved. The application program is rescheduled if it was an MPP. For a BMP region, the job must be restarted. This process is transparent to application programs and terminal operators.

There are two situations where the enqueue/dequeue routines of program isolation are not used in processing a data base call:

1. If PROCOPT=GC (read only) is specified for the referenced segment(s) of the call.

2. If PROCOPT=E (exclusive) is specified for the referenced segment(s) in the call.

Notice that possible conflicts with exclusive extent are resolved during scheduling time and as such cannot occur at call time.

Notes:

1. With the GO option, a program can retrieve data which has been altered or modified by another program still active in another region, and data base changes made by that program are subject to being backed out.

2. Exclusive intent may be required for long-running BMP programs that do not issue checkpoint calls. Otherwise, an excessively large enqueue/dequeue table in main storage may result.

3. Even when PROCOPT=E is specified, dynamic logging will be done for data base changes. The ultimate way to limit the length of the dynamic log chain in a BMP is by using the XRST/CHKP calls. The chain is deleted at each CHKP call because it constitutes a synchronization point.

4. If, as can occur in our subset, one MPP and one BMP get involved in a deadlock situation, the MPP will be subject to the abnormal termination, back out and reschedule process.

Application Program Abnormal Termination

Upon abnormal termination of a message or batch-message processing application program for other reasons than deadlock resolution, internal commands are issued to prevent rescheduling. These commands are the equivalent of a /STOP command. 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 data base changes of a failing program are dynamically backed-out. Also, its output messages inserted in the message queue since the last synchronization point are cancelled.

Conversational Processing

A transaction code can be defined as belonging to a conversational transaction during IMS/VS system definition. If so, an application program that processes that transaction, can interrelate messages from a given terminal. The vehicle to accomplish this is the scratchpad area (SPA). A unique scratchpad area is created for each physical terminal which starts a conversational transaction. Each time an input message is entered from a physical terminal in conversational mode, its SPA is presented to the application program as the first message segment (the actual input being the second segment). Before terminating or retrieving another message (from another terminal), the program must return the SPA to the control region with a message ISRT call. The first time a SPA is presented to the application program when a conversational transaction is started from a terminal, IMS/VS will format the SPA with binary zero's (X'00'). If the program wishes to terminate the conversation, it can indicate this by inserting the SPA with a blank transaction code.

OUTPUT MESSAGE PROCESSING

As soon as an application reaches a synchronization point, its output messages in the message queue become eligible for output processing. A synchronization point is reached whenever the application program terminates or requests a new message/SPA from the input queue via a GU call.

In general, output messages are processed by message format service before they are transmitted via the telecommunications access method.

Different output queues can exist for a given LTERM, depending on the message origin. They are, in transmission priority:

1. Response messages, that is, messages generated as a direct response (same PCB) to an input message from this terminal.

2. Command responses.

3. Alternate output messages, that is, messages generated via an alternate PCB.

Note: The printing of "DFS059 TERMINAL STARTED" messages on the 3270 printer terminals will be suppressed in our subset. This is done to protect preprinted forms.

LOGGING AND CHECKPOINT/RESTART

To ensure the integrity of its data bases and message processing, IMS/VS uses logging and checkpoint/restart. In case of system failure, either software or hardware, IMS/VS can be restarted. This restart includes the repositioning of users' terminals, transactions, and data bases.

Logging

During IMS/VS execution all information necessary to restart the system in the event of hardware or software failure, is recorded on a system log data set. In our subset, this log data set must be on a magnetic tape unit.

The following critical system information is recorded on the log tape (see Figure 3-8):

- The receipt of an input message in the input queue
- The start of an MPP/BMP
- The receipt of a message by the MPP for processing
- Before and after images of data base updates by the MPP/BMP
- The insert of a message into the queue by the MPP
- The termination of an MPP/BMP
- The successful receipt of an output message by the terminal

In addition to the above logging, all previous data base record images are written to a separate dynamic log. This log information is only used for dynamic back-out processing of a failing MPP/BMP. As soon as the MPP/BMP reaches a synchronization point, the dynamic log information of this program is discarded.

Checkpointing

At regular intervals during IMS/VS execution, checkpoints are written to the log tape. This is to limit the amount of reprocessing required in the case of emergency restart. A checkpoint is taken after a specified number of log records are written to the log tape or after a checkpoint command. A special checkpoint command is available to stop IMS/VS in an orderly manner.

A special disk restart data set is used to record the checkpoint identification and log tape volume serial numbers. This restart data set (IMSVS.RDS) is used during restart for the selection of the correct restart checkpoint and restart log tape(s).

Note: Although IMS/VS itself provides for full disk logging/restart with the IMSVS.RDS data set, this function is not included in our subset.

Cold Start

An IMS/VS CTL region cold start is done at the first time you start the system. During cold start, we format (initialize) the message queue, dynamic log and restart data sets.

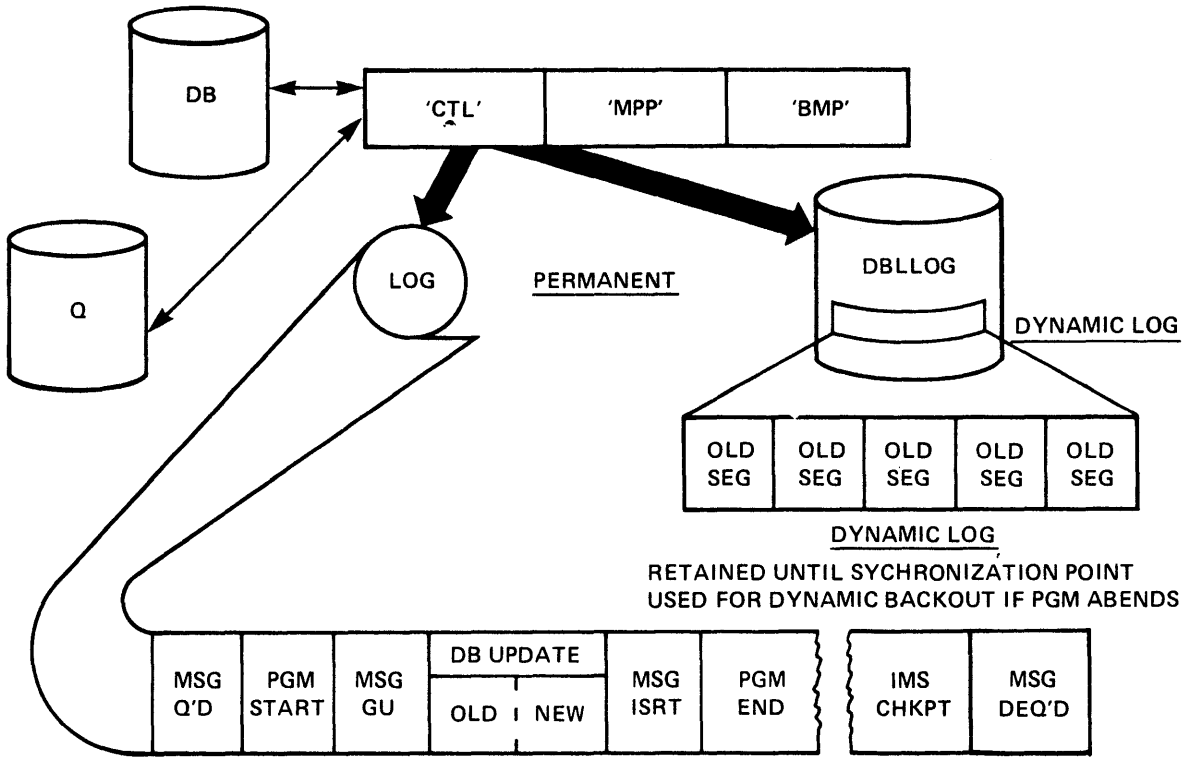


Figure 3-8. IMS/VS Logging

Emergency Restart

In case of failure, IMS/VS is restarted with the log tape active at the time of failure. Restart processing will back-out the data base changes of incomplete MPPs and BMPs. The output messages inserted by these incomplete MPPs will be deleted.

After back-out, the input messages are re-enqueued, the MPPs restarted and the pending output messages are (re)transmitted. If a BMP was active at the time of failure, it must be resubmitted via OS/VS job management. If the BMP uses the XRST/CHKP calls, it must be restarted from its last successful checkpoint. In this way missing or inconsistent output is avoided. For more details, see Chapter 8, "Operations."

Normal Restart

Normal restart or warm start is done from a previous normal IMS/VS termination. The message queues are preserved in this way.

SECURITY

In our subset we will only consider password and terminal security. For a description of these security provisions, see the "IMS/VS Security Maintenance Utility" description in Chapter 7, "Installing IMS/VS."

IMS/VS itself has more extensive security features for user signon and support of user exits and the RACF program product (OS/VS2 MVS only). For more details on these additional security features, see the *IMS/VS General Information* manual and the *IMS/VS System Application Design Guide*.

THE MASTER TERMINAL

The IMS/VS master terminal in our subset consists of two components:

- The primary components, a 3270 display terminal of 1920 characters (24 lines by 80 columns).
- The secondary component, a 3270 printer terminal.

All messages are routed to both the primary and secondary components. Special MFS support is used for the master terminal. The display screen of the master terminal is divided into four areas. See Figure 3-9.

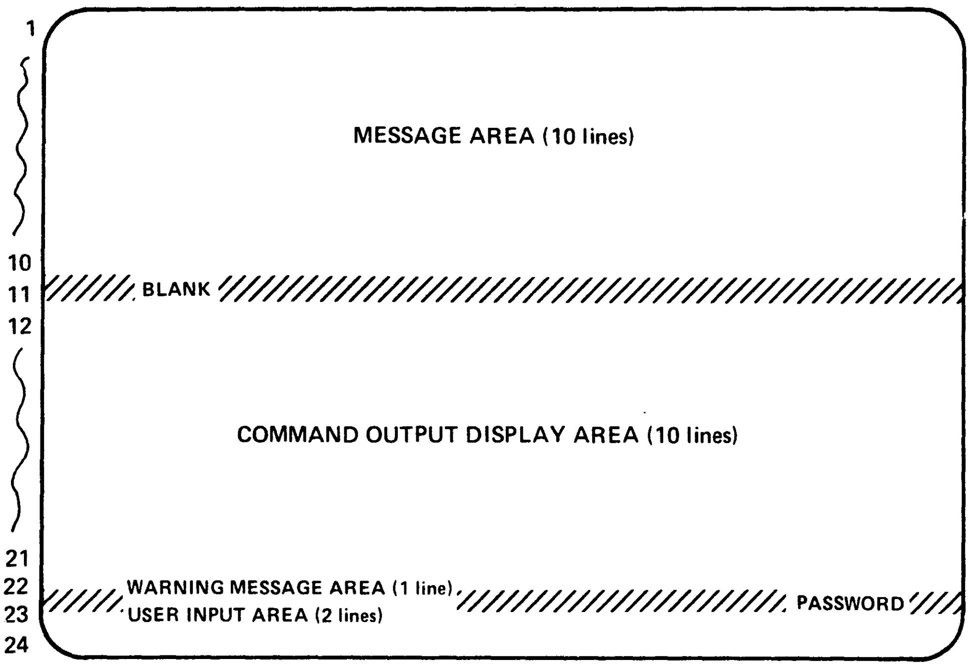


Figure 3-9. 3270 Master Terminal Format

The message area is for IMS/VS command output (except /DISPLAY and /RDISPLAY), message switch output, application program output that uses a message output descriptor name beginning with DFSMO (see MFS), and IMS/VS system messages.

The display area is for /DISPLAY and /RDISPLAY command output.

The warning message area is for the following warning messages: MASTER LINES WAITING, MASTER MESSAGE WAITING, DISPLAY LINES WAITING, and USER MESSAGE WAITING. To display these messages or lines, press PA1. An IMS/VS password may also be entered in this area after the "PASSWORD" literal.

The user input area is for operator input.

Program function key 11 or PA2 requests the next output message and program function key 12 requests the Copy function if it is a remote terminal.

For more details on the use of the master terminal refer to Chapter 8, "Operations."

Using the OS/VS Console as Master Terminal

IMS/VS always has a communication path with the OS/VS system console. The write-to-operator (WTO) and write-to-operator-with-reply (WTOR) facilities are used for this. Whenever the IMS/VS CTL region is active, there is an outstanding message requesting reply on the OS/VS system console. This can be used to enter commands for the CTL region. 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 defined as IMS/VS line number one during system definition.

3270 REMOTE COPY FUNCTION

For remote 3270 display terminals IMS/VS provides a copy function. By pressing PFK12 (PA3 on data entry keyboard), the operator can cause the contents of the screen to be copied to a printer attached to the same control unit. Which printer is selected is determined by terminal status and system definition sequence. In general the first ready terminal on the control unit is selected. This function should only be used for occasional hard copies. For production applications it is generally better to perform printing under application program control.

MESSAGE SWITCHING

The basic format of a message switch is the destination LTERM name followed by a blank and the message text. In our subset, using 3270s and message format service, we will include a sample message switch format. The advantage of using the sample format, is that it automatically provides the originating LTERM name and location. The use of this format is discussed in detail in the *IMS/VS Primer Remote Terminal Operator's Guide*.

MESSAGE FORMAT SERVICE OVERVIEW

Through the Message Format Service (MFS), a comprehensive facility is provided for IMS/VS users of 3270 and other terminals/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. 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 3-10.

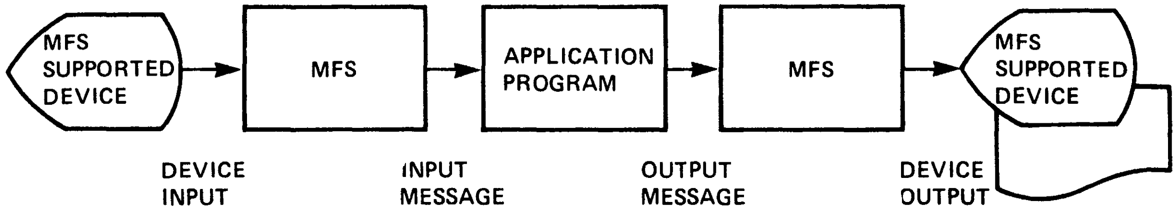


Figure 3-10. Message Formatting Using MFS

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. 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.

<u>Notes</u>:

1. The DIF and the DOF control blocks are generated as the result of the format (FMT) statement.
2. The MID and the MOD are generated as a result of different message (MSG) statements.
3. The initial formatting of a 3270 display is done via the "/FORMAT modname" command. This will format the screen with the specified MOD, as if a null message was sent.

Figure 3-11 provides an overview of the MFS operations.

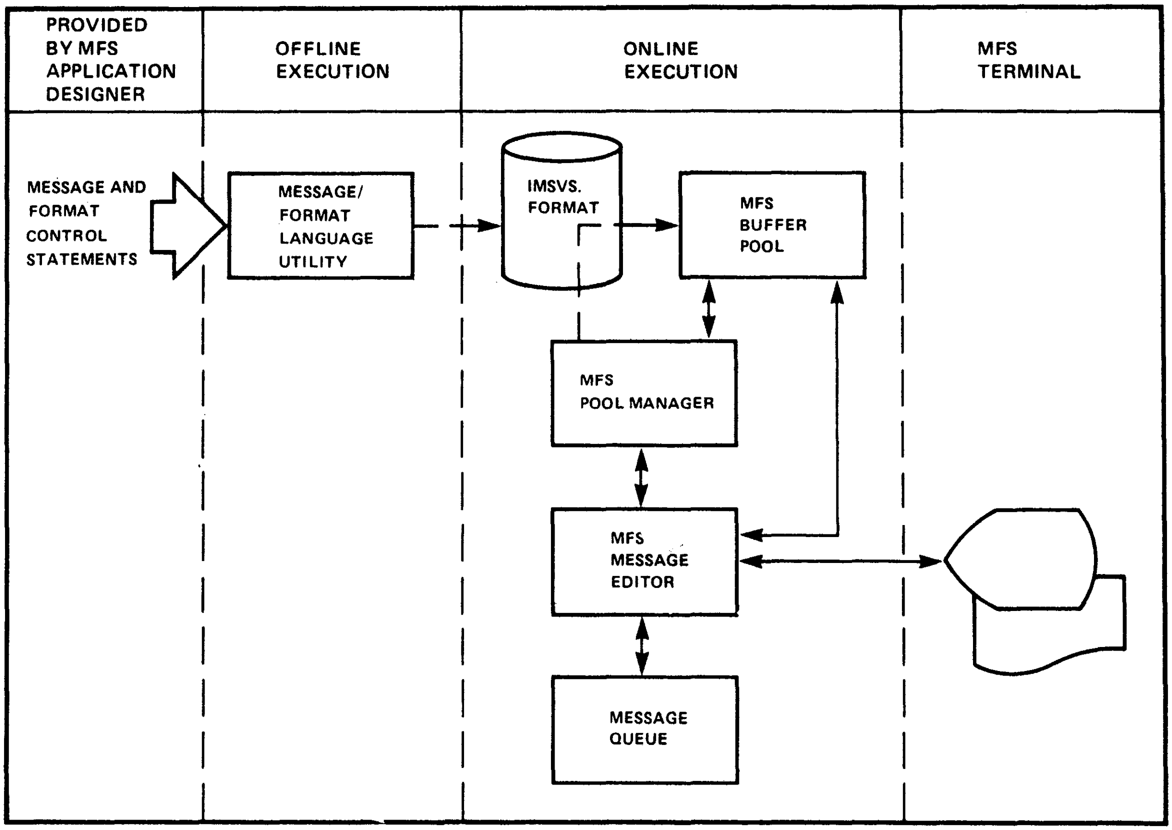


Figure 3-11. Overview of Message Format Service

MFS AND THE 3270

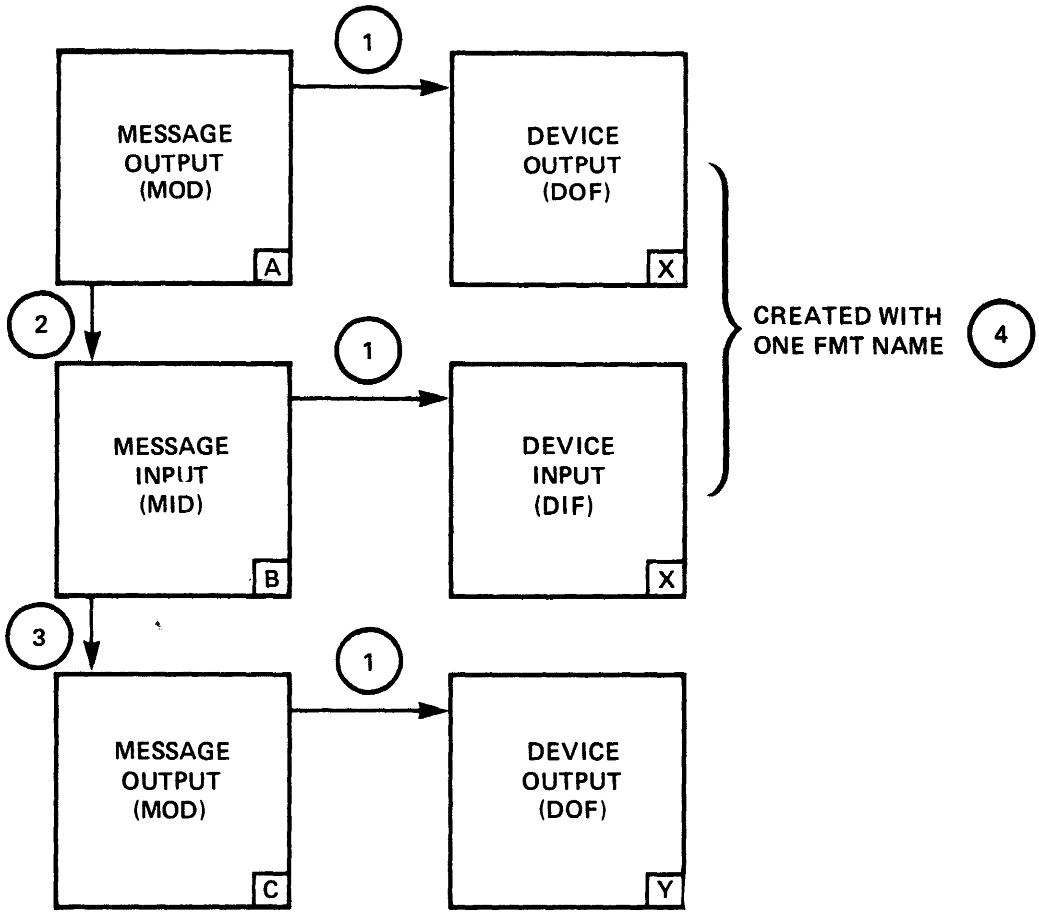
IMS/VS Message Format Service (MFS), described in the previous section, is always used to format data transmitted between IMS/VS and the devices of the 3270 information display system. MFS provides a high level of device independence for the application programmers and a means for the application system designer to make full use of the 3270 device capabilities in terminal operations. Although our subset only considers the 3270, its use of MFS is such that it is open-ended to the use of other MFS supported terminals when required. See the *IMS/VS General Information Manual* for a list of these terminals.

RELATIONSHIP BETWEEN MFS CONTROL BLOCKS

Several levels of linkage exist between MFS control blocks, as described in the following sections.

MFS Control Block Chaining

Figure 3-12 shows the highest-level linkage, that of chained control blocks.



Legend:

1. This linkage must exist.

2. If the linkage does not exist, device input data from 3270 devices is not processed by MFS. It is always used in our subset.

3. This linkage is provided for application program convenience. It provides a MOD name to be used by IMS/VS if the application program does not provide a name via the format name option of the insert call. The default MOD, DFSM02, will be used if none is specified at all, or if the input is a message switch to an MFS-supported terminal.

4. The user-provided names for the DOF and DIF used in one output/input sequence are normally the same. The MFS language utility alters the internal name for the DIF to allow the MFS pool manager to distinguish between the DOF and DIF.

   The direction of the linkage allows many message descriptions to use the same device format if desired. One common device format can be used for several application programs whose output and input message formats, as seen at the application program interface, are quite different.

Figure 3-12. Chained Control Block Linkage

Linkage between DFLD and MFLD

Figure 3-13 shows the second level of linkage, that between message fields and device fields. The arrows show the direction of reference in the MFS language utility control statements, not the direction of data flow.

References to device fields by message fields need not be in any particular sequence. An MFLD need not refer to any DFLD, in which case it simply defines space in the application program segment to be ignored if the MFLD is for output, and padded if the MFLD is for input. Device fields need not be referenced by message fields, in which case they are established on the device, but no output data from the output message is transmitted to them. Device input data is ignored if the DFLD is not referenced by an input MFLD.

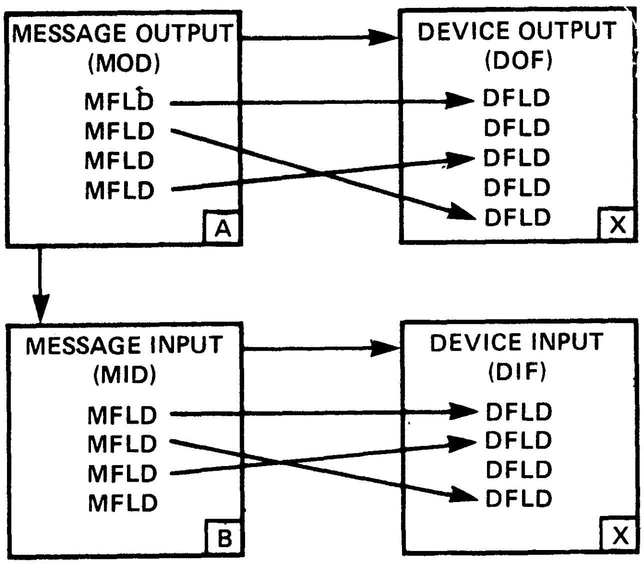


Figure 3-13. Linkage between Message Fields and Device Fields

Linkage between LPAGE And DPAGE

Figure 3-14 shows a third level of linkage, one which exists between the LPAGE and the DPAGE.

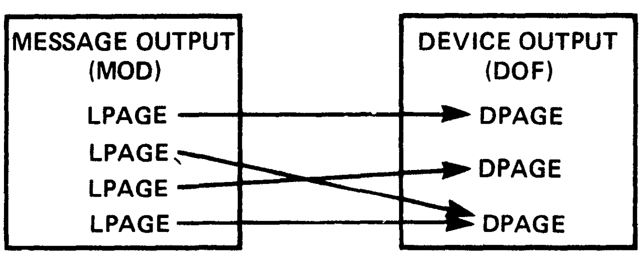


Figure 3-14. LPAGE -- DPAGE Linkage

The LPAGE in the MOD must refer to a DPAGE in the DOF. However, all DPAGEs need not be referred to from a given MOD.

Because we will always have single segment input in our subset, the defined MFLDs in the MID may refer to DFLDs in any DPAGE. But input data for any given input message from the device is limited to fields defined in a single DPAGE.

Optional Message Description Linkage

Figure 3-15 shows a fourth level of linkage. It is optionally available to allow selection of the MID based on which MOD LPAGE is displayed when input data is received from the device.

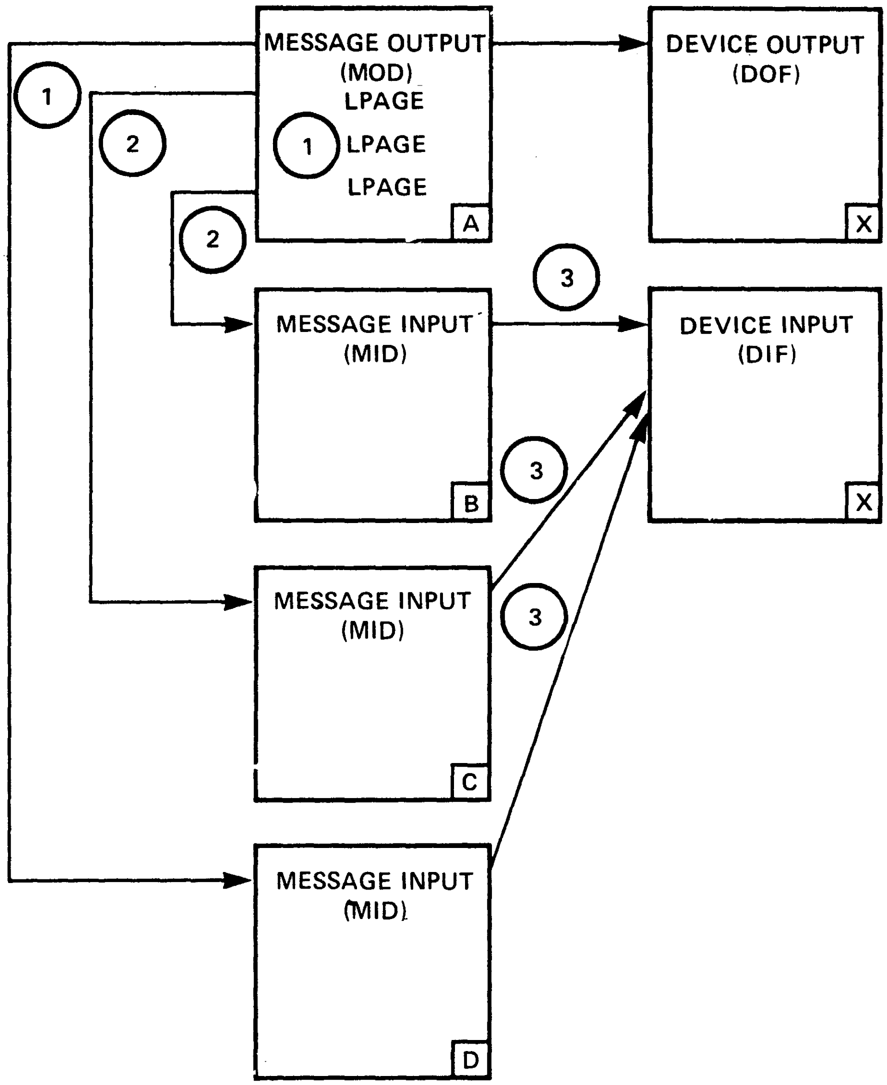


Figure 3-15. Optional Message Description Linkage

Legend:

1. The next MID name provided with the MSG statement is used if no name is provided with the current LPAGE.

2. If a next MID name is provided with the current LPAGE, input will be processed using this name.

3. For 3270 devices, all MIDs must refer to the same DIF. This is the same user-provided name used to refer to the DOF when the MOD was defined.

3270 Device Considerations Relative to Control Block Linkage

Since output to 3270 display devices establishes fields on the device using hardware capabilities, and field locations cannot be changed by the operator, special linkage restrictions exist. Because formatted input can only occur from a screen formatted by output, the IPAGE and physical page description used for formatting input is always the same as that used to format the previous output. The MFS language utility enforces this restriction by ensuring that the format name used for input editing is the same as the format name used for the previous output editing. Furthermore, if the DIF corresponding to the previous DOF cannot be fetched during online processing, an error message is sent to the 3270 display.

MFS FUNCTIONS

The following sections contain a description of the basic MFS functions.

INPUT MESSAGE FORMATTING

All device input data received by IMS/VS is edited before being passed to an application program. The editing is performed by either IMS/VS basic edit or MFS. This section describes the input message editing performed by MFS. It tells how the use of MFS is determined and how, when MFS is used, the desired message format is established based on the contents of two MFS control blocks -- the device input format (DIF) and the message input descriptor (MID).

All 3270 devices included in an IMS/VS system use MFS. The 3270s always operate in formatted mode except when first powered on, after the CLEAR key has been pressed, or when the MOD used to process an output message does not name a MID to be used for the next input data. While in unformatted mode, you can still enter commands and transactions, but they will not be formatted by MFS.

Input Data Formatting Using MFS

Input data from terminals in formatted mode is formatted based on the contents of two MFS control blocks, the MID and the DIF. The MID defines how the data should be formatted for presentation to the application program and points to the DIF associated with the input device. See Figure 3-16.

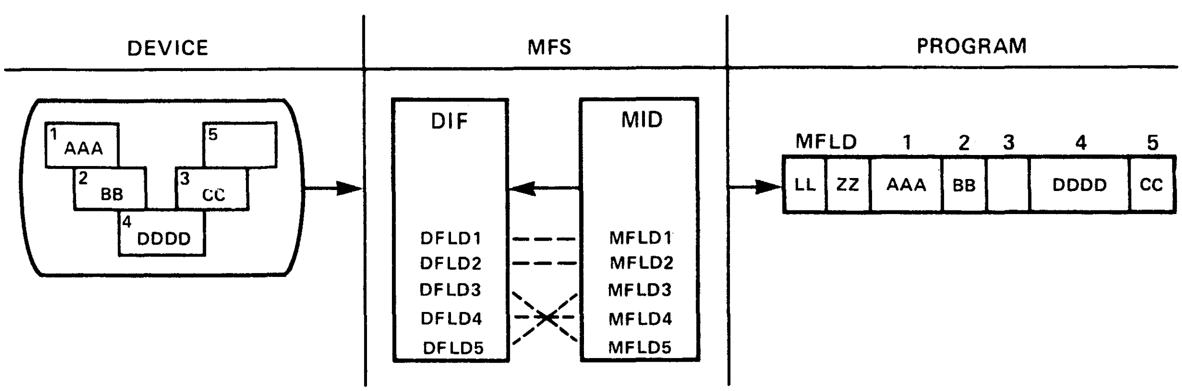


Figure 3-16. MFS Input Formatting

The MID contains a list of message descriptor fields (MFLDs) which define the layout of the input segment as is to be seen by an application program. The DIF contains a list of device descriptor fields (DFLDs) which define what data is to be expected from which part of the device (that is, the location on the screen). MFS maps the data of the DFLDs into the corresponding MFLDs. The application program is largely device independent because different physical inputs can be mapped into the same input segment.

MFLD statements are to define:

- The device fields (DFLDs) defined in the DIF which contents will be included in the message presented to the application program.
- Constants, defined as literals to be included in the message; a common use of literals is to specify the transaction code.

In addition, the MFLD statement defines:

- The length of the field expected by the application program.
- Left or right justification and the fill character to be used for padding the data received from the device.
- A 'nodata' literal for the MFLD if the corresponding DFLD does not contain any input data.

It should be noted that all message fields as defined by MFLD statements will be presented to the application program in our subset. Furthermore, there will always be only one input message segment, except for a conversational transaction, in which case the first segment presented to the program is the SPA. The SPA is never processed by MFS, however.

Input Message Field Attribute Data

Sometimes input messages are simply updated by an application program and returned to the device. In such a case, it may simplify message definition layouts in the MPP if the attribute data bytes are defined in the message input descriptor as well as in the message output descriptor.

Non-literal input message fields can be defined to allow for 2 bytes of attribute data. When a field is so defined, MFS will reserve the first 2 bytes of the field for attribute data to be filled in by the application program when preparing an output message. In this way, the same program area can be conveniently used for both input and output messages. When attribute space is specified, the specified field length must include the 2 attribute bytes.

IMS/VS Passwords

If the input data is for a password protected transaction, a device field should be designated for the password. The device field in which the operator keys in the password will not be displayed on the screen.

OUTPUT MESSAGE FORMATTING

All output messages for 3270 devices are processed by MFS in a way similar to input.

Output Data Formatting Using MFS

All MFS output formatting is based on the contents of two MFS control blocks -- the message output descriptor (MOD) and the device output format (DOF). See Figure 3-17. The MOD defines output message content and optionally, literal data to be considered part of the output message. Message fields (MFLDs) refer to device field locations via device field (DFLD) definitions in the DOF. The DOF specifies the use of hardware features, device field locations and attributes, and constant data considered part of the format.

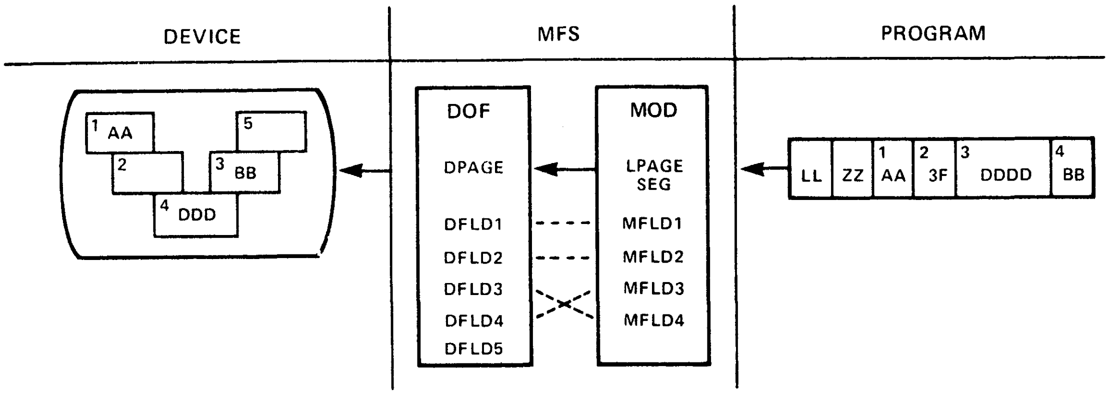


Figure 3-17. MFS Output Formatting

The layout of the output message segment to be received by MFS from the program is defined by a list of MFLDs in the MOD. The DOF in turn contains a list of DFLDs which define where the data is to be displayed/printed on the output device. MFS maps the data of the MFLDs into the corresponding DFLDs.

All fields in an output message segment must be defined by MFLD statements. Fields can be truncated or omitted by two methods. The first method is to insert a short segment. The second method is to place a NULL character (X'3F') in the field. Fields are scanned left (including the attribute bytes, if any) to right for a NULL character. The first NULL character encountered terminates the field. If the first character of a field is a NULL character, no data is sent to the screen for this field. This means that if the field is protected and the same device format is used, the old data remains on the screen. To erase the old data of a protected field the application program must send X'403F' to that field. Positioning of all fields in the segment remains the same regardless of NULL characters. Truncated fields are padded with a program tab character in our subset. Furthermore, we always specify erase-unprotected-all in the display device format. This erases all old data in unprotected fields on the screen.

Notes:

1. Device control characters are invalid in output message fields under MFS. The control characters HT, CR, LF, NL, and BS will be changed to null characters (X'CC'). All other nongraphic characters are changed to blanks before transmission. Graphic characters are X'40' through X'FE'.

2. With MFS, the same output message can be mapped on different device types with one set of formats. This will not be covered in our subset. The formatting discussed will cover one device type per device format, not a mixture. However, the mixture can be implemented later by changing the formats.

In addition to MFLD data, constants can be mapped into DFLDs. These constants are defined as literals in DFLD or MFLD statements.

Multiple-Segment Output Messages

MFS allows mapping of one or more output segments of the same message onto a single or multiple output screens. In our subset, we will limit ourselves to a one-to-one relationship between output message segments and logical output pages. Also, one logical output page is one physical output page (one screen).

Logical Paging of Output Messages

Logical paging is the way output message segments are grouped for formatting. When logical paging is used, an output message descriptor is defined with one or more LPAGE statements. Each LPAGE statement relates a segment produced by an application program to a corresponding device page.

Using logical paging, the simplest message definition consists of one LPAGE and one segment description. As shown in Figure 3-18, each segment produced by the application program is formatted in the same manner using the corresponding device page.

```
MSG                    Device          Application
Definition             Page            Program Output

LPAGE1--------------->DPAGE1           Segment 1
  SEG1
                                       or

                                       Segment 1
                                       Segment 1
                                       Segment 1
```

Figure 3-18. An Output Message Definition with one LPAGE

With the definition shown in Figure 3-18, each output segment inserted by the MPP will be displayed with the same and only defined MOD/DOF combination.

If different formats are required for different output segments, one LPAGE and SEG statement combination is required for each different format. Each LPAGE can link to a different DPAGE if desired. (This would not be required if only defined constants and MFLDs differ in the MOD.)

The selection of the LPAGE to be used for formatting is based on the value of a special MFLD in the output segment. This value is set by the MPP. If the LPAGE to be used cannot be determined from the segment, the last defined LPAGE is used. See also the description of the COND parameter of the LPAGE statement. Each LPAGE can refer to a corresponding DPAGE with unique DFLDs for its own device layout. See Figure 3-19.

MSG Definition	Device Page	Application Program Output
LPAGE1-------->DPAGE1		Segment 1* (LPAGE1 condition specified)
SEG1		

Figure 3-19. An Output Message Definition with Multiple Pages

Operator Paging Of Output Messages

If an output message contains multiple pages, the operator requests the next one with the program access key 1 (PA1). If PA1 is pressed after the last page is received, IMS/VS will send a warning message in our subset. If PA1 is then pressed again, IMS/VS will send the first page of the current output message again.

The operator can always request the next output message by pressing the PA2 key. Also, in our subset, when the operator enters data, the current output message is dequeued.

Output Message Literal Fields

Output message fields can be defined to contain literal data specified by the user during definition of the MOD. MFS will include the specified literal data in the output message before sending the message to the device.

MFS users may define their own literal field and/or select a literal from a number of literals provided by MFS. The MFS-provided literals are referred to as system literals and include various date formats, a time stamp, the output message sequence number, the logical terminal name, and the number of the logical page.

Output Device Field Attributes

Device field attributes are defined in DFLD statements. For 3270 display devices, specific attributes may be defined in the ATTR= keyword of the DFLD statement. If not, default attributes will be assumed. The message field definition (MFLD) corresponding to the device field (DFLD) may specify that the application program can dynamically modify the device field attributes.

When a field is so defined, the first 2 data bytes of the field are reserved for attribute data. Any error in the 2-byte specification causes the entire specification to be ignored, and the attributes defined or defaulted for the device field are used.

Note: The two attribute bytes should not be included in the length specification of the device field (DFLD) in the DOF.

The default attributes for non-literal 3270 display device fields are alphabetic, not-protected, normal display intensity, and not-modified. Literal device fields have forced attributes of protected and not-modified and default attributes of numeric and normal display intensity. Numeric protected fields provide an automatic skip function on display terminals.

Cursor Positioning

The positioning of the cursor on the 3270 display device is done in either of two ways:

1. The DPAGE statement defines the default cursor position.

2. The program can dynamically set the cursor to the beginning of a field via its attribute byte.

System Message Field (3270 Display Devices)

Output formats for 3270 display devices may be defined to include a system message field. If so defined, all IMS/VS messages except DFS057 REQUESTED FORMAT BLOCK NOT AVAILABLE are sent to the system message field whenever the device is in formatted mode. Providing a system message field avoids the display of an IMS/VS message elsewhere on the screen, thereby overlaying the screen data.

When MFS sends a message to the system message field, it activates the device alarm (if any) but does not reset modified data tags (MDTs) or move the cursor. Since an IMS/VS error message is an immediate response to input, MDTs remain as they were at entry and the operator merely has to correct the portion of the input in error.

In our subset we will always reserve the bottom line of the screen for the system message field. This field can also be used to enter commands, for example, /FORMAT.

Printed Page Format Control

The 3270 printer devices are also supported via MFS. Three basic options can be specified in the DEV statement (PAGE= operand):

- A defined fixed number of lines should always be printed for each page (SPACE). This is the recommended option because it preserves forms positioning.

- Only lines containing data should be printed. Blank lines are deleted (FLOAT).

- All lines defined by DFLDs should be printed, whether or not the DFLDs contain data (DEFN).

MFS FORMATS SUPPLIED BY IMS/VS

Several formats are included in the IMSVS.FORMAT library during IMS/VS system definition. They are used mainly for the master terminal, and for system commands and messages. All these formats start with the characters DFS. One of the most interesting in our subset is the default output message format. This format is used for broadcast messages from the master terminal and application program output messages with no MOD name specified. It permits two segments of input, each being a line on the screen. DFSDF2 is the format name, DFSMO2 the MOD and DFSMI2 the MID name.

When the master terminal format is used, any message whose MOD name begins with DFSMO (except DFSMO3) is displayed in the message area. Any message whose MOD name is DFSDSPC1 is displayed in the display area. Messages with other MOD names cause the warning message USER MESSAGE WAITING to be displayed at the lower portion of the display screen.

MFS CONTROL STATEMENTS

This section describes the control statements used by the MFS language utility. There are two major categories of control statements:

- Definition statements are used to define message formats and device formats.
- Compiler statements are used to control the compilation and listings of the definition statements.

The definition of message formats and device formats is accomplished with separate hierarchical sets of definition statements. The statement set used to define message formats consists of the following statements:

MSG	Identifies the beginning of a message definition.
LPAGE	Identifies a related group of segment/field definitions.
PASSWORD	Identifies a field to be used as an IMS/VS password.
SEG	Identifies a message segment.
MFLD	Defines a message field. Iterative processing of MFLD statements can be invoked by specifying DO and ENDDO statements. To accomplish interative processing, the DO statement is placed before the MFLD statement(s) and the ENDDO after the MFLD statement(s). See following discussion on compilation statements.
MSGEND	Identifies the end of a message definition.

The statement set used to define device formats consists of the following statements:

FMT	Identifies the beginning of a format definition.
DEV	Identifies the device type and operational options.
DIV	Identifies the format as input, output, or both.
DPAGE	Identifies a group of device fields corresponding to an LPAGE group of message fields.
DFLD	Defines a device field. Iterative processing of DFLD statements can be invoked by specifying DO and ENDDO statements. To accomplish iterative processing, the DO statement is placed before the DFLD statement(s) and the ENDDO after the DFLD statement(s). See the following discussion on compilation statements.

FMTEND	Identifies the end of a format definition.

Compilation statements have variable functions. The most common ones are:

DO	Requests iterative processing of MFLD or DFLD definition statements.
EJECT	Ejects SYSPRINT listing to the next page.
END	Defines the end of data for SYSIN processing.
ENDDO	Terminates iterative processing of MFLD or DFLD definition statements.
PRINT	Controls SYSPRINT options.
SPACE	Skips lines on the SYSPRINT listing.
TITLE	Provides a title for the SYSPRINT listing.

Compilation statements are to be inserted at logical points in the sequence of control statements. For example, TITLE could be placed first, and EJECT could be placed before each MSG or FMT statement.

RELATIONS BETWEEN SOURCE STATEMENTS AND CONTROL BLOCKS

In general, the following relations exists between the MFS source statements and control blocks:

- One MSG statement and its associated LPAGE, SEG, and MFLD statements generate one MID or MOD.
- One FMT statement and its associated DEV, DIV, DPAGE and DFLD statements generate one DIF and/or DOF. For displays, both the DIF and DOF are generated, because the output screen is used for input too.

In addition the MFS utilities will establish the linkages between the MID, MOD, DIF, and DOF. These are the result of the symbolic name linkages defined in the source statements.

Naming Conventions

The names of format blocks must be unique. The MID and MOD names, specified as the label of the MSG statement must be 1 to 8 alphanumeric characters. The DIF and DOF names are derived from the 1 to 6 alphanumeric character label of the FMT statement.

With reference to our naming convention in Chapter 1, we will use in the samples:

- OE4aaa for the FMT (DIF/DOF)
- OE4aaaIn for the MID
- OE4aaaOn for the MOD.

where:

aaa identifies the application

n   is a sequence number

UTILITY SYNTAX

The MFS language utility uses the syntax common to Assembler language. In addition, it should be noted:

- There is no limit to the number of continuation cards.
- There is no limit to the total number of characters in the operand field. Individual operand items cannot exceed 256 characters.
- Literal length restrictions do not include leading, trailing, and imbedded second quote characters.
- If a nonstandard character, such as a multipunch, is detected in a literal, a severity 4 warning message is issued.
- Positional parameters, if specified, must precede keyword parameters.

MFS DEFINITION STATEMENTS

Following is a detailed description of each of the MFS language definition statements. This description should be used as a reference when you are coding your own formats. You can skip this section at initial study. A coding sample is provided in Figure 3-21 at the end of this section.

MSG Statement

The MSG statement initiates and names a message input or output description.

```
 /--------------------------------------------------------------------
/          |           |                                              |
| label    |   MSG     |   [TYPE={INPUT }]                            |
|          |           |   [     {OUTPUT}]                            |
|          |           |                                              |
|          |           |   ,SOR=(formatname,IGNORE),CPT=2             |
|          |           |                                              |
|          |           |   [,NXT=msgdescriptionname]                  |
|          |           |----------------------------------------------|
|          |           |   FOR MSG TYPE=OUTPUT ONLY                   |
|          |           |                                              |
|          |           |   ,PAGE=YES                                  |
|          |           |                                              |
----------------------------------------------------------------------
```

label
    a 1- to 8-character alphameric name must be specified. This label may be referred to in the NXT operand of another message description. It is the name of the MID or MOD which are stored in the IMSVS.FORMAT library.

TYPE=
defines this description as message INPUT or OUTPUT. Default value is INPUT.

SOR=
formatname is the name of the FMT statement which, with the DEV and DFLD statements, defines the terminal data fields processed by this message description. IGNORE should be specified as shown in our subset.

OPT=2
should be specified as shown in our subset.

NXT=
specifies a message description to be used to map the next expected message as a result of processing a message using this message description. If TYPE=INPUT, NXT= specifies a message output description. In that case, the MOD can be overridden by the application program. If TYPE=OUTPUT, NXT= specifies a message input description.

If TYPE=OUTPUT and the formatname specified in the SOR= operand contains formats for 3270 and/or 3270P device types, the msgdescription name referred to by NXT=, (the message input description) must use the same formatname. This parameter should be coded if TYPE=OUTPUT.

PAGE=YES
should be specified as shown in our subset for all output message descriptions.

LPAGE Statement

The LPAGE statement defines a group of segments comprising a logical page. The LPAGE statement is optional and in our subset only applicable to output messages.

	LPAGE	SOR=dpagename
		[,COND=(mfldname,=,'value')]
		[,NXT=msgdescriptionname]

SOR=
specifies the name of the DPAGE statement that defines the display format for this logical page.

COND=
this parameter controls the selection of the message output formats to be used for each logical page occurrence. Mfldname must be the name of an MFLD defined in this LPAGE. The length of this MFLD must be equal to the length of the value literal. This parameter works as follows: If the content of the mfldname is equal to the specified value, then this LPAGE and its associated segment, field, and format description are used for formatting of the output message. A one character field with values A, B, C,..., etc., is recommended. Example: COND=(PAGETYPE,=,'A'), where PAGETYPE is a defined MFLD of one character in this LPAGE. If the conditional tests for all LPAGEs fail, the last defined LPAGE is used for formatting of the message.

NXT=
specifies the name of the message description to be used to map the next message if this logical page is processed. This name will override any NXT= name specified on the previous MSG statement.

PASSWORD Statement

The PASSWORD statement identifies a field to be used as an IMS/VS password. When used, the PASSWORD statement and its associated MFLD must precede the first SEG statement in a MSG definition. The total password length may not exceed 8 characters. The first 8 characters of data after editing will be used for the IMS/VS password.

```
  -------------------------------------------------------------
 /        |          |                                        |
/         | PASSWORD |   Blanks or comments                   |
|         |          |                                        |
---------------------------------------------------------------
```

SEG Statement

The SEG statement delineates message segments and is required only if multisegment message processing is used by the application program. Output message segments processed by MFS cannot exceed the logical record length of the long message queue data set. This maximum is in our subset 1388 bytes. Only one segment should be defined for TYPE=INPUT MSGs, and each LPAGE statement.

```
  -------------------------------------------------------------
 /        |          |                                        |
/         |   SEG    |                                        |
|         |          |                                        |
---------------------------------------------------------------
```

DO Statement

The DO statement causes repetitive generation of MFLD statements between the DO and ENDDO statements.

```
  -------------------------------------------------------------
 /        |          |                                        |
/         |    DO    |   count                                |
|         |          |                                        |
|         |          |   [,SUF={01    }]                      |
|         |          |   [     {number}]                      |
|         |          |                                        |
---------------------------------------------------------------
```

count
specifies how many times to generate the following MFLD statement(s). The maximum count that may be specified is 99.

SUF=
specifies the 2-digit suffix to be appended to the MFLD label and dfldname of the first group of generated MFLD statements. Default value is 01. MFS increases the suffix by 1 on each subsequent generation of statements.

If the specified suffix exceeds 2 digits, MFS uses the rightmost 2 digits.

If the specified count is such that the generated suffix eventually exceeds 2 digits, MFS reduces the count to the largest legitimate maximum value. For example, if count equals 8 and SUF=95, invalid suffixes of 100,101, and 102 would result. In this instance, MFS reduces the count to 5, processes the statement, and issues an error message.

MFLD STATEMENT

The MFLD statement defines a message field as it will be presented to an application program as part of a message input segment or received from an application program as part of a message output segment. At least one MFLD statement must be specified for each MSG description.

```
/[label] | MFLD | FOR MSG TYPE=INPUT
         |      |
         |      | [{dfldname                }]
         |      | [{'literal'               }]
         |      | [{(dfldname,'literal')    }]
         |      |
         |      | [,LTH=nn]
         |      |
         |      | [,JUST={L}]
         |      | [      {R}]
         |      |
         |      | [,ATTR={NO }]
         |      | [      {YES}]
         |      |
         |      | [      {C' '}]
         |      | [,FILL={NULL}]
         |      | [      {C'c'}]
         |      |------------------------------------------
         |      | FOR MSG TYPE=OUTPUT
         |      |
         |      | [{dfldname                        }]
         |      | [{(dfldname,'literal')            }]
         |      | [{(dfldname,system-literal)       }]
         |      |
         |      | [,LTH=nn]
         |      |
         |      | [,ATTR={NO }]
         |      | [      {YES}]
```

label
    a 1- to 8-character alphameric name may be specified. This label is required if it is referred to in the COND operand of the previous LPAGE statement. It may be used simply to uniquely identify this statement. If the MFLD is between the DO and ENDDO statements, this label should be restricted to 6 characters or less. DO statement processing appends a 2-digit suffix to the label and prints the label as part of the generated MFLD statement.

dfldname specifies the device field name (defined via the DEV or DFLD statement) from which input data is extracted or into which output data is placed. If this parameter is omitted when defining a message output descriptor, the data supplied by the application program is not displayed on the output device. If the repetitive generation function of MFS is used (DO and ENDDO statements), this dfldname should be restricted to 6 bytes maximum length. When each repetition of the statement is generated, a 2-character sequence number (01 to 99) is appended to the dfldname. If the dfldname specified here is greater than 6 bytes and repetitive generation is used, the dfldname is truncated at 6 characters and a 2-character sequence number is appended to form an 8-character name. No error message is provided if this occurs. This parameter may be specified in one of the following formats:

dfldname
: identifies the device field name from which input data is extracted or into which output data is placed.

'literal'
: may be specified if a literal value is to be inserted in an input message.

(dfldname,'literal')
: If TYPE=OUTPUT, this describes the literal data to be placed in the named DFLD. When this form is specified, space for the literal must not be allocated in the output message segment supplied by the application program. Normally, such a literal should be defined with a DFLD statement. Specifying it in the MOD allows different literal values (in different MODs) to be displayed with the same device format.

  If TYPE=INPUT, this describes the literal data to be placed in the message field when no data for this field is received from the device.

  In both cases, if the LTH= operand is specified, the length of the literal will be truncated or padded as necessary to the length of the LTH= specification. If the literal length is less than the defined field length, the literal is padded with blanks if TYPE=OUTPUT and with the specified fill character (FILL=) if TYPE=INPUT. If no fill character is specified for input, the literal is padded with blanks (the default value). The literal length may not exceed 256 characters.

(dfldname,system-literal)
: specifies a name from a list of system literals. A system literal functions like a normal literal except that the literal value is created during formatting prior to transmission to the device. The LTH= and ATTR= operands may not be specified. When this form is specified, space for the literal must not be allocated in the output message segment supplied by the application program.

The system literals and their associated length and format are:

SYSTEM LITERAL NAME	PRODUCES LITERAL OF LENGTH	FORMAT	COMMENTS
LTNAME	8	aaaaaaaa	See note 1
TIME	8	HH:MM:SS	
DATE1	6	YY.DDD	
DATE2	8	MM/DD/YY	
DATE3	8	DD/MM/YY	
DATE4	8	YY/MM/DD	
LPAGENO	4	nnnn	See note 2

Notes:

1. Messages generated by the IMS/VS control region in response to terminal input (error messages, most command responses) will use DFSM01 and have an LTNAME of blanks.

2. LPAGENO specifies that the current logical page number of the message be provided as a system literal. The literal produced will be a 4-digit number with leading zeros converted to blanks.

LTNAME is the logical terminal (LTERM) name of the LTERM for which this message is being formatted.

Note: In our subset, the first MFID in the MID must define the system message field used for command input. This complies with our IMS/VS Primer Remote Terminal Operator's Guide. The following MFID(s) in the MID must define the transaction code. This can be a literal, DFID(s), or a combination. The total length in the MID must be 9 characters, 8 for the transaction code, and one blank as delimiter. If necessary, the transaction code must be padded with blanks.

LTH=
: specifies the length of the field to be presented to an application program on input or received from an application program on output. Default or minimum value is 1 if it is not a literal. Maximum value is 8000. The maximum message length must not exceed 32767. In our subset, the maximum output segment length is 1388.

JUST=
: specifies that the input data field is to be left-justified (L) or right-justified (R) and right or left truncated as required, depending upon the amount of data expected by the device format descriptor. Default value is L. R is recommended for numeric fields and L for other fields.

ATTR=
: specifies whether (YES) or not (NO) the first 2 bytes of this field should be reserved for attribute data to be filled in by the application program (TYPE=OUTPUT). Default value is NO. Requests that can be made in the field attribute data are described in Chapter 4 under the topic "Dynamic Attribute Modification and Cursor Control." These two bytes must be included in the LTH= operand value. ATTR=YES is invalid if a literal value has been specified through the positional parameter in an output message.

FILL=

specifies a character to be used to pad this field when the length of the data received from the device is less than the length of this field. This character is also used to pad when no data is received for this field. This operand is only valid if TYPE=INPUT. Default value is blank.

C'c'

character 'c' will be used to fill fields. Recommended: Zero for numeric fields that are right justified, and blank for all others.

NULL

must be specified in our subset for the first MFLD, the system message field used for command input. This will completely suppress the field if no command input is received.

ENDDO Statement

The ENDDO statement terminates the group of MFLD statements that are to be repetively generated. The generated MFLD statements are printed immediately following the ENDDO statement.

```
 -------------------------------------------------------------------
 /          |         |                                             |
/           |  ENDDO  |  Blanks or comments                         |
|           |         |                                             |
--------------------------------------------------------------------
```

MSGEND Statement

The MSGEND statement terminates a message input or output description and is required as the last statement in the description.

```
 -------------------------------------------------------------------
 /          |         |                                             |
/           |  MSGEND |  Blanks or comments                         |
|           |         |                                             |
--------------------------------------------------------------------
```

FMT Statement

This statement delineates and names a device format which defines data formats as they are received from or displayed on specific devices. A device format is referred to by message descriptions to format input or output messages for an application program.

```
 -------------------------------------------------------------------
 /          |         |                                             |
/ label     |   FMT   |  Blanks or comments                         |
|           |         |                                             |
--------------------------------------------------------------------
```

label

a 1- to 6-character alphameric name must be specified. This name is referred to by message descriptions in the SOR= operand of MSG statements.

This name becomes part of the member name used for the resulting device output format and device input format blocks that are stored in the IMSVS.FORMAT library.

DEV Statement

The DEV statement defines device and data characteristics for a specific device type. The DFLD statements following this DEV statement are mapped using these characteristics until the next DEV or FMTEND statement is encountered. In our subset, we will not consider mixing of device types, so only one DEV statement per FMT should be coded.

```
        |  DEV  |  FOR 3270 DISPLAYS
        |       |
        |       |  TYPE=3270-An
        |       |
        |       |  ,FEAT=IGNORE,DSCA=X'00A0'
        |       |
        |       |  [,SYSMSG=dfldlabel]
        |       |--------------------------------------------
        |       |  FOR 3270 PRINTERS
        |       |
        |       |  TYPE=(3270P, 2 )
        |       |               1
        |       |
        |       |  ,FEAT=IGNORE
        |       |
        |       |  [,PAGE=([{55    }] [,{DEFN }])]
        |       |          [{number}]   {FLOAT}
        |       |                       {SPACE}
```

TYPE=

specifies the 3270 display screen size or printer model.

Based on the display screen size or printer model used, specify:

TYPE=	for	Screen size/Printer model
3270-A1		12x80
3270-A2		24x80
3270-A3		32x80
3270-A4		43x80
3270-A5		12x40
3270-A6		6x40
3270P,1		3284-1
		3284-3 attached to a 3275-1
		3286-1
3270P,2		3284-2
		3284-3 attached to a 3275-2
		3286-2
		3287
		3288
		3289

Notes:

1. The IBM 3270 Information Display System provides a hardware compatibility for displaying a small screen size format on a large screen display. A 24x80 screen format will be displayed on the top part of a 32x80 or 43x80 display, whether or not that display station was defined as a 24x80, a 32x80, or a 43x80 display type to IMS/VS. Also, a 12x40 screen format will be displayed on the top left part of a 12x80 display unit.

2. If you are an existing IMS/VS user, using TYPE=(3270,1) or TYPE=(3270,2), you do not need to change your existing formats. They are still valid and are equally subject to note 1 above.

FEAT=IGNORE
: should be specified as shown in our subset.

DSCA=X'00AC'
: should be specified as shown in our subset. It forces the erase unprotected all option.

SYSMSG=
: specifies the label of the DFLD statement(s) that defines the device field in which IMS/VS system messages are to be displayed. A DFLD with this label must be defined for each DPAGE. DFLDs for SYSMSG should normally be at least LTH=79 to prevent message truncation. We will always reserve the last line of each screen for this purpose. As we will also use this field for command input, it should not be protected (see the DFLD statement).

PAGE=
: number
  : defines the number of print lines on a printed page. This value is used for validity checking. The number specified must be greater than or equal to 1. Default value is 55.

  DEFN
  : specifies that lines are to be printed as defined by DFLD statements (no lines are to be removed or added to the output page).

  SPACE
  : specifies that each output page will contain the exact number of lines specified in the 'number' parameter. This is the recommended option.

  FLOAT
  : specifies that lines with no data (all blank or NULL) after formatting are to be deleted, that is, will not be printed.

DIV Statement

The DIV statement defines device formats within a device format description. The formats are identified as input, output, or both input and output. Only one DIV statement per DEV is allowed.

```
 ------------------------------------------------------------------
 /        |         |                                              |
/         |   DIV   |   TYPE={INOUT }                              |
|         |         |        {OUTPUT}                              |
|         |         |                                              |
 ------------------------------------------------------------------
```

TYPE=
    in our subset, INOUT should be specified for display (DEV TYPE=3270-An) and OUTPUT for printers (DEV TYPE=3270P).

DPAGE Statement

The DPAGE statement defines a physical page. This statement can be omitted if none of the message descriptors referring to this device format (FMT) contain LPAGE statements, and cursor position is under program control.

```
 -----------------------------------------------------------------
|  /          |           |                                       |
| / [label]   |  DPAGE    |   [CURSOR=((ll,cc))]                  |
|             |           |                                       |
 -----------------------------------------------------------------
```

label
    a 1- to 8-byte alphameric name may be specified. This name can be omitted if there are no message descriptions for this device format that contain LPAGE SOR= references, or if only one DPAGE statement is defined for the device.

CURSOR=
    specifies the position of the cursor on the screen. The value ll specifies line number and cc specifies column. Both ll and cc must be greater than or equal to 1. The default ll,cc value for DEV=3270-An is 1,2. Value 1,1 is invalid for screens.

    Note: Typically, the cursor position would be controlled by the program via the attribute byte in the required field. The cursor position via the DPAGE is used for initial formats, requested via the /FORMAT command.

DO Statement

The DO statement causes repetitive generation of DFLD statements between the DO and ENDDO statements. When DO is used, there are restrictions in the naming of DFLDs (refer to "DFLD Statement").

```
 -----------------------------------------------------------------
|  /          |           |                                       |
| /           |   DO      |      count                            |
|             |           |                                       |
|             |           |                                       |
|             |           |  [,{ 1              }]                |
|             |           |  [ {line-increment  }]                |
|             |           |                                       |
|             |           |  [,SUF={01    }]                      |
|             |           |  [     {number}]                      |
|             |           |                                       |
 -----------------------------------------------------------------
```

count
    specifies how many times to generate the statement(s).

line-increment
    specifies how much to increase the line position after the first cycle. The first cycle uses the lll value specified in the POS=

keyword of the DFLD statement. Default value is 1. The column position is not incremented in this way.

SUF=
: specifies the 2-digit suffix to be appended to the dfldname of the first generated DFLD statement. Default value is 01. MFS increments the suffix by one on each subsequent DFLD statement generation.

  If the specified suffix exceeds 2 digits, MFS uses the rightmost 2 digits.

  If the specified count is such that the generated suffix eventually exceeds 2 digits, MFS reduces the count to the largest legitimate maximum value. For example, if count equals 8 and SUF=95, invalid suffixes of 100, 101, and 102 would result. In this instance, MFS reduces the count to 5, processes the statement, and issues an error message.

DFLD Statement

The DFLD statement defines a field within a device format descriptor which is read from or written to the terminal. Only those areas which are of interest to the application program should be defined. Null space in the format need not and should not be defined.

```
+--------+--------+----------------------------------------------------+
| /      |        |  FOR DEV TYPE=3270-An                               |
|/       |        |                                                    |
| label  | DFLD   |  ['literal',]                                      |
|        |        |                                                    |
|        |        |  POS=(lll,ccc)                                     |
|        |        |                                                    |
|        |        |  [,LTH=nnn]                                        |
|        |        |                                                    |
|        |        |                                  [  { NORM   } ]   |
|        |        |  [,ATTR= ([{ALPHA}]  {NOPROT}    [, { NODISP } ]   |
|        |        |  [         [{NUM  }]  {PROT  }    [  { HI     } ]   |
|        |        |                                                    |
|        |        |          [,{NOMOD}])]                              |
|        |        |          [ { MOD }  ]                              |
|        |        |----------------------------------------------------|
|        |        |  FOR DEV TYPE=3270P                                |
|        |        |                                                    |
|        |        |  ['literal',]                                      |
|        |        |                                                    |
|        |        |  POS=(lll,ccc)                                     |
|        |        |                                                    |
|        |        |  [,LTH=nnn]                                        |
+--------+--------+----------------------------------------------------+
```

label
: a 1- to 8-character alphameric name may be specified. This label (dfldname) can be referred to by a message descriptor in transferring data to and from a terminal. If the repetitive generation function of MFS is used (DO and ENDDO statements), this dfldname should be restricted to 6 characters maximum length. When each repetition of the statement is generated, a 2-character

sequence number (01-99) is appended to the dfldname. If the dfldname specified here is greater than 6 characters and repetitive generation is used, the dfldname is truncated to 6 characters, and a 2-character sequence number is appended to form the 8-character name. No error message is provided if this occurs. The label should be omitted for literals.

'literal'
: specifies a literal character string to be presented to the device. The literal length cannot exceed 256 characters for 3270 display devices and the line width -1 for 3270 printer devices.

  For DEV TYPE=3270-An, literal fields will have the PROT attribute whether specified or not. The NUM attribute will be assumed if ALPHA is not specified. A literal field cannot be referred to by a message descriptor.

POS=
: defines the first data position of this field in terms of line (lll) and column (ccc). lll and ccc must be greater than or equal to 1.

  For a TYPE=3270-An device, POS=(1,1) cannot be specified. Fields cannot be defined such that they wrap from the bottom to the top of the display screen.

  A 3270 display screen cannot be copied when a field starting on line 1, column 2, has both alphabetic and protect attributes.

LTH=
: specifies the length of the field. The specified LTH= cannot exceed the physical page size of the device.

__Note__: POS= and LTH= do not include the attribute character position reserved for a 3270 display device. The inclusion of this byte in the design of display formats is necessary since it occupies the screen page position preceding each displayed field even though it is not necessarily accessible by an application program.

When defining DFLDs for 3270 printers, a hardware ATTRIBUTE character is not used. Therefore, fields may be defined with a juxtaposition that does not allow for the attribute character. However, the last column of a print line cannot be used. It is reserved for carriage control operations performed by IMS/VS.

ATTR=
: defines the display attribute of this field. This parameter is only applicable to displays in our subset. Attribute keywords may be specified in any order and only those desired need be specified. The underlined keywords need not be specified since they are defaults. Only one value in each vertical list may be specified.

  __ALPHA__
  NUM
  : specifies whether or not the field should have the numeric attribute. It is only relevant to input data. The numeric attribute specifies that the numeric lock feature and/or auto-skip features will be used. For a discussion of numeric lock and auto-skip, refer to __Operator's Guide for IBM 3270 Information Display Systems__, GA27-2742.

  __NOPROT__
  PROT
  : specifies whether or not the field is protected from operator modification. For literal fields, PROT is used and specification of NOPROT is ignored.

NORM
NODISP
HI
: specifies the field's display intensity as normal (NORM), high intensity (HI), or nondisplayable (NODISP).

NOMOD
MOD
: defines whether or not the field modified attribute should be assumed for this field. MOD causes the terminal to assume the field has been modified by the operator even though it was not. This should not be confused with the PROT attribute which prevents operator modification. MOD is ignored for literal fields.

Notes:

1. In general, device fields which should not be changed by the operator should have the PROT attribute. This avoids accidental change of such a field by the operator. Remember, the attribute bytes can be set by the MPP. Proper use of this can significantly reduce the number of different formats otherwise required to meet application needs.

2. The recommended attribute specification for the system message field (see DEV statement) is ATTR=HI. This will display these critical messages with high intensity and allow this field to be used for command input.

ENDDO Statement

The ENDDO statement terminates the group of DFLD statements that are to be repetitively generated. The generated DFLD statements are printed immediately following the ENDDO statement.

```
   ------------------------------------------------------------------
 /        |       |                                                  |
/         | ENDDO |                                                  |
|         |       |                                                  |
--------------------------------------------------------------------
```

FMTEND Statement

The FMTEND statement terminates a device format description and is required as the last statement in the device format description.

```
   ------------------------------------------------------------------
 /        |        |                                                 |
/         | FMTEND |                                                 |
|         |        |                                                 |
--------------------------------------------------------------------
```

COMPILATION STATEMENTS

The following compilation statements are the most frequently used ones and are included in the samples.

TITLE Statement

The TITLE statement is used to specify the heading to appear on the SYSPRINT listing.

```
 ------------------------------------------------------------------
  /        |  TITLE   |   'character sequence'                      |
 /         |          |                                             |
 |         |          |                                             |
 ------------------------------------------------------------------
```

'character sequence'
specifies the heading to be printed on the output listing.

PRINT Statement

The PRINT statement can be used to suppress the detailed output listing of the MFS language processor. It is recommended to include this statement at the beginning.

```
 ------------------------------------------------------------------
  /        |          |                                             |
 /         |  PRINT   |   NOGEN                                     |
 |         |          |                                             |
 ------------------------------------------------------------------
```

SPACE Statement

The SPACE statement specifies the number of lines to skip when output is printed. The SPACE statement is printed before the skip occurs.

```
 ------------------------------------------------------------------
  /        |          |                                             |
 /         |  SPACE   |      1                                      |
 |         |          |      number                                 |
 |         |          |                                             |
 |         |          |                                             |
 ------------------------------------------------------------------
```

1
number
specifies how many lines to skip after this statement is encountered. Default number is 1.

EJECT Statement

The EJECT statement is used to eject a page in an output listing. The EJECT statement is printed before the actual eject.

```
 ------------------------------------------------------------------
  /        |          |                                             |
 /         |  EJECT   |                                             |
 |         |          |                                             |
 ------------------------------------------------------------------
```

END Statement

The END statement must be used to define the end of the MFS input statements. It must be the last statement.

/	END	

Sample Formats

Figure 3-20 shows the sample formats for the Customer Name Inquiry and Address program. The lines illustrate the symbolic linkages between the different format blocks and fields. Figure 3-21 shows the screen layout of this format as printed by the MFS language utility. The complete set of these MFS source statements is included as member DE4CNI01 in IMSVS.PRIMESRC.

MOD

```
OE4CNIO1 MGS   TYPE=OUTPUT,                 *
               SOR=(OE4CNI,IGNORE),         *
               NXT=OE4CNII1,                *
               OPT=2
         LPAGE SOR=A1200
         SEG
         MFLD  CNUM,LTH=6
         MFLD  CNAM,LTH=20
         MFLD  CADR,LTH=20
         MFLD  CCTY,LTH=20
         MFLD  CPCD,LTH=6
         MFLD  ERROR,LTH=35
         MSGEND
```

MID

```
OE4CNII1 MSG   TYPE=INPUT,                  *
               SOR=(OE4CNI,IGNORE),         *
               NXT=OE4CNIO1,                *
               OTP=2
         LPAGE SOR=A1200
         SEG
TRANSACTION MFLD SYSMSG,LTH=79,FILL=NULL
   -->   MFLD  'TE4CNINQ',LTH=9
CODE     MFLD  CUSID,LTH=6
         MSGEND
```

DIF/DOF

```
OE4CNI   FMT
         DEV   TYPE=3270-A2,                            *
               DSCA=X'00A0',                            *
               SYSMSG=SYSMSG,                           *
               FEAT=IGNORE
         DIV   TYPE=INOUT
A1200    DPAGE CURSOR=((23,20))
         DFLD  'CUSTOMER INQUIRY',POS=(2,26)
         DFLD  'NUMBER',POS=(4,21)
CNUM     DFLD  POS=(4,33),LTH=6,ATTR=(HI,PROT,NUM)
         DFLD  'NAME',POS=(6,21)
CNAM     DFLD  POS=(6,33),LTH=20,ATTR=(HI,PROT,NUM)
         DFLD  'ADDRESS',POS=(8,21)
CADR     DFLD  POS=(8,33),LTH=20,ATTR=(HI,PROT,NUM)
         DFLD  'CITY',POS=(10,21)
CCTY     DFLD  POS=(10,33),LTH=20,ATTR=(HI,PROT,NUM)
         DFLD  'POSTAL CODE',POS=(12,21)
CPCD     DFLD  POS=(12,33),LTH=6,ATTR=(HI,PROT,NUM)
ERROR    DFLD  POS=(22,21),LTH=35,ATTR=(HI,PROT,NUM)
         DFLD  'ENTER CUSTOMER NO',POS=(23,2)
CUSID    DFLD  POS=(23,20),LTH=6,ATTR=HI
SYSMSG   DFLD  POS=(24,2),LTH=79,ATTR=(HI)
         FMTEND
         END
```

Figure 3-20. Format Language Statement Sample

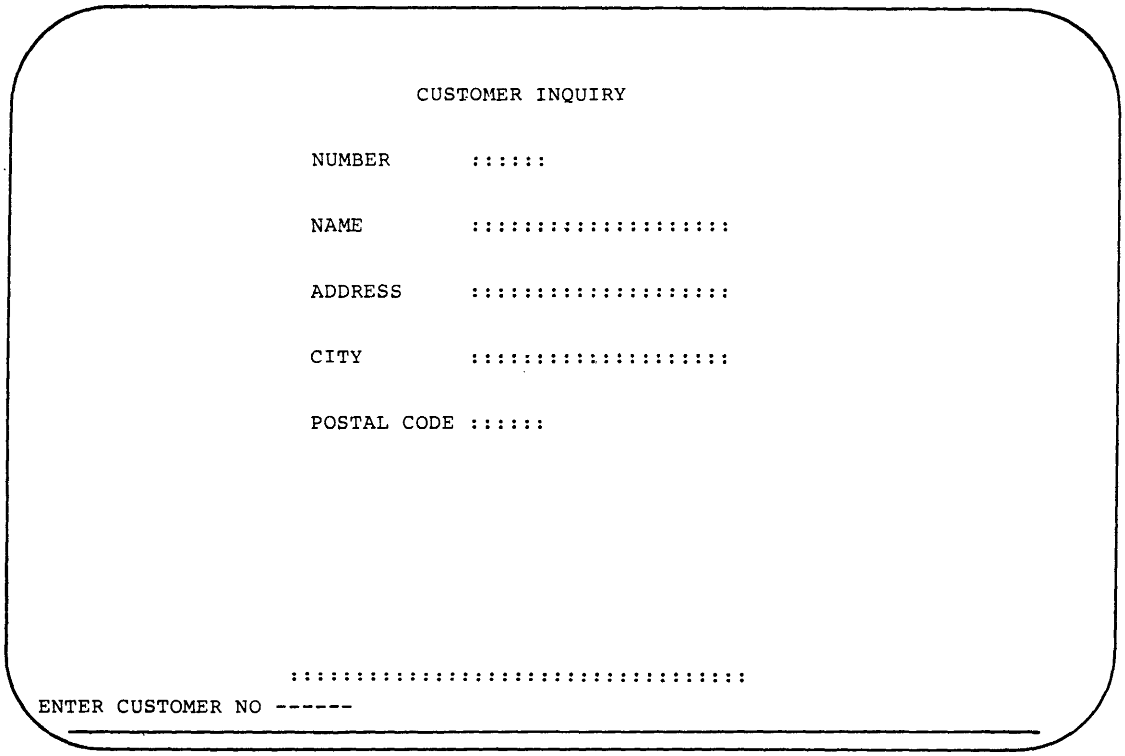


Figure 3-21. Sample Display Format

MFS CONTROL BLOCK GENERATION

MFS control blocks are generated by execution of the MFS language utility program. This is a two-stage process. See Figure 3-22.

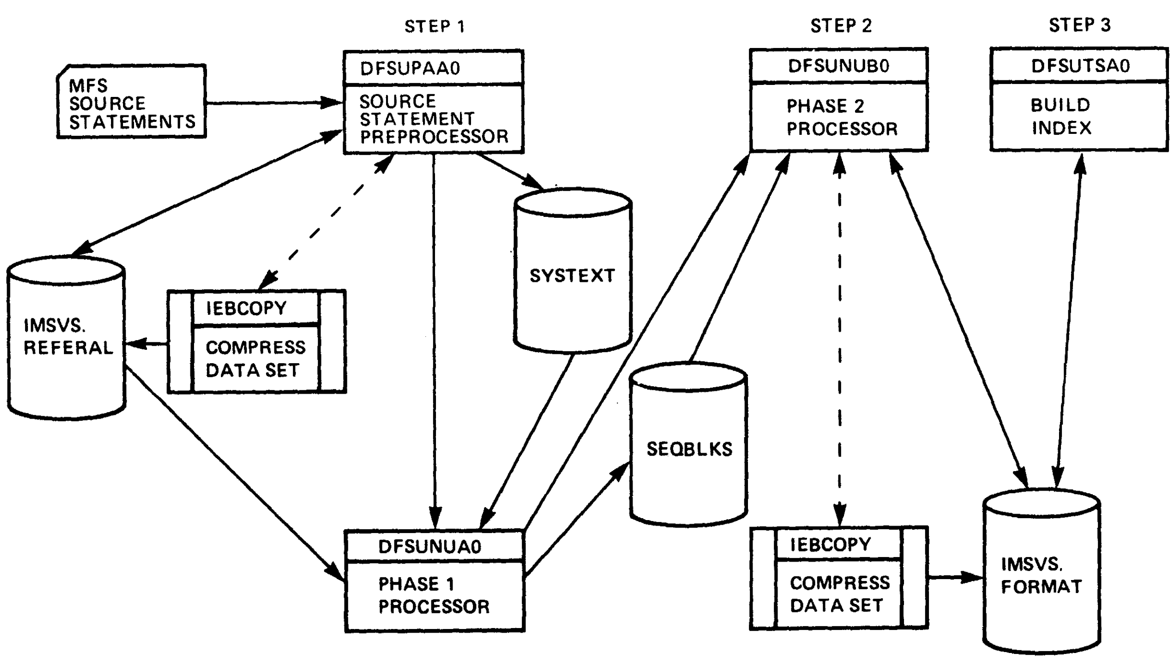


Figure 3-22. Creation of MFS Control Blocks

The MFS control block generation can be executed by an IMS/VS supplied catalogued procedure: MFSUTL. For a description of this procedure, see Chapter 7, "Installing IMS/VS." Multiple formats can be generated with one execution. In general you would process a complete format set, i.e, the related message and format descriptions, in one execution of MFSUTL. Sample job //SAMP425 in IMSVS.PRIMEJOB shows the use of this procedure for our sample applications. Three executions of MFSUTL are involved to process the three sample format sets.

STEP 1

Preprocessor

The MFS language utility preprocessor generates intermediate text blocks (ITBs), based on the MFS language source statements. Definitions of the MFS language utility source input are contained in this chapter under the topic "MFS Control Statements." The primary function of the preprocessor is to perform syntax and relational validity checks on user specifications and generate ITBs. The ITBs are then processed by phase 1 of the utility to generate message (MSG) and format (FMT) descriptors. An ITB generated for each MSG and FMT source definition processed and is stored in the historical reference library, IMSVS.REFERAL. An ITB for a particular MSG or FMT description can be re-used by the same or another format set, once it has been successfully added to the IMSVS.REFERAL data set. Each such description must start with a MSG or FMT statement and end with a MSGEND or FMTEND statement.

Phase 1

The preprocessor invokes phase 1 if the highest return code generated by the preprocessor is less than 16. Phase 1 places the newly constructed descriptors on the SEQBLKS data set. Each member processed has a control record placed on the SEQBLKS data set identifying the member, its size, and the date and time of creation. This control record is followed by the image of the descriptor as constructed by phase 1. Alternatively, if an error is detected during descriptor building, an error control record is placed on the SEQBLKS data set for the description in error, identifying the member in error, and the date and time the error control record was created. In addition, phase 1 returns a completion code of 12 to OS/VS. If execution of step 2 is forced, phase 2 will delete descriptors with build errors.

STEP 2

Phase 2

Phase 2 receives control as a job step following phase 1. After final processing, it will place the new descriptors into the IMSVS.FORMAT library. Phase 2 passes a completion code to OS/VS for step 2 based on all the descriptor maintenance to IMSVS.FORMAT for a given execution of the MFS language utility.

STEP 3

In our subset, we will always execute the MFS service utility after MFS control block generation. This utility will build a new index directory block which will eliminate the need for directory search operations during the IMS/VS online operation.

SAMPLE MFS GENERATION JOB

Job //SAMP425 in IMSVS.PRIMEJOB shows the JCL for the complete MFS control block generation process. This job uses the MFSUTL and MFSTVC procedures which are placed in IMSVS.PROCLIB during Stage 2 of IMS/VS system definition. The output generated by the second execution of the MFSUTL procedure and the MFSTVC procedure are listed in Chapter 3 of the *IMS/VS Primer Sample Listings*. This is the output of the processing of the customer order entry formats.

MFS LIBRARY MAINTENANCE

The IMSVS.FORMAT and IMSVS.REFERAL libraries are standard OS/VS partitioned data sets. Backup and restore operations can be done with the proper OS/VS utility (IEBCOPY). However, care must be taken that both the IMSVS.FORMAT and the IMSVS.REFERAL data sets are dumped and restored *at the same time*.

PSBGEN FOR MPPs AND BMPs

As for each DL/I batch program, a PSB is needed for each MPP or BMP. In addition to data base PCBs (see Chapter 2), a PSB for a MPP/BMP contains one or more data communication PCBs. The overall generation of the PSB remains as described in Chapter 2. However, there are a few additions to the data base PCB statements, and a new statement for the data communication PCB. In addition, you must perform an *application control blocks generation* (ACBGEN) for all DBDs and PSBs to be used by the CTL region. This is discussed at the end of this section.

ADDITIONAL PSB CODING CONVENTIONS

In addition to the PSB coding conventions stated in Chapter 2, the following rules must be observed.

- The name of the PSB as specified in the PSBNAME= keyword of the PSBGEN and the MBR= keyword in the PSBGEN procedure must be exactly the same as the program load module name in case of an MPP. This name is defined during IMS/VS system definition via the PSB= keyword of the APPLCTN statement.

- The order of the PCBs in the PSB must be:

    1. Data communication PCBs,

    2 Data base PCBs,

    3. GSAM PCBs (not allowed for MPPs).

- One data communication PCB is always automatically included by IMS/VS at the beginning of each PSB of an MPP or BMP. This default data communication PSB is used to insert output messages back to the originating LTERM.

    Note: By use of the CMPAT=YES keyword on the batch PSBGEN statement, we already provided this PCB to the batch program. In this way, a batch program can be run as a BMP without change. The relative positions of the data base PCBs remain the same.

THE DATA COMMUNICATION PCB

Besides the default data communication PCB, which does not require a PCB statement, additional PCBs can be coded. These PCBs are used to insert output messages to LTERMs other than the LTERM which originated the input message. A typical use of an _alternate_ PCB is to send output to a 3270 printer terminal.

The destination of the output LTERM can be set in two ways:

- During PSBGEN by specifying the LTERM name in an alternate PCB.
- Dynamically by the MPP during execution, by using a change call against a modifiable alternate PCB.

The method used depends on the PCB statement.

The PCB Statement

This is the only statement required to generate an alternate PCB (multiple occurrences are allowed). Its format is:

```
      ---------------------------------------------------------------
   /      |           |                                              |
  /       |           |                                              |
 /        |   PCB     |   TYPE=TP,{LTERM=name}                       |
|         |           |           {MODIFY=YES}                       |
|         |           |                                              |
-----------------------------------------------------------------------
```

Where:

TYPE=TP

is required for all alternate PCBs.

LTERM=name
MODIFY=YES

specifies this is a modifiable alternate PCB (MODIFY=YES) or a preset destination alternate PCB, where name specifies the output LTERM. MODIFY=YES is the basic recommendation.

Note: If MODIFY=YES is specified, the MPP must specify a valid alternate output LTERM with a change call before inserting any message via this PCB.

THE DATA BASE PCB

The data base PCB for an MPP or BMP is basically the same as discussed in Chapter 2. Two additional processing intent options can be specified with the PROCOPT=keyword of the PCB and/or SENSEG statement.

Additional Processing Intent Options

The PROCOPT= keyword is extended with two additional processing intent options, "O" and "E".

Their meanings are:

O - Read only; no dynamic enqueue is done by program isolation for calls against this data base. Can be specified with only the G intent option, as GO or GOP. This option is only valid for the PCB statement.

E - Forces exclusive use of this data base or segment by the MPP/BMP. No other program which references this data base/segment will be scheduled in parallel. No dynamic enqueue by program isolation is done, but dynamic logging of data base updates will be done. E can be specified with G, I, D, R, and A.

CAUTION: If the 'O' option (read-only) is used for a PCB, the data that is read should not be used as a basis for updating records in any data base. With this option, IMS/VS does not check the ownership of the segments returned. This means that the read-only user might get a segment that had been updated by another user. If the updating user should then abnormal terminate, and be backed out, the read-only user would have a segment that did not (and never did) exist in the data base. Therefore, the 'O' option user should not perform updates based on data read with that option. An ABEND can occur with PROCOPT=GO if another program updates pointers when this program is following the pointers. Pointers are updated during insert, delete and backout functions.

THE PSBGEN STATEMENT

This is basically the same as for a data base PCB. The IOEROPN= parameter must be omitted, the CMPAT=YES parameter is ignored.

EXAMPLE OF AN ONLINE PSB

Figure 3-23 shows an example of a online PSB. This PSB, PE4CORDR is to be used with the online customer order MPP. Its PSBGEN can be performed with job //SAMP401 (COBOL) or job //SAMP402 (PL/I) in IMSVS.PRIMEJOB. You should compare this PSB with the Phase 2 batch PSB, PE2CORDR, in Chapter 2.

```
*                          PROGRAM SPECIFICATION BLOCK FOR PHASE 4
*                          ORDER UPDATE PROGRAM PE4CORDR.
*
*                     ALTERNATE MESSAGE OUTPUT TERMINAL
*
         PCB    TYPE=TP,MODIFY=YES
*
*                     CUSTOMER DATABASE VIEW
*
         PCB    TYPE=DB,DBDNAME=BE2PCUST,PROCOPT=GO,KEYLEN=6
        SENSEG NAME=SE2PCUST
*
*                     ORDER DATABASE VIEW
*
         PCB    TYPE=DB,DBDNAME=BE4LORDR,KEYLEN=14
        SENSEG NAME=SE2ORDER,PROCOPT=AP
        SENSEG NAME=SE2OPART,PARENT=SE2ORDER,PROCOPT=A
        SENSEG NAME=SE2OSHIP,PARENT=SE2ORDER,PROCOPT=GI
*
*                     PARTS DATABASE VIEW
*
         PCB    TYPE=DB,DBDNAME=BE4LPART,KEYLEN=20
        SENSEG NAME=SE1PART,PROCOPT=GRP
        SENSEG NAME=SE1PSTOK,PARENT=SE1PART,PROCOPT=GR
*
*
*
        PSBGEN  LANG=COBOL,CMPAT=YES,PSBNAME=PE4CORDR
         END
```

Figure 3-23. Example of an Online PSB.

APPLICATION CONTROL BLOCK GENERATION (ACBGEN)

Before PSBs and DBDs can be used by the CTL region, they must be expanded to an internal control block format. This expansion is done by the Application Control Block generation (ACBGEN) utility. The expanded control blocks are maintained in the IMSVS.ACBLIB. This is a standard OS/VS partitioned data set. The OS/VS IEHMOVE and IEBCOPY utilities can be used for its maintenance.

JCL REQUIREMENTS

An ACBGEN procedure is placed in IMSVS.PROCLIB during IMS/VS system definition. Job //SAMP425 in IMSVS.PRIMEJOB shows how to use this procedure.

Note: Multiple BUILD statements can be coded for both DBDs and PSBs, but the ones for DBDs must be first.

Required Control Statements

The utility control statements for this program are free-form. A statement is coded as a card image and is contained in columns 1-71. The control statement may optionally contain a name starting in column 1. The operation field must be preceded by and followed by one or more blanks. The operand is composed of one or more PSB/DBD names and must be preceded by and followed by one or more blanks. Commas, parenthesis, and blanks can be used only as delimiting characters. Comments may be written following the last operand of a control statement, separated from the operand by one or more blanks. A control statement may be continued by inserting a comma after the last operand of the statement, inserting a non-blank character in column 72 and continuing the statement in column 16 of the next control statement. Columns 1-15 of the continuation statement must be blank.

In our subset, two control statements are required in the following order:

```
 -----------------------------------------------------------------
  /         |          |                                          |
 /          | BUILD    |  DBD=(dbdname,....)                      |
|           |          |                                          |
|           | BUILD    |  PSB=(psbname,....)                      |
|           |          |                                          |
 -----------------------------------------------------------------
```

DBD=
: must list all dbdnames of data bases used by the online IMS/VS system. Logical DBDs and GSAM DBDs must not be listed.

PSB=
: must list all psbnames of MPPs and BMPs; for example, those defined in APPLCTN statements during IMS/VS system definition. See Chapter 7.

ACBGEN Execution

Only one ACBGEN is required in case of multiple PSB/DBD changes as long as it is done after the last PSBGEN/DBDGEN and before the CTL region is started.

THE DATA COMMUNICATION DESIGN PROCESS

This part of Chapter 3 is complementary to the section "The Data Base Design Process" in Chapter 2. It is assumed that you have a clear understanding of Chapter 2 before reading this part.

We will distinguish between the following areas in the IMS/VS data base/data communication design process:

- Program design
- Message format service design
- Data base design

The data base design process is essentially the same as outlined in Chapter 2. We will not repeat this, but merely provide additional guidelines.

In the program design section, we will concentrate on the design of message processing programs (MPPs).

The MFS design will discuss the 3270 screen layouts and operator interaction.

Although we will cover each of the above areas in separate sections, it should be realized that they are largely dependent on each other. Therefore, an overall system design must be performed initially and an overall system review must follow the design phase of each section.

CONCEPTS OF ONLINE TRANSACTION PROCESSING

In an IMS/VS online environment, one can view a transaction from three different points:

- The application, that is, its processing characteristics and data base accesses.
- The terminal user.
- The IMS/VS system.

Each of the above constitutes a set of characteristics. A description of each set will follow.

Application Characteristics

From an application point of view, we can identify:

- Data collection with no previous data base access). This is not a typical IMS/VS application but can be part of an IMS/VS application system.
- Inquiry (data base retrieve-only processing). Inquiry/report programs like GIS/VS should be considered for this if inquiry is on a more or less ad hoc basis.
- Update. This normally involves data base reference and the subsequent updating of the data base. This is the environment of most IMS/VS applications.

In a typical IMS/VS multi-application environment, the above characteristics are often combined. However, a single transaction normally has only one of the above characteristics.

Terminal User Characteristics

From the terminal user's point of view, we distinguish:

- Single-interaction transactions.
- Multi-interaction transactions.

The single interaction transaction does not impose any dependency between an input message and its corresponding output, and the next input message. The multi-interaction transaction constitutes a dialogue between the terminal and the message processing program(s). Both the terminal user and the message processing program rely on a previous interaction for the interpretation/processing of a subsequent interaction.

IMS/VS Characteristics

From the IMS/VS system point of view, we distinguish:

- Non-response transactions.
- Response transactions.
- Conversational transactions.

Note: These IMS/VS transaction characteristics are defined for each transaction during IMS/VS system definition.

With non-response transactions, IMS/VS accepts multiple input messages (each being a transaction) from a terminal without a need for the terminal to first accept the corresponding output message, if any These non-response transactions will not be further considered in our sample.

With response transactions, IMS/VS will not accept further transaction input from the terminal before the corresponding output message is sent and interpreted by the user.

For conversational transactions, which are always response transactions, IMS/VS provides a unique scratch pad area (SPA) for each user to store vital information across successive input messages.

Transaction Response Time Considerations

In addition to the above characteristics, the transaction *response time* is often an important factor in the design of online systems. The response time is the elapsed time between the entering of an input message by the terminal operator and the receipt of the corresponding output message at the terminal. Two main factors constitute in general the response time:

1. The telecommunication transmission time, which is dependent on such factors as:
   - Terminal and network configuration
   - Data communication access method and data communication line procedure
   - Amount of data transmitted, both input and output
   - Data communication line utilization
2. The internal IMS/VS processing time, which is mainly determined by the MPP service time. The *MPP service time* is the elapsed time required for the processing of the transaction in the MPP region.

Chapter 9, "Optimization," contains a basic assessment of the above two factors in the section entitled "Data Communication Design Optimization."

Choosing the Right Characteristics

Each transaction in IMS/VS can and should be categorized by one characteristic of each of the previously discussed 3 sets.

Some combinations of characteristics are more likely to occur than others, but all of them are valid.

In general, it is the designer's choice as to which combination is attributed to a given transaction. Therefore, it is essential that this selection of characteristics is a deliberate part of the design process, rather than determined after implementation.

Following are some examples, based on our sample:

1. Assume an inquiry for the customer name and address with the customer number as input. The most straightforward way to implement this is clearly a non-conversational response-type transaction.

2. The entry of new customer orders could be done by a single response transaction. The order number, customer number, detail information, part number, quantity, etc., could all be entered at the same time. The order would be processed completely with one interaction. This is most efficient for the system, but it may be cumbersome for the terminal user because she or he has to re-enter the complete order in the case of an error.

Quite often, different solutions are available for a single application. Which one to choose should be based on a trade-off between system cost, functions, and user convenience. The following sections will highlight this for the different design areas.

ONLINE PROGRAM DESIGN

This design area is second in importance to data base design. We will limit the discussion of this broad topic to the typical IMS/VS environment. We will first discuss a number of considerations so that you become familiar with them. Next, we will discuss the design of the two online sample programs. You will notice that some discussions are quite arbitrary and may have to be adjusted for your own environment. Do remember, however, that our prime objective is to make you aware of the factors which influence these decisions.

Single Versus Multiple Passes

A transaction can be handled with one interaction or pass, or with two or more passes (a pass is one message in and one message out). Each pass bears a certain cost in line time and in IMS/VS and MPP processing time. So, in general, you should use as few passes as possible. Whenever possible you should use the current output screen to enter the next input. This is generally easy to accomplish for inquiry transactions, where the lower part of the screen can be used for input and the upper part for output. (See "Basic Screen Design" later in this chapter.) For update transactions, the choice is more difficult. The basic alternatives are:

One-Pass Update: After input validation, the data base updates are all performed in the same pass. This is the most efficient way from the system point of view. However, correcting errors after the update confirmation is received on the terminal requires additional passes or re-entering of data. An evaluation of the expected error rate is required.

Two-Pass Update: On the first pass, the input is validated, including data base access. A status message is sent to the terminal. If the terminal operator agrees, the data base will be updated in the second pass. With this approach, making corrections is generally much simpler, especially when a scratch pad area is used. However, the data base is accessed twice.

You should realize, that, except for the SPA, no correlation exists between successive interactions from a terminal. So, the data base can be updated by somebody else and the MPP may process a message for another terminal between two successive passes.

Multi-Pass Update: In this case, each pass does a partial data base update. The status of the data base and screen is maintained in the SPA. This approach should only be taken for complex transactions. Also, remember that the terminal operator experiences response times for each interaction. You also must consider the impact on data base integrity. IMS/VS will only back-out the data base changes of the current interaction in the case of program or system failure.

Notes:

1. IMS/VS emergency restart with a complete log tape will reposition the conversation. The terminal operator can proceed from the point where he or she was at the time of failure.

2. When a conversational application program terminates abnormally, only the last interaction is backed out.

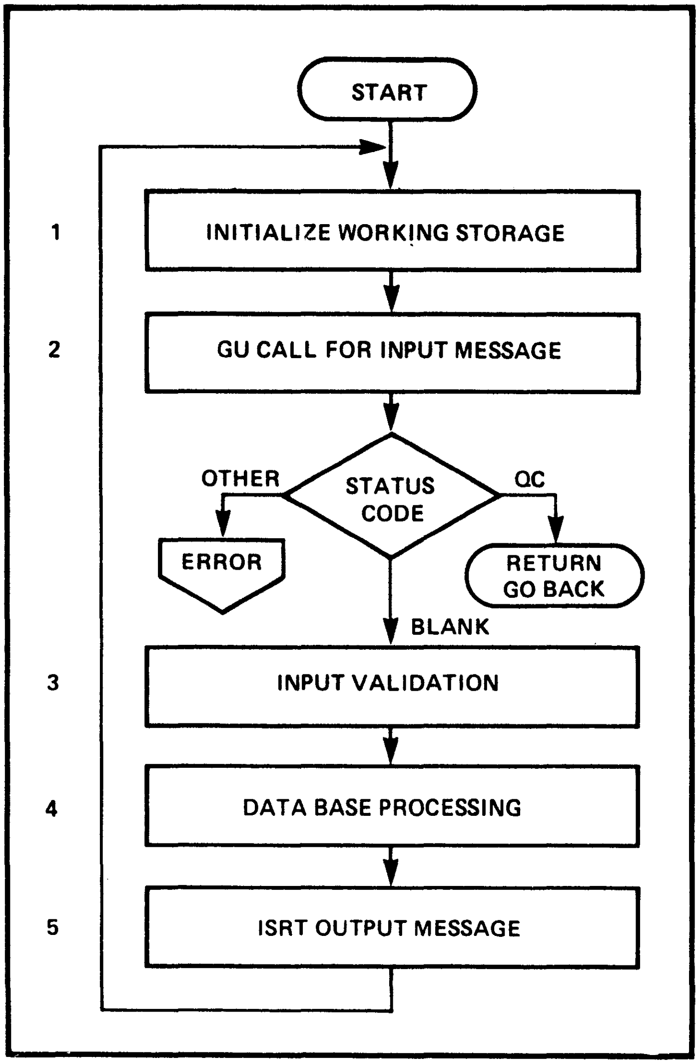
   The application must reposition the conversation after correction. For complex situations, IMS/VS provides an abnormal transaction exit routine. This is not covered in our subset.

Conversational Versus Non-Conversational

Conversational transactions are generally more expensive in terms of system cost than non-conversational ones. However, they give better terminal operator service. You should only use conversational transactions when you really need them. Also, with the proper use of MFS, the terminal operator procedures sometimes can be enhanced to almost the level of conversational processing. This will be discussed in the section about MFS Design.

General MPP Structure/Flow

Basically, the MPP processing can be divided into five phases. See Figure 3-24.



1. Initialization: The clearing of working storage, which may contain data left-over by the processing of a message from another terminal.

2. Retrieval of the SPA (optional) and the input message.

3. Input syntax check. All checks which can be done without accessing the data base, including a consistency check with the status of the conversation as maintained in the SPA.

4. Data base processing, preferably in one phase. This means that the retrieval of a data base segment is immediately followed by its update. Compare this to an initial retrieve of all required segments followed by a second retrieve and then update.

5. Output processing. The output message is built and inserted together with the SPA (only for conversational transactions).

<u>Note</u>: After finishing the processing of one input message, the program should go back to step 1 and request a new input message. If there are no more input messages, IMS/VS will return a status code indicating that. At that time, the MPP must return control to IMS/VS.

Figure 3-24. General MPP Structure and Flow

Transaction/Program Grouping

It is the designer's choice how much application function will be implemented by one transaction and/or program. The following considerations apply:

- Inquiry-only transactions should be separated from update transactions. These should be normally implemented as non-conversational transactions. Also, they can be defined as "non-recoverable inquiry-only" (See Chapter 7, "Installing IMS/VS," the TRANSACT macro). If, in addition, the associated MPPs specify PROCOPT=GO in all their data base PCBs, no dynamic enqueue and logging will be done for these transactions.

- Limited-function MPPs are smaller and easier to maintain. However, a very large number of MPPs costs more in terms of IMS/VS resources (control blocks and path lengths).

- Transactions with a long MPP service time (many data base accesses) should be handled by separate programs. Chapter 9, "Optimization," contains a discussion of MPP service time and its implications.

Note: IMS/VS provides a program-to-program message switch capability. This is not part of our subset. With this facility, you can split the transaction processing in two (or more) phases. The first (foreground) MPP does the checking and switches a message (and, optionally, the SPA) to a (background) MPP in a lower priority partition which performs the lengthy part of the transaction processing. In this way the foreground MPP is more readily available for servicing other terminals. Also, if no immediate response is required from the background MPP and the SPA is not switched, the terminal is more readily available for entering another transaction.

MESSAGE FORMAT SERVICE DESIGN

Basic Screen Design

Generally, a screen can be divided into five areas, top to bottom:

1. Primary output area, contains general, fixed information for the current transaction. The fields in this area should generally be protected.

2. Detail input/output area, used to enter and/or display the more variable part of the transaction data. Accepted fields should be protected (under program control); fields in error can be displayed with high intensity and unprotected to allow for corrections.

3. MPP error message area. In general, one line is sufficient. This can be the same line as 5 below.

4. Primary input, that is, requested action and/or transaction code for next input, and primary data base access information.

5. System message field, used by IMS/VS to display system messages and by the terminal operator to enter commands.

For readability, the above areas should be separated by at least one blank line. The above screen layout is a general one, and should be evaluated for each individual application. It is recommended to develop a general screen layout and set of formats to be used by incidental programs and programs in their initial test.

This can significantly reduce the number of format blocks needed and maintenance. In any case, installation standards should be defined for a multi-application environment.

MFS Subset Restriction

Our subset use of IMS/VS imposes the following major restrictions:

1. The maximum output length of a message segment is 1388 bytes; this is related to our long message record length of 1500 bytes.

2. A format is designated for one screen size. This can be later changed via additional MFS statements to support both screens and other devices with the same set of format blocks. A 1920 character format can be displayed on the top part of a 2560 or 3440 character display, and a 480 character format can be displayed on the top of a 960 character display.

3. A segment is one physical page, which is one logical page.

General Screen Layout Guidelines

The following performance guidelines should be observed when making screen layouts:

1. Avoid full-format operations. IMS/VS knows what format is on the screen. So, if the format for the current output is the same as the one on the screen, IMS/VS need not retransmit all the literals and unused fields.

2. Avoid unused fields, for example, undefined areas on the screen. Use the attribute byte (non-displayed) of the next field as a delimiter, or expand a literal with blanks. Each unused field causes additional control characters (5) to be transmitted across the line during a full-format operation.

   Note: This has to be weighed against user convenience. For example, our sample customer name inquiry format does not have consecutive fields but it is user convenient. Also, this application rarely needs a new format so we are not so much concerned with unused fields.

Including the Transaction Code in the Format

IMS/VS requires a transaction code as the first part of an input message. With MFS, this transaction code can be defined as a literal. In doing so, the terminal operator always enters data on a preformatted screen. The initial format is retrieved with the /FORMAT command. To allow for multiple transaction codes on one format, part of the transaction code can be defined as a literal in the MID. The rest of the transaction code can then be entered via a DFID. This method is very convenient for the terminal operator because the actual transaction codes are not of his concern. An example of such a procedure is shown in our sample customer order entry application.

DESIGN OF A SAMPLE INQUIRY TRANSACTION

The sample inquiry transaction we will consider in the following is the customer name inquiry, TE4CNINQ. It is a very simple transaction. Upon entry of the customer number, the program should retrieve the customer name and address and display it. If the customer number is not found in the data base, an error message should be sent. The design decisions are straightforward:

- Non-conversational. There is no need to correlate two successive interactions. In case of error, just enter a new customer number.

- The transaction can be defined as non-recoverable inquiry-only, because logging is not required. In case of system failure, the lost input or output is not important.

- One format can be used for both input and output. The output fields are protected. However, the customer number input field should not be protected, because it is to be used to enter another customer number. The transaction code is defined as a literal in the MID, thus avoiding the need for re-entering it each time that input is submitted. Switching to another application can be easily done by requesting another format via the system message field, the bottom line in our subset.

  The formats of this sample are included as member OE4CNIO1 in IMSVS.PRIMESRC. They can be generated with job //SAMP425 in IMSVS.PRIMEJCE. The programs, PE4CNINQ for COBOL and PE4PNINQ for PL/I are provided in IMSVS.PRIMESRC and can be compiled with jobs //SAMP441 and //SAM451 in IMSVS.PRIMEJOB, respectively. The remote terminal operator instructions can be found in the _IMS/VS Primer Remote Terminal Operators Guide_, Chapter 5.

DESIGN OF A SAMPLE UPDATE TRANSACTION

The sample update transaction chosen for this discussion is the entry of new customer orders, TE4CCNEW. There are two categories of data to be entered when adding a new customer order to the data base:

- Header information, such as customer number and customer order number.

- Detail information for each order line: The part number, part quantity, etc.

The data base processing of the program involves:

- Retrieval of the customer name and address for terminal operator verification and printing on a packing slip (optional).

- Insertion of the customer order root segment.

- For each order line, retrieval of the requested part segment and its associated stock segment for verification.

- Insertion of the order line segment and update of the stock information for that part.

The output sent to the terminal is the order confirmation and/or error messages. The error message can range from "customer unknown" to "not enough parts in stock". Also, upon request, a packing slip is printed on a 3270 terminal printer.

This transaction can be implemented in many ways. We will discuss first the most distinctive alternatives and then our choice.

Alternative 1, Single-Pass Update

With this alternative, the header and all detail lines are entered at the same time. If everything is correct, the program inserts the complete customer order and displays it on the terminal. In case of errors, the whole order must be re-entered if it is a non-conversational transaction. If conversational, the SPA can be used to store the correct order items and the necessary data base status information, to allow for convenient terminal operator corrections.

Alternative 2, Two-Pass Update

With this alternative, the input and data base are checked in the first pass; no data base updates are done. The order is stored in the SPA. The second pass is a confirmation of the terminal operator that he accepts the proposed data base update. Next, the actual data base update is done. With this approach, most errors will be detected before any data base updates are done. However, the number of data base accesses is higher than with alternative 1.

Also, all checking must be done again in the second phase, because the data base contents may have been changed in the meantime by another transaction.

Alternative 3, Multi-Pass Update

With this alternative, the order entry is done with multiple passes. The first pass checks and inserts the header information. Each successive pass checks and inserts one order line. This is a rather costly approach. Also, it is generally cumbersome for the terminal operator because he must wait for response after each order line is entered. This approach should only be used for very complex transactions with significant operator "think-time". Remember, also, that typically the error rate in predetermined transactions is quite low (<10%). So the normal operator procedures should be as smooth as possible. On the other hand, the error correction procedure could very well limit itself to one correction at a time, since the change on multiple errors is typically very low (<1%).

Which One to Choose

It is clear that alternative 1 should be the first choice from a system performance point of view. If the amount of data entered is significant, an SPA can be used to avoid the re-entering of all input data in case of errors. The correct part of the input could be kept in the SPA and should be displayed on the screen with a protected field. If there are errors, the field in error should be displayed with high-intensity and the cursor should be positioned on the first field to be corrected.

The basic principle behind this is:

"Do as much as you can immediately."

In our sample customer order entry program, we made a compromise. We selected alternative 3 but entering of the detail lines is in one pass. One reason for this is that we want to show a more elaborate use of the SPA. Remember that in a straightforward use of alternative 1, the SPA is only used in case of errors. Another reason is that we want to be able to enter the next order using the same screen format as used for the display of the last order. With the amount of data for the full order, this would require in any case an extra operator action to erase all unprotected fields.

Our Sample Conversational Program

This program, DFS4CNEW in IMSVS.PRIMESRC (renamed to PE4CORDR during linkage editing) has two basic passes:

1. The initial input is the customer number and the customer order number. The customer name is retrieved and stored in the SPA. The customer order root segment is inserted and the output, including customer name and address, is displayed.

2. The second input is all the orderlines. The order lines are processed completely (that is, checking, stock update and order line segment insert), one at a time.

   In case of error, pass 2 is repeated until all detail lines are correct. Already processed order lines are maintained. The SPA is used for keeping track of the status. In case the insert of an order line fails, the stock update is reversed. Remember, in case of abend, IMS/VS will backout all data base changes of only the current pass.

Optionally, a packing slip is produced. The following MFS considerations apply:

- The same format is used for both passes, both input and output, thus avoiding full screen formatting.

- The screen layout is such that upon completion of an order, the heading data for the next order (customer number and customer order number) can be entered on the very same screen; the cursor is already in place.

Miscellaneous Design Considerations

The following design considerations should also be noted:

- The conversation will be terminated (insert blank transaction code in SPA) after each successful order entry. This is transparent to the terminal operator, because the output format is linked to a MID which contains the transaction code, so the operator need not re-enter it.

- Each output message should contain all the data (except the MOD-defined literals) to be displayed. You should never rely on already existing data on the screen, because a clear or (re)start operation may have destroyed it.

ONLINE DATA BASE DESIGN

The transaction/data element approach for data base design as introduced in Chapter 2 is fully applicable to the IMS/VS online environment. We will not repeat it here but will extend it with some additional guidelines.

Using Secondary Indexes

Using secondary indexing can significantly increase the accessibility of online data bases. Therefore, a wider use of this facility is likely in the online data base design. However, its use in our subset is limited to the creation of additional non-root key access paths to the data base record.

Preferable Data Base Organization

Even more than for batch operations, the preferable data base organization for online data base operations is HDAM. HIDAM should be considered only if the online processing requirement is low and the requirement for key sequential batch processing is high.

Remember that a secondary index can be used for incidental key sequential access such as needed for generic search (search with a partial key).

Limitation of Online SHISAM Usage

IMS/VS will schedule any application programs with a delete or insert (PROCOPT=I, D, or A), against a SHISAM data base with the exclusive intent option. This implies, for instance, that if a BMP is delete or insert sensitive to a SHISAM data base, no MPP will be scheduled that references that SHISAM data base in its PSB.

Using an Intermediate Data Base

In some applications it may be necessary to use an _intermediate_ data base. Such a data base is used to store online transactions for later batch processing. The transaction processing is split in two phases:

1. The online part, in which the transaction is checked and verified using the online data bases. Accepted transactions are stored in an intermediate data base.

2. The batch part, in which the transactions in the intermediate data base are further processed via a batch program or BMP.

The main reason for this approach is generally the access requirements to non-DL/I files. Remember, GSAM is not available in MPP regions.

We recommend a simple structure for such an intermediate data base. The most straightforward implementation would be a root-only HDAM data base with a simple numeric root key, ranging between 1 and N (N=number of maximum expected transactions). In this situation, a simple linear randomizing module such as sample module DFSOALIN in IMSVS.PRIMESRC can be used. It would be more efficient to load the intermediate data base, periodically, with "empty" segments. A GHU + REPL call can then be used instead of an ISRT call.

There is one common problem with intermediate data bases and that is: "How does the MPP know what the next-to-be-used-root-key is?" The simplest solution is to have the latest used root key value in the first root segment of the data base. This value must then be updated by the MPP at the end of transaction processing, before a new GU to the message queue.

CHAPTER 4. DATA BASE PROCESSING

STRUCTURE OF THIS CHAPTER

This chapter is divided into two major sections -- Data Base Processing and Data Communication Application Programming. Both sections apply to the IMS/VS DB/DC user. If you are a DB-only user, however, you may skip the second section.

The section covering Data Base Processing is further divided into four parts. The first part deals with a general introduction to DL/I data base processing. It defines the basic structure of a DL/I application program. The second part introduces basic DL/I calls against a single hierarchical data base structure. It therefore uses the phase 1 sample environment. It also gives guidelines and samples for Assembler, COBOL, and PL/I application programs. However, the visualization of each DL/I call in particular is done following the COBOL syntax. The third part covers the processing of logical data bases which are implemented with the DL/I logical relationships function. The fourth part deals with the use of secondary indexes.

INTRODUCTION TO DATA BASE PROCESSING

In general, data base processing is transaction oriented. You should refer to "Concepts of Data Base Design" in Chapter 2, "Data Base Design," for a more detailed discussion of transactions and data bases. Generally, an application program accesses one or more data base records for each transaction it processes. There are two basic types of DL/I application programs:

- The direct access program
- The sequential access program

A direct access program accesses, for every input transaction, some segments in one or more data base records. These accesses are based on data base record and segment identification. This identification is essentially derived from the transaction input. Normally it is the root-key value and additional (key) field values of dependent segments. For more complex transactions, segments could be accessed in several DL/I data bases concurrently.

A sequential application program accesses sequentially selected segments of all or a consecutive subset of a particular data base. The sequence is usually determined by the key of the root-segment. A sequential program can also access other data bases, but those accesses are direct, unless the root-keys of both data bases are the same. Most sequential application programs are report programs, which list some part of the data base. For such programs, you should consider PL/I, the report feature of COBOL, or the more extended facilities of GIS/VS.

A DL/I application program normally processes only particular segments of the DL/I data bases. The portion that a given program processes is called an *application data structure*. This application data structure is defined in the *program specification block* (PSB). There is one PSB defined for each application program. An application data structure always consists of one or more hierarchical data structures, each of which is derived from a DL/I physical or logical data base.

PROGRAM STRUCTURE AND INTERFACE TO DL/I

LANGUAGE AND COMPILATION

Application programs are written in one of three languages: PL/I, COBOL, or Assembler Language. The program is compiled through the user-selected language compiler and is placed in the appropriate program library, after it is link-edited with the DL/I language interface module. In our subset we will only consider ANS COBOL with the OS ANS Version 4 or the OS/VS COBOL compilers or the PL/I optimizer compiler.

INTERFACE COMPONENTS

A DL/I batch application program executes in a manner similar to any other OS/VS job in a region/partition. It executes, however, under the control of DL/I. To perform the data base accesses as required by the application program, DL/I uses its own processing modules which in turn invoke OS/VS services. Also DL/I relies on the defined DBD and PSB control blocks to determine the data base organization and the program's access characteristics. Figure 4-1 presents an overview of DL/I and the application program during execution.

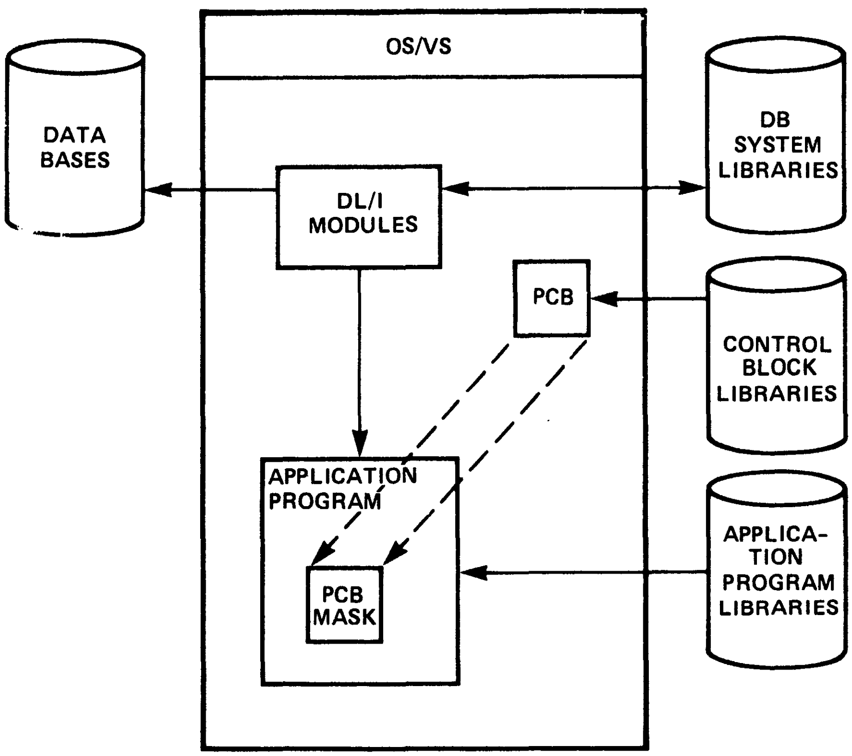


Figure 4-1. DL/I Interface with an Application Program

Before you execute an application program, a *program specification block generation* (PSBGEN) must be performed to create the *program specification block* (PSB) for the program. The PSB contains one PCB for each DL/I data base (logical or physical) the application program will access. The PCBs specify which segments the program will use and the kind of access (retrieve, update, insert, delete) the program is authorized to. The PSBs are maintained in an IMS/VS system library (IMSVS.PSBLIB). The coding and generation of PSBs is described in Chapter 2, "Data Base Design," of this manual.

During initialization, both the application program and its associated PSB are loaded from their respective libraries by the IMS/VS batch system. The DL/I modules, which reside together with the application program in one partition/region, interpret and execute data base CALL requests issued by the program.

The application program interfaces with DL/I via the following program elements:

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

The PCB mask(s) and I/O areas are described in the program's data declaration portion. Program entry, calls to DL/I, processing, and program termination are described in the program's procedural portion. Calls to DL/I, processing statements, and program termination may reference PCB mask(s) and/or I/O areas. In addition, DL/I may reference these data areas. Figure 4-2 illustrates how these elements are functionally structured in a program and how they relate to DL/I. The elements are discussed in the text that follows.

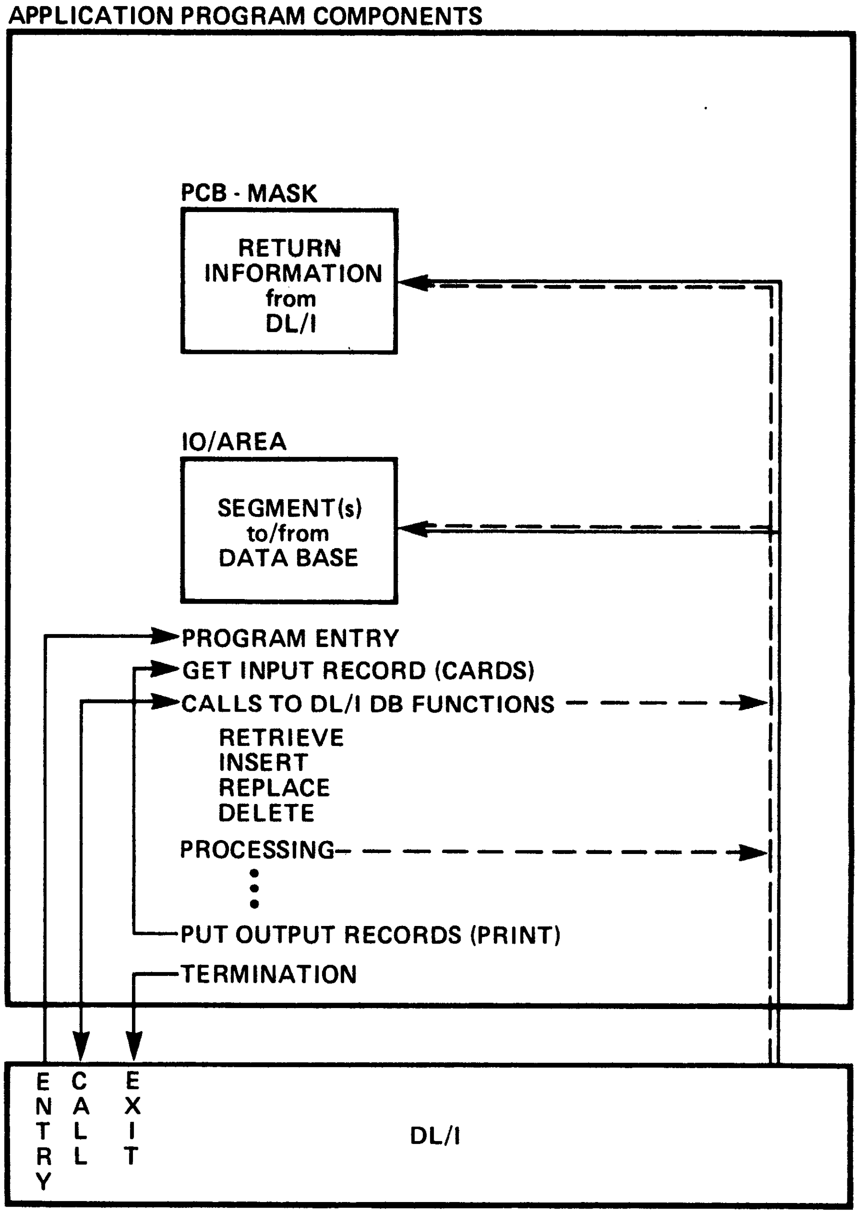


Figure 4-2. Structure of a Batch Application Program

Entry to Application Program

Referring to Figure 4-2, when the operating system gives control to the DL/I control facility, the DL/I control program in turn passes control to the application program (through the entry point as defined below). At entry, all the PCB-names used by the application program are specified. The order of the PCB-names in the entry statement must be the same as in the PSB for this application program. The sequence of PCBs in the linkage section or declaration portion of the application program need not be the same as the sequence in the entry statement.

Notes:

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

2. Programs that operate as 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.

PCB-Mask

A mask or skeleton data base PCB must be provided in the application program. The program views a hierarchical data structure via this mask. One PCB is required for each data structure. The details are shown in Figure 4-3.

As the PCB does not actually reside in the application program, care must be taken to define the PCB-mask as an Assembler dsect, a COBOL linkage section entry, or a PL/I based variable.

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

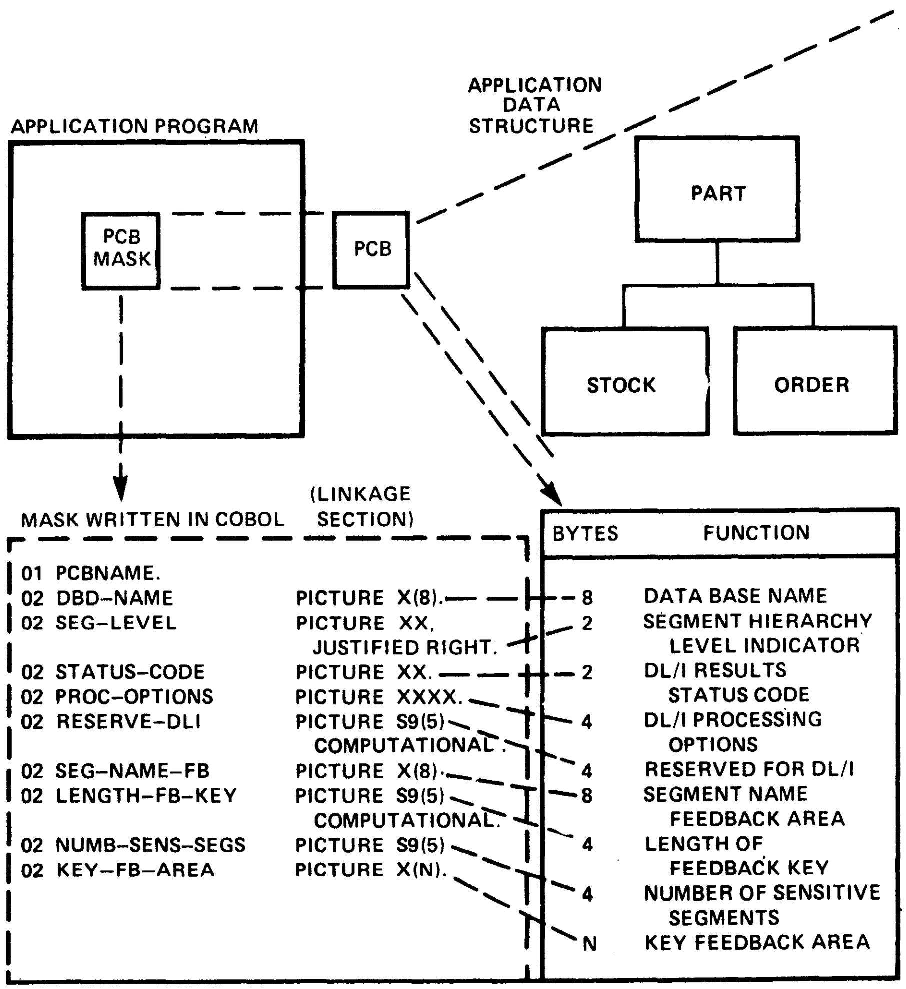


Figure 4-3. Application Program Data Base PCB Mask

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

1. Name of the PCB -- This is the name of the area which refers to the entire structure of PCB fields. It is used in program statements. This name is not a field in the PCB. It is the 01 level name in the COBOL mask in Figure 4-3.

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

3. Segment Hierarchy Level Indicator -- DL/I uses this area to identify the level number of the last segment encountered which satisfied a level of the call. When a retrieve is successfully completed, the level number of the retrieved segment is placed here. If the retrieve is unsuccessful, the level number returned is that of the last segment that satisfied the search criteria along the path from the root (the root segment level being '01') to the desired segment. If the call is completely unsatisfied, the level returned is '00'. This field contains character data; it is two bytes long and is a right-justified numeric value.

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

5. DL/I Processing Options -- This area contains a character code which tells DL/I the "processing intent" of the program against this data base (that is, the kinds of calls that may be used by the program for processing data in this data base). This field is four bytes long. It is left-justified. It does not change from call to call. It gives the default value coded in the PCB PROCOPT parameter (see Chapter 2), although this value may be different for each segment. DL/I will not allow the application to change this field, nor any other field in the PCB.

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

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

8. Length of Key Feedback Area -- This entry specifies the current active length of the key feedback area described below. This field is one fullword (4 bytes), binary.

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

10. Key Feedback Area -- DL/I places in this area the concatenated key of the last segment encountered which satisfied a level of the call. When a retrieve is successful, the key of the requested segment and the key field of each segment along the path to the requested segment are concatenated and placed in this area. The key fields are positioned from left to right, beginning with the root segment key and following the hierarchical path. When a retrieve is unsuccessful, the keys of all segments along the path to the requested segment, for which the search was successful, are placed in this area. Segments without sequence fields are not represented in this area.

    Note: This area is never cleared, so it should not be used after a completely unsuccessful call. It will contain information from a previous call. See Figure 2-5 for an explanation of concatenated keys.

Calls To DL/I

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

A call request is composed of a CALL statement with an argument list. The argument list specifies the processing function to be performed, the hierarchic path to the segment to be accessed, and the segment occurrence of that segment. One segment or multiple segments along the hierarchical path of segments may be operated upon with a single DL/I call. However, a single call never will return more than one occurrence of one segment type.

The arguments contained within any DL/I call request include:

- For PL/I, a field containing the number of call arguments in the statement, exluding itself
- The input/output function to be performed
- The PCB name
- The segment input/output work area
- The identification of the data segment(s) to be operated upon.

Following is a sample of a basic CALL statement for COBOL:

```
CALL 'CBLTDLI' USING function,PCB-name,I/OArea,SSA1,...,SSAn.
```

function
: identifies the DL/I function to be performed. This argument is the name of a four-character field which describes the desired I/O operation. The DL/I functions are described briefly below, and in full detail later in this chapter.

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

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

SSA1 through SSAn
are the names of Segment Search Arguments, and these are optional. There can be a maximum of 1 SSA per level along the hierarchic path being accessed. See the section "Segment Search Arguments" below.

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

Following are the basic DL/I call functions to request DL/I data base services:

Meaning	DL/I Call Function
GET UNIQUE	'GUbb'
GET NEXT	'GNbb'
GET HOLD UNIQUE	'GHUb'
GET HOLD NEXT	'GHNb'
INSERT	'ISRT'
DELETE	'DLET'
REPLACE	'REPL'

Note: b stands for blank; each CALL function is always 4 characters.

The above calls constitute four categories of segment access:

- Retrieve a segment: GU, GN, GHU, GHN
- Replace a segment: REPL
- Delete a segment: DLET
- Insert a segment: ISRT

In addition to the above data base calls, there are the system service calls. These are used for requesting systems services such as checkpoint/restart. All of the above calls and some basic system service calls will be discussed in detail in the following sections.

PCB-name Argument: "PCB-name" is the second (third in PL/I) argument in the CALL statement. It is the name of the PCB within the PSB that identifies for DL/I which specific hierarchical data structure the application program wishes to process.

I/O Work Area Argument: The I/O work area name is the third (fourth in PL/I) argument in the CALL statement. The work area is an area in the application program into which DL/I puts a requested segment, or from which DL/I takes a designated segment. If a common area is used to process multiple DL/I calls, it must be as long as the longest path of segments to be processed. The work area name points to the leftmost byte of the area. Segment data is always left-justified within a work area.

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

Note: It is a good practice to make the length of a general IOAREA large enough to accommodate future segment extensions. An installation standard could be set for this.

Segment Search Arguments: For each segment accessed in a hierarchical path, one SSA can be provided. The purpose of the SSA is to identify by segment name and, optionally, by a field value, the segment to be accessed.

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

- Narrow the field of search to a particular segment type, or to a particular segment-occurrence
- Request that either one segment or a path of segments be processed
- Alter DL/I's position in the data base for a subsequent call

Segment Search Argument (SSA) names represent the fourth (fifth for PL/I) through last arguments (SSA1 through SSAn) in the call statement. There can be 0 or 1 SSA per level, and, since DL/I permits a maximum of 15 levels per data base, a call may contain from 0 to 15 SSA names. In our subset, an SSA consist of one, two or three elements: The segment name, command code(s) and a qualification statement, as shown in the following diagram:

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

where:

SEGMENT NAME
The segment name must be eight bytes long, left-justified with trailing blanks as required. This is the name of the segment as defined in a physical and/or logical DBD referenced in the PCB for this application program.

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

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

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

Field Name
is the name of a field which appears in the description of the specified segment type in the DBD. The name is up to eight characters long, left-justified with trailing blanks as required. The named field may be either the key field

(preferably) or another data field within a segment. The field name is used for searching the data base, and must have been defined in the physical DBD.

RO = Relational Operator
is a set of two characters which express the manner in which the contents of the field, referred to by the field name, is to be tested against the comparative-value.

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

Note: As used above, the lowercase b represents a blank character.

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

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

Qualification

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

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

Example: SEGNAMEbb last character blank to unqualify

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

Example: SEGNAMEb(FIELDXXX>=value)

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

Command Codes

Both unqualified and qualified SSAs may contain one or more optional command codes which specify functional variations applicable to the call function or the segment qualification. The command codes are discussed in detail later in this chapter.

General Characteristics of segment search arguments

- An SSA may consist of the segment name only (unqualified). It may optionally also include one or more command codes and a qualification statement.
- SSAs following the first SSA must proceed down a hierarchical path. Not all SSAs in the hierarchical path need be specified. That is, there may be missing levels in the path. DL/I will provide, internally, SSAs for missing levels according to the rules given later in this chapter. However, it is strongly recommended to always include SSAs for every segment level.

Examples of SSAs will be given with the sample calls at each DL/I call discussion in the following section.

Termination

At the end of processing of the application program, control must be returned to the DL/I control program.

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

Warning: Since DL/I links to your application program, return to DL/I causes storage occupied by your program to be released. Therefore you should close all non-DL/I data sets for COBOL and Assembler before return, to prevent abnormal termination during close processing by OS/VS. PL/I automatically causes all files to be closed upon return.

STATUS CODE HANDLING

After each DL/I call, a two-byte status code is returned in the PCB which is used for that call. We distinguish between three categories of status codes:

- The blank status code, indicating a successful call
- Exceptional conditions and warning status codes, that is, valid status codes from an application point of view
- Error status codes, specifying an error condition in the application program and/or DL/I.

The grouping of status codes in the above categories is somewhat installation dependent. We will, however, give a basic recommendation after each specific call function discussion. It is also recommended that you use a standard procedure for status code checking and the handling of error status codes. The first two categories should be handled by the application program after each single call. Figure 4-4 gives an example.

```
CALL 'CBLTDLI' USING ....
IF PCB-STATUS EQ 'GE' PERFORM PRINT-NOT-FOUND.
IF PCB-STATUS NE 'bb' PERFORM STATUS-ERROR.
everything okay, proceed . . . . .
```

Figure 4-4. Testing Status Codes

Notice that it is more convenient to directly test the regular exceptions in-line instead of branching to a status code check routine. In this way, you clearly see the processing of conditions that you wish to handle from an application point of view, leaving the real error situations to a central status code error routine. A detailed discussion of the error status codes and their handling will be presented later in this chapter.

SAMPLE PRESENTATION OF A CALL

DL/I calls will be introduced in the following sections. For each call we will give samples. These samples will be in a standard format as shown in Figure 4-5.

```
77  GU-FUNC PICTURE XXXX VALUE 'GUbb'.

01  SSA001-GU-SE1PART.
    02  SSA001-BEGIN PICTURE ....
    02  ....
    02  ....

01  IOAREA   PICTURE X(256).
-----------------------------------------------------------------

CALL 'CBLTDLI' USING GU-FUNC,PCB-NAME,IOAREA,SSA001-GU-SE1PART.

-----------------------------------------------------------------

STATUS CODES:

    bb:  successful call
    --:  exceptional but correct condition
 other:  error situation
```

Figure 4-5. Sample Call Presentation

All the calls in the samples are presented in COBOL format. The coding of a call in PL/I or Assembler will be presented later. Each call example contains three sections. The first section presents the essential elements of working storage as needed for the call. The second part, the processing section, contains the call itself. Note that the PCB-NAME parameter should refer to the selected PCB defined in the Linkage Section. Sometimes we will add some processing function description before and/or after the call, in order to show the call in its right context. The third section contains the status codes and their interpretation, which can be expected after the call. The last category of status code, labeled "other: error situation," would normally be handled by a status code error routine. We will discuss these error status codes with the presentation of such a routine later in this chapter.

BASIC DATA BASE PROCESSING

DL/I POSITIONING CONCEPT

To satify a call, DL/I relies on two sources of segment identification:

- The established position in the data base as set by the previous call against the PCB
- The segment search arguments as provided with the call.

The data base position is the knowledge of DL/I of the location of the last segment retrieved and all segments above it in the hierarchy. This position is maintained by DL/I as an extension of, and reflected in, the PCB. When an application program has multiple PCBs for a single data base, these positions are maintained independently. For each PCB, the position is represented by the concatenated key of the hierarchical path from the root segment down to the lowest level segment accessed. It also includes the positions of non-keyed segments.

If no current position exists in the data base, then the assumed current position is the start of the data base. This is the first physical data base record in the data base. With HDAM this is not necessarily the root-segment with the lowest key value.

SAMPLE ENVIRONMENT

The phase 1 sample environment is used to exemplify the basic DL/I calls presented in the following sections. The data base used is the PARTS data base as shown in Figure 4-6.

```
                  ┌----------┐
                  |          |
                  | SE1PART  |
                  |          |
                  └----------┘
                       |
       ┌---------------|---------------┐
       |               |               |
┌----------┐     ┌----------┐     ┌----------┐
|          |     |          |     |          |
|SE1PSTOK  |     | SE1PPUR  |     |SE1PGDSC  |
|          |     |          |     |          |
└----------┘     └----------┘     └----------┘
```

Figure 4-6. The Phase 1 PARTS Data Base

The following programs are part of the IMS/VS Primer phase 1 sample application and are included in IMSVS.PRIMESRC:

- DFSOADBL, a data base load program written in Assembler,
- PE1CPINV (member DFS1CINV in IMSVS.PRIMESRC), a COBOL program which gives a parts inventory report for some (transaction TE1INVQ) or all (transaction TE1INVPP) of the parts in the PARTS data base,
- PE1CPPUR (member DFS1CPUR in IMSVS.PRIMESRC), a COBOL program which processes the purchase orders (transactions: TE1PONEW, TE1PCCNG, TE1PCDEL). This program utilizes GSAM and the batch checkpoint/restart function of DL/I.

For more details on these programs, you are referred to "Phase 2 Sample Requirements" in Chapter 2, "Designing Data Bases."

RETRIEVING SEGMENTS

There are two basic functions in retrieving a segment:

- Retrieve a specific segment: GU
- Retrieve the next segment in the hierarchy: GN

The Get Unique Call -- GU

The basic get unique (GU) call, function code 'GUbb', retrieves one segment in a hierarchical path. The segment retrieved is identified by an SSA for each level in the hierarchical path down to and including the requested segment. Each should contain at least the segment name. The SSA for the root-segment should provide the root-key value. Figure 4-7 shows an example of the get unique call.

The main use of the GU call is to position yourself to a data base record and obtain (a path of) segment(s). Typically, the GU call is used only once for each data base record you wish to access. Additional segments within the data base record would then be retrieved by means of get next calls (See the following section.) The GU call can also be used for retrieving a dependent segment, by adding additional SSAs to the call. For example, if you add a second SSA which specifies the stock location, you would retrieve a STOCK segment below the identified part. If the SSA did not provide a stock location number, this would be the first STOCK segment for this part.

```
77  GU-FUNC PICTURE XXXX VALUE 'GUbb'.

01  SSA001-GU-SE1PART.
  02  SSA001-BEGIN PICTURE X(19) VALUE 'SE1PARTb(FE1PGPNRb='.
  02  SSA001-FE1PGPNR PICTURE X(8).
  02  SSA001-END PICTURE X VALUE ')'.

01  IOAREA PICTURE X(256).
---------------------------------------------------------------------

MOVE PART-NUMBER TO SSA001-FE1PGPNR.

CALL 'CBLTDLI' USING GU-FUNC,PCB-NAME,IOAREA,SSA001-GU-SE1PART.
---------------------------------------------------------------------

STATUS CODES:

   bb:  requested PART segment has been moved to IOAREA
   GE:  segment not found; supplied part number not in data base
other:  error situation
```

Figure 4-7. Basic GU Call

The Get Next Call -- GN

The get next (GN) call, function code 'GNbb', retrieves the next segment in the hierarchy as defined in the PCB. To determine this next segment, DL/I relies on the previously established position.

Unqualified Get Next Call

```
 77  GN-FUNC PICTURE XXXX VALUE 'GNbb'.

 01  IOAREA    PICTURE X(256).
--------------------------------------------------------------------

 CALL 'CBLTDLI' USING GN-FUNC,PCB-NAME,IOAREA.
--------------------------------------------------------------------

 STATUS CODES:

    bb:  if previous call retrieved a PART, then a STOCK segment
         will be retrieved
    GK:  a segment is returned in IOAREA, but it is a different
         type at the same level, for instance, a PURCHASE
         ORDER segment after the last stock segment
    GA:  segment returned in IOAREA, but it is of a higher level
         than the last one, that is, a new PART segment
    GB:  end of data base reached, no segment retrieved
 other:  error situation
```

Figure 4-8. Unqualified Get Next Call

The above get next call with no SSAs at all will, if repeated, return the segments in the data base in hierarchical sequence. Only those segments are returned to which the program is defined sensitive in its PCB. If this call was issued after the get unique call of Figure 4-7, then it would retrieve the first STOCK segment for this part (if one existed). Subsequent calls would retrieve all other STOCK segments, PURCHASE ORDER, and DESCRIPTION segments for this part. After this, the next part would be retrieved and its dependent segments, etc., until the end of the data base is reached. Special status codes will be returned whenever a different segment type at the same level or a higher level is returned. No special status code is returned when a different segment at a lower level is returned. You can check for reaching a lower level segment type in the segment level indicator in the PCB. Remember, only those segments to which the program is sensitive via its PCB are available to you.

Although the above unqualified GN call may be efficient, especially for report programs, you should use a qualified GN call whenever possible.

The Qualified Get Next Call: This qualified GN call should at least identify the segment you want to retrieve. In doing so, you will achieve a greater independence toward possible data base structure changes in the future. If you supply only the segment name in the SSA, then you will retrieve all segments of that type from all data base records with subsequent get next calls (see Figure 4-9).

```
77  GN-FUNC PICTURE XXXX VALUE 'GNbb'.

01  SSA002-GN-SE1PPUR PICTURE X(9) VALUE 'SE1PPURbb'.

01  IOAREA PICTURE X(256).
```

```
CALL 'CBLTDLI' USING GN-FUNC,PCB-NAME,IOAREA,SSA002-GN-SE1PPUR.
```

```
STATUS CODES:

   bb:  next PURCHASE ORDER segment has been moved to IOAREA
   GB:  end of data base reached, no more PURCHASE ORDER segments
other:  error situation
```

Figure 4-9. Qualified Get Next Call

Repetition of the above GN call will retrieve all subsequent PURCHASE ORDER segments of the data base, until the end of the data base is reached. To limit this to a specific part, you could add a fully qualified SSA for the PART segment. This would be the same SSA as used in Figure 4-7.

An example of a qualified get next call with a qualified SSA is shown in Figure 4-10. This fully qualified get next call should be primarily used. It always clearly identifies the hierarchical path and the segment you want to retrieve.

```
77  GN-FUNC PICTURE XXXX VALUE 'GNbb'.

01  SSA001-GU-SE1PART.
  02  SSA001-BEGIN PICTURE X(19) VALUE 'SE1PARTb(FE1PGPNRb='.
  02  SSA001-FE1PGPNR PICTURE X(8).
  02  SSA001-END PICTURE X VALUE ')'.
01  SSA002-GN-SE1PPUR PICTURE X(9) VALUE 'SE1PPURb'.
01  IOAREA PICTURE X(256).
```

```
CALL 'CBLTDLI' USING GN-FUNC,PCB-NAME,IOAREA,SSA001-GU-SE1PART,
                     SSA002-GN-SE1PPUR.
```

```
STATUS CODES:

   bb:  next PURCHASE ORDER segment is in IOAREA
   GE:  segment not found; no more purchase orders for this part,
        or part number in SSA001 does not exist
other:  error situation
```

Figure 4-10. GN Call with Qualified SSA

Get Hold Calls

To change the contents of a segment in a data base through a replace or delete call, the program must first obtain the segment. It then changes the segment's contents and requests DL/I to replace the segment in the data base or to delete it from the data base.

This is done by using the get hold calls. These function codes are like the standard get function, except the letter 'H' immediately follows the letter 'G' in the code (that is, GHU, GHN). The get hold calls function exactly as the corresponding get calls for the user. For DL/I they indicate a possible subsequent replace or delete call.

After DL/I has provided the requested segment to the user, one or more fields, but not the sequence field, in the segment may be changed.

After the user has changed the segment contents, he can call DL/I to return the segment to, or delete it from the data base. If, after issuing a get hold call, the program determines that it is not necessary to change or delete the retrieved segment, the program may proceed with other processing, and the "hold" will be released by the next DL/I call against the same PCB.

UPDATING SEGMENTS

Segments can be updated by application programs and returned to DL/I for restoring in the data base, with the replace call, function code 'REPL'. Two conditions must be met:

- The segment must first be retrieved with a get hold call, GHU or GHN; no intervening calls are allowed referencing the same PCB.
- The sequence field of the segment cannot be changed; this can only be done with combinations of delete and insert calls for the segment and all its dependents.

Figure 4-11 shows an example of a combination of a GHU and REPL call. Notice that the replace call must not specify a SSA for the segment to be replaced. If, after retrieving a segment with a get hold call, the program decides not to update the segment, it need not issue a replace call. Instead the program can proceed as if it were a normal get call.

Because there is only a very small performance difference between the get and the get hold call, you should use the get hold call whenever there is a reasonable chance (about 5% or more) that you will change the segment.

```
 77  GHU-FUNC PICTURE XXXX VALUE 'GHUb'.
 77  REPL-FUNC PICTURE XXXX VALUE 'REPL'.
 01  SSA001-GU-SE1PART.
   02  SSA001-BEGIN PICTURE X(19) VALUE 'SE1PARTb(FE1PGPNRb='.
   02  SSA001-FE1PGPNR PICTURE X(8).
   02  SSA001-END PICTURE X VALUE ')'.
 01  SSA002-GN-SE1PPUR PICTURE X(9) VALUE 'SE1PPURbb'.

 01  IOAREA PICTURE X(256).
-----------------------------------------------------------------
 MOVE PART-NUMBER TO SSA001-FE1PGPNR.

 CALL 'CBLTDLI' USING GHU-FUNC,PCB-NAME,IOAREA,SSA001-GU-SE1PART,
                SSA002-GN-SE1PPUR.

    The retrieved PURCHASE ORDER segment can now be changed by the
    program in the IOAREA.

 CALL 'CBLTDLI' USING REPL-FUNC,PCB-NAME,IOAREA.
-----------------------------------------------------------------
 STATUS CODES (after REPL call):

    bb:  segment is replaced with contents in IOAREA
 other:  error situation
```

Figure 4-11. Basic REPL Call

DELETING SEGMENTS

To delete the occurrence of a segment from a data base, the segment must first be obtained by issuing a get hold (GHU, GHN) call through DL/I. Once the segment has been acquired, the DLET call may be issued.

No DL/I calls which use the same PCB must intervene between the get hold call and the DLET call, or the DLET call is rejected. Quite often a program may want to process a segment prior to deleting it. This is permitted as long as the processing does not involve a DL/I call which refers to the same data base PCB used for the get hold/delete calls. However, other PCBs may be referred to between the get hold and DLET calls.

DL/I is advised that a segment is to be deleted when the user issues a call that has the function DLET. The deletion of a parent, in effect, deletes all the segment occurrences beneath that parent, whether or not the application program is sensitive to those segments. If the segment being deleted is a root segment, that whole data base record is deleted. The segment to be deleted must still be in the IOAREA of the delete call (with which no SSA is used), and its sequence field must not have been changed. Figure 4-12 gives an example of a DLET call.

```
 77  GHU-FUNC PICTURE XXXX VALUE 'GHUb'.
 77  DLET-FUNC PICTURE XXXX VALUE 'DLET'.
 01  SSA001-GU-SE1PART.
   02  SSA001-BEGIN PICTURE X(19) VALUE 'SE1PARTb(FE1PGPNRb='.
   02  SSA001-FE1PGPNR PICTURE X(8).
   02  SSA001-END PICTURE X VALUE ')'.
 01  SSA002-GN-SE1PPUR PICTURE X(9) VALUE 'SE1PPURbb'.

 01  IOAREA PICTURE X(256).
```

```
 CALL 'CBLTDLI' USING GHU-FUNC,PCB-NAME,IOAREA,SSA001-GU-SE1PART,
                 SSA002-GN-SE1PPUR.

    The retrieved PURCHASE ORDER segment can now be processed by
    the program in the IOAREA

 CALL 'CBLTDLI' USING DLET-FUNC,PCB-NAME,IOAREA.
```

STATUS CODES (after DLET call):

bb:	requested purchase order segment is deleted from the data base; all its dependents, if any, are deleted also.
other:	error situation

Figure 4-12. Basic DLET Call

INSERTING SEGMENTS

Adding new segment occurrences to a data base is done with the insert call, function code 'ISRT'.

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

When loading or inserting, the last SSA must specify only the name of the segment being inserted. It should specify only the segment name, not the sequence field. Thus an unqualified SSA is always required.

Up to the level to be inserted, the SSA evaluation and positioning for an insert call is exactly the same as for a GU call. For the level to be inserted, the value of the sequence field in the segment in the user I/O area is used to establish the insert position. If no sequence field was defined, then the segment is inserted at the end of the physical twin chain. If multiple non-unique keys are allowed, then the segment is inserted after existing segments with the same key value.

Figure 4-13 shows an example of an ISRT call. The status codes in this example are applicable only to non-initial load inserts. The status codes at initial load time will be discussed under the topic "Loading A Basic Data Base" later in this chapter.

```
77  ISRT-FUNC PICTURE XXXX VALUE 'ISRT'.
01  SSA001-GU-SE1PART.
  02  SSA001-BEGIN PICTURE X(19) VALUE 'SE1PARTb(FE1PGPNRb='.
  02  SSA001-FE1PGPNR PICTURE X(8).
  02  SSA001-END PICTURE X VALUE ')'.
01  SSA002-GN-SE1PPUR PICTURE X(9) VALUE 'SE1PPURbb'.

01  IOAREA PICTURE X(256).
---------------------------------------------------------------
MOVE PART-NUMBER TO SSA001-FE1PGPNR.

MOVE PURCHASE-ORDER TO IOAREA.

CALL 'CBLTDLI' USING ISRT-FUNC,PCB-NAME,IOAREA,SSA001-GU-SE1PART,
               SSA002-GN-SE1PPUR.
---------------------------------------------------------------
STATUS CODES:

   bb:  new PURCHASE ORDER segment is inserted in data base
   II:  segment to insert already exists in data base
   GE:  segment not found; the requested part number (that is,
        a parent of the segment to be inserted) is not in the
        data base
other:  error condition
```

Figure 4-13. Basic ISRT Call

Note: There is no need to check the existence of a segment in the data base with a preceding retrieve call. DL/I will do that at insert time, and will notify you with an II or GE status code. Checking previous existence is only relevant if the segment has no sequence field.

CALLS WITH COMMAND CODES

Both unqualified and qualified SSAs may contain one or more optional command codes which specify functional variations applicable to either the call function or the segment qualification. Command codes in an SSA are always prefixed by an asterisk (*), which immediately follows the 8 byte segment name. Figure 4-14 illustrates this. Following are some important command codes.

D Command Code

The 'D' command code is the one most widely used. It requests DL/I to issue path calls. A "path call" enables a hierarchical path of segments to be inserted or retrieved with one call. (A "path" was defined earlier as the hierarchical sequence of segments, one per level, leading from a segment at one level to a particular segment at a lower level.) The meaning of the 'D' command code is as follows:

For retrieval calls, multiple segments in a hierarchical path will be moved to the I/O area with a single call. The first through the last segment retrieved are concatenated in the user's I/O area. Intermediate SSAs may be present with or without the 'D' command code. If without, these segments are not moved to the user's I/O area. The segment named in the PCB "segment name feedback area" is the lowest-level segment retrieved, or the last level satisfied in the call in case of a non-found condition. Higher-level segments associated with SSAs having the 'D' command code will have been placed in the user's I/O area even in the not-found case. The 'D' is not necessary for the last SSA in the call, since the segment which satisfies the last level is always moved to the user's I/O area. A processing option of 'P' must be specified in the PSBGEN for any segment type for which a command code 'D' will be used.

For insert calls, the 'D' command code designates the first segment type in the path to be inserted. The SSAs for lower-level segments in the path need not have the D command code set, that is, the D command code is propagated to all specified lower level segments.

Figure 4-14 shows an example of a path call.

```
77  GU-FUNC PICTURE XXXX VALUE 'GUbb'.

01  SSA004-GUD-SE1PART.
  02  SSA004-BEGIN PICTURE X(21) VALUE 'SE1PARTb*D(FE1PGPNRb='.
  02  SSA004-FE1PGPNR PICTURE X(8).
  02  SSA004-END PICTURE X VALUE ')'.

01  SSA005-GN-SE1PGDSC PICTURE X(9) VALUE 'SE1PGDSCb'.

01  IOAREA PICTURE X(256).
--------------------------------------------------------------------
CALL 'CBLTDLI' USING GU-FUNC,PCB-NAME,IOAREA,
                 SSA004-GUD-SE1PART,SSA005-GN-SE1PGDSC.
--------------------------------------------------------------------
STATUS CODES:

   bb:  both segments (PART and DESCRIPTION) have been placed
        in IOAREA
   GE:  segment not found; PART segment may be retrieved in
        IOAREA; check segment name and level indicator in PCB.
other:  error condition
```

Figure 4-14. Sample Path Retrieve Call

The above example shows a common usage of the path call. Although we don't know if the requested part has a separate DESCRIPTION segment (SE1PGDSC), we retrieve it at almost no additional cost if there is one.

The correct usage of path calls can have a significant performance advantage. You should use it whenever possible, even if the chance of the existence or the need for the dependent segment(s) is relatively small. For instance, if you would need, in 10% or more of the occurrences, the first dependent segment after you inspect the parent, then it is generally advantageous to use a path call to retrieve them both initially.

N Command Code

When a replace call follows a path retrieve call, it is assumed that all segments previously retrieved with the path call are being replaced. If any of the segments have not been changed, and, therefore, need not be replaced, the 'N' command code may be set at those levels, telling DL/I not to replace the segment at this level of the path. The status codes returned are the same as for a regular replace call.

F Command Code

This command code allows you to back up to the first occurrence of a segment under its parent. It has meaning only for a get next call. A get unique call always starts with the first occurrence. Command code F is disregarded for the root segment.

L Command Code

This command code allows you to retrieve the last occurrence of a segment under its parent. This command code should be used whenever applicable.

- Command Code

The hyphen is a null command code. Its purpose is to simplify the maintenance of SSAs using command codes.

DATA BASE POSITIONING AFTER A DL/I CALL

As stated before, the data base position is used by DL/I to satisfy the next call against the PCB. The segment level, segment name and the key feedback areas of the PCB are used to present the data base position to the application program.

The following basic rules apply:

1. If a get call is completely satisfied, current position in the data base is reflected in the PCB key feedback area.

2. A replace call does not change current position in the data base.

3. Data base position after a successful insert call is immediately after the inserted segment.

4. Data base position after return of an II status code is immediately prior to the duplicate segment. This positioning allows the duplicate segment to be retrieved with a GN call.

5. Data base position after a successful delete call is immediately after all dependents of the deleted segment. If no dependents existed, data base position is immediately after the deleted segment.

6. Data base position is unchanged by an unsuccessful delete call.

7. After an (partial) unsuccessful retrieve call, the PCB reflects the lowest level segment which satisfied the call. The segment name or the key feed back length should be used to determine the length of the relevant data in the key feedback area. Contents of the key feedback area beyond the length value must not be used, as the feedback area is never cleared out after previous calls. If the

level-one (root) SSA cannot be satisfied, the segment name is cleared to blank, and the level and key feedback length are set to 0.

In considering 'current position in the data base', it must be remembered that DL/I must first establish a starting position to be used in satisfying the call. This starting position is the current position in the data base for get next calls, and is a unique position normally established by the root SSA for get unique calls.

The following are clarifications of 'current position in the data base' for special situations:

- If no current position exists in the data base, then the assumed current position is the start of the data base.
- If the end of the data base is encountered, then the assumed current position to be used by the next call is the start of the data base.
- If a get unique call is unsatisfied at the root level, then the current position is such that the next segment retrieved would be the first root segment with a key value higher than the one of the unsuccessful call, except when end of the data base was reached (see above) or for HDAM, where it would be the next segment in physical sequence.

You can always reestablish your data base positioning with a GU call specifying all the segment key values in the hierarchical path. It is recommended that you use a get unique call after each not found condition.

USING MULTIPLE PCBs FOR ONE DATA BASE

Whenever there is a need to maintain two or more independent positions in one data base, you should use different PCBs. This avoids the reissue of get unique calls to switch forward and backward from one data base record or hierarchical path to another. There are no restrictions as to the call functions available in these multiple PCBs. However, to avoid "position confusion" in the application program, you should not apply changes via two PCBs to the same hierarchical path. For simplicity reasons you should limit the updates to one PCB unless this would cause additional calls.

SYSTEM SERVICE CALLS

Besides call functions for manipulating data base segments, DL/I provides special system service calls. The most common ones are:

- STATISTICS (STAT) -- This call is used to obtain various statistics from DL/I.
- CHECKPOINT (CHKP) -- CHKP informs DL/I that the user has "checkpointed" his program and that it thus may be restarted at this point. The current position is maintained in GSAM data bases. For all other data bases, you must reposition yourself after each checkpoint call with a get unique call.
- RESTART (XRST) -- XRST requests DL/I to restore checkpointed user areas and reposition GSAM data bases for sequential processing if a checkpoint ID for restarting has been supplied by the call or in the JCL.

The XRST and CHKP calls will be discussed under the topic "Batch Checkpoint/Restart" later in this chapter.

The STAT Call

The STAT call retrieves the statistics information of the data base buffer pool(s). A discussion of those pools and their statistics can be found in Chapter 9: "Optimization." We will not include a detailed discussion of the STAT call. Instead we provide a general subroutine, DFSOAST in IMSVS.PRIMESRC, which performs the STAT call, formats and prints the statistics. This subroutine can be called from any DL/I batch program. To obtain the print of the statistics you must include a SYSOUT card in the JCL with ddname of //DCCSTAT. If you don't want the statistics, just leave out this DD statement.

The basic format of the call statement to call this subroutine in COBOL is:

```
CALL 'DFSOAST' USING pcb-name.
```

pcb-name: can be any data base PCB in your program.

No status code checking should be done after return. Typically, the statistics will be requested at the end of each batch program.

PROCESSING GSAM DATA BASES

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

Calls to be used for GSAM accessing are:

```
CALL 'CBLTDLI' USING call-func,pcb-name,ioarea.
```

where:

call-func

is the name of the field which contains the call function:

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

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

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

- pcb-name
  is the name of the GSAM PCB

- ioarea
  is the name of the I/O area for GN/ISRT calls, or the optional address of the OPEN-option for an OPEN call. The OPEN option is either INP, OUT, or, in the case of SYSOUT type data sets, OUTA or OUTM to include the standard print or punch control characters (A for ASA, M for Machine).

STATUS CODES:

- bb: OK, proceed
- GB: end of data (get next only)
- other: error situation

RECORD FORMATS:

Records may be fixed or variable length, blocked or unblocked. Records must be unkeyed. The inclusion of carriage control characters may also be indicated in the JCL RECFM subparameter (for example, RECFM=FBA) for all record formats. The record in the IOAREA includes a halfword record length for variable length records.

Sample GSAM processing is shown in programs PE1CPPUR and PE3CPPUR (members DFSICPUR and DFS3CPUR, respectively) in IMSVS.PRIMESRC.

The use of GSAM data sets in a checkpoint/restart environment is further discussed later in this chapter.

LOADING A BASIC DATA BASE

After generating the physical DBD, you can load your data base using a load program. Basically the load program reads a sequential file with the data base record contents; it builds the segments and inserts them in the data base in hierarchical order. Quite often the data to be stored in the data base already exists in one or more files, but merge and sort operations may be required to present the data in the correct sequence. Sometimes even clean-up and correction activities are required, especially when multiple files with redundant data are merged into one data base (see Figure 4-15).

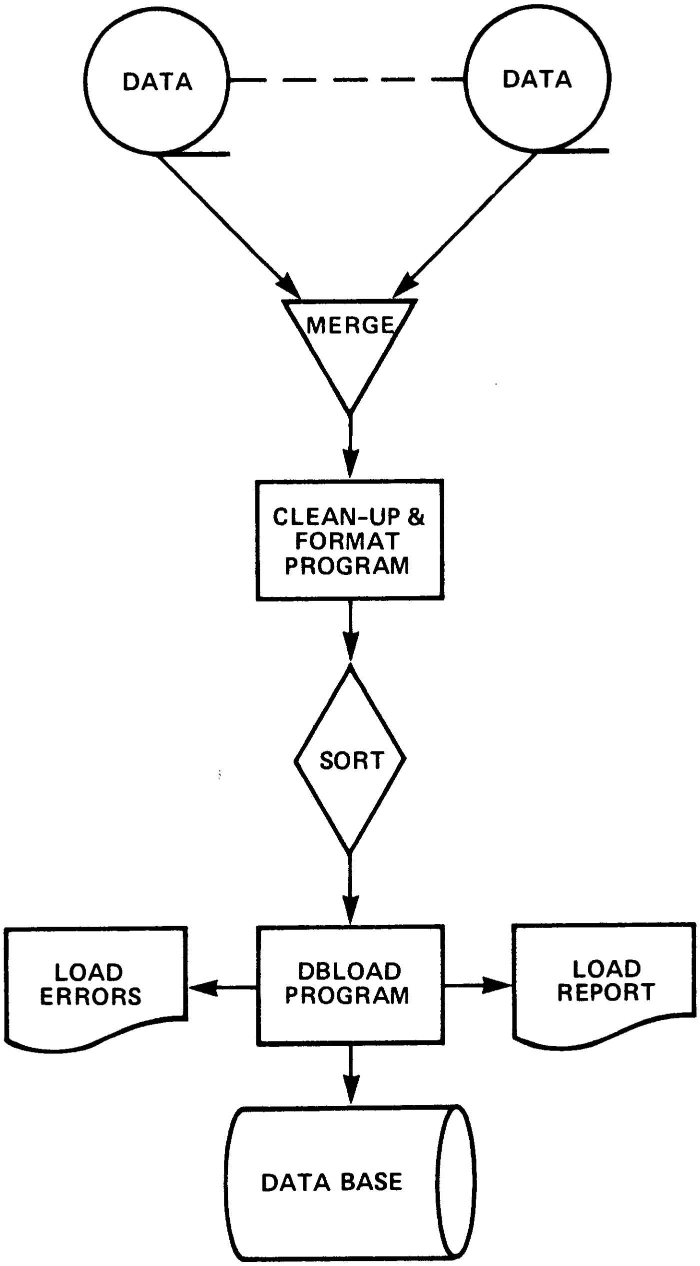


Figure 4-15. Basic Data Base Load Process

Sample Data Base Load Program

The sample data base load program DFSOADBL in IMSVS.PRIMEJOB may be used to load the sample data bases. It may also be used as a general data base load program to load your own data bases. Furthermore, you will find this program, due to its modular structure, easy to modify should you wish to do so. To use the program as it is, use the following guidelines:

- The input data can be any OS/VS sequential file supported by QSAM. Each segment must be stored in one record with the following format:
  - Positions 1 through 3: segment code (see Figure 2-4), zoned decimal, 001 is the root segment, maximum 255

- Positions 4 to N: not used by the load program; can be used to store additional sequence fields for sort purposes. N is defined for each segment in the control input.

- Position N and beyond: the segment data in exactly the format you wish it stored in the data base.

- The input control file contains one card for each segment type to be loaded, with the following format:

  - Positions 1-3: segment code, 001-255
  - Position 4: blank or comma
  - Positions 5-12: segment name
  - Position 13: blank or comma
  - Positions 14-17: position of first byte of input data in the input record; default is 0004 (this is N as defined for input data above)
  - Positions 18-26: blanks
  - Positions 27-80: not used

Note: This load program does not manipulate the actual data base data; however, it does provide the hooks to add such functions easily. The program listing should be consulted for guidance.

Loading a HIDAM Data Base

When loading a HIDAM data base initially, you must specify PROCOPT=LS in the PCB. Also, the data base records must be inserted in ascending root key sequence, and the segments must be inserted in their hierarchical sequence.

Sorting segments in hierarchical sequence: If there is a need to sort on a segment level, you must provide the following sort control fields with each segment (Figure 4-16):

ROOT KEY	LEVEL 2 SEGMENT CODE	LEVEL 2 KEY	LEVEL 3 SEGMENT CODE	LEVEL 3 KEY	etc.

Figure 4-16. Control Field for Sorting Segments into Hierarchical Sequence

Notes:

1. The level 2 segment code for the root segment should contain a lower value than other level 2 segment codes.

2. For every level, the key field length should be equal to the largest segment key field on that level. Shorter keys should be left adjusted and padded with low value characters.

3. Segments on the lowest level need not have a key field if no sequence field is defined; however, their sequence below their parent might be different after the sort. If no sequence field is

available in the segment itself, you should provide one. This could be a simple dependent segment counter provided by the "clean up and format" program in Figure 4-15.

4. The above fields must be filled in for every segment. So a level n segment has the segment code of its superior segment at phase 2 (that is, the phase 2 segment it is a dependent of) stored in the "phase 2 segment code" control field, and so forth.

5. When using the sample data base load program (DFSOADBI), the above sort control field, can be within the same input record as the segment itself.

When loading the HIDAM data base, DL/I will also load the primary index data base. You must, however, provide a DD statement for the data space of this primary index with your job. Job //SAMP270 in IMSVS.PRIMEJCB can be used to load our sample phase 2 customer order HIDAM data base.

Note: When loading a HIDAM data base, DL/I will automatically insert a high key (X'FF...') at the end of the data base. This is for its chain maintenance, it is completely transparent for your program. But you should not use this key in your application.

Loading a HDAM Data Base

When initially loading a HDAM data base, you should specify PROCOPT=L in the PCB. There is no need for DL/I to insert the data base records in root key order, but you must still insert the segments in their hierarchical order. For performance reasons it is advantageous to sort the data base records into physical sequence. The physical sequence should be the ascending sequence of the block and root anchor point values as generated by the randomizing algorithms. This can be achieved by an E61 type sort exit routine, which gives each root key to the randomizing module for address conversion, and then directs SORT to sort on the generated address + root key value. Such a general exit routine is provided as sample DFSOASRT in IMSVS.PRIMESRC.

Job //SAMP170 in IMSVS.PRIMEJCE shows the JCL to load the phase 1 HDAM parts data base. Module DFSOASRT is used, as an E61 sort exit routine, to sort the segments in the desired sequence, and program DFSOADBI is used for the actual load.

Loading a SHISAM Data Base

Loading a SHISAM data base is the same as loading a root only HIDAM data base. Just insert the root segments in ascending key sequence. No sample is provided for this because the SHISAM data base was created as a KSDS.

Status Codes for Data Base Loading

The following status codes can be expected when loading basic data bases after the ISRT call:

bb: OK, segment is inserted in data base

LB: the segment you tried to insert already exists in the data base

LC: key field of segment is out of sequence

LD: no parent has been inserted for this segment in the data base.

other: error situation

STATUS CODE ERROR ROUTINE

There are essentially two categories of error status codes: those caused by application program errors and those caused by system errors. Sometimes, however, a clear split cannot be made immediately. Appendix B contains a listing of the status codes in both of these categories, together with an explanation and suggested actions.

This listing is not complete, but does contain all the status codes you should expect using our subset of DL/I. You should refer to Appendix B of the "IMS/VS Application Programming Reference Manual," if you should need a complete listing of all possible status codes.

To aid the debugging of programs SSAs, a status code error routine should print critical system information like CALLID, IOAREA, PCB, etc. The sample error status code routine, DFSOAER, in IMSVS.PRIMESRC provides such services. To call this routine from your application program code (COBOL):

```
CALL 'DFSOAER' USING pcb-name,call-label,area1,options,area2,..area9
```

where:

pcb-name
: name of PCB used for the preceding DL/I call

call-label
: name of symbolic label identifier of the preceding DL/I call. Required format: Dxxxxxxx, when using a DB PCB; Cxxxxxxx, when using a DC PCB.

options
: address of a 4 byte option field.

byte 1:	C'1'	Abnormal termination after print; recommended for production.
	C'0'	Return to caller after print. This enables multiple invocations for testing purposes. A final invocation is required.
	C'2'	Final invocation to close print data set, program gets control back.
	C'3'	Message DFS3125A will be issued. This is used by the sample programs for testing recovery procedures. See the <u>IMS/VS Messages and Codes Reference Manual</u> for more details.
byte 2,3,4:	reserved	

area 1, ...area9
: program areas to be printed by DFSOAER. The first 76 characters of each area will be printed. At least 1 and a maximum of 9 areas should be specified.

Notes:

1. The status code error routine will return to your program on request. This may be valuable in a test environment; however, for a production program the abnormal termination option should be selected.

2. For PL/I, you should declare DFSOAER to be an ENTRY with OPTIONS (ASSEMBLER). Moreover, you should pass the actual PCB-name and not the pointer variable on which the PCB is based, unlike calls to DL/I.

For the programming details refer to the sample application programs in IMSVS.PRIMESRC.

ASSEMBLER PROGRAMMING CONSIDERATIONS

When writing a DL/I application program in Assembler, the following should be observed (for an example see program DFSOADBI in IMSVS.PRIMESRC).

- You should supply an entry statement:

```
ENTRY DLITASM
```

- At entry to your program, register 1 contains the address of a list of PCB addresses in standard OS/VS convention. The high order byte of the last fullword in this address list is set to X'80' to indicate end of list. You should not change this list but save these PCB addresses for later reference.

- Each call statement should be coded as follows:

```
CALL ASMTDLI,(function,(pcbreg),ioarea,ssa1,....,ssan),VL
```

where:

function is the name of a field containing the DL/I call function

pcbreg is the register containing the PCB address

ioarea is the name of input/output area

ssa1-ssan are the names of segment search arguments

- After each call, you should check the status code as returned in the PCB. On error conditions you should invoke a status code error routine.

- At the end of your program, you should always return to DL/I. You can set a return code, but if DL/I has encountered an error condition (for example, data base I/O error), your return code will be overriden by DL/I's.

Using the Sample Routines

Several Assembler routines are provided with the sample programs in IMSVS.PRIMESRC. Most of these can be used as they are, or with minor modifications depending on your installation's standards.

The following general routines are available:

- DFSOAER, a status code error routine, discussed earlier in this chapter.

- DFSOAST, a general data base buffer pool statistics print routine; see the discussion of the STAT call in this chapter.

JCL for Assembly and Linkage Editing

The Sample JCL for assembly and linkage editing can be found in job //SAMP034 in IMSVS.PRIMEJCB, which can be used to assemble and link-edit the sample data base load program.

COBOL PROGRAMMING CONSIDERATIONS

There are a few considerations that apply when you are coding DL/I programs in COBOL. Refer to figure 4-17 for this discussion. The numbers between parenthesis in the text below refer to the corresponding code lines in Figure 4-17. Specific parameter values and formats are explained elsewhere throughout this chapter.

```
0000001 ID DIVISION.
0000002      :
0000003 ENVIRONMENT DIVISION.
0000004      :
0000005 DATA DIVISION.
0000006 WORKING-STORAGE SECTION.
0000007 77  GU-FUNC       PIC XXXX VALUE 'GU  '.
0000008 77  GN-FUNC       PIC XXXX VALUE 'GN  '.
0000009 77  ERROPT        PIC XXXX VALUE '1   '.
0000010 77  DERRID        PIC X(8) VALUE 'DERROR01'.
0000011 01  IOAREA        PIC X(256) VALUE SPACES.
0000012 01  SSA001-GU-SE1PART.
0000013     02  SSA001-BEGIN    PIC X(19) VALUE 'SE1PART (FE1PGPNR ='.
0000014     02  SSA001-FE1PGPNR PIC X(8).
0000015     02  SSA001-END      PIC X VALUE ')'.
0000016      :
0000017 LINKAGE SECTION.
0000018 01  D1PC.
0000019         02  D1PCDBDN PIC X(8).
0000020         02  D1PCLEVL PIC 99.
0000021         02  D1PCSTAT PIC XX.
0000022         02  D1PCPROC PIC XXXX.
0000023         02  D1PCRESV PIC S9(5) COMP.
0000024         02  D1PCSEGN PIC X(8).
0000025         02  D1PCKFBL PIC S9(5) COMP.
0000026         02  D1PCNSSG PIC S9(5) COMP.
0000027         02  D1PCKFBA PIC X(20).
0000028
0000029 PROCEDURE DIVISION.
0000030     ENTRY 'DLITCBL' USING D1PC.
0000031     CALL 'ILBOSPIO'.
0000032      :
0000033     CALL 'CBLTDLI' USING GU-FUNC, D1PC, IOAREA,
0000034                                  SSA001-GU-SE1PART.
0000035      :
0000036     CALL 'CBLTDLI' USING GN-FUNC, D1PC, IOAREA.
0000037     IF D1PCSTAT NOT = '  ',
0000038       CALL 'DFSOAER' USING D1PC, DERRID, IOAREA, ERROPT,
0000039       MOVE +4 TO RETURN-CODE.
0000040      :
0000041     CALL DFSOAST USING D1PC.
0000042      :
0000043     CALL 'ILBOSPIO'.
0000044     GOBACK.
```

Figure 4-17. COBOL Batch Program Structure

- The DL/I function codes (7), IOAREA (11), and Segment Search Arguments (12) should be defined in the Working-Storage Section of the Data Division. Typically, either the IOAREA would be REDEFINED to provide addressability to the fields of each segment, or separate IOAREAs would be defined for each segment.

- The Program Communication Blocks (PCBs) should be defined in the Linkage Section of the Data Division (18). When there are multiple database structures (thus multiple PCBs) in a program, there must be one PCB defined in the Linkage Section for each PCB in the PSB. However, these PCBs need not be in any specific order.

- An ENTRY statement (30) should be coded at the entry to your program. A parameter of the USING clause should exist for each database structure (PCB) that is used in your program. The order of PCBs in this clause must be the same as specified in the Program Specification Block (PSB) for your program.

- Each DL/I CALL statement should be coded as in statement (33). The parameters of the DL/I call are explained elsewhere in this chapter, and differ in number for different functions.

- The status code in the PCB should be checked after each call (37). The status-code error routine is discussed below (38).

- At the end of processing, control must be returned to DL/I via a GOBACK statement (44). If you wish, you may set the COBOL 'RETURN-CODE' (39). If DL/I detects no errors, and thus does not set the return code, the COBOL 'RETURN-CODE' value will be passed on to the next job step.

- The Status-Code Error Routine, DFSOAER (38) may be called if an unexpected status code is returned by your program. This routine is discussed earlier in this chapter.

- The Buffer-Pool Statistics Print Routine, DFSOAST, (41) may be called from your program. Its usage is discussed in this chapter under the STAT call.

- The Symbolic-Debugging facility of COBOL can be used. A call to COBOL module ILBOSPIO (31) should be made immediately after entry to and before exit from your program. This facility is further described in the Programmers Guide for your COBOL program product.

- The sample programs in IMSVS.PRIMESRC use a form of structuring. This is not classical "structured programming", but the examples are modular and should be readily maintainable.

JCL For Compile And Linkage Editing

Sample JCL for compiling and link editing a COBOL program can be found in job //SAMP140 in IMSVS.PRIMEJOB.

JCL For Program Execution

Job //SAMP171 in IMSVS.PRIMEJOB shows the JCL for the Parts Inventory report program. Job //SAMP173 shows the JCL for the Purchase Order program.

PL/I PROGRAMMING CONSIDERATIONS

This section refers to Figure 4-18. The numbers between parenthesis in the text refer to the corresponding code lines in Figure 4-18.

When DL/I invokes your PL/I program it will pass the addresses, in the form of pointers, to each PCB required for execution. These will be passed in the same sequence as specified in the PSB. To use the PCBs, you must code parameters in your PROCEDURE statement, and declare them to have the attribute POINTER. In the example in Figure 4-18, DC_PTR and DB_PTR are specified in the PROCEDURE statement (6) and declared POINTER variables (15 and 16). These pointer variables should be used in declaring the PCBs as BASED structures (18 and 21), and in calling DL/I (55).

The format of the PL/I CALL statement to invoke DL/I (55) is:

```
CALL PLITDLI (parmcount,function,pcb-ptr,io-area,ssa1,...,ssan);
```

where:

- **parmcount** is the number of arguments in this call following this argument. It must have the attributes FIXED BINARY (31). See (38).
- **function** is the DL/I function code. It must be a fixed length character string of length 4.
- **pcb-ptr** is a pointer variable containing the address of the PCB. This is normally the name of one of the parameters passed to your program at invocation.
- **io-area** is the storage in your program into/from which DL/I is to store/fetch data. It can be a major structure, a connected array, a fixed-length character string (CHAR(100)), an adjustable character string (CHAR(N)), a pointer to any of these or a pointer to a minor structure. It cannot be the name of a minor structure or a character string with the attribute VARYING.
- **ssa1,...** is one or more optional segment search arguments. Each SSA argument must be one of the same PL/I forms allowed for io-areas, described above. See (47) in the example.

Upon completion of your program, you should return either via a RETURN statement or by executing the main procedure END statement.

```
/*-----------------------------------------------------------------*/
/*                      SAMPLE PL/I PROGRAM                        */
/*-----------------------------------------------------------------*/

PE2PORD:
  PROCEDURE (DC_PTR,DB_PTR) OPTIONS (MAIN);

/*...... DECLARE POINTERS AND PCBS .................................*/

  DECLARE

    PLITDLI ENTRY,                            /* DL/I WILL BE CALLD*/
    DFSOAST ENTRY OPTIONS (ASSEMBLER INTER),  /* STATISTICS PRINT  */
    DFSOAER ENTRY OPTIONS (ASSEMBLER INTER),  /* STATUS CODE PRINT */
    DC_PTR POINTER,                           /* CMPAT IN PSB      */
    DB_PTR POINTER,                           /* ORDER DB PCB      */

    1 C1PC BASED (DC_PTR),                    /* NOT USED IN       */
      2 DUMMY CHAR (32),                      /*    BATCH DL/I     */

    1 D1PC BASED (DB_PTR),                    /* PHASE 2 ORDER DB  */
      2 D1PCDBDN CHAR (8),                    /*   DBD NAME        */
      2 D1PCLEVL CHAR (2),                    /*   SEGMENT LEVEL   */
      2 D1PCSTAT CHAR (2),                    /*   STATUS CODE     */
      2 D1PCPROC CHAR (4),                    /*   PROCESSING OPTN */
      2 D1PCRESV FIXED BINARY (31),           /*   RESERVED        */
      2 D1PCSEGN CHAR (8),                    /*   SEGMENT NAME    */
      2 D1PCKFBL FIXED BINARY (31),           /*   KEY FEEDBACK LNG*/
      2 D1PCNSSG FIXED BINARY (31),           /*   NO. OF SENSEGS  */
      2 D1PCKFBA CHAR (14);                   /*   KEY FEEDBACK    */

/*...... DECLARE FUNCTION CODES, I/O AREA, CALL ARG LIST LENGTHS ....*/

  DECLARE

    IO_AREA CHAR (256),                       /* I/O AREA          */
    GU_FUNC STATIC CHAR (4) INIT ('GU'),      /* CALL FUNCTION     */
    FOUR STATIC FIXED BINARY (31) INIT (4),   /* ARG LIST LENGTH   */
    ERROPT1 CHAR (4) INIT ('0') STATIC,       /* OPTN FOR DFSOAER  */
    ERROPT2 CHAR (4) INIT ('2') STATIC,       /* FINAL OPTN-DFSOAER*/
    DERRID CHAR (8) INIT ('DERROR01') STATIC; /* ID FOR DFSOAER    */

/*...... DECLARE SEGMENT SEARCH ARGUMENT (SSA) - ORDER SEGMENT ......*/

  DECLARE

    1 SSA007_GU_SE2ORDER,
      2 SSA007_BEGIN CHAR (19) INIT ('SE2ORDER(FE2OGREF ='),
      2 SSA007_FE2OGREF CHAR (6),
      2 SSA007_END CHAR (1) INIT (')');

/*...... PROCESSING PORTION OF THE PROGRAM ..........................*/

  SSA007_FE2OGREF = 'XXXXXX';                 /* SET SSA VALUE     */
  CALL PLITDLI (FOUR,GU_FUNC,DB_PTR,IO_AREA,  /* THIS CALL WILL    */
                SSA007_GU_SE2ORDER);          /* RETURN 'GE' STAT  */
  IF D1PCSTAT ¬= '  ' THEN                    /* CALL ERROR PRINT  */
     CALL DFSOAER (D1PC,DERRID,IO_AREA,ERROPT1);
     CALL DFSOAER (D1PC,DERRID,IO_AREA,ERROPT2); /* FINAL CALLTO ERR*/

/*...... BEFORE ENDING, LIST BUFFER POOL STATISTICS .................*/

  CALL DFSOAST (D1PC);                        /* CALL STATS PRINT  */

/*...... RETURN TO CALLER ...........................................*/

  END PE2PORD;

/*-----------------------------------------------------------------*/
/*                   END OF PL/I SAMPLE PROGRAM                    */
/*-----------------------------------------------------------------*/
```

Figure 4-18. PL/I Batch Program Structure

Other PL/I Considerations

- Programs that are OS/VS subtasks of an application program called by IMS/VS must not issue DL/I calls. If they do, the results will be unpredictable. With PL/I, whenever PL/I multitasking is used, all tasks, even the apparent main task, operates as subtask to a hidden PL/I control task. PL/I tasking is therefore not allowed in an IMS/VS program.

- Because the normal method of passing parameters to a main procedure is not available in IMS/VS, you must use the PLIXOPT facility to specify PL/I run-time options. See the _PL/I Optimizing Compiler Programmer's Guide_, SC33-0006, for details.

- You should consider using the PL/I Fast Path Initialization/Termination Option in an IMS/VS DB/DC environment. Consult the appropriate PL/I documentation for details.

Using the Sample Routines

The Status Code Error Print program (DFSOAER) may be called from PL/I programs as shown in (9). Similarly, the Statistics Print program (DFSOAST) can be called. See (10). See "Status Code Error Routine" and "The STAT Call" discussions earlier in this chapter for a description of the formats for these calls. In both cases, no parmcount argument is allowed, as is required in the DL/I call. In addition, the name of the PCB must be passed rather than the name of the pointer variable on which the PCB is based. DFSOAER and DFSOAST should be declared as ENTRY constants with OPTIONS (ASSEMBLER).

Link-Editing PL/I Programs for DL/I

The DL/I language interface program must be included at link-edit time. In addition, the normal entry point for PL/I Optimizer programs (PLISTART) must be overridden, specifying PLICALLA. The sample job //SAMP254 linkage editor step shows an example of the required linkage editor control statements. They are:

```
INCLUDE   ddname(DFSOAER)         SC Error Print

INCLUDE   ddname(DFSOAST)         Statistics Print

INCLUDE   RESLIB(PLITDLI)         Language Interface

ENTRY     PLICALLA

NAME      load-module-name(R)
```

The above three INCLUDE statements can be omitted if the modules or aliases are members of libraries which are concatenated as part of the ddname SYSLIB during link-editing. References to them will be resolved via automatic library call. When link-editing PL/I-F programs, the load module ENTRY must be specified as either IHESAPB (OPT=0) or IHESAPD (OPT=1).

SAMPLE PHASE 1 PROGRAMS

Besides the sample data base load program DFSOADBL, two batch programs both in COBOL and PL/I are included in IMSVS.PRIMESRC.

Program PE1CPINV (member DFS1CINV for COBOL, or DFS1PINV for PL/I in IMSVS.PRIMESRC) is a read only parts inventory report program. This program can be compiled with job //SAMP140 (COBOL) or //SAMP150 (PL/I) in IMSVS.PRIMEJOB. Job //SAMP171 in IMSVS.PRIMEJOB can be used for its execution.

Program PE1CPPUR (member DFS1CPUR for COBOL, or DFS1PPUR for PL/I in IMSVS.PRIMESRC) is an update parts purchase order sample program. This program uses the batch checkpoint/restart facility of IMS/VS. The program can be compiled with job //SAMP141 (COBOL), or //SAMP151 (PL/I). Job //SAMP173 can be used for its execution, and job //SAMP178 for its restart.

For more details on these programs and their operation you should consult their listing.

Notice that the data base update jobs specify a log data set in the //IEFRDER DD statement. For a discussion of the DL/I logging facility, refer to Chapter 6, "Data Base Recovery."

PROCESSING WITH LOGICAL RELATIONSHIPS

Generally, there is no difference between the processing of physical data bases and logical data bases; all call functions are available for both. Some considerations do apply, however, when accessing a logical child or a concatenated segment. For a definition of these terms see "DL/I Logical Relationships" in Chapter 2.

ACCESSING A LOGICAL CHILD IN A PHYSICAL DBD

When accessing a logical child in a physical DBD, you should remember the layout of the logical child. It always consists of the logical parent concatenated key (that is, all the consecutive keys from the root segment down to and including the logical parent) plus the logical child itself; the intersection data (see Figure 2-10). This is especially important when inserting a logical child. You will also get an IX status code when you try to insert a logical child and its logical parent does not exist (except at initial load time). This will typically happen when you forget the LPCK in front of the LCHILD.

Note: In general, physical data bases should not be used when processing logical relationships.

ACCESSING SEGMENTS IN A LOGICAL DBD

The following considerations apply for each call function when accessing segments in logical DBDs.

Retrieve Calls

These calls function as before with the same status codes. Remember, however, that the concatenated segment always consists of the logical child segment plus, optionally (dependent on the logical DBD), the destination parent segment (see Figure 2-13).

Replace Calls

In general, these calls function the same as before. When replacing a concatenated segment you may replace both the logical child segment and the destination parent. Remember, however, that you never can change a sequence field. The following sequence fields can occur in a concatenated segment (see also Figure 2-16):

- Destination parent concatenated key
- Real logical child sequence field, (that is, the sequence of the physical twin chain as defined for the real logical child). This field can (partially) overlap the logical parent concatenated key.
- Virtual logical child sequence field, (that is, the sequence of the logical twin chain as defined for the virtual logical child). This field can (partially) overlap the physical parent concatenated key.
- The key of the destination parent itself.

If any of the above fields is changed during a replace operation, a DA status code will be returned, and no data will be changed in the data base.

Delete Calls

In general, these calls function the same as before. If, however, you delete a concatenated segment (either of the two versions), only the logical child and its physical dependents (that is, the dependents of the real logical child) will be deleted. The destination parent can be deleted only via its physical path. In other words: "The delete is not propagated upwards across a logical relation." You can delete only those dependents of concatenated segments which are real dependents of the logical child. Examples:

- If in the logical DBD of Figure 2-25, a PART segment was deleted, the associated STOCK, PURCHASE ORDER, DESCRIPTION, and ORDER LINE segments are deleted, too. However, the associated CUSTOMER ORDER and SHIPMENT segments remain.
- If in the logical DBD of Figure 2-26, a CUSTOMER ORDER segment was deleted, the associated ORDER LINE and SHIPMENT segments are deleted, too. However, the associated PART, STOCK, PURCHASE ORDER, and DESCRIPTION segments remain.

Notice, the logical child (and its physical dependents) is always deleted whenever one of its parents is deleted.

Note: The above discussion of the DL/I calls is only applicable to our subset environment. This is explicitly related to the coding of the "RULES=" parameter as specified in Chapter 2 under the topic "Coding A Logical Relationship In A Physical DBD."

Insert Calls

Whenever you insert a concatenated segment, the destination parent must already exist in the data base. You can provide the destination parent together with the logical child in the ICAREA, but it is not used. Besides the normal status codes, an IX status code is returned when the destination parent does not exist.

LOADING DATA BASES WITH LOGICAL RELATIONSHIPS

To establish the logical relationships during initial load of data bases with logical relationships, DL/I provides a set of utility programs. These are necessary because the sequence in which the logical parent is loaded is normally not the same as the sequence in which the logical child is loaded. To cope with this, DL/I will automatically create a workfile whenever you load a data base which contains a logical child and/or logical parent. This workfile contains the necessary information to update the pointers in the prefixes of the logically related segments. Before doing so, the workfile is sorted in physical data base sequence with the prefix resolution utility (DFSURG10). This utility also checks for missing logical parents. Next, the segment prefixes are updated with the prefix update utility (DFSURGP0). After this, the data base(s) are ready to use. The above data base load, prefix resolution and update should be preceded by the prereorganization utility (DFSURPR0). This utility generates a control data set to be used by data base load, DFSURG10 and DFSURGP0. A detailed discussion of this data base load process and the associated utilities can be found in Chapter 5: "Data Base Reorganization."

Loading the Phase 2 Data Bases

Both the phase 2 data bases BE2PARTS and BE2ORDER (Figure 2-24) can be loaded with the sample data base load program DFSOADBL. Job //SAMP270 in IMSVS.PRIMEJOB shows the JCL for loading both data bases including all necessary DL/I utilities.

Notice, there is no difference (in comparison to Phase 1) in the PARTS data base application (load) program. This is because its user data has not been changed. Also, remember, the virtual logical child does not actually exist and must not be loaded. However, the real logical child in the CUSTOMER ORDER data base must be loaded as it is defined in the physical BE2ORDER data base (that is, including the logical parent's concatenated key.)

Notes:

1. You cannot use a logical DBD when initially loading a data base (PROCOPT=L(S) in the PCB).
2. The logical relationship in the PARTS data base could also be implemented with the aid of the DL/I reorganization utilities, thus avoiding a new initial load of the PARTS data base. This will be discussed in detail in Chapter 5: "Data Base Reorganization."

SAMPLE PHASE 2 PROGRAMS

Sample program BE2CORDR (member DFS2CORD for COBOL, or DFS2PCRD for PL/I) in IMSVS.PRIMESRC shows the processing of the Phase 2 logical data bases as specified in Figure 2-25 and Figure 2-26. This is the customer order processing program as defined in Chapters 1 and 2. Guidelines for its use are in the program listing. The program can be compiled and link-edited with job //SAMP242 (COBOL) or //SAMP254 (PL/I) and executed with job //SAMP272 in IMSVS.PRIMEJOB.

PROCESSING WITH SECONDARY INDEXES

For a review of the terminology and functions of secondary indexes see "DL/I Secondary Indexes" in Chapter 2. The sample environment to be used in this section is the phase 3 environment as introduced in Chapters 1 and 2. In discussing the DL/I calls in the following sections, you should refer to the phase 3 sample DBDs of Figure 2-29.

As discussed before, DL/I will always maintain the secondary index, whether or not the program making the change is using the index. As a consequence, DL/I must have access to the index data bases when processing the main data base. So, the DD statements for the index data bases must be supplied in the JCL of every job which could change the secondary index.

ACCESSING SEGMENTS VIA A SECONDARY INDEX

Retrieving Segments

The same calls are used as before. However, the index search field, defined by an XDFLD statement in the DBD will be used in the SSA for the get unique of the root segment. It defines the secondary processing sequence. (See Figure 2-34, second PCB). Figure 4-19 shows an example.

After the successful completion of this get unique call, the PCB and IOAREA look the same as after the basic GU of Figure 4-7, except that the key feedback area now starts with the purchase order number.

When using the secondary processing sequence, consecutive get next calls for the PARTs segment will present only those parts with a PURCHASE ORDER segment, the sequence being the purchase order number. It is as if the purchase order number has taken over the role of root-key from the part number. As a consequence, the key feedback area in the PCB now contains the purchase order number instead of the part number. Remember: The sequence of the parts within one specific order is undetermined. In addition, you should not use a get unique with the part number for accessing the parts segment with the secondary processing PCB. This would result in a full data base scan.

If both the primary and the secondary processing sequence are needed in one program, you should use two PCBs as in Figure 2-34.

```
77  GU-FUNC PICTURE XXXX VALUE 'GUbb'.

01  SSA005-GU-SE1PART.
  02  SSA005-BEGIN PICTURE X(19) VALUE 'SE1PARTb(FE3PSID1b='.
  02  SSA005-FE3PSID1 PICTURE X(8).
  02  SSA005-END PICTURE X VALUE ')'.

01  IOAREA   PICTURE X(256).
--------------------------------------------------------------------
MOVE PORDER-NUMBER TO SSA005-FE3PSID1.

CALL 'CBLTDLI' USING GU-FUNC,PCB-NAME,IOAREA,SSA005-GU-SE1PART.
--------------------------------------------------------------------
STATUS CODES:

   bb:  requested PART segment has been moved to IOAREA
   GE:  segment not found, requested purchase order number not in
        data base
other:  error situation
```

Figure 4-19. GU Call Using a Secondary Index

Replacing Segments

To replace segments in the indexed data base a combination of get hold and replace calls can be used as before. Again, no sequence fields may be changed. The index search fields, however, can be changed. If an index search field is changed, DL/I will automatically update the index data base via a delete old and insert new pointer segment.

Note: When using a secondary processing sequence, this could result in the later reaccessing of a data base record.

Deleting Segments

When using a secondary processing sequence, you cannot delete the index target segment (that is, the root segment). If you have a need to do so, you should use a separate PCB with a primary processing sequence.

Inserting Segments

Again, when using a secondary processing sequence, you cannot insert the index target segment. In all other cases, the ISRT call will function as before.

SAMPLE PHASE 3 PROGRAMS

Program PE3CPPUR (member DFS3CPUR for COBOL, or DFS3PPUR for PL/I) in IMSVS.PRIMESRC, shows the use of a secondary index for the purchase order processing sample. This program processes transaction TE3PCINQ as defined in Chapter 2. For more details, see the program listing. This program can be compiled and link-edited with job //SAMP341 (COBOL) or //SAMP351 (PL/I) in IMSVS.PRIMEJOB. It can be executed with job //SAMP373.

SECONDARY INDEX CREATION

A secondary index can be created during initial load of the indexed data base or later. The secondary index data base is created with the DL/I reorganization utilities. No application program is required for this creation. Chapter 5, "Reorganization," will cover this in detail.

BATCH CHECKPOINT/RESTART

The batch checkpoint/restart facility of DL/I allows long running programs to be restarted at an intermediate point in case of failure. At regular intervals (CHKP calls) during application program execution, DL/I saves, on its log tape, designated working storage areas in the user's program, the position of GSAM data bases, and the key feedback areas of non-GSAM data bases.

For each checkpoint, a checkpoint ID (message DFS681I) will be written to the OS/VS system console and to the job system output.

At restart, the restart checkpoint ID is supplied in the PARM field of the EXEC statement of the job. DL/I will then reposition the GSAM data bases and restore the designated program areas. This is accomplished with a special restart call (XRST) which must be the very first DL/I call in the program. At initial program execution, the XRST call identifies the potential program areas to be checkpointed by later CHKP calls.

USING THE XRST AND CHKP CALLS

To utilize the checkpoint/restart function of DL/I for batch programs, you should consider the following guidelines:

1. All the data sets that the program uses must be DL/I data bases. GSAM should be used for sequential input and output files, including SYSIN and SYSOUT. Any other file cannot be repositioned by DL/I and can result in duplicate or lost output.

2. The GSAM output data sets should use DISP=(NEW,KEEP,KEEP) for the initial run and DISP=(OLD,KEEP,KEEP) at restart(s).

3. SYSOUT should not be used directly. The output should be written to a GSAM file (as in 2) and be printed with an additional jobstep. IEBGENER can be used for this purpose.

4. The first call issued to DL/I must be a XRST call. Its format will be discussed later.

5. The frequency of the checkpoint call is your choice. A basic recommendation is one checkpoint for every 50 to 500 update transactions. It is good practice to program for an easy adjustment of this frequency factor.

6. After each checkpoint call, you must reposition yourself in the non-GSAM data bases by issuing a get unique call for each of those data bases. Repositioning of GSAM data bases is done by DL/I, and you should proceed with a get next (input) or an insert (output) call.

The Restart Call

Upon receiving the restart call (XRST), DL/I checks whether a checkpoint ID has been supplied in the PARM field of the EXEC card or in the workarea pointed to by the XRST call. If no ID has been supplied, a flag is set to trigger storing of repositioning data and user areas on subsequent CHKP calls (that is, DL/I assumes that this is the initial program execution, not a restart).

If the checkpoint at which restart is to occur has been supplied, the IMS/VS batch restart routine reads backwards on the log defined in the //IMSLOGR DD card to locate the checkpoint records. User program areas are restored.

The GSAM data bases active at the checkpoint are repositioned for sequential processing. Key feedback information is provided in the PCB for each data base active at the checkpoint. The user program must reposition itself on all non-GSAM data bases, just as it must do after taking a checkpoint.

The format of the XRST call is:

COBOL:

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

PL/I:

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

ASSEMBLER:

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

where:

parmcount
: is the name of a binary fullword field containing the number of arguments following. PL/I only.

call-func
: is the name of a field which contains the call function 'XRST'.

ICPCB-name
: is the name of the I/O PCB or the "dummy" I/O PCB supplied by the CMPAT option in PSBGEN (C1PCB in the sample programs).

I/O-area-len
: is the name of the length field of the largest I/O area used by the user program; must be a fullword.

work-area
: is the name of a 12 byte work area. This area should be set to blanks (X'40') before the call and tested on return. If the program is being started normally, the area will be unchanged. If the program is being restarted from a checkpoint, the ID supplied by the user in that CHKP call and restart JCL will be placed in the first 8 bytes. If the user wishes to restart from a checkpoint using a method other than IMS/VS Program Restart, he may use the XRST call to reposition GSAM data bases by placing the checkpoint ID in this area before issuing the call. This ID is the 8-byte left-aligned, user supplied ID.

1st-area-len
: is the name of a field which contains the length of the first area to be restored; must be a fullword.

1st-area
: is the name of the first area to be restored

.
.
.

nth-area-len
: is the name of a field which contains the length of the nth area to be restored (max n=7); must be a fullword.

nth-area

: is the name of the nth area to be restored (max n=7).

Notes:

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

2. The lengths of the areas specified on the XRST call must equal to or larger than the lengths of the corresponding (in sequential order) areas of any CHKP call.

3. The XRST call is issued only once and it must be the first request made to DL/I.

4. The only correct status code is bb; any other implies an error condition.

5. All "area-len" fields in PL/I must be defined as substructures. The name of the major structure should, however, be specified in the call.

   Example:

   ```
   DCL  1    I/O-AREA-LEN,
        2    L NTH FIXED BIN(31) INIT(length);
   ```

The Checkpoint Call

When DL/I receives a CHKP call from a program which initially issued a XRST call, the following actions are taken:

- All data base buffers modified by the program are written to DASD.
- A log record is written with the checkpoint ID; message DFS601I is written, specifying this ID to the OS/VS system console and job sysout.
- The user-specified areas (for example, application variables and control tables) are recorded on the DL/I log tape. They should be specified in the initial XRST call.
- The fully-qualified key of the last segment processed by the program on each DL/I data base is recorded on the DL/I log tape.

The format of the CHKP call is:

COBOL:

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

PL/I:

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

ASSEMBLER:

```
    CALL ASMTDLI,(call-func,IOPCB-name,I/O-area-len,I/O-area
                   where:area-
```

parmcount
: is the name of a binary fullword field containing the number of arguments following; PL/I only.

call-func
: is the name of a field with the call function 'CHKP'.

IOPCB-name
: is the name of the I/O PCB or dummy PCB in batch.

I/O-area-len
: is the name of the length field of the largest I/O area used by the application program; must be a fullword.

I/O-area
: is the name of the I/O area. The I/O area must contain the 8 byte checkpoint ID. This is used for operator or programmer communication and should consist of EBCDIC characters. In PL/I, this parameter should be specified as a pointer to a major structure, an array, or a character string. Recommended format:

    MMMMnnnn

    MMMM = 4 character program identification

    nnnn = 4 digit checkpoint sequence number, incremented at each CHKP call.

1st-area-len (optional)
is the name of a field that contains the length of the first area to checkpoint; must be a fullword.

1st-area (optional)
is the name of the first area to checkpoint

.
.
.

nth-area-len (optional)
is the name of a field that contains the length of the nth area to checkpoint (max n=7); must be a fullword.

nth-area (optional)
is the name of the nth area to checkpoint (max n=7).

Notes:

1. The only correct status code in batch is bb; any other specifies an error situation.

2. Before restarting a program after failure, you always must first correct the failure and recover your data bases. This is discussed in Chapter 6: "Data Base Recovery."

3. You must reestablish your position in all IMS/VS data bases (except GSAM) after return from the checkpoint (that is, issue a get unique).

4. All "area-len" fields in PL/I must be defined as substructures, see the example under note 5 of the XRST call.

5. Because the log tape is read forward during restart, the checkpoint ID must be unique for each checkpoint.

USING GSAM WITH CHECKPOINT/RESTART

Sequential Input Files

At restart time, GSAM will reposition the sequential input file. For card input, SYSIN, the whole original input deck must be used.

Sequential Output Files

At restart time, GSAM will resume writing to the output data set (DISP=OLD), based on its output record count which was written on the DL/I log tape.

Note: DL/I does not provide recovery of GSAM data sets.

SAMPLE BATCH CHECKPOINT/RESTART PROGRAMS

Program DE1CPPUR (member DFS1CPUR for COBOL of DFS1PPUR for PL/I) in IMSVS.PRIMESRC shows how to use the XRST and CHKP calls. It also shows the use of GSAM in this environment. Job //SAMP174 in IMSVS.PRIMEJCE can be used to execute this program. Job //SAMP178, shows the restart of this program. For details on exercising this program, see its program listing in IMSVS.PRIMESRC.

Note: To be compatible with IMS/VS data communication operation, the first PCB in the PSB of a batch program is a dummy PCB. (CMPAT=YES in the PSBGEN statement).

DATA COMMUNICATION APPLICATION PROGRAMMING

In the following sections, we extend the IMS/VS data base processing into the online environment. To process data bases online, message calls are added to the DL/I interface. Message calls are used to send messages to terminals and to retrieve messages sent to IMS/VS from terminals. The data base calls discussed in the first part of this chapter remain the same.

APPLICATION PROGRAMMING AND MFS

In our subset, we will exclusively use message format services (MFS) for the IBM 3270 Information Display System display and printer terminals. Therefore it is recommended that you are familiar with the section entitled "Message Format Service Overview" in Chapter 3 before using this section.

APPLICATION PROGRAM TYPES

As defined in Chapter 1, "Introduction," there are three types of IMS/VS programs:

- The Batch Processing Program (DLI) for batch processing of data bases. This program type is solely discussed in the first part of this chapter.
- The Batch Message Processing Program (BMP) for batch processing of online data bases.
- The Message Processing Program (MPP) for processing transactions entered from terminals.

There is no difference in the program structure of a DLI and BMP program. In fact, the very same program can be executed as a DLI or a BMP program if you follow the guidelines of the data base part of this chapter.

GENERAL MPP CONSIDERATIONS

All data base functions previously discussed are available in the MPP, except:

- GSAM data bases and OS/VS files cannot be used.
- The XRST call cannot be used in a MPP.
- The CHKP call should not be used in MPPs in our subset.
- The STAT call should not be used in a MPP. Its results bear no direct relationship to the data base accesses of the MPP. Information on these accesses are available via the DC Monitor. See Chapter 9, "Optimization."

GENERAL BMP CONSIDERATIONS

Any DL/I batch program written according to the guidelines in the data base part of this chapter can be executed as a BMP.

The XRST and CHKP calls and GSAM for Non-DL/I files are required if the program is to be restartable. This is especially important for update programs. The reason is that the IMS/VS program isolation function will create an enqueue element in main storage for every data base change. In addition, the old data base image will be saved on the dynamic log. Both the enqueue and the old data base image will be freed at the next synchronization point. As each CHKP call constitutes a synchronization point, these resources are freed at each CHKP call. Our subset selection of the main storage pool for enqueue elements and the size of the dynamic log is related to a CHKP frequency of one for every 100 or less data base changes.

Additional CHKP Status Code In A BMP

In comparison to the IMS/VS-DB system, one additional status code should be expected after the CHKP call in a BMP: "XD".

This XD status code signals that the IMS/VS control region is shutting down. No more DL/I calls are accepted by the CTL region from the BMP. The BMP should terminate as soon as possible.

MPP STRUCTURE AND IMS/VS INTERFACE

When the IMS/VS Data Communication feature is used, application programs can communicate with devices as well as access data bases. The program communicates logically with a device through IMS/VS rather than directly to the device. This is made possible by the IMS/VS concept of logical terminals. A logical terminal is a name related to the actual device, the physical terminal. One physical terminal can have one or more associated logical terminal names. The logical terminal name or names for each physical terminal are defined by the IMS/VS system programmer during IMS/VS system definition.

The logical terminal concept allows an application program to be independent of the characteristics of a particular physical terminal.

Generally, you need not be concerned with the actual location or address of the device. If a physical terminal becomes inoperative, its associated logical terminal(s) can be reassigned to another physical terminal, thereby causing output messages to be sent to another physical terminal. Also, each logical terminal can have unique security checking associated with it.

To an application program, therefore, a logical terminal can be viewed as just another sequential data input source or output destination. The application program interface to the logical terminal is through essentially the same call interface mechanism that was described for data base access. Access to a data base requires the use of a data base Program Communication Block (DB-PCB). Accordingly, communications with a DC device requires the use of a data communication PCB (DC-PCB).

MPPs normally reference both DB-PCBs and DC-PCBs, and must contain a mask to handle each PCB type. Figure 4-20 shows that the MPP views terminals and data from a logical view point. Any changes to the physical terminal configuration or to the actual data structures have a minimal effect on the application program.

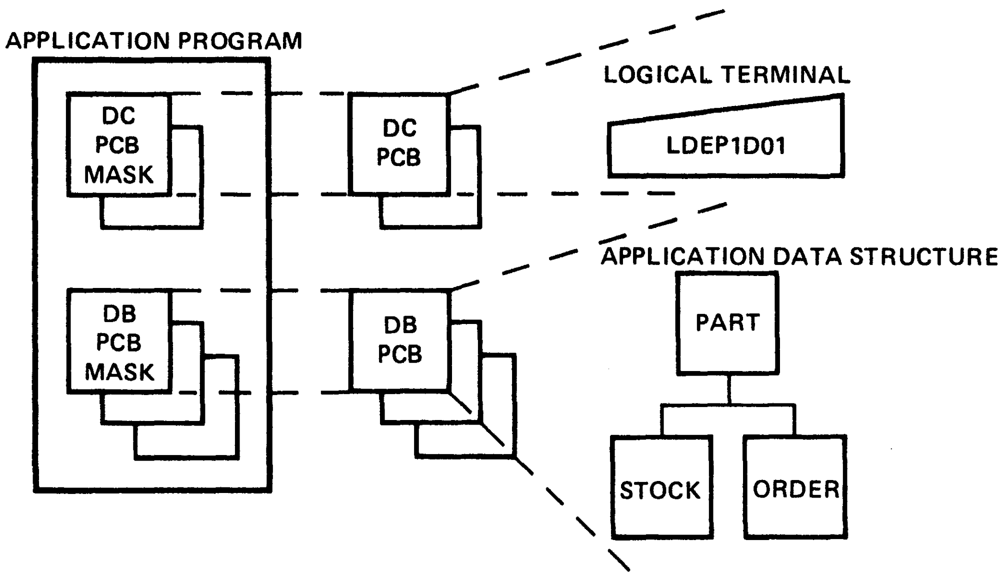


Figure 4-20. PCB Masks fcr a MPP

As for the batch system, both the DB PCBs and the DC PCBs are part of the program specification block (PSB). A PSB is required for each MPP and is created by the PSBGEN utility. See Chapters 2 and 3.

DC PCEs

There are two types of DC PCBs -- the I/O PCB and the alternate PCB. An I/O PCB is always provided by IMS/VS to the application program that executes in a DC envircnment. Alternate PCBs are optional and must be coded separately in the PSB.

I/O PCB

The I/O PCB must be used by the MPP to:

- Obtain an input message frcm a terminal.
- Return a reply tc the terminal that originated the input message. In our subset this will be required before new input is accepted from the terminal (respcnse mode).

When IMS/VS receives an input message, it queues the message according tc transaction code and schedules the application program that processes that transaction. When scheduling the program, IMS/VS passes to the program the address

of its I/O PCB plus the alternate PCB(s), if any, and the DB-PCB(s), if any, defined in its PSB. The I/O PCB contains the name of the logical terminal that entered the message (source) and can receive the reply (destination).

ALTERNATE PCB

An alternate PCB must be used by the MPP to send an output message to a destination other than the terminal that originated the input message. An alternate PCB specifies a logical terminal destination. The destination can be specified during PSB generation or during program execution.

To be able to specify a destination during program execution, the alternate PCB must be defined as modifiable during PSB generation. When an application program uses modifiable alternate PCBs, the program must set the output message destination before inserting the output message.

THE DC-PCB MASK

To support communication with IMS/VS, the MPP must contain a DC-PCB mask. As shown in Figure 4-21, a DC-PCB mask distinguishes seven fields which are filled in by IMS/VS during each message call. These fields should not be changed by the program; they are only for reference.

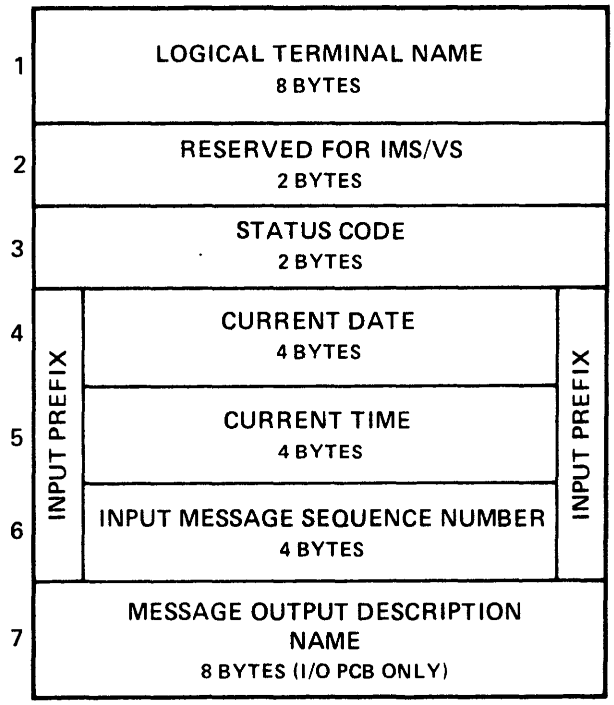


Figure 4-21. Layout of a DC-PCB Mask

1. LOGICAL TERMINAL NAME -- This field contains the name of the logical terminal that entered or will receive the message. The name is 1 to 8 bytes long, left-justified, and padded with blanks.

2. RESERVED AREA -- A 2-byte area reserved for IMS/VS.

3. STATUS CODE -- A code showing the status of the result of a DC call is placed in this 2-byte field. When a call is executed successfully, this field is set to blanks. A non-blank status code is returned on an unsuccessful call.

4 through 6.

INPUT PREFIX -- Is available only for the I/O PCB. The length of the input prefix is 12 bytes.

4. 4 bytes - Julian date (YYDDD-packed decimal, right aligned) when the input message was completely received from the physical terminal.

5. 4 bytes - Time (HHMMSS.S-packed decimal) when the input message was completely received from the physical terminal.

6. 4 bytes - Sequence number (binary) of the input message. Per terminal, since last IMS/VS start-up.

7. MESSAGE OUTPUT DESCRIPTION NAME -- Is available only for the I/O PCB. This field has meaning only when output messages are sent to terminals that use the IMS/VS Message Format Service (MFS).

   When IMS/VS supplies the first segment of an input message, it fills this field with either the name of a message output description (MOD) or blanks. The MOD name can be changed by using the output MOD name parameter of the DC output call that contains the first segment of an output message. This will be discussed later in this chapter.

COBOL Example of a DC-PCB Mask

The following example is an I/O PCB mask for COBOL message processing program. This mask would be found in the linkage section of the program. A mask for an alternate PCB would be similar but without the IN-PREFIX and MOD-NAME fields.

```
DATA DIVISION.

LINKAGE SECTION.

   01     IO-PCB.
      02  LTERM-NAME  PICTURE X(8).
      02  DLI-RESERVE PICTURE XX.
      02  STATUS-CODE PICTURE XX.
      02  IN-PREFIX.
          03  JULIAN-DATE PICTURE S9(7)       COMPUTATIONAL-3.
          03  TIME-OF-DAY PICTURE S9(7)       COMPUTATIONAL-3.
          03  MSG-COUNT      PICTURE S9(7)    COMPUTATIONAL.
      02  MOD-NAME PICTURE X(8).
```

PL/I Example of a DC-PCB Mask

The following is an example for PL/I Optimizing Compiler message processing programs. A mask for an alternate PCB would be similar but without the IN_PREFIX and MOD_NAME fields.

```
     DECLARE 1 IO_PCB BASED (IO_PCBPTR),
               2  LTERM_NAME  CHARACTER (8),
               2  DLI_RESERVE CHARACTER (2),
               2  STATUS_CODE CHARACTER (2),
               2  IN_PREFIX,
                  3  JULIAN_DATE   FIXED DECIMAL (7),
                  3  TIME_OF_DAY   FIXED DECIMAL (7),
                  3  MSG_COUNT     FIXED BINARY (31),
               2  MOD_NAME    CHARACTER (8);
```

ENTRY TO THE MPP

The entry statement to a MPP must name the DC-PCBs and the DB-PCBs. The DC-PCBs must precede the DB-PCBs, and at least one DC-PCB must be specified to provide for the I/O PCB.

- The format for an COBOL program is:

```
ENTRY 'DLITCBL' USING IO-PCB, ALT-PCB1, ALT-PCBn, DB-PCB1, DB-PCBn.
```

- The format for a PL/I optimizing compiler program is:

```
DLITPLI: PROCEDURE (IO_PCBPTR,ALT1_PCBPTR,...ALTn_PCBPTR,
                    DB1_PCBPTR,...DBn_PCBPTR) OPTIONS (MAIN);
```

Programs that are OS/VS subtasks of an application program called by IMS/VS must not issue DL/I calls. If they do, the results will be unpredictable. With PL/I, whenever PL/I multitasking is used, all tasks, even the apparent main task, operate as subtasks to a hidden PL/I control task. PL/I multitasking is therefore not allowed in an IMS/VS program.

THE DC CALLS

In addition to the DB calls, an MPP uses DC calls for the retrieval and insertion of messages. These DC calls must reference a DC-PCB as discussed in the previous section. They relate to messages.

A message is comprised of one or more segments. Figure 4-22 shows two messages: Message A is made up of Segment A1. Message B is made up of segments B1, B2 and B3.

```
MESSAGE A            MESSAGE B

┌────────────┐       ┌────────────┐
| SEGMENT A1 |       | SEGMENT B1 |
└────────────┘       |────────────|
                     | SEGMENT B2 |
                     |────────────|
                     | SEGMENT B3 |
                     └────────────┘
```

Figure 4-22. Single and Multi Segment Message

In our subset, using 3270 terminals and MFS, an input message will always be a single segment. The basic DC calls are:

- GU (get unique): to retrieve the first input message segment.
- GN (get next): to retrieve the second input message segment, in our subset, only used for conversational programs.
- ISRT (insert): to insert a message segment into the output message queue.
- CHNG (change destination): to set the LTERM destination of an alternate PCB.

The DC call format is slightly different from DB calls because there is no hierarchical structure with which to be concerned. SSAs (Segment Search Arguments) are not used for DC calls.

The format for a COBOL program is:

```
CALL 'CBLTDLI' USING CALL-FUNC, IO-PCB, IOAREA.
```

The format for a PL/I program is:

```
CALL PLITDLI (PARM_COUNT,CALL_FUNC,IO_PCBPTR,IOAREA);
```

When a transaction or input message is available for processing, the associated application program is scheduled into a message processing region. After being loaded, the program should issue a get unique (GU) call to obtain the first segment of its input message. A subsequent segment of that message is obtained with a get next (GN) call. GU and GN calls cannot be made to an alternate PCB.

If the program is serially reusable or reenterable between GU calls, GU calls can be issued for subsequent input messages until all messages are retrieved. If a program is not serially-reusable or reenterable between GU calls, the program must terminate after each GU call so that it will be reloaded and re-initialized.

It is highly recommended that the MPPs be at least serially-reusable. We will provide guidelines for this in the section "Basic MPP Flow" later on in this chapter.

Get Calls (GU, GN)

The get calls are used to retrieve segments of an input message. For each get unique (GU) or get next (GN) call, one segment is returned to the application program. IMS/VS returns the retrieved segment to a work area defined in the application program. Since the length of a message segment is variable, the work area must be large enough to contain the longest segment expected by the program.

In addition, the program should check the length field of the input message.

The first segment of an input message is obtained with a GU call against the I/O PCB. In response to a GU call, IMS/VS returns the first message segment and fills in the following I/O PCB fields:

- Source name (name of the logical terminal that originated the message).
- Status code.
- Input prefix.
- Message output description name (when present).

The format for a COBOL program is:

```
CALL 'CBLTDLI' USING GET-UNIQUE-FUNC, IO-PCB, IOAREA.
```

The format for a PL/I program is:

```
CALL PLITDLI (PARM_COUNT,GET_UNIQUE_FUNC,IO_PCBPTR,IOAREA);
```

STATUS CODES:

bb: Call successful, message segment returned in IOAREA

QC: No more input messages for this transaction; the program must terminate

other: Error situation

For programs that process multiple transaction codes, the text of the input message can be examined to determine the transaction code.

The second segment of an input message is retrieved with a GN call.

The format for a COBOL program is:

```
CALL 'CBLTDLI' USING GET-NEXT-FUNC, IO-PCB, IOAREA.
```

The format for a PL/I program is:

```
CALL PLITDLI (PARM_COUNT,GET_NEXT_FUNC,IO_PCBPTR,IOAREA);
```

STATUS CODES:

bb: Call successful, message segment returned in IOAREA

other: Error situation

Notes:

1. The get next call should only be used for conversational transactions within our subset. In that case, the GU call will retrieve the scratch pad area (SPA) and the GN call the actual input message segment. See the section on conversational programs later in this chapter.

2. The program must check the status code after each call. The handling of error status calls is discussed later in this chapter.

Insert Call (ISRT)

The insert call is used to build output messages. To build an output message in reply to the terminal that originated the input message, output message segments must be inserted to the I/O PCB. Output message segments can also be inserted to alternate PCBs. If an alternate PCB has been defined as modifiable, a change call must be used before the first insert call against the alternate PCB. The change call sets the destination of the output message.

The ISRT call format is similar to that for message get calls.

The format for a COBOL program is:

```
CALL 'CBLTDLI' USING ISRT-FUNC, DC-PCB, IOAREA, MODNAME.
```

The format for a PL/I program is:

```
CALL PLITDLI (PARM_COUNT,ISRT_FUNC,DC_PCBPTR,IOAREA,MODNAME);
```

STATUS CODES:

bb: Call successful, message segment inserted

other: Error situation

MODNAME is the label of an 8-byte field containing the name of the message output description; the name must be left-justified and padded with blanks.

This parameter should only be specified at the insert call of the first segment of a message. Although the MODNAME may be already defined in the message input description (MID) used for the input message, it is recommended that you always specify it on the ISRT call of the first message output segment. It can also be used for the alternate PCBs.

Note: For a detailed discussion on MIDs and MODs and their linkages, refer to the section "Message Format Service Overview" in Chapter 3, DC Design.

Output message segments cannot be distinguished as first and subsequent segments by the insert call. Any required distinction must be made by the program. All message segments inserted to a given DC-PCB during the processing of a single input message are treated by IMS/VS as a single output message.

At least one output message segment should be inserted to the I/O PCB.

If the MPP has DB-PCBs defined, one or more data base calls may be executed. The normal sequence of operation is to obtain the input message, issue data base calls based upon input message content, and create an output message based upon input message content and data obtained with data base calls.

Change Call (CHNG)

The change call is used to set the destination of a modifiable alternate PCB to any valid logical terminal in the system. To use the change call, the alternate PCB must have been defined as modifiable during PSB generation. The destination of the modifiable PCB must be set with the change call before any segments are inserted.

The new destination remains set until either the application program issues another CHNG, issues a GU, or terminates. At that time, IMS/VS resets the destination to blanks.

A change call for an alternate PCB cannot be issued while that PCB is being used to form a message. Therefore, multiple modifiable PCBs must be defined if messages are to be sent to several destinations while processing a single input message.

The format for a COBOL call is:

```
CALL 'CBLTDLI' USING CHNG-FUNC, ALT-PCB, DEST-NAME.
```

The format for a PL/I call is:

```
CALL PLITDLI (THREE,CHNG_FUNC,ALT_PCBPTR,DEST_NAME);
```

STATUS CODES:

bb: Call successful, destination set

other: Error situation

The destination name parameter (DEST_NAME) specifies the name of an 8-byte field containing the name of the logical terminal. The name may be 1- to 8-bytes long, uppercase EBCDIC, left-justified, and padded with blanks.

BASIC MESSAGE FORMATS

The following message formats are presented to or received by IMS/VS with Message Format Service. For a detailed discussion of MFS, see Chapter 3.

INPUT MESSAGE FORMAT

MFS edits input data from the terminal as defined in the device input format (DIF) and the corresponding message input description (MID).

The format of the one input message segment is:

LL	ZZ	TRANCODE	MFLDs

LL

is a 2-byte binary field representing the total length of the message segment, including LL and ZZ. The LL value is provided by IMS/VS for input messages.

When PL/I is used, the LL field must be declared FIXED BINARY (31), a binary fullword. The value contained in the LL field is the actual segment length minus 2 bytes. For example, if the input message segment is 20 bytes, LL is equal to 18 and represents the sum of the lengths of LL (4 bytes minus 2 bytes), ZZ (2 bytes), TRANCODE (9 bytes), and MFLDs (5 bytes).

ZZ is a 2-byte field reserved for IMS/VS.

TRANCODE

contains the transaction code as defined during IMS/VS system definition. This field is 9 bytes; the transaction code is padded with blanks. The transaction code should be defined as a 9-byte MFLD literal in the MID.

MFLDs

are the message fields as defined in the MID.

__Notes__:

1. When the MFLD contains an attribute byte (ATTR=YES), the first two bytes of this field are reserved for attribute data to be filled in by the MPP.

2. Following our MFS guidelines, all MFLDs will appear in the message segment whether or not input data is received from the terminal. Therefore the input segment length is related to the transaction code. At least a check on maximum segment length should be done by the MPP.

OUTPUT MESSAGE FORMAT

MFS edits output segments created by an application program into a device-dependent format suitable for the device to which the message is destined. Normally, the output segments contain no device-related data. All device-dependent information is provided when the message format is defined to MFS.

An output message consists of all segments presented to IMS/VS with an ISRT call between a GU call to the I/O PCB and another GU call to the I/O PCB, or normal program termination.

The layout of the output segment is:

LL	Z1	Z2	MFLDs

LL

is a 2-byte binary field representing the total length of the message segment, including LL, Z1, and Z2. The value of LL equals the number of bytes in text (all MFLDs) plus 4. The application program must fill in this count. The segment must be less than 1388 in our subset. The segment length may be less than the length defined to the MFS language utility.

When PL/I is used, the LL field must be declared FIXED BINARY (31), a binary fullword. The value provided by the PL/I application program must represent the actual segment length minus 2 bytes. For example, if an output message segment is 16 bytes, LL is equal to 14 and represents the sum of the length of LL (4 bytes minus 2 bytes, Z1 (1 byte), Z2 (1 byte), and TEXT (10 bytes).

Z1

is a 1-byte field reserved for IMS/VS; it must contain binary zeros.

Z2

is a 1-byte field, which always should be a blank (bit 1 on, X'40') in our subset, as we only allow for one segment per logical=physical page.

MFLDs

are the message fields as defined in the message output descriptor.

Field Format

All fields in output segments are defined as fixed length and fixed position. Fields can be truncated or omitted by two methods: The first method is by inserting the appropriate LL field, which truncates the segment. The second method is by placing a NULL character (X'3F') in the field. Fields are scanned left to right for a null character; the first null encountered terminates the field.

Note: The above two methods will not clear the contents of protected fields on the screen. To completely clear such a field, a blank followed by a NULL character (X'403F') should be inserted in the first two positions of the field, that is, immediately following the attribute bytes, if any. Positioning of all fields in the segment remains the same regardless of null characters. Fields truncated or omitted are padded by MFS with a program tab character for display terminals, and blanks for printer terminals.

Dynamic Attribute Modification And Cursor Control

An option of MFS allows you to dynamically modify the attributes of a device field, although attribute byte characteristics are normally specified in the MOD. This option reserves the first 2 data bytes of an output message field for attribute definition. You must add these 2 bytes to the normal field length. Any errors detected in the 2-byte specification cause the entire request to be ignored and the attributes defined on the appropriate DFLD statement for the device format will be used. Any output field can have attribute bytes defined.

The 2 attribute bytes are defined as follows:

Byte	Bit	
0	0-1	Both bits are on, requests that the cursor be placed on the first position of this field on the device. Only the first MFLD with a cursor-positioning request in the MOD is used to position the cursor. These bits must be 00 or 11.
	2-7	Must be off

Byte	Bit	
1	0	Must be on
	1	a) If on, these attribute specifications are to replace the attribute byte defined for the field.
		b) If off, these attribute specifications are to be added to the attribute byte defined for the field (logical OR operation)
	2	Protected
	3	Numeric
	4	High-intensity
	5	Nondisplayable
	6	Detectable (not included in subset)
	7	Premodified

Bits 4, 5, and 6 are mutually exclusive. If more than one is set, bit 4 takes precedence over bits 5 and 6; bit 5 takes precedence over bit 6. Note: If a message field is to be omitted from the device output data, the attribute bytes preceding the NULL character must be binary zeros, or the first attribute byte must be a NULL character itself.

Multiple Page Output Messages

With MFS, you can easily build a multiple page output message. After insertion of such a message, the terminal operator can view it, one page at a time. He also can go back to the beginning of the message if desired.

In our subset each segment is one logical page (one LPAGE statement in the MOD) and is also one physical page (one DPAGE statement in the DOF) or one physical screen or printer page.

LPAGE Selection: When a message has multiple LPAGEs, the value of a special message segment field is used for the selection of the LPAGE which will be used for the formatting of that segment. For a detailed description see Chapter 3, the COND= parameter of the LPAGE statement. If the condition value as specified by the program does not match any of the LPAGE defined values, the last defined LPAGE will be used.

WRITING A SIMPLE MPP

The basic flow of a MPP and the message calls used are shown in Figure 4-23 and described below.

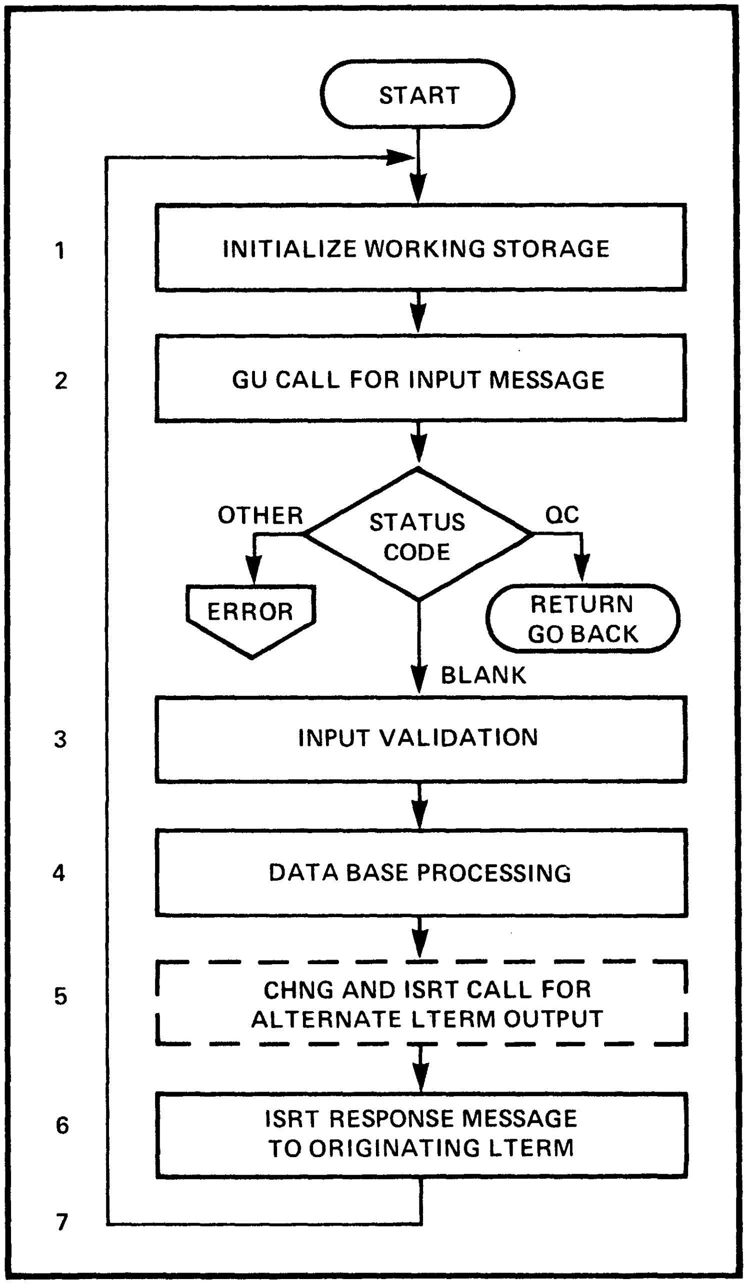


Figure 4-23. Basic MPP Flow and Calls

Legend:

1. After getting control, the MPP must initialize its working storage, as this may contain leftover data from a previously processed message (likely from another terminal). This is also a requirement for reusability.

2. The MPP retrieves the one input message segment with a GU call referencing the I/O-PCB. A blank status code means the message is placed by IMS/VS in the MSG-AREA specified in the call. A QC status code means there are no more messages in the input queue. The MPP must then return control to IMS/VS. Any other status code is an error condition and should be handled by an error code status routine.

3. The input is validated. This should include:

   - Checking the length of input message

   - Checking the format, value and consistency of input data fields

   This validation should be as complete as possible and be done before any data base access.

4. The data base processing is performed. For data base calls and status code handling see the data base calls at the first part of this chapter.

5. Optionally, a CHNG call to the alternate I/O-PCB is used to set an alternate destination. This, for instance, is required to print output on a 3270 printer terminal. The change call must not be used against the I/O-PCB. All non blank status codes should be handled by an error code status routine.

   The CHNG call is followed by one or more ISRT calls for message output segments to this alternate output terminal.

   Note: Only one CHNG call is allowed per alternate PCB for each input message.

6. The response output message is inserted to the originating LTERM via the I/O-PCB. One ISRT call is required for each output message segment. Any non-blank status code is an error condition.

7. The processing of the current input message is now completed. The program should now go back to the initialization of its working storage and the retrieval of the next input message (if any).

   Note: The first thing IMS/VS does after receiving the GU call is the synchronization point processing of the previous input message. This includes the release of the data base change enqueue elements. As a consequence, the data base positions of all DB PCBs are cleared. So after each message GU you should start with data base GU(s) to access the data base segments as requested by the new input message. This new input message is almost certainly from another user (LTERM).

SAMPLE COBOL INQUIRY MPP

Listed on the next two pages is a sample COBOL MPP, PE4CNINQ (member DFS4CNAM in IMSVS.PRIMESRC). This program expects a terminal to input the customer's number. It will display the customer name and address. It uses the sample formats listed in member OE4CNIO1 in IMSVS.PRIMESRC. Job //SAMP441 can be used for its compilation. The same COBOL programming considerations used for the batch DL/I program apply. See COBOL Programming Considerations in the first part of this chapter.

COBOL Compile Options for MPPs

Due to the way IMS/VS loads the MPP, only the following combinations of COBOL compile options should be used in our subset:

- NORES,NODYNAM,NOENDJOB

- RES,DYNAM,ENDJOB

The first combination is recommended, as the second combination must be changed when the IMS/VS program preload option is used later.

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. 'PE4CNINQ'.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  GU               PIC X(4)     VALUE 'GU'.
       77  ISRT             PIC X(4)     VALUE 'ISRT'.
       77  END-SWITCH  PIC X    VALUE '0'.
           88  NO-MORE-INPUT    VALUE '1'.
       77  NOT-FOUND-MSG   PIC X(35) VALUE
             'INVALID NUMBER - PLEASE RE-ENTER'.
       77  ERROPT  PIC X(4)     VALUE '1   '.
       77  MODNAME      PIC X(8)  VALUE 'OE4CNIO1'.
       77  BAD-CALL     PIC X(8)  VALUE 'BAD CALL'.

       01  INPUT-MESSAGE.
           04  FILLER  PIC X(4).
           04  TRANS-CODE  PIC X(9).
           04  FE00GCNR PIC X(6).
           04  FILLER  PIC X(60).

       01  OUT-MESSAGE.
           02  OUT-LL  PIC S9(3) COMP VALUE +111.
           02  OUT-ZZ  PIC S9(3) COMP VALUE +0.
           02  OUT-DETAILS.
               04  FE2PCNUM    PIC X(6).
               04  FE2PCNAM    PIC X(20).
               04  FE2PCADR    PIC X(20).
               04  FE2PCCTY    PIC X(20).
               04  FE2PCPCD    PIC X(6).
           02  OUT-ERROR     PIC X(35).

       01  SE2PCUST.
           04  FE2PCNUM    PIC X(6).
           04  FE2PCNAM    PIC X(20).
           04  FE2PCADR    PIC X(20).
           04  FE2PCCTY    PIC X(20).
           04  FE2PCPCD    PIC X(6).
           04  FILLER      PIC X(40).

       01  CUSTOMER-SSA.
           04  FILLER  PIC X(19) VALUE 'SE2PCUST(FE2PCNUM ='.
           04  SSA-CNUM  PIC X(6).
           04  FILLER  PIC X VALUE ')'.

       LINKAGE SECTION.
      *       PCB FOR INPUT OUTPUT LOGICAL TERMINAL
       01  C1PC.
           02  FILLER      PIC X(10).
           02  C1PCSTAT    PIC X(2).
      *       PCB FOR CUSTOMER DATABASE
       01  D1PC.
           02  FILLER      PIC X(10).
           02  D1PCSTAT    PIC X(2).
           EJECT

       PROCEDURE DIVISION.

           ENTRY 'DLITCBL' USING C1PC, D1PC.
```

```
    PERFORM READ-MESSAGE.
    PERFORM PROCESS-MESSAGES UNTIL NO-MORE-INPUT.
    GOBACK.
READ-MESSAGE.
    CALL 'CBLTDLI' USING GU, C1PC, INPUT-MESSAGE.
    IF C1PCSTAT = 'QC'
        THEN MOVE '1' TO END-SWITCH
        ELSE IF C1PCSTAT NOT = SPACES
             THEN CALL 'DFSOAER' USING
                   C1PC, BAD-CALL, INPUT-MESSAGE, ERROPT.
PROCESS-MESSAGES.
    MOVE FEROGCNR TO SSA-CNUM.
    PERFORM READ-CUSTOMER-DB
    IF D1PCSTAT = SPACES
        THEN MOVE CORR SE2PCUST TO OUT-DETAILS
             MOVE SPACES TO OUT-ERROR
        ELSE MOVE NOT-FOUND-MSG TO OUT-ERROR
             MOVE SPACES TO OUT-DETAILS.
    PERFORM ISRT-MESSAGE.
    PERFORM READ-MESSAGE.
READ-CUSTOMER-DB.
    CALL 'CBLTDLI' USING GU, D1PC, SE2PCUST, CUSTOMER-SSA.
    IF D1PCSTAT = SPACES OR 'GE'
        THEN NEXT SENTENCE
        ELSE CALL 'DFSOAER' USING D1PC, BAD-CALL,
              SE2PCUST, ERROPT.
ISRT-MESSAGE.
    CALL 'CBLTDLI' USING ISRT, C1PC, OUT-MESSAGE, MODNAME.
    IF C1PCSTAT NOT = SPACES
        THEN CALL 'DFSOAER' USING C1PC, BAD-CALL,
                   OUT-MESSAGE, ERROPT.
```

SAMPLE PL/I INQUIRY MPP

Listed below is a sample PL/I MPP, PE4PNINQ (member DFS4PNAM in IMSVS.PRIMESRC). This program expects a terminal to input the customer number. It will display the customer name and address. It uses the sample formats of member OE4CNI01 in IMSVS.PRIMESRC. Job //SAMP451 can be used for its compilation.

```
PE4NINQ: PROCEDURE (C1PC_PTR,D1PC_PTR) OPTIONS (MAIN);

/* * * D E C L A R A T I O N S * * */

  DCL 1 C1PC BASED (C1PC_PTR),
        2 FILL CHAR (10),
        2 STAT CHAR (2),
      1 D1PC BASED (D1PC_PTR) LIKE C1PC;

  DCL 1 INPUT_MESSAGE,
        2 FILL1 CHAR (6),
        2 TRANS_CODE CHAR (9),
        2 FE00GCNR CHAR (6),
        2 FILL2 CHAR (60),

      1 OUT_MESSAGE,
        2 OUT_LL INIT (111) FIXED BINARY (31),
        2 OUT_ZZ INIT (0) FIXED BINARY (15),
        2 OUT_DETAILS,
          3 FE2PCNUM CHAR (6),
          3 (FE2PCNAM,
             FE2PCADR,
             FE2PCCTY) CHAR (20),
          3 FE2PCPCD CHAR (6),
        2 OUT_ERROR CHAR (35),

      1 SE2PCUST,
        2 CUST_DETAILS LIKE OUT_DETAILS,
        2 FILL CHAR (40),

      1 CUSTOMER_SSA,
        2 FILL1 CHAR (19) INIT ('SE2PCUST(FE2PCNUM ='),
        2 SSA_CNUM CHAR (6),
        2 FILL2 CHAR (1) INIT (')');

  DCL ((GU INIT ('GU'),
        ISRT INIT ('ISRT'),
        ERROPT INIT ('1')) CHAR (4),
       (MODNAME INIT ('OE4CNIO1'),
        BAD_CALL INIT ('BAD CALL')) CHAR (8),
       (THREE INIT (3),
        FOUR INIT (4)) FIXED BINARY (31)) STATIC,
      (C1PC_PTR,D1PC_PTR) POINTER,
      (PLITDLI, DFSOAER OPTIONS (ASSEMBLER)) ENTRY;

/* * * P R O C E S S    M E S S A G E S * * */

READ_MESSAGE:

  CALL PLITDLI (THREE,GU,C1PC_PTR,INPUT_MESSAGE);
  IF C1PC.STAT = 'QC' THEN RETURN;
  IF C1PC.STAT ¬= ' '
     THEN CALL DFSOAER (C1PC,BAD_CALL,INPUT_MESSAGE,ERROPT);
  SSA_CNUM = FE00GCNR;

/* * * R E A D   C U S T O M E R   D A T A   B A S E * * */

  CALL PLITDLI (FOUR,GU,D1PC_PTR,SE2PCUST,CUSTOMER_SSA);
  IF D1PC.STAT = ' ' THEN DO;
     OUT_DETAILS = CUST_DETAILS;
     OUT_ERROR = ' ';
     END;
  ELSE IF D1PC.STAT = 'GE' THEN DO;
     OUT_ERROR = 'INVALID NUMBER - PLEASE RE-ENTER';
     OUT_DETAILS = ' ';
     END;
  ELSE CALL DFSOAER (D1PC,BAD_CALL,SE2PCUST,ERROPT);

/* * * I N S E R T   M E S S A G E * * */

  CALL PLITDLI (FOUR,ISRT,C1PC_PTR,OUT_MESSAGE,MODNAME);
  IF C1PC.STAT ¬= ' '
     THEN CALL DFSOAER (C1PC,BAD_CALL,OUT_MESSAGE,ERROPT);
  GO TO READ_MESSAGE;

  END PE4NINQ;
```

HANDLING ERROR STATUS CODES

The handling of error status codes in an MPP is the same as previously discussed for a DL/I batch program. The same status code error routine can be used. See the section "Status Code Error Routine" earlier in this chapter.

CONVERSATIONAL PROCESSING

For an introduction to conversational processing, see Chapter 3, "Data Communication Design," in the section entitled "Conversational Processing."

RETRIEVING THE SPA AND TERMINAL INPUT

When an MPP that processes a conversational transaction code receives control, the GU call against the I/O PCB retrieves the scratch pad area (SPA). The subsequent GN call will retrieve the actual input message. Data saved in the SPA can be in any form. The GU call for retrieving the SPA in COBOL is:

```
CALL 'CBLTDLI' USING GU-FUNC,IO-PCB,SPA-AREA.
```

In PL/I:

```
CALL PLITDLI (THREE,GU_FUNC,IO_PCBPTR,SPA_AREA);
```

Status Codes:

bb: SPA retrieved in working storage field SPA-AREA.

QC: No more input transactions; return control to IMS/VS.

other: Error situation

SCRATCHPAD AREA FORMAT

The SPA format is:

LL	XXXX	TRAN CODE	USER WORK AREA

where:

LL

is a halfword binary field containing the total number of characters in the SPA, including LL, XXXX, TRAN CODE, and USER WORK AREA. This field must not be modified by the user. The size of all SPAs in our subset is fixed at 1300 bytes. When PL/I is used, the LL field must be declared FIXED BINARY (31), a binary fullword. The two extra bytes must not be included in the LL value.

XXXX

is a 4-byte area reserved for IMS/VS. XXXX must not be modified by the user.

TRAN CODE

is an 8-byte field containing the transaction code that caused the program to be scheduled. The transaction code can be from 1 to 8 bytes, left-justified, and padded with blanks.

Note: If the MPP processes both conversational and non-conversational transactions, the TRANCODE should be checked after the GU to determine if a GN is required.

USER WORK AREA

This area is for retaining user information (for example, intermediate calculations, data retrieved from the data base, or previous input data) required by the MPP for processing of subsequent input data from the same terminal. This area is cleared to binary zeros on the initial entry of the conversation.

After the input scratchpad area and input message have been obtained, one or more data base calls may be made and one output message may be built. The application program may wish to retain data entered from the terminal or obtained from data bases. This data is saved in the user work area portion of the scratchpad.

Layout Of SPA User Work Area

In general, three different categories of data can be stored in the SPA:

- Conversation control data, used to interrelate the successive input messages of a terminal.
- Input data saved from previous input messages, not yet stored in the data base.
- Data base information already retrieved in the processing of previous input from the terminal.

The conversational control data is used to keep track of the conversation. You should record which input fields were in error, what the next expected input would be, etc.

You must also save data base position information (for example, root-key values) as IMS/VS will have cleared the data base position during synchronization point processing. This will occur between terminal interactions.

Input data must be saved in the SPA if you don't want to update the data base until all input is received and successfully processed.

Saving data base data in the SPA should only be done if doing so would save DL/I data base calls during processing of subsequent input messages of this terminal.

Input Message Format

From a terminal operator's viewpoint, the format of the input data at the terminal is the same as any nonconversational transaction-type message. IMS/VS removes the transaction code from the message segment and places it in the scratchpad area. The message segment is left justified to remove the transaction code. It is retrieved by the GN call issued after the GU call that retrieved the scratchpad. The layout of the input message segment data processed by MFS is as defined in the MID.

Note: If the transaction code is defined in the MID (as we do), IMS/VS will only remove this transaction code at the first pass. If the same MID is used for subsequent passes the 9 byte TRANCODE field defined in the MID will be present. See sample program PE4CORDR (member DFS4CNEW for COBOL, or DFS4PNEW for PL/I) in IMSVS.PRIMESRC for more details.

DATA BASE PROCESSING IN CONVERSATIONAL MODE

The actual DL/I data base calls for a conversational program are exactly the same as before.

Remember, the MPP's data base position is cleared by IMS/VS synchronization point processing between successive terminal input messages (interactions).

INSERTING THE SPA AND TERMINAL OUTPUT

If the application program modifies or initializes any SPA fields, it must return the SPA to IMS/VS before issuing another GU or terminating. A SPA is returned to IMS/VS by inserting it to the I/O PCB.

The insert (ISRT) call for COBOL is coded as follows:

```
CALL 'CBLTDLI' USING ISRT, IO-PCB, SPA-AREA.
```

or, in PL/I:

```
CALL PLITDLI (THREE,ISRT_FUNC,IO_PCBPTR,SPA_AREA);
```

Status Codes:

bb: Call successful, SPA accepted by IMS/VS.

other: Error situation.

Output Message Format

A response to the originating terminal is required to allow the conversation to continue. The terminal operator is prevented from entering more data to be processed (except IMS/VS commands) until he has received this response.

The output message segment format for a conversational application program is the same as for any nonconversational output message.

Terminating The Conversation

A conversation may be terminated by the conversational program, terminal operator, master terminal operator, or IMS/VS. A conversational program terminates a conversation by:

- Blanking the transaction code in the SPA and returning the SPA to IMS/VS through an ISRT call. This terminates the conversation as soon as the terminal has received the message response. This is the recommended procedure.

The terminal operator terminates a conversation by:

- Entering a /EXIT command from the terminal participating in the conversation.

The master terminal operator terminates a conversation by:

- Entering a /EXIT command which specifies the terminal in conversation.

- Entering a /START LINE (no PTERM specified) for the line of a terminal in conversation.

IMS/VS terminates a conversation if, after a successful GU or insertion of the SPA, a conversational application program fails to insert a message. When this situation occurs, IMS/VS sends the message DFS3272I NO RESPONSE, CONVERSATION TERMINATED to the terminal, ends the conversation, and completes synchronization point processing.

When terminating the conversation, IMS/VS deletes the current SPA. If the next terminal input message is for a conversational transaction, a fresh SPA is made available to the program. It is recommended that you terminate the conversation at each logical end (for example, when an order is stored in the data base) of an interactive session. This can best be done by the MPP. Because the transaction code is defined in the MID, no special terminal operator action is required to restart the same conversation (for example, entry of next order). A transaction code password is required for each first pass of the conversation if it is a password protected transaction.

Notes:

1. You should implement a standard subfunction code (for example, END) in a predefined input format field to allow the terminal operator to request the MPP to terminate the conversation.

2. A help function (for example, HELP) is recommended for complex conversations. The MPP could resend the latest message based on SPA content together with advice about the next possible action.

RULES FOR WRITING CONVERSATIONAL PROGRAMS

The following rules should be observed when writing conversational programs within our subset.

- The first 6 bytes of the SPA cannot be modified in any way by the application program. (IMS/VS uses these 6 bytes to identify the SPA.)

- To terminate a conversation, the transaction code (beginning in position 7) should be changed to blanks.

- If modified by an application program, the SPA must be returned to IMS/VS through an ISRT call in order to make the updated SPA available during the next interaction of the conversation.

- The SPA cannot be returned to IMS/VS more than once (that is, for every GU call for the SPA, there is only one ISRT call for the SPA).

- One and only one response output message must be inserted to the I/O-PCB for each SPA/MSG input. This message can consist of as many segments as required.

- Conversational programs must be designed to handle the condition in which the first GU call to the I/O-PCB produces no message to process. This condition can occur if the operator cancels the conversation through an /EXIT command, prior to the program issuing a GU call, and this was the only message in the queue to be processed.

WRITING A CONVERSATIONAL MPP

The basic flow of a conversational MPP and the message calls used are shown in Figure 4-24 and described in the following.

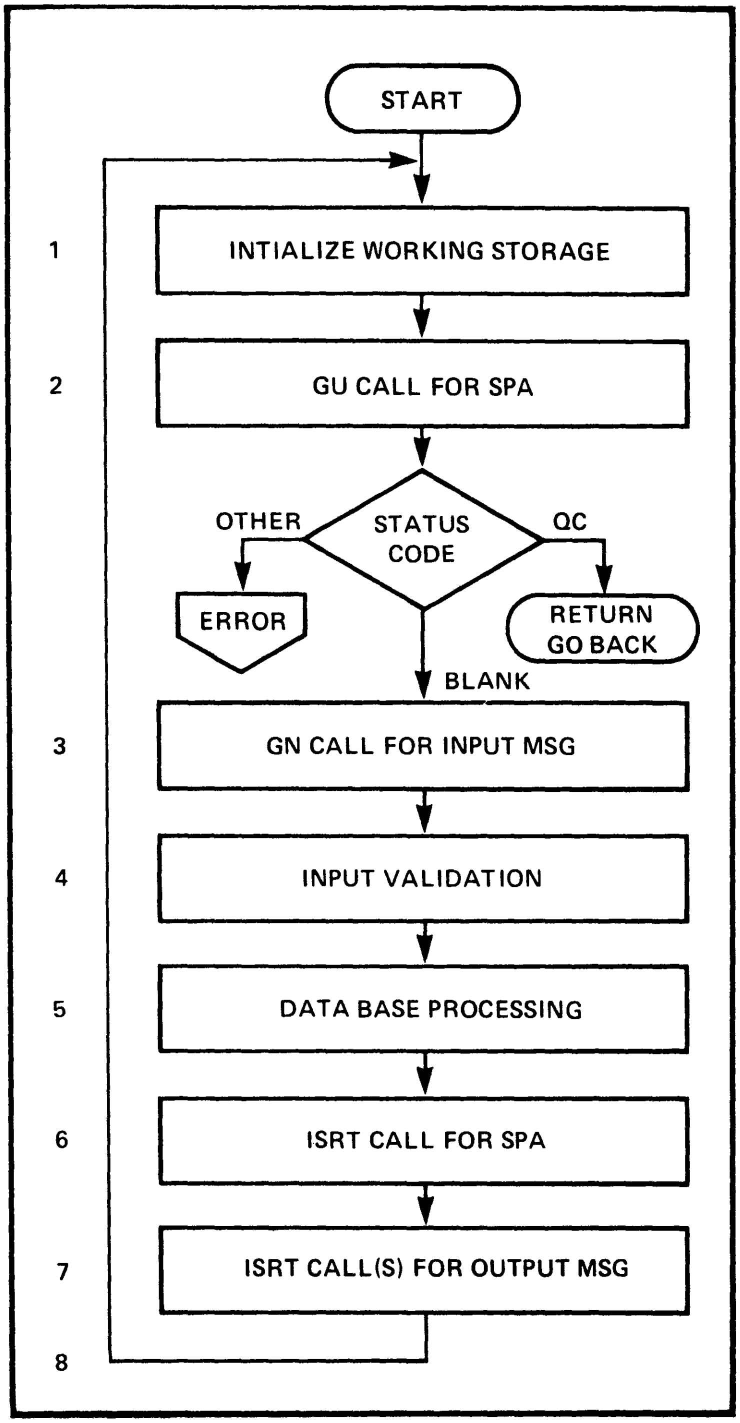


Figure 4-24. Conversational MPP Flow and Calls.

The following notes relate to the numbers in Figure 4.24:

1. After receiving control, the MPP must initialize its working storage as this may contain leftover data from a previously processed message (likely from another terminal).

2. The MPP retrieves the SPA with a GU call, referencing the IO-PCB. A blank status code means the SPA is placed by IMS/VS in the SPA-AREA specified in the call. A QC status code means, there are no more messages in the input queue. The MPP must then return control to

IMS/VS. Any other status code is an error condition and should be handled by an error code status routine (see "handling Error Status Codes" earlier in this chapter).

3. The actual terminal input is retrieved by a GN call, referencing the same IO-PCB. A blank status code means the one input message segment is placed by IMS/VS in the MSG-AREA specified in the call. Any other status code is an error condition.

4. The input is validated. This should include:

   - Checking the SPA for status of conversation; what was the expected input
   - Checking the length of input message
   - Checking the format, value and consistency of input field using conversation control information in the SPA.

   This validation should be as complete as possible and be done before any data base accesses.

5. The data base processing is done as before. Data base elements and their update status required for subsequent input message processing can be saved in the SPA.

6. The updated SPA is returned to IMS/VS with a ISRT call. Only a blank status code is acceptable. If the SPA content is of no use in the processing of the next terminal input, the conversation should be terminated by blanking the transaction code in the SPA before the ISRT call.

7. The response output message is inserted to the originating LTERM via the I/O-PCB. One ISRT call is required for each output message segment. Any non-blank status code is an error condition.

8. The processing of the current input message is now completed. The program should now go back to the initialization of its working storage and the retrieval of the next SPA + input message (if any).

SAMPLE CONVERSATIONAL MPPs

Two COBOL and PL/I conversational MPPs are included in IMSVS.PRIMESRC:

- PE4CORDR (member DFS4CNEW for COBOL and DFS4PNEW for PL/I), which processes transaction TE4CORDR for the insertion of new customer orders into the data base.
- PE4COCDEL for COBOL and DFS4PDEL for PL/I (member DFS4DEL), which processes transactions TE4CODEL and TE4COCNG for the deletion and change, respectively, of customer orders in the data base.

You should study these programs, especially the way they handle the input message, output messages, and the SPA.

TESTING YOUR MPP

Testing of a MPP can most efficiently be done in batch mode using a terminal simulator program, such as the FDP, Batch Terminal Simulator II, 5796-PGT.

For more information see SH20-1844, "BTS II Program Description And Operation Manual."

CHAPTER 5. DATA BASE REORGANIZATION/LOAD PROCESSING

ABOUT THIS CHAPTER

This chapter consists of three parts:

1. Introduces the function of data base reorganization in a DL/I environment. It is a first-time introduction into the requirements for, and the process of, data base reorganization. It gives an overview of the DL/I utilities for reorganization to be used in the subset.

2. Gives a formal description of the available DL/I utilities for data base reorganization. As such, it is the main source of reference for the actual use of the utilities.

3. Introduces the use of the utilities for a particular environment. It proceeds along the three phases of our subset sample environment from the reorganization of one data base up to the transition of one phase into another. Samples are provided for each function. It contains guidelines for the design of your own reorganization procedures.

WHAT IS REORGANIZATION

Reorganization is the process of changing the physical storage and/or structure of a data base to better achieve the application's performance requirements. We distinguish between the following two types:

- Physical reorganization, to optimize the physical storage of the data base.

- Logical reorganization, to optimize the data base structure.

WHEN TO REORGANIZE

Physical reorganization is normally required by one of the following:

- Degradation in processing program performance due to degraded data base storage, that is, the segments belonging to one data base record are stored over excessive CI/blocks. This is normally shown by an increase in the number of physical I/Os per transaction. Chapter 9, "Optimization," provides guidelines for monitoring this. Additionally, the VSAM catalog contains the number of control interval (CI) and control area (CA) splits. This information can be printed with access method services.

- Lack of free space in the data base, caused by (foreseen) large quantities of segment inserts. Again, for VSAM, the catalog will provide information on this. For HDAM/OSAM the VTOC can be checked for the use of secondary extents. Also, for HDAM, an increase in the number of I/Os per transaction could indicate an RAA (root addressable area) which is too small.

Note: Should an abnormal termination due to lack of disk space occur during normal processing, the standard recovery procedures of Chapter 6, "Data Base Recovery," should be used. The allocated space to the affected data base must of course be increased.

Logical reorganization is normally caused by design changes in the data base. In our subset we will address some changes under the topic "Applying Structural Changes" later in this chapter.

THE FREQUENCY OF REORGANIZATION

The frequency of reorganizing is largely dependent on the application environment. However both VSAM and DL/I contain special facilities to minimize the need for reorganization. If the initial allocation of space includes sufficient (distributed) free space, the need for physical reorganization would normally be quite low (typically once or twice a year).

STEPS IN REORGANIZATION

There are three major steps in reorganization:

1. Unload the data bases.

2. Delete the old space, redefine the new space, and optionally change DBD parameters (DBDGEN).

3. Restore the data bases.

Because step 2 above deletes the existing data base, it is recommended that you make an image copy (see Chapter 6, "Data Base Recovery") just before you do the unload. Another method would be to rename the old data space and define new data space. The old data space can then be deleted after reorganization and subsequent image copies are made.

You should also make image dumps of all your data bases immediately after the reload and before any application program is executed.

OVERVIEW OF THE REORGANIZATION/LOAD UTILITIES

The DL/I reorganization utilities provide three basic functions:

1. The reorganization of DL/I data bases.

2. Establishing logical relationship connections when initially loading data bases having logical relationships.

3. Creation of secondary INDEX data base(s) when you load data bases or when you reorganize them.

The seven basic utility programs involved in data base reorganization/load processing are:

1. INDEX Reorganization Unload (DFSURUL0)

2. INDEX Reorganization Reload (DFSURRL0)

3. HD Reorganization Unload (DFSURGU0)

4. HD Reorganization Reload (DFSURGL0)

5. Data Base Prereorganization (DFSURPR0)

6. Data Base Prefix Resolution (DFSURG10)

7. Data Base Prefix Update (DFSURGP0)

Of these utilities, there are two types: physical reorganization utilities and logical relationship resolution utilities.

PHYSICAL REORGANIZATION UTILITY PROGRAMS

There are two sets each of two physical reorganization utilities.

The INDEX Reorganization Utilities

The INDEX Reorganization Unload utility (DFSURUL0) and the corresponding INDEX Reload utility (DFSURRL0) can be used to:

- Reorganize the primary index data base of a HIDAM data base.
- Create a secondary index data base.
- Reorganize a secondary index data base.

The HD Reorganization Utilities

The HD Reorganization Unload utility (DFSURGU0) and the corresponding Reload utility (DFSURGL0) can be used to:

- Reorganize a HDAM, HIDAM, or SHISAM data base.
- Create work data sets if the data base being reloaded includes logical relationships and/or secondary indexes.

Notes:

- The HD utilities should be used for the unload/reload of a SHISAM data base only if this data base is to be converted to a HDAM or HIDAM data base.
- Use of the HD Unload/Reload utilities in making structural changes to a data base is discussed later in this chapter under "Applying Structural Changes."

LOGICAL RELATIONSHIP RESOLUTION UTILITY PROGRAMS

The following three logical relationship resolution utilities programs are required when initial loading or reorganizing data bases with logical relationships: (1) Data Base Prereorganization, (2) Data Base Prefix Resolution, and (3) Data Base Prefix Update.

Data Base Prereorganization Utility

This utility creates a control data set that is used by other utilities. This control data set is needed when initially loading or reorganizing data base(s) with logical relationships and/or secondary indexes.

Data Base Prefix Resolution Utility

This utility combines and sorts all work data sets generated by the HD Reload utility or by the initial data base load process. This utility generates an output work data set that contains the prefix information needed to complete the loading and/or reorganization of data bases which contain logical relationships. If secondary indexes are present, a separate output data set is also generated, used to build these indexes.

Data Base Prefix Update Utility

This utility uses the output data set generated by the Data Base Prefix Resolution utility to update the prefix of each segment whose prefix information is affected by a data base load and/or reorganization.

INDEX REORGANIZATION UNLOAD UTILITY (DFSURUL0)

A flow diagram of the INDEX Reorganization Unload utility is shown in Figure 5-1.

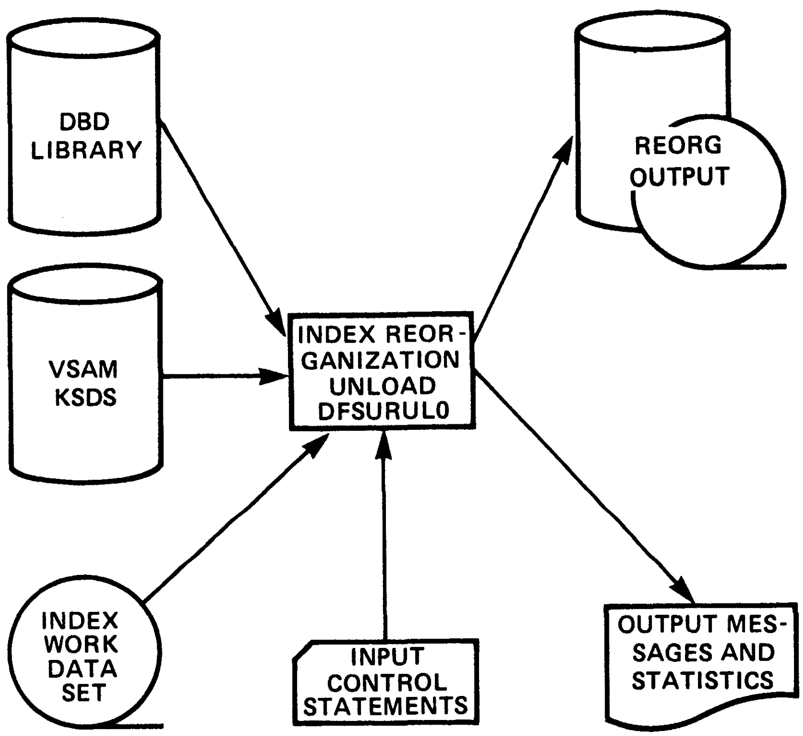


Figure 5-1. INDEX Reorganization Unload Utility

JCL STATEMENTS

The INDEX Reorganization Unload utility is executed as a standard OS/VS job. The following JCL statements are required:

EXEC	This statement must be in the form: PGM=DFSRRC00,PARM='ULU,DFSURUL0'
IMS DD	Defines the library containing the DBD that describes the data base to be reorganized (that is, DSN=IMSVS.DBDLIB,DISP=SHR). This data set must reside on a direct access device.
SYSPRINT DD	Defines the output message and statistics data set.
SYSIN DD	Defines the input control statement data set.

vsamin DD — Defines the VSAM KSDS data set to be reorganized. The ddname must be the same as the name in the DBD that describes this data set. It must also appear on the utility control statement in the SYSIN data set of this job step.

dataout DD — Defines the reorganized output data set. It can be any name, but the name must appear in the associated utility control statement. The data set must reside on either tape or a direct access device.

This sequential data set is blocked to the blocksize of the output device, up to a maximum of 16K. Since the blocking factor is determined at execution time, standard labels must be used on all output volumes.

indexwrk DD — Describes the output data set (DFSURIDX) from the Prefix Resolution program which contains secondary index information. This statement is required if the utility control statement is type "X"; otherwise, it is optional. The ddname must be the same as the name starting in position 40 of the control statement.

DFSVSAMP DD — Describes the data set that contains the buffer information required by the DL/I Buffer Handler. This DD statement is required. (For additional information, see "Defining the IMS/VS Data Base Buffer Subpools" in Chapter 7.)

UTILITY CONTROL STATEMENT

```
  1 2 4        13       22                  40        50               80
┌──────────────────────────────────────────────────────────────────────┐
│                                                                      │
│ R 1 dbdname vsamin  dataout             indexwrk [comments]          │
│ X           ddname  ddname              ddname                       │
│                                                                      │
└──────────────────────────────────────────────────────────────────────┘
```

Position	Description
1	This must be either 'R' or 'X'. An 'R' should be coded if this is a separate reorganization of an existing INDEX data base. An 'X' should be coded if this is the creation of an INDEX data base, that is, if the VSAM KSDS is "empty."
2	This must be a 1. There is no default, and if this field is left blank an error message is generated.
4	This must be the name of the DBD that includes the KSDS to be reorganized.
13	This must be the ddname of the KSDS to be reorganized. It must appear in the referenced DBD, and a corresponding DD statement must have been provided.
22	This must be the ddname of the output data set. A corresponding DD statement must have been provided.

Position	Description
40	This must be the ddname of the secondary index work data set if this control statement is type "X".
50	Comments can be placed in positions 50 through 80.

Note: All other positions must be blank.

RETURN CODES

This program returns codes preceded (in the case of errors) by numbered messages to the SYSPRINT data set which more fully explain the results of program execution. The return codes are as follows:

Code	Meaning
0	All requested operations have successfully completed.
4	One or more operations have not successfully completed.
8	Severe errors have occurred causing job termination.
12	A combination of error codes 4 and 8 has occurred.

OUTPUT MESSAGES AND STATISTICS

The INDEX Reorganization Unload utility provides messages and statistics on data base content for each data set. In addition, it provides an audit trail capability.

EXAMPLE

A discussion of the use of this utility, together with an example, can be found under the topics "Reorganizing an INDEX Data Base" and "Initial Data Base Load Processing" later in this chapter.

INDEX REORGANIZATION RELOAD UTILITY (DFSURRL0)

A flow diagram of the INDEX Reorganization Reload utility is shown in Figure 5-2.

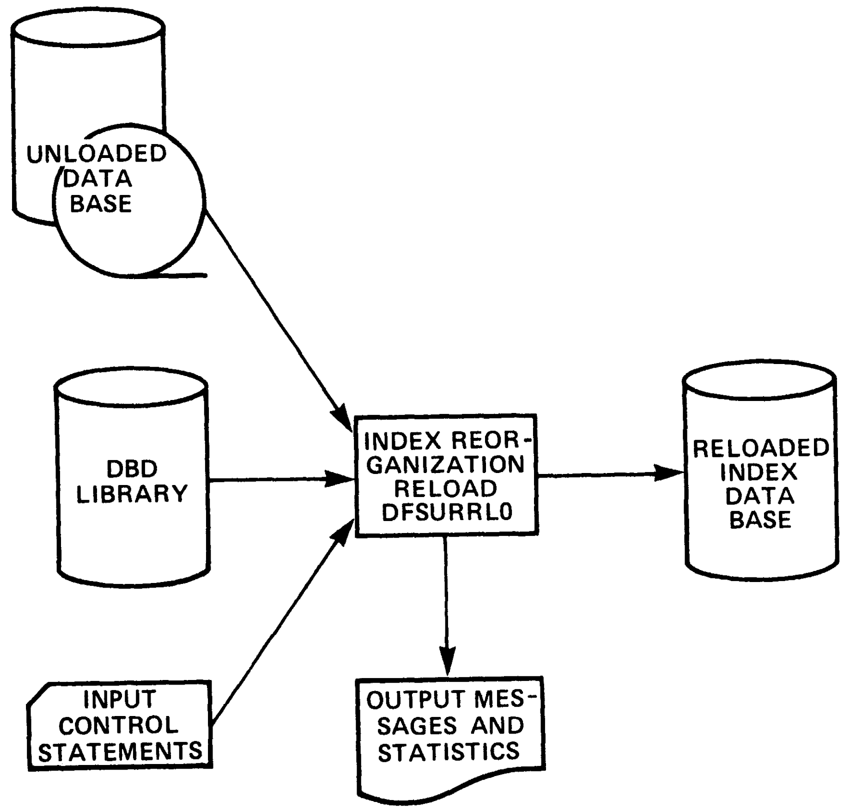


Figure 5-2. INDEX Reorganization Reload Utility

JCL STATEMENTS

The INDEX Reogranization Reload utility is executed as a standard OS/VS job. A JOB statement (defined by the using installation), an EXEC statement, and DD statements that define inputs and outputs are required.

EXEC — This statement must be in the form:

PGM=DFSRRC00,PARM='ULU,DFSURRL0'

IMS DD — Defines the library containing the DBD which describes The data base being reorganized. This data set must reside on a direct access device.

SYSPRINT DD — Defines the output message and statistics data set.

DFSUIN01 DD — Defines the unloaded input data set. This data set must be created by DFSURUL0.

vsamout DD — Defines the KSDS output data set to be reloaded. The ddname must be the same as the name in the DBD that was referenced when this data set was unloaded.

DFSVSAMP DD — Describes the data set that contains the buffer information required by the DL/I Buffer Handler. This DD statement is required. (For additional information, see "Defining the IMS/VS Data Base Buffer Subpools" in Chapter 7, "Installing IMS/VS.")

Note: No SYSIN DD statement or utility control statements are required for this utility.

RETURN CODES

The following return codes are provided at program termination:

Code	Meaning
0	All operations have successfully completed.
4	One or more warning messages issued.
8	One or more operations have not completed successfully.
16	Severe errors have occurred causing program termination.

OUTPUT MESSAGES AND STATISTICS

The INDEX Reorganization Reload utility provides messages, statistics and an audit trail for the data set being reloaded.

EXAMPLE

A discussion of the use of this utility, together with an example can be found under the topics "Reorganizing an INDEX Data Base" and "Initial Data Base Load Processing" later in this chapter.

HD REORGANIZATION UNLOAD UTILITY (DFSURGU0)

The HD Reorganization Unload utility can be used to unload an HDAM, HIDAM, or SHISAM data base to a QSAM formatted data set. There are no utility control statements for this utility.

A flow diagram of the HD Reorganization Unload utility is shown in Figure 5-3.

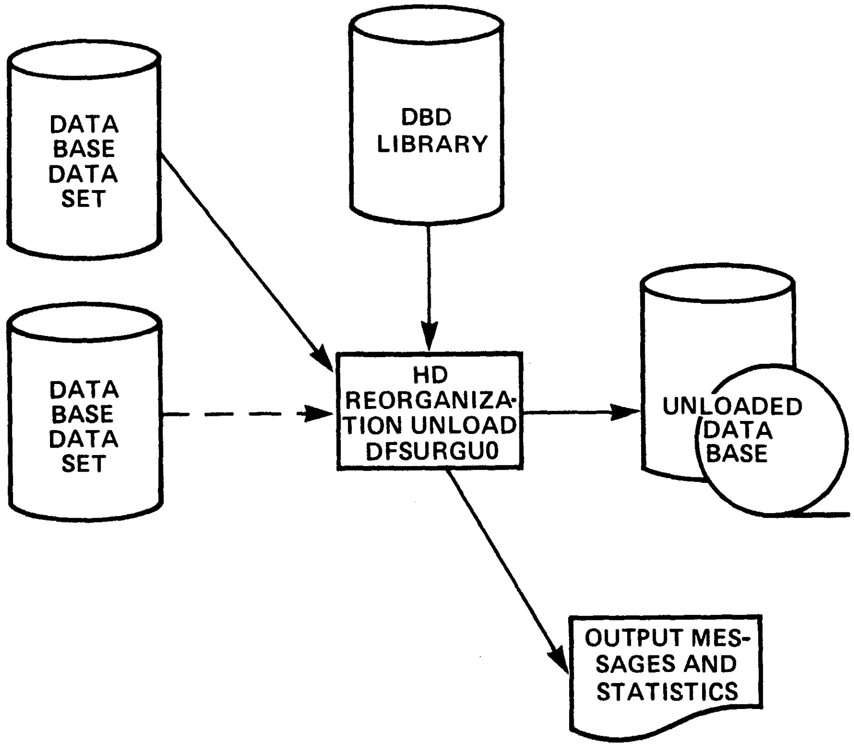


Figure 5-3. HD Reorganization Unload Utility

JCL STATEMENTS

The HD Reorganization Unload utility is executed as a standard OS/VS job. The following JCL statements are required.

EXEC	This statement must be in the following form: PGM=DFSRRC00,PARM='ULU,DFSURGU0,dbdname' where the parameters ULU and DFSURGU0 describe the utility region, and dbdname is the name of the DBD which describes the data base to be reorganized.
IMS DD	Defines the library (IMSVS.DBDLIB) containing the DBD which describes the data base to be reorganized. This data set must reside on a direct access device.
SYSPRINT DD	Defines the message and statistics output data set.
DFSURGU1 DD	Defines the sequential, variable blocked output data set. This DD statement must be supplied. The data set can reside on either tape or a direct access device. Since output is blocked to the maximum size the output device can handle, up to 8K, standard labels must be used on output volumes. Standard labels must be used on output volumes.
data base DD	Defines the data base data set to be reorganized. The ddname must match the ddname in the DBD.

If this is a HIDAM data base, you must also include a DD statement for the primary index. That DD name must be the same as specified in the primary INDEX DBD. No DD statements are required for any secondary indexes associated with this data base. JCL must be included for all logically related data bases, even though these data bases are not unloaded.

This data set must reside on a direct access device.

DFSVSAMP DD	Describes the data set that contains the buffer pool information required by the DL/I Buffer Handler. This DD statement is required. (For additional information, see "Defining the IMS/VS Data Base Buffer Subpools" in Chapter 7, "Installing IMS/VS.")

RETURN CODES

The following return codes are provided at program termination:

Code	Meaning
0	Data base unload successful.
4	One or more warning messages issued.
8	Severe error has occurred.
12	Multiple warning and/or error messages issued.
16	Data base unload not successful.

OUTPUT MESSAGES AND STATISTICS

The HD Reorganization Unload utility provides output messages and statistics. An example of the messages and statistics obtained from this utility, accompanied by explanatory information, is provided in Chapter 3 of the IMS/VS Primer Sample Listings manual.

EXAMPLE

A discussion of the use of this utility, together with an example, can be found under the topic "Reorganizing a HDAM or HIDAM Data Base" later in this chapter.

HD REORGANIZATION RELOAD UTILITY (DFSURGL0)

The HD Reorganization Reload utility can be used to: (a) reload an HDAM or HIDAM data base from a data set created by the HD Unload utility, and (b) create work data sets (if the data base to be reloaded includes logical relationships or secondary indexes) which are to be used as input to the logical relationship resolution utilities.

A flow diagram of the HD Reorganization Reload utility is shown in Figure 5-4.

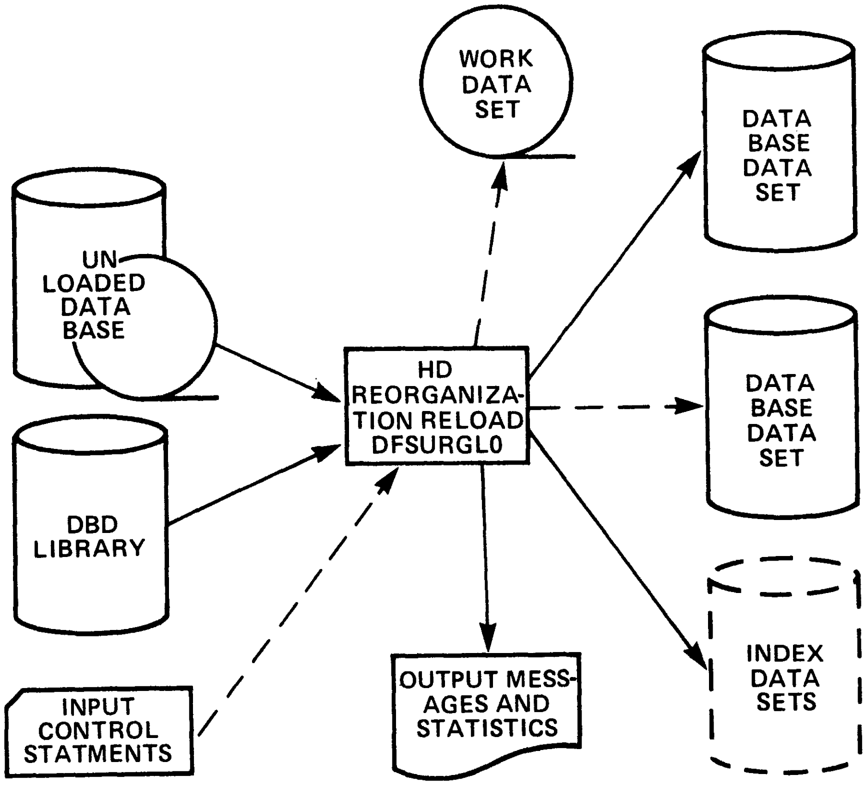


Figure 5-4. HD Reorganization Reload Utility

JCL STATEMENTS

The HD Reorganization Reload utility is executed as a standard OS/VS job. The following JCL statements are required.

EXEC — This statement must be in the form:

PGM=DFSRRC00,PARM='ULU,DFSURGL0,dbdname'

where dbdname is the name of the DBD which includes the data base to be reloaded.

IMS DD — Describes the library containing the DBD referenced in the EXEC statement PARM field (normally this is IMSVS.DBDLIB). This data set must reside on a direct access device.

SYSPRINT DD — Defines the message output data set.

DFSUINPT DD — Describes the input data set containing the data to be reloaded. This is the data set created by the HD Reorganization Unload utility. The data set must reside either on tape or a direct access device.

DFSURWF1 DD — Describes the data set to be created during reload that will be used as input by the Prefix Resolution utility (DFSURG10) to resolve logical or secondary index relationships. This DD statement must always be present. It can specify DD DUMMY if the data base is not involved in logical relationships or secondary indexes.

The DCB parameters for the DD statement must include LRECL=300, RECFM=VB, and BLKSIZE specified to be the same as that specified for the work data set of the user's initial load program. A full track blocksize or 8-16K for tape is recommended.

The data set must reside either on tape or a direct access device.

database DD	Defines the data base data set to be reorganized. One statement of this type must be present for each data set that appears in the DBD which describes this data base. The ddname must match the ddname in the DBD. If this is a HIDAM data base, a DD statement must also be included for the KSDS of the primary index. This ddname is specified in the DBD for the index data base. No DD statements are required for any secondary indexes associated with this data base. This data set must reside on a direct access device.
DFSURCDS DD	Defines the control data set for this program. This data set must be created by the Prereorganization utility (DFSURPRO). This DD statement must be included if logical relationships exist. This data set must reside on either tape or a direct access device.
DFSVSAMP DD	Describes the data set that contains the buffer pool information required by the DL/I Buffer Handler. This DD statement is required. (For additional information, see "Defining the IMS/VS Data Base Buffer Subpools" in Chapter 7, "Installing IMS/VS.")

RETURN CODES

The following return codes are provided at program termination:

Code	Meaning
0	Data base reload successful.
16	Data base reload unsuccessful.

OUTPUT MESSAGES AND STATISTICS

The HD Reorganization Reload utility provides output messages and statistics. An example of the messages and statistics obtained from this utility, is provided in Chapter 3 of the IMS/VS Primer Sample Listings.

EXAMPLE

A discussion of the use of this utility, together with an example can be found under the topic "Reorganizing a HDAM or HIDAM Data Base," later in this chapter.

DATA BASE PREREORGANIZATION UTILITY (DFSURPRO)

The Data Base Prereorganization utility creates a control data set that is used by the other logical relationship resolution utilities. This utility must be executed before you initially load or reorganize any data base which contains logical relationships and/or secondary indexes.

The input to this utility is a data set which consists of the utility control statements that name the data base(s) being loaded and/or reorganized. The DBDs that are used for the data bases named on these statements must define each data base as it is to exist after the logical relationships and/or secondary indexes are established. These DBDs must not be modified until the Prefix Update utility has been successfully executed.

The output is a control data set that is used by the HD Reorganization Reload and by the Data Base Prefix Resolution utilities. It is also used during the initial load of data bases with logical relationships and/or secondary indexes.

A flow diagram of the Data Base Prereorganization utility is shown in Figure 5-5.

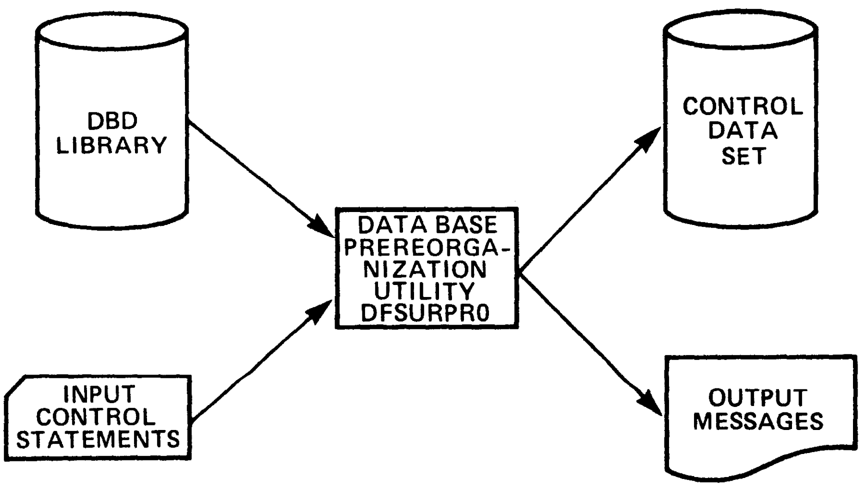


Figure 5-5. Data Base Prereorganization Utility

JCL STATEMENTS

The Data Base Prereorganization utility is executed as a standard OS/VS job. A JOB statement (defined by the using installation), an EXEC statement, and DD statements that define inputs and outputs are required.

EXEC — This statement must be in the form:

PGM=DFSRRC00,PARM='ULU,DFSURPRO'

IMS DD — Defines the library containing the DBDs which describe the data bases named on the input control statements. This DD statement must always be included. The data set must reside on a direct access device.

SYSIN DD — This data set will contain input control statements. DCB parameters specified within this program are RECFM=FB and LRECL=80. BLKSIZE must be provided on this DD statement. If BLKSIZE is not specified, there is no default and the results are unpredictable.

SYSPRINT DD — Define the message output data set. The DCB parameters specified within this program are RECFM=FB and LRECL=120. BLKSIZE must be provided on this DD statement. If BLKSIZE is not specified, there is no default and the results are unpredictable.

DFSURCDS DD — Defines the output data set for this program. This data set is the control data set used by subsequent utilities. This DD statement must always be included.

DCB parameters specified within this program are RECFM=FB and LRECL=1600. BLKSIZE must be provided on this DD statement.

UTILITY CONTROL STATEMENTS

```
  1                                                                    80
 ---------------------------------------------------------------------
|                                                                     |
| DBIL=database name1,database name2,...    [comments]                |
|                                                                     |
 ---------------------------------------------------------------------
```

This utility control statement names data bases to be <u>initially loaded</u>. One or more of these statements can be provided. Each DBD name must be left-justified to provide a total length of 8 characters. If the DBD name is less than 8 characters, sufficient trailing blank characters must be provided to make a total of 8 characters. A blank must follow the last entry on each statement. If a HIDAM data base is to be initially loaded, only its DBD name should be listed on a DBIL control statement. Neither the HIDAM primary index nor any secondary index DBD names should be listed.

```
  1                                                                    80
 ---------------------------------------------------------------------
|                                                                     |
| DBR=database name1,database name2,...     [comments]                |
|                                                                     |
 ---------------------------------------------------------------------
```

This utility control statement names data bases to be <u>reorganized</u>. One or more of these statements can be provided. Each DBD name must be left-justified to provide a total length of 8 characters. If the DBD name is less than 8 characters, sufficient trailing blank characters must be provided to make a total of 8 characters. A blank must follow the last entry on each statement.

If a HIDAM data base is to be reorganized, only its DBD name should be listed on a DBR control statement. (Neither the HIDAM primary index nor secondary index DBD names should be listed.)

```
  1                                                                          80
┌------------------------------------------------------------------------------┐
|                                                                              |
| OPTIONS=(NOPUNCH,STAT,SUMM)                                                  |
|                                                                              |
└------------------------------------------------------------------------------┘
```

This utility control statement must be coded as shown above in our subset. It directs the prefix resolution utility to provide statistics on the number of segments being updated and the number of logical parents without logical children.

RETURN CODES

The following return codes are provided at program termination:

Code	Meaning
0	No errors detected.
8	One or more error messages have been issued.

OUTPUT MESSAGES

The output messages issued by this utility indicate the data base operations that must be performed prior to execution of the Prefix Resolution and the Prefix Update utilities. For instance:

- Data bases listed after the characters DBIL= in message DFS861I must be initially loaded.
- Data bases listed after the characters DBR= in message DFS861I must be reorganized using the HD Reorganization Unload/Reload utilities.
- Data bases listed after the characters DBS= in message DFS862I must be specified on a DBR= control card, and the utility must be re-executed. If this occurs, you may have omitted a data base to be reorganized.

DATA BASE PREFIX RESOLUTION UTILITY (DFSURG10)

The Data Base Prefix Resolution utility accumulates the information generated on work data sets during the load and/or reorganization of one or more data bases. It produces an output data set that contains the prefix information needed to complete the logical relationships defined for the data base(s) and, optionally, an output data set containing information needed to (re)create secondary index data bases. There are no utility control statements for this utility.

RESTRICTIONS

The Data Base Prefix Resolution utility uses the OS/VS Sort/Merge programs. Since the maximum sort field permitted by Sort/Merge is 256 characters, certain limits must be observed. The following restrictions apply in our subset:

1. For any given logical parent/logical child pair, the sum of items a and b below must not exceed 200 characters (the balance of 56 characters is used by IMS/VS for control purposes):

   a. The length of the logical parent's concatenated key.

b. The length of the sequence field of the logical child as seen by its logical parent.

2. The sum must be computed once for the logical parent and once for the logical child. These summations are treated separately.

The Data Base Prereorganization utility performs the above limit check for logical parent/logical child combinations affected by an intended data base initial load or reload. It should be noted that the limit check is a worst-case check. If the limit check fails for a logical parent/logical child combination, message DFS885 will be issued. Refer to the _IMS/VS Messages and Codes Reference Manual_ for an explanation of the message.

A flow diagram of the Data Base Prefix Resolution utility is shown in Figure 5-6.

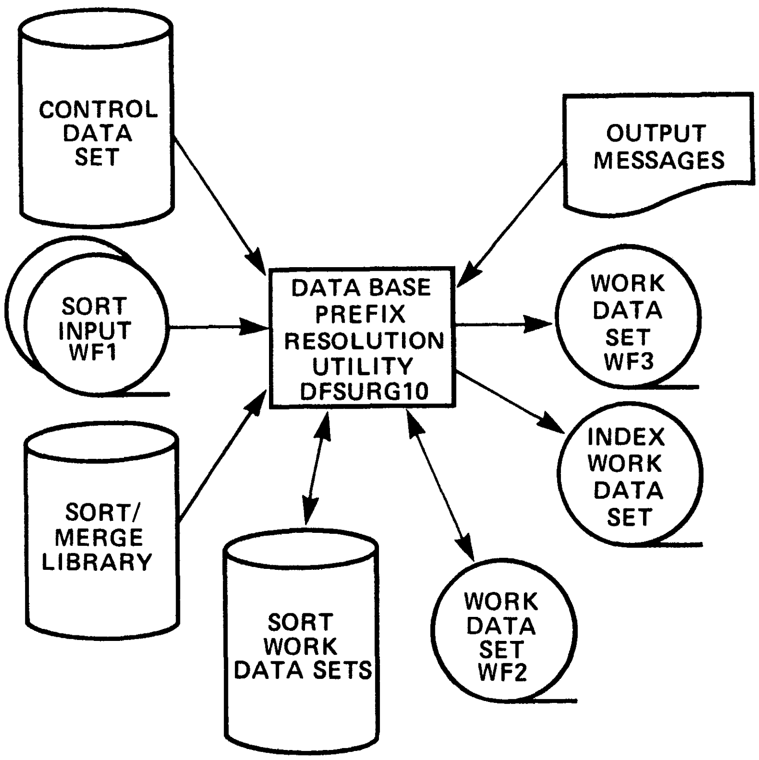


Figure 5-6. Data Base Prefix Resolution Utility

JCL STATEMENTS

The Data Base Prefix Resolution utility is executed as a standard OS/VS job. A JOB statement (defined by the using installation), an EXEC statement, and DD statements that define inputs and outputs are required.

EXEC This statement must be of the form:

PGM=DFSURG10,PARM='options'

Since this program invokes the operating system Sort, program efficiency can be improved by increasing the partition/region size.

The PARM field can be used to specify options for the SORT/MERGE program. A recommended option is

PARM='SIZE=En'

n is the estimated number of records to be sorted. Specification of this parameter improves significantly the SORT/MERGE performance. Guidelines for calculating the number of SORT/MERGE input records are provided under the topic "Work Data Set Allocation" later in this chapter.

SYSPRINT DD — Defines the message output data set for this program.

DCB parameters specified within this program are RECFM=FB and LRECL=120. BLKSIZE must be provided on the SYSPRINT DD statement and must be a multiple of 120.

SYSOUT DD — Defines the message output data set for Sort/Merge.

SORTLIB DD — Defines a data set containing load modules for the operating system Sort/Merge program. This DD statement must always be included.

SORTWKnn DD — Defines intermediate storage data sets for the operating system Sort/Merge program. Refer to the appropriate operating system Sort/Merge manual regarding specification of number and size of intermediate storage data sets. These DD statement(s) must be included.

SORTIN DD — Defines the input data set for this program. This DD statement must always be included. It is referenced by the operating system Sort/Merge program and must conform to its JCL requirements. The data set(s) referenced by this DD statement must be the output work data set(s) produced during a data base initial load or reload operation; those work data sets must be concatenated to form the SORTIN data set.

DCB parameters specified within this program are RECFM=VB and LRECL=300. The BLKSIZE must be the same as that specified for the work data sets (WF1s) written during initial data base load, or data base reload.

DFSURWF2 DD — Defines an intermediate sort work data set. This DD statement must always be included. The data set can reside on a tape or direct access device. The size of the data set will be approximately the same as that of the input data set defined by the SORTIN DD statement.

DCB parameters specified within this program are RECFM=VB and LRECL=300. BLKSIZE must be provided on this DD statement. If BLKSIZE is not specified, there is no default and the results are unpredictable.

DFSURWF3 DD — The output data set defined by this statement will be supplied as input to the Prefix Update utility. This statement must always be included. The data set can reside on tape or direct access device. Its size will be approximately the same as that of the input data set defined by the SORTIN DD statement.

DCB parameters specified within this program are RECFM=VB and LRECL=300. BLKSIZE must be provided on the DFSURWF3 DD statement. If BLKSIZE is not specified, there is no default and the results are unpredictable.

DFSURCDS DD — Defines the control data set for this program. It must be the output control data sets generated by the Prereorganization utility. This DD statement must always be included.

DFSURIDX DD — Defines an output work data set which will be used if secondary indexes are present in the DBDs being reorganized/loaded. All notes on DFSURWF3, above, apply to this data set also. This data set must be used as input to the INDEX Unload program (DFSURUL0) for (re)creating secondary indexes. This DD statement is required only if secondary indexes are present.

RETURN CODES

The following return codes are provided at program termination:

Code	Meaning
0	No errors detected.
4	Returned when either one or both of the following messages have been issued during program execution: DFS878, DFS885
8	Returned when one or more of the following messages has been issued during program execution: DFS852, DFS855, DFS857, DFS876, DFS877, DFS879, DSF880, DFS881 or if no data is written to the WF3 data set.
12	Returned when either one or both of the messages listed under return code 4 <u>and</u> any one or more of the messages listed under return code 8 have been issued.
16	Returned by OS/VS Sort/Merge program. This return code takes precedence over the above return codes.

<u>Note</u>: For return codes larger than 16, the same meaning stated above for return code 16 applies.

If either an 8, 12, or 16 return code is returned by the Prefix Resolution utility (DFSURG10), the Prefix Update utility (DFSURGP0) should not be executed since the input work data set required by DFSURGP0 will not have been generated by DFSURG10. The errors indicated by the diagnostic messages should be corrected, and the data base operations should be redone before the Prefix Resolution utility is again attempted. Generally, execution is satisfactory if a return code of 4 is set. However the SYSPRINT list should be checked. Refer to the <u>IMS/VS Messages and Codes Reference Manual</u> for an explanation of the DFS878 and DFS885 cautionary messages.

OUTPUT MESSAGES AND STATISTICS

If no errors are detected by this program, statistics and a normal program termination message will be printed.

DATA BASE PREFIX UPDATE UTILITY (DFSURGP0)

The Data Base Prefix Update utility uses the output generated by the Prefix Resolution utility to update the prefix of each segment whose prefix information was affected by a data base load and/or reorganization.

The output of the Prefix Resolution utility consists of one or more update records to be applied to each segment that contains logical relationship prefix information. The update records have been sorted into data base and segment physical location order by the Prefix Resolution utility.

A flow diagram of the Data Base Prefix Update utility is shown in Figure 5-7.

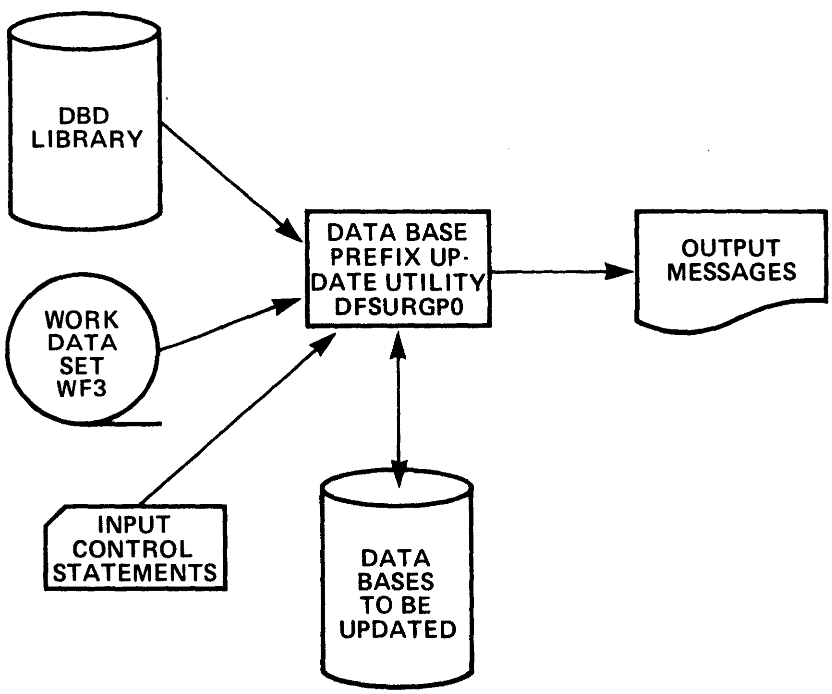


Figure 5-7. Data Base Prefix Update Utility

JCL STATEMENTS

The Data Base Prefix Update utility is executed as a standard OS/VS job. A JOB statement, an EXEC statement, and DD statements that define inputs and outputs are required.

EXEC This statement must be in the form:

PGM=DFSRRC00,PARM='ULU,DFSURGP0'

IMS DD	Defines the library containing the DBDs which describe the data base(s) that were loaded and/or reorganized. This DD statement must always be included. The data set must reside on a direct access device.
SYSPRINT DD	Defines the output message data set.
	DCB parameters supplied by the program are RECFM=FB and LRECL=120. BLKSIZE must be specified on this DD statement and must be a multiple of 120. If BLKSIZE is not specified, there is no default and the results are unpredictable.
DFSURWF3 DD	Defines the input work data set for this program. It must be the output data set generated by the Prefix Resolution utility. The data set can reside on a tape or direct access device. This DD statement must always be included.
database DD	References the data base(s) that were initially loaded and/or reorganized. One DD statement must be present for each data set of a data base that has logical relationships. The ddname must match the DDNAME indicated in the DBD. If an HIDAM data base is operated upon with this utility, a DD statement must also be supplied for the KSDS of its primary index data base.
	This data set must reside on a direct access device.
DFSVSAMP DD	Describes the data set that contains the buffer information required by the DL/I Buffer Handler. This DD statement is required. (For additional information, see "Defining the IMS/VS Data Base Buffer Subpools" in Chapter 7, "Installing IMS/VS.")

RETURN CODES

The following return codes are provided at program termination.

Code	Meaning
0	No errors detected.
8	One or more error messages issued. The messages contain details of the error(s) and are printed as part of the system output.

OUTPUT MESSAGES

If no errors are detected by this program, the only output message issued will be a normal program termination message that indicates the number of records processed.

PHYSICAL REORGANIZATION

REORGANIZING AN INDEX DATA BASE

The steps to be taken for the physical reorganization of a primary or a secondary index data base are the same.

Step 1: Unload the Data Base

Job //SAMP287 in IMSVS.PRIMEJOB shows an example of how to do this. This job will unload the primary index of the sample CUSTOMER ORDER data base.

Step 2: Change Physical Parameters

Using access method services, the KSDS cluster must be deleted and redefined. Only the following physical attributes can be changed before the reload:

- The amount of DASD space: via access method services DEFINE.
- The CI size: via the SIZE parameter in the DBD and access method services DEFINE.

Note: If the CI SIZE is changed, a DBDGEN of the altered data base must be executed here.

Step 3: Reload the Data Base

Job //SAMP288 in IMSVS.PRIMEJOB shows an example. This job will reload the primary index of the sample Customer Order data base. The job also includes the necessary access method services delete and define commands.

REORGANIZING A HDAM OR HIDAM DATA BASE

The 3 basic steps in reorganizing a HDAM or HIDAM data base are:

Step 1: Unload the Data Base

Job //SAMP185 in IMSVS.PRIMEJOB shows an example of how to do this. This job will unload the phase 1 Parts data base, BE1PARTS.

Step 2: Change Physical Parameters

The following physical parameters can be changed before the reload:

- The amount of DASD space: access method services or JCL.
- The CI size: SIZE parameter in DBD and access method services DEFINE.
- Size of the root addressable area and/or number of root anchor points (HDAM only).

Step 3: Reload the Data Base

Job //SAMP186 in IMSVS.PRIMEJOB shows the reload of the phase 1 Parts data base.

Note: In addition, several structural changes in the data base can be made. See "Applying Structural Changes" later in this chapter.

INDICATIONS THAT DATABASES MAY NEED REORGANIZATION

To determine the need for data base reorganization, certain indicators should be monitored. These indicators are different for OSAM data bases and for VSAM data bases.

In our subset, HIDAM, SHISAM and INDEX data bases will always be VSAM. HDAM data bases may be VSAM or OSAM. As GSAM data bases are never reorganized, we are not concerned about them here.

For each access method (VSAM, OSAM) there are two sources of information which can signal the need to reorganize:

- The DASD volume table of contents or VSAM catalog data, which does not relate to a specific execution of a job, and
- The buffer pool statistics that do relate to execution of a specific job.

OSAM Data Bases -- (HDAM only)

The VTOC of the DASD volume on which an OSAM data base resides may be retrieved by the OS/VS utility IEHLIST, with a control record as in the following example for the phase 1 PARTS data base:

```
LISTVTOC FORMAT,VOL=3330=IMSPRM,DSNAME=IMSPRIME.DB1PARTS.DBASE
```

A growth in the number of extents (the field identified by "NO EXT") may indicate that a reorganization is needed. Typically an OSAM data set has only one extent. The message about the number of empty cylinders and tracks in this dataset is not necessarily accurate for a data base, so it should be ignored.

The buffer pool statistics obtained by the use of sample routine DFSOAST and printed on DD statement //DOOASTA, also provides indicators if well monitored. Chapter 9, "Optimization," provides more guidelines for this.

VSAM Data Bases

Statistical data about VSAM clusters is maintained in the VSAM catalog and is available by running VSAM's Access Method Services with a control card such as the following for the phase 2 Customer Orders data base:

```
LISTCAT CLUSTER ALL ENTRIES (IMSPRIME.DB2PCUST.DBASE)
```

The major indicators can be found in the DATA portion of the cluster, under the STATISTICS heading. The RECORDS DELETED and INSERTED fields contain counts of this activity from the last creation (initial load or reorganization) of this data base. A large number, relative to data base size, in either or both fields may indicate a need for reorganization.

More important are the SPLITS counts for CA and CI. Control Area (CA) splits indicate that significant space is being claimed and that it is reorganization time. Control Interval (CI) splits indicate the same, but to a somewhat lesser extent. The NUMBER of EXTENTS should not grow. If they do, reorganize.

There is one other field, applicable to both the DATA and INDEX portions of a cluster, that will vary with the number of EXTENTS. This field, called TOTAL BYTES IN DATASET, indicates the number of free bytes left in the DATA or INDEX portion. As this freespace approaches zero, you are approaching the requirement for a new extent and you should consider reorganizing.

The buffer pool for VSAM is organized into subpools, with one subpool for each control interval size. In our examples, we have used 1024 byte and 2048 byte CI sizes, thus we have two subpools. These were defined in each job by the //DFSVSAMP DD statement.

These statistics are obtained during the execution of a job by calling the sample routine DFSOAST, and are listed on the DOOASTA DD statement. Two sets of statistics will be listed in our sample -- one for each subpool. A detailed discussion of these statistics and their interpretation can be found in Chapter 9, "Optimization."

In the HIDAM organization, the primary INDEX data base can be reorganized separately from the main HIDAM data base. (See jobs //SAMP287 and //SAMP288 in IMSVS.PRIMEJOB.) Because this is normally a small data space, you can do this weekly or even daily.

INITIAL DATA BASE LOAD PROCESSING

The initial load of a physical data base which does not contain logical relationships or secondary indexes is discussed in Chapter 4 under the topic "Loading a Data Base." None of the utilities of this chapter is required to load a single physical data base, which does not contain any logical relationships or secondary indexes.

LOADING DATA BASES WITH LOGICAL RELATIONSHIPS

Whenever you are loading one or more data bases which contain logical relationships, you must use the logical relationship resolution utilities. This is necessary because, when loading a logical child segment, the logical parent segment may not have been loaded, and vice versa. So DL/I cannot make the pointer connection at that time. Therefore, when loading a logical child or logical parent, DL/I will (automatically) write a control record to a workfile (DFSURWF1). This workfile is later sorted and used in a prefix update utility. Exactly which control records need to be generated is established beforehand by the prereorganization utility. Figure 5-8 gives an overview of this process.

Notes:

1. The job for loading the data base must contain DD statements with the ddnames of DFSURWF1 and DFSURCDS. The DFSURWF1 DD statement describes a data set which is automatically created by IMS as the result of the user's ISRT calls to DL/I at initial load. The DCB parameters for this statement must include LRECL=300, RECFM=VB, and BLKSIZE specified to be the same as that specified for any other WF1. A basic recommendation is full track blocksize or 8-16K for tape.

2. When loading 2 or more logically related data bases, the DFSURWF1 files must be concatenated. This concatenated data set (including all created WF1's) must be specifed to the Prefix Resolution utility as input.

3. The DFSURCDS DD statement must reference the control data set created by the prereorganization utility.

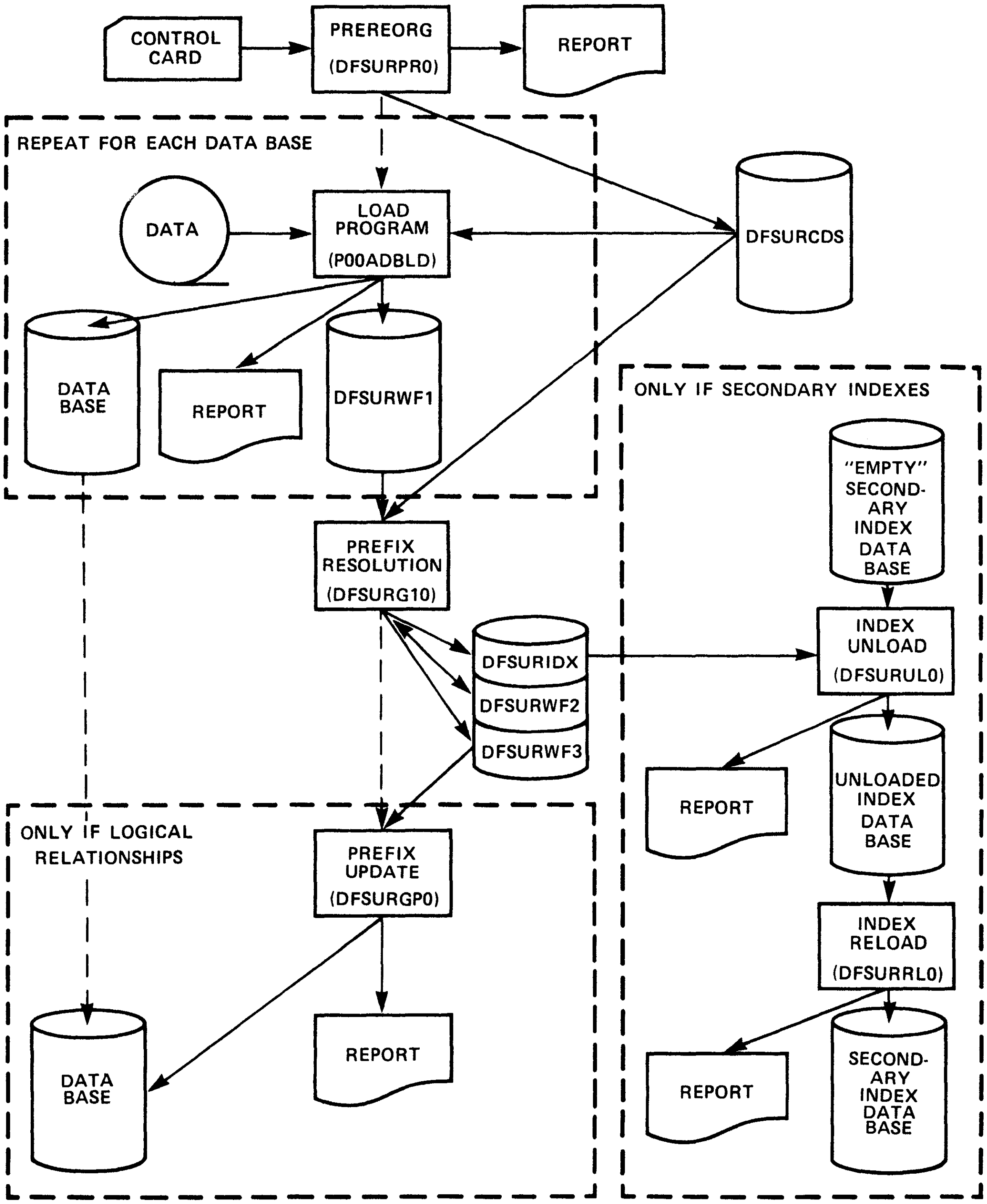


Figure 5-8. Initial Data Base Load with Logical Relationships and/or Secondary Indexes

Job //SAMP270 in IMSVS.PRIMEJCE can be used for loading the Level 2 Parts and Customer Orders data bases.

LOADING DATA BASES WITH SECONDARY INDEXES

When initially loading a data base with one or more secondary indexes, you must use the logical relationship resolution utilities. The basic process is the same as the one for loading data bases with logical relationships (see Figure 5-8).

However, some additions are required:

1. The prefix resolution utility (DFSURG10) creates an index workfile. Its ddname is DFSURIDX.

2. The above workfile must be used as input for the index unload utility, DFSURUL0.

   Note: The unload must be done from a newly allocated, empty KSDS.

3. The secondary index data base must be reloaded using the Index Reload utility.

Example

Job //SAMP37C in IMSVS.PRIMEJOB can be used for the initial load of the sample phase 3 data bases, including both logical relationships and secondary indexes.

WORK DATA SET ALLOCATION

The following guidelines should be observed for a good performance of the data base load process, especially for large data bases:

1. For the initial data base load job, the input file, the data base data set, and the workfile 1 should be on separate volumes.

2. For the prefix resolution:

   - Workfiles 1, 2, and 3 can be located at the same device since they don't interfere with each other. But they should be separated from the index workfile (DFSURIDX) if one exists.

   - The SORT/MERGE workfiles, SORTWKnn, should be kept on a separate device from workfiles 1, 2, and 3. The normal SORT/MERGE guidelines for the placement of SORTWKnn apply. Typically three SORT/MERGE workfiles on separate direct access devices give good performance.

3. For the prefix update execution, workfile 3 should be on a different volume from the data base data set(s).

4. The location of the control data set DFSURCDS is not important for performance; it is used only at the beginning of the utilities.

Size of Workfile 1

The records, and their size, which will be written to workfile 1 by DL/I during initial load or reload are:

Type 00. One record will be written for each logical parent occurrence. If the logical parent has multiple logical child segment types, the record is written once for each logical segment type.

The size will be 48 bytes + the length of the logical parent concatenated key if the data base is being initially loaded.

Type 10. One record will be written for each logical child occurrence. The size will be 43 bytes plus the length of the logical parent concatenated key (only initial load) plus the length of the virtual child sequence field if one is defined.

Type 20. One record will be written for each logical child occurrence which has a LTF pointer and no virtual child sequence field defined. The size is 43 bytes plus the length of the logical parent concatenated key if in initial load.

Type 30. One record will be written for each logical child occurrence which has a LTB pointer and no virtual child sequence field defined. The size is 43 bytes plus the length of the logical parent concatenated key if initial load.

Type 40. One record will be written for each index source segment occurrence. The size is 42 bytes, plus the length of the index search field(s), including the four bytes for the /SXname field if any. This is the only record which will be written to the index work file.

Note: The actual size of work file 1 can be found in the tape trailer label or the VTOC if it is on a direct access device.

REORGANIZING DATA BASES WITH LOGICAL RELATIONSHIPS/SECONDARY INDEXES

The following steps should be executed when reorganizing data bases with logical relationships and/or secondary indexes:

1. Start with the prereorganization utility, DFSURPR0. Specify all the related DBD names in a DBR control statement(s). However, no primary or secondary index DBD names should be specified.

2. Unload all related HDAM/HIDAM data base(s), using the HD Reorganization Unload utility, DFSURGU0. This should be done at the same time, that is, no data base processing between the unload and the prefix update of all connected data bases. The primary (HIDAM) and secondary index data bases need not and should not be unloaded separately.

3. Change any physical attributes as needed. Refer to the section "Reorganizing a HDAM or HIDAM Data Base" earlier in this chapter for the allowed changes.

4. Reload the HDAM/HIDAM data base(s), using the HD reorganization reload utility, DFSURGL0. Each reload of a data base will create a DFSURWF1 workfile.

5. The other steps are exactly the same as for the initial load process (see Figure 5-8), that is, prefix resolution, prefix update, index unload and index reload.

   Note: When unloading the existing secondary index data base, it must be a newly defined "empty" KSDS. So you should first delete the KSDS and redefine its space before the execution of the index unload utility (DFSURUL0).

APPLYING STRUCTURAL CHANGES

The HD reorganization utilities can be used to implement many different design changes to your data bases. The most common changes are discussed in the following sections.

CHANGING A PHYSICAL DBD

The following rules and restrictions should be observed when applying structural changes to a physical data base:

A. The HD unload utility must have been executed against the DBD describing the current structure, and no data base updates should have occurred since the unload.

B. An existing segment type can be deleted from the DBD provided all occurrences of this segment were deleted prior to execution of the HD unload utility.

C. New segment types can be added to the new DBD provided they do not change either the hierarchic relationship among existing segment types or the concatenated keys of logically related segments. (You cannot add a parent to existing segments.)

D. Names of existing segments must not be changed during reorganization, that is, between unload and reload. Segment names can be changed before or after the reorganization.

E. Any field statement except the sequence field (key field) can be changed, added, or deleted. However, DL/I will not change any segment data except as in (F) below.

F. Existing segment lengths can be altered. If the segment is made smaller, DL/I simply truncates the segment. If the segment is extended, it will be filled with whatever exists in main storage beyond the end of the segment. The user should replace this via an update program.

G. The access method may be changed. SHISAM/HIDAM may be changed to HDAM. HDAM can be changed to either indexed method only if the randomizing module maintains root key sequence.

ADDING LOGICAL RELATIONSHIPS/SECONDARY INDEXES

The following rules and guidelines should be observed when adding a logical relationship and/or a secondary index to an existing data base:

A. Before unloading the data base which contains the logical child, all the logical children must already exist in that data base. This segment, which at unload time is still a regular dependent segment, must start with the concatenated key of the "would be" logical parent. Remember, the HD reorganization utilities process only the segment prefix, never the segment data.

   If a logically related data base is to be added, its initial load process will generate a DFSURWF1 file. No additional unload/reload of that data base is required.

B. The HD unload utility must have been executed against the DBD describing the current structure, and no data base updates should have occurred since the unload.

C. *After* the HD unload, the DBD(s) are changed. And the prereorganization utility (DFSURPRO) must be run with the new DBD(s) before the reload/initial load. This could also be done initially if the new DBD(s) have different names. Notice, the HD unload utility does not need the control data set created by the prereorganization utility.

D. Prefix resolution (DFSURG10), prefix update (DFSURGP0), and index creation (optional: DFSURUI0/DFSURRL0) should then be performed as in Figure 5-8.

Examples

1. Job //SAMP289 in IMSVS.PRIMEJOB shows how to add a logical relationship to the Parts data base together with the initial load of the related Customer Orders data base.

2. Job //SAMP389 in IMSVS.PRIMEJOB shows how to convert the phase 2 Parts and Customer Orders data bases to their phase 3 versions by adding a secondary index to the Parts data base. Notice that no application program is required to add a secondary index.

REORGANIZING IN AN ONLINE ENVIRONMENT

In our subset, we will not consider the reorganization of a data base while it is allocated to the online IMS/VS control region. Therefore, the reorganization procedures in the IMS/VS-DB/DC environment are exactly the same as for the IMS/VS-DB only environment.

CHAPTER 6. DATA BASE RECOVERY

WHAT IS RECOVERY?

Data base recovery is, in its simplest form, the restoration of a data base after its (partial) destruction due to some failure. The preceding sentence defines the three basic elements in recovery:

- The data base
- The failure
- The restoration

Their relationship is: "The restoration eliminates the effect of the failure on the data base."

The basic principle of almost any data base recovery mechanism is maintaining duplicate data. Periodically, a copy of the data in the data base is saved. This copy is normally referred to as a back-up or image copy. These image copies normally reside on magnetic tapes. In addition to this, the changes made to the data in the data base are saved, at least until the next image copy. See Figure 6-1 for an overview.

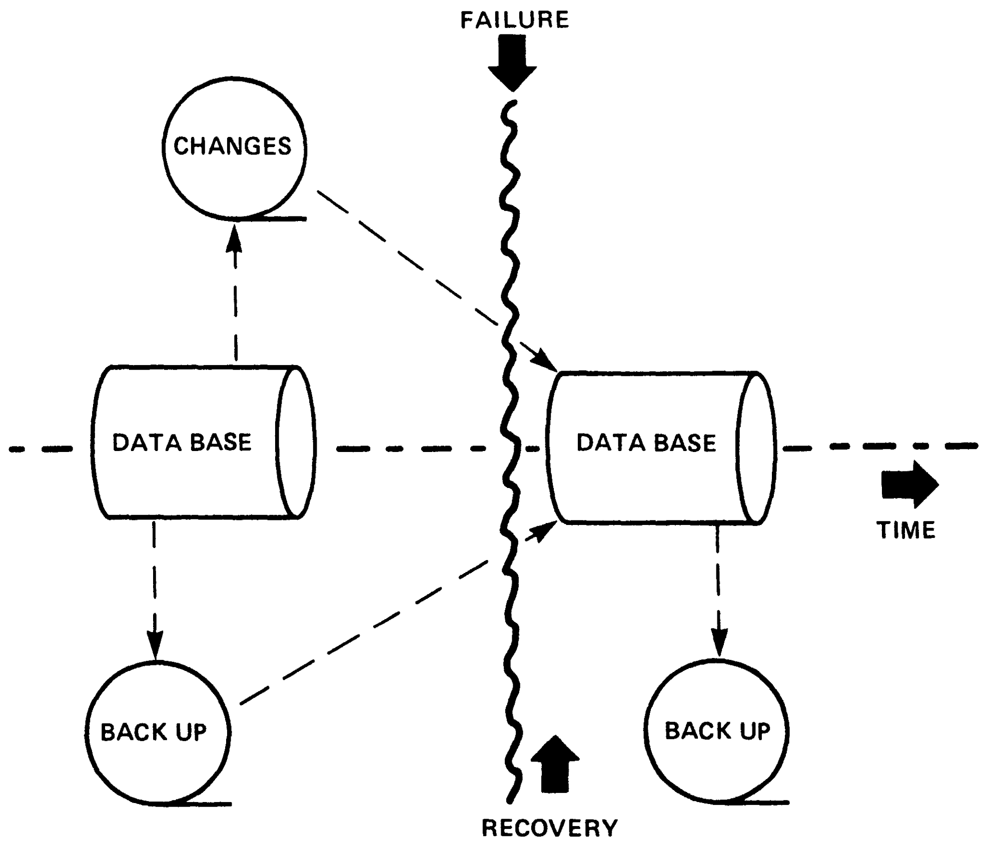


Figure 6-1. Concepts of Data Base Recovery

The recovery process then includes the following four steps:

1. Determine the cause of the failure.

2. Correct the cause of the failure.

3. Reconstruct the data base using the image copy and the saved changes.

4. Resume processing at the point of failure.

Each of the above steps can cause, in practice, a variety of activities. The intention of this chapter is to provide you with guidelines for, and examples of, procedures to handle these activities.

TWO APPROACHES

With DL/I, two approaches are, in general, available to protect the integrity of your data bases: basic recovery and DL/I recovery.

BASIC RECOVERY

After each back-up copy is made, all input data to the data base update programs are saved until the next back-up copy is made. In case of failure, the data base is restored, using the back-up copy. Next, all update programs executed during that period are re-executed, with exactly the same input data and in exactly the same sequence. The regenerated output replaces the original output. DL/I provides utilities to create the back-up copy and to restore the data base. This approach is referred to in the following discussion as basic recovery. See Figure 6-2.

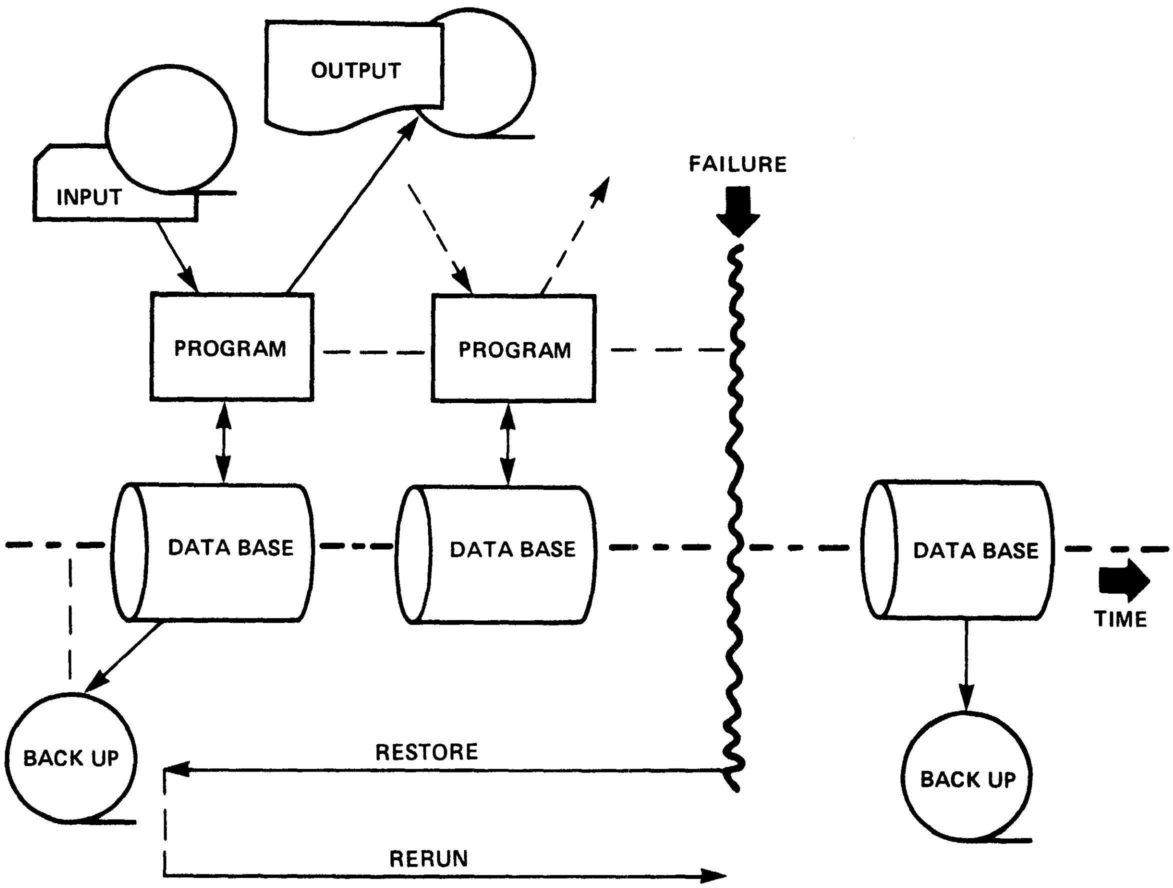


Figure 6-2. Basic Data Base Recovery

DL/I RECOVERY

The second approach uses the DL/I log facility. During processing of the data bases by application programs, all changes made to the data base are automatically logged on the DL/I log data set. A DL/I utility is provided which allows you to accumulate all changes made to the data base by all processing programs in a single change accumulation data set. Only the last copy of a data base change is kept in this data set, thus reducing the volume of tape required. When a failure occurs, you restore the latest back-up copy of the data base, using a DL/I utility, which at the same time merges the change accumulation data set into the restored data base. This brings the data base up to the point of the failure. In addition, a separate utility is available to back out (undo) the data base changes of a failing program. This approach is referred to in the following discussion as DL/I recovery. See Figure 6-3.

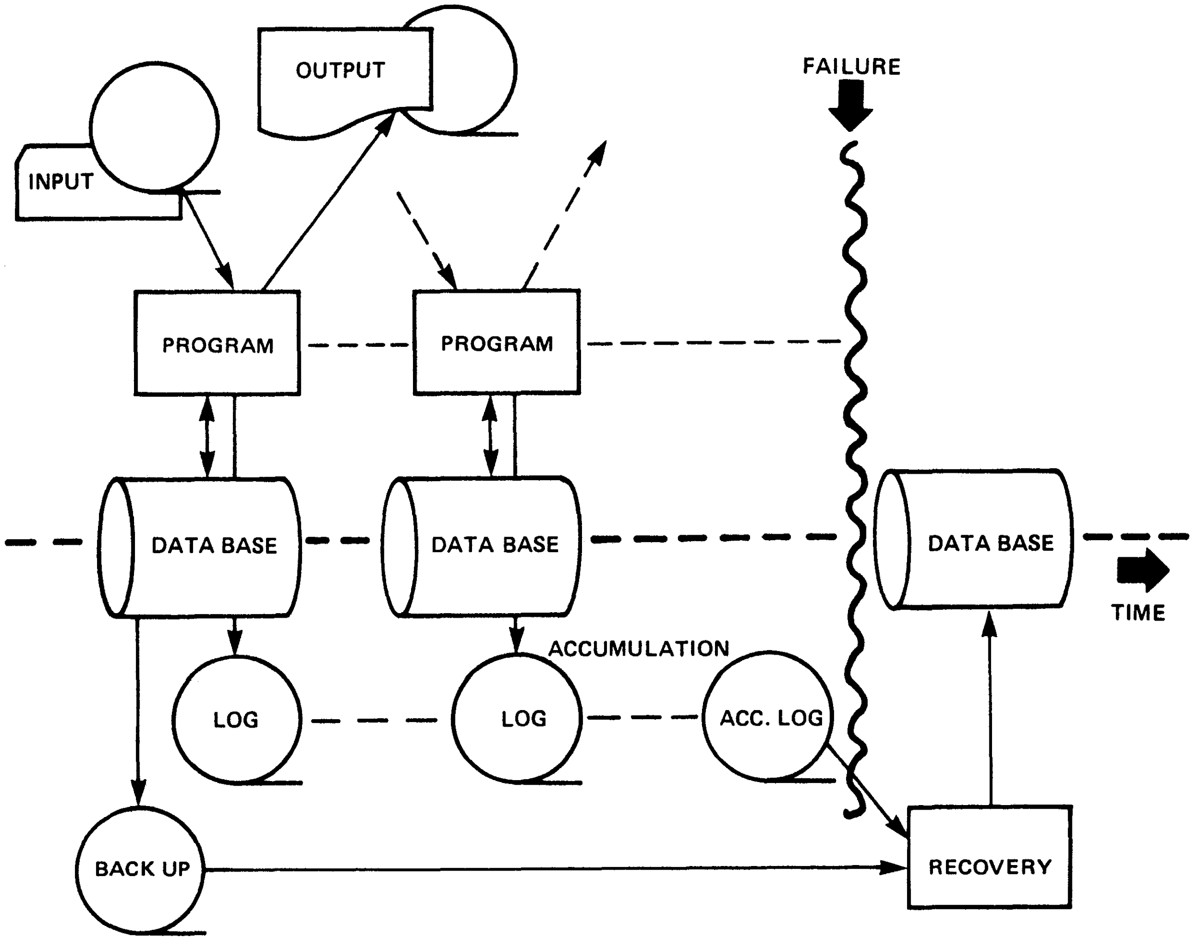


Figure 6-3. DL/I Recovery

WHICH ONE TO CHOOSE

DL/I recovery has several advantages over the basic recovery:

- No need to retain the input data sets
- No rerun of update processing programs
- Only the affected data sets need to be recovered
- No time synchronization problems if the programs are date dependent or have been modified in the interim
- Simpler administration: only the back-up copies and log data sets are required
- No duplicate output
- For the IMS/VS data communication facility, a log data set for the online data bases is mandatory. There is normally no retained input from terminal transactions. It is recommended that you establish DL/I recovery procedures before going online.

Based on the above advantages, the reccmmendation is to implement DL/I reccvery unless you have:

- Only one or two data bases, and,
- Low data base update rate, and,
- Very frequent (daily) back-ups, and,
- No plans for online processing.

Before describing the two recovery approaches, we will first discuss the DL/I logging facility and associated recovery utilities.

THE DL/I LOGGING FACILITY

The DL/I logging facility is the cornerstone of the recovery utilities of DL/I. This facility, operating as an integral part of DL/I, automatically writes all before and after images of updated data base segments to a central log data set. Each log data set, created with a DL/I batch program execution, is one sequential data set. It must reside on magnetic tape in our subset. You must specify DISP=(,KEEP,KEEP) or DISP=(,CATLG,CATLG) for the log data set, that is, DISP=MOD is not allowed.

In our subset, we will assume the use of the log tape write ahead function (specify: OPTIONS LTWA=YES in the DFSVSAMP control data set). For more details, see "Defining the IMS/VS Data Base Buffer Subpools" in Chapter 7, "Installing IMS/VS."

The log tape write ahead (LTWA) function will assure that no data base change will be written to physical data base storage before the corresponding log records are physically written to the log data set. With this provision and the log close function of the log recovery utility, there is no risk of lost data base changes, even in the case of an abrupt system breakdown.

A log data set is created by adding a //IEFRDER DD... statement to the JCL of the batch execution job.

Notes:

1. A log data set is not created when a data base is initially loaded (that is, when the processing option "L" or "LS" is selected in the PCB).

2. IMS/VS uses the OS/VS ESTAE facility to flush the log buffers and close the log data set in the event of an abnormal termination. In addition, for batch jobs only, the data base log buffers will be written to DASD.

THE DL/I RECOVERY UTILITIES

DL/I provides four utilities for recovering a data base. The diagram in Figure 6-4 illustrates the relationship between these utilities.

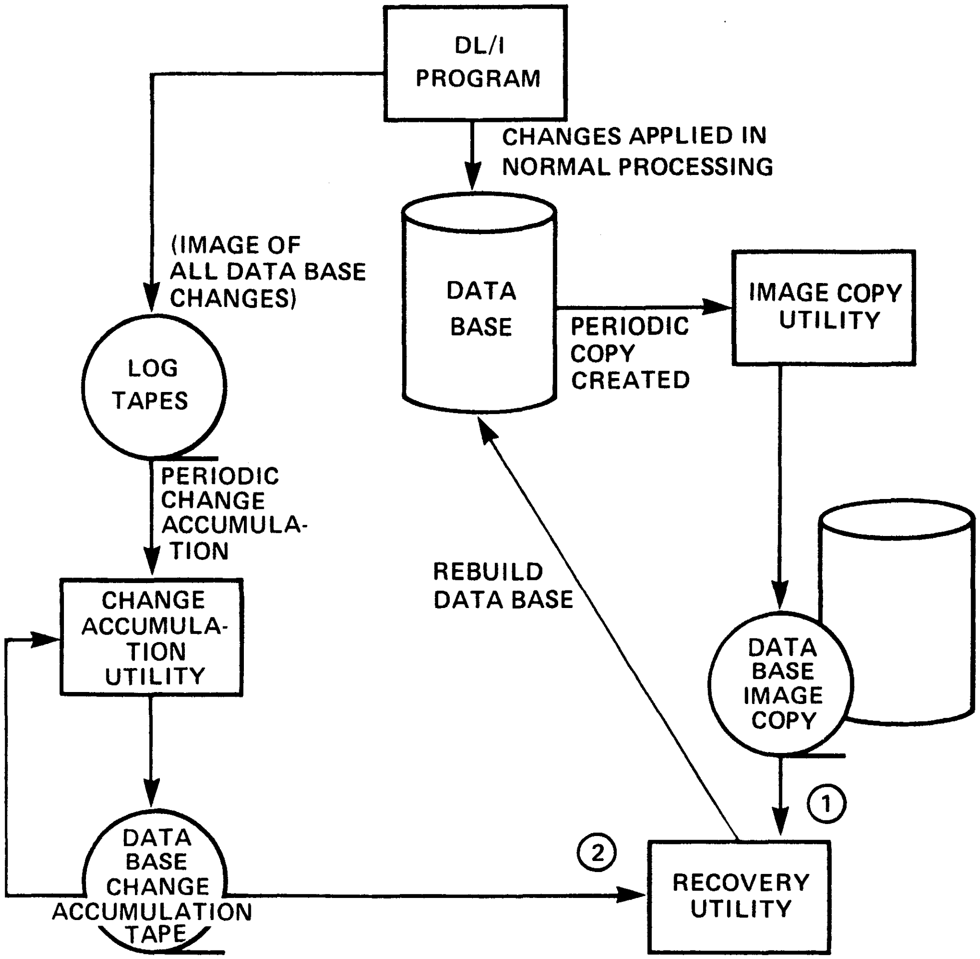


Figure 6-4. Data Base Recovery Utilities

A description of these utilities and their basic functions follows:

1. Data Base Image Copy utility for creation of image copies of data bases.

2. Data Base Change Accumulation utility for accumulation of data base changes from DL/I log tapes since the last complete image copy.

3. Data Base Recovery utility for restoration of the data base, using a prior data base image copy and the accumulated changes from DL/I log tapes.

4. Data Base Backout utility for removal of changes made to data bases by a specific application program.

A fifth utility program, the DL/I System Log Recovery Utility (DFSULTR0), is used to close the DL/I Log in the event of an operating system or hardware failure, thus enabling use of the log by the four principal programs of the recovery system.

For those data bases which consist of multiple data sets, recovery is done by individual data set. To recover a complete data base composed of multiple data sets, data base recovery must be performed for each of its component data sets.

DATA BASE IMAGE COPY UTILITY (DFSUDMP0)

The Data Base Image Copy utility creates a copy of the data sets within the data bases. As illustrated previously (see Figure 6-4), this output is used as input to the Data Base Recovery utility.

Multiple data sets can be copied with one execution of the Image Copy utility. All data sets of a data base should be copied at the same time. In our subset, we presume that all data base data sets are dumped at the same time, that is, no intervening data base processing.

A flow diagram of the Data Base Image Copy utility is shown in Figure 6-5.

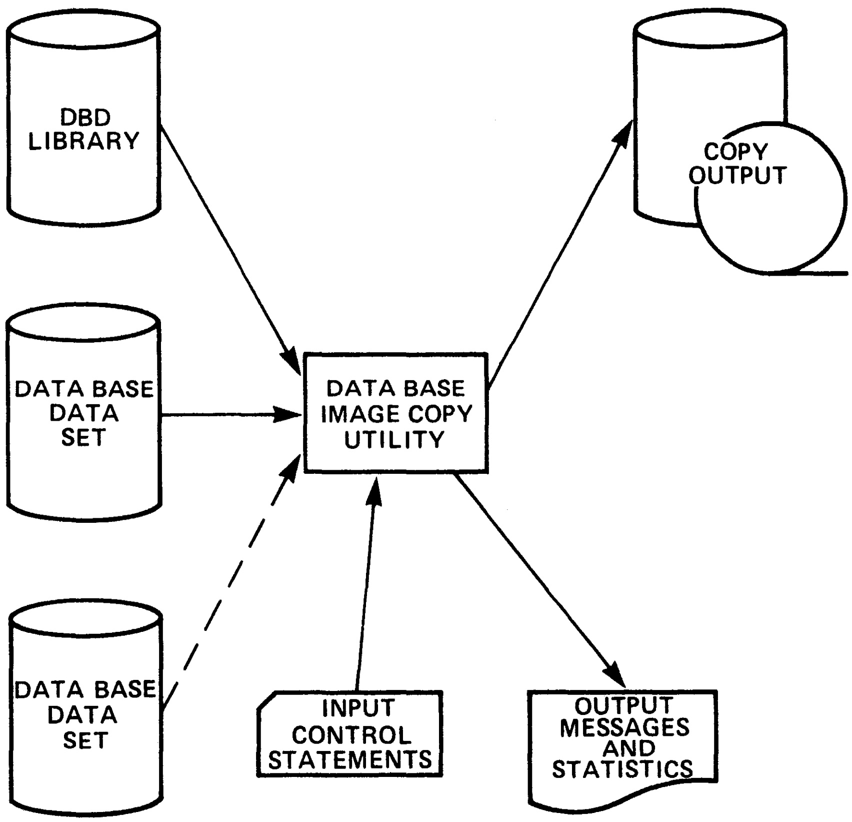


Figure 6-5. Data Base Image Copy Utility

JCL Statements

The Data Base Image Copy utility is executed as a standard OS/VS job. The following JCL statements are required:

EXEC	This statement must be in the following form: PGM=DFSRRC00,PARM='ULU,DFSUDMP0'
IMS DD	Defines the library containing the DBD that describes the data base to be dumped. This is usually DSNAME=IMSVS.DBDLIB.
SYSPRINT DD	Defines the output message data set.

datain DD — Defines the input data set to be dumped. The ddname on this statement must be the same as the name in the DBD that describes this data set; the ddname must also appear on the utility control statement. One DD statement of this type must be present for each data set to be dumped. The data set must reside on a direct access volume.

dataout DD — Defines the image copy output data set. One DD statement is required for each data set to be dumped. The ddname may be any 1- to 8-character string, but the ddname must appear in the associated utility control statement. The output device must be either direct access or tape. Standard labels must be used. LRECL and BLKSIZE are calculated by the utility and should not be provided in the JCL.

SYSIN DD — Defines the input control statement data set.

Utility Control Statement

One control statement is required for each data set to be dumped.

```
 1  4        13       22                                                80
+------------------------------------------------------------------------+
|                                                                        |
| D1 DE1PARTS DE1PARTS DFSUDUMP                                          |
|                                                                        |
+------------------------------------------------------------------------+
```

Position	Description
1	This must be the characters 'D1'.
4	This must be the name of the physical DBD that includes the ddname of the data set to be dumped.
13	This must be the ddname of the input data set to be dumped. It must appear in the referenced DBD, and a corresponding DD statement must have been provided.
22	This must be the ddname of the output data set. A corresponding DD statement must have been provided.

Note: All other fields must be blank in our subset.

Return Codes

The Data Base Image Copy utility provides the following return codes:

Code	Meaning
0	Successful completion of all operations.
4	Warning message issued.
8	One or more operations not successful.
16	Severe errors have caused the job to terminate without completing all operations.

These return codes can be tested by the COND= parameter on the EXEC statement of a subsequent job step.

Examples

Job //SAMP180 in IMSVS.PRIMEJCB shows the JCL for the image copy job of the phase 1 Parts data base. Job //SAMP380 shows the image copy of all our sample phase 3 data bases.

DATA BASE CHANGE ACCUMULATION UTILITY (DFSUCUM0)

The function of the data base change accumulation utility is to create a sequential data set that contains only that information from the log tapes which is necessary for recovery. This accumulation log data set is to be used by the data base recovery utility. This accumulation is done by sorting only the required log records (latest version) in physical record within data set sequence. This provides efficient data base recovery whenever needed. The number of log tapes will be significantly reduced. This utility invokes the Sort/Merge Program in your OS/VS installation.

The change accumulation utility can be run independently of DL/I application programs. The new output data set created by Change Accumulation is used by the data base recovery utility. Figure 6-6 depicts the sources of input to the data base change accumulation utility and the output created by this utility.

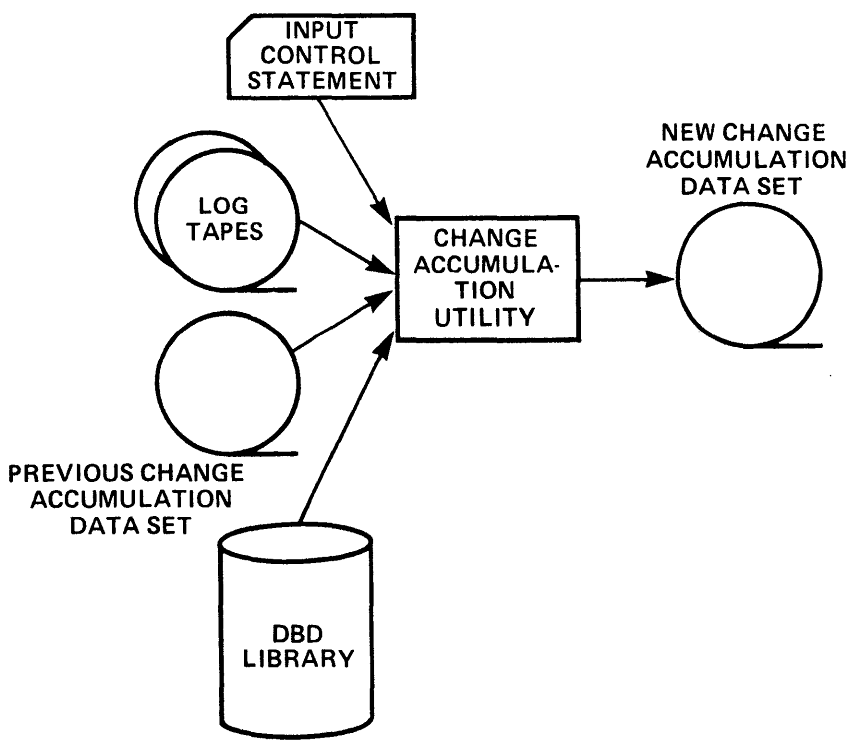


Figure 6-6. Data Base Change Accumulation Utility

The input to the data base change accumulation utility consists of:

1. All log tapes created since either the last image copy utility execution or the last execution of this utility.

2. The previous change accumulation data set. This would be the output from the last execution of this utility. The first change accumulation run after a new image dump _must_ _not_ include any old change accumulation data set, that is, those created during the previous period.

3. The DBD library which is normally called IMSVS.DBDLIB.

4. An optional control statement (ID).

Output from the data base change accumulation utility consists of a new change accumulation data set. This is a sequential data set containing the combined data base records for all data base data sets.

JCL Statements

The data base change accumulation utility is executed as a standard OS/VS job. The following JCL statements are required:

EXEC	This statement must be in the following form: PGM=DFSUCUMC
SYSPRINT DD	Defines the output message data set.
IMS DD	Defines the library containing the DBDs that describe all data bases to be accumulated. This is usually DSNAME=IMSVS.DBDLIB.
SYSOUT DD	Defines the output message data set for the Sort/Merge program. The Sort/Merge program specifies AP (all messages to printer).
SORTLIB DD	Defines a data set containing load modules for the execution of Sort/Merge. This is usually DSNAME=SYS1.SORTLIB.
SORTWKnn DD	These DD statements define the intermediate storage data sets for the Sort/Merge program. The data sets normally reside on a direct access volume; however, tape can be used. (For specification of number and size of intermediate storage data sets, refer to the applicable Sort/Merge manual.
DFSUCUMN DD	Defines the new accumulated change output data set. The data set can reside on a tape or a direct access volume. The output is blocked to maximum device capacity, up to a maximum of 8K. Standard labels must be used.
DFSUCUMO DD	Defines the old accumulated change input data set that is to be merged with the new accumulated change data set. If no old changes are to be merged, the following DD statement must be used: //DFSUCUMO DD DUMMY,DCB=BLKSIZE=100 This is required in the first change accumulation execution after each new image dump of the data bases. This data set can reside on tape or a direct access volume.

DFSULOG DD	Defines the log input data set containing the change records to be accumulated. This data set can reside on tape or a direct access volume.
SYSIN DD	Defines the control statement data set.

Utility Control Statement

An optional control statement can be used to describe additional table requirements for this change accumulation execution. If it is not included, default values are assigned as described below.

```
  1                        31-33                        80
+-------------------------------------------------------+
|                                                       |
| ID                       Max Seq Length               |
|                                                       |
+-------------------------------------------------------+
```

Position	Description
1	Positions 1 and 2 must contain the characters "ID".
31	Positions 31-33 contain the maximum length root sequence field contained within the log records to be processed. This value is used to pad the sequence field with binary zeros for sorting purposes. If there are no VSAM KSDS records to be processed, this value should be specified as 4, the length of the Relative Block Number field. This value must be in the range 1-236 and must be left-justified or supplied with leading zeros. The default value for this entry is 10. In our subset you must specify the maximum root sequence field of any HIDAM and/or SHISAM data base.

Note: All other columns must be blank in our subset.

Return Codes

The data base change accumulation utility provides the following return codes:

Code	Meaning
0	Successful completion.
16	Unsuccessful Completion.

These return codes can be tested by the COND= parameter on the EXEC statement of a subsequent job step.

Example

Jobs //SAMP181 and //SAMP381 in IMSVS.PRIMEJOB show the JCL to accumulate a log data set, with a previous accumulated log data set, into a new accumulated log data set.

Note: The change accumulation utility can accept multiple log data sets as input. These log data sets must be specified as concatenated data sets in the DFSULOG DD statement. The sequence of these data sets should be in the correct date and time sequence, that is, the oldest first.

DATA BASE RECOVERY UTILITY (DFSURDB0)

The data base recovery utility program will restore a physically damaged data set which is part of a DL/I data base. This utility does not provide a means of recovery from application logic errors; it is the user's responsibility to ensure the integrity of *the data* in the data base.

The data base recovery utility achieves a high rate of throughput by manipulating data sets individually in a physical sequential manner. The basic input consists of an image copy data set and, optionally, an accumulated change data set.

The data base recovery utility program is executed in a DL/I batch region. Its flow diagram is shown in Figure 6-7.

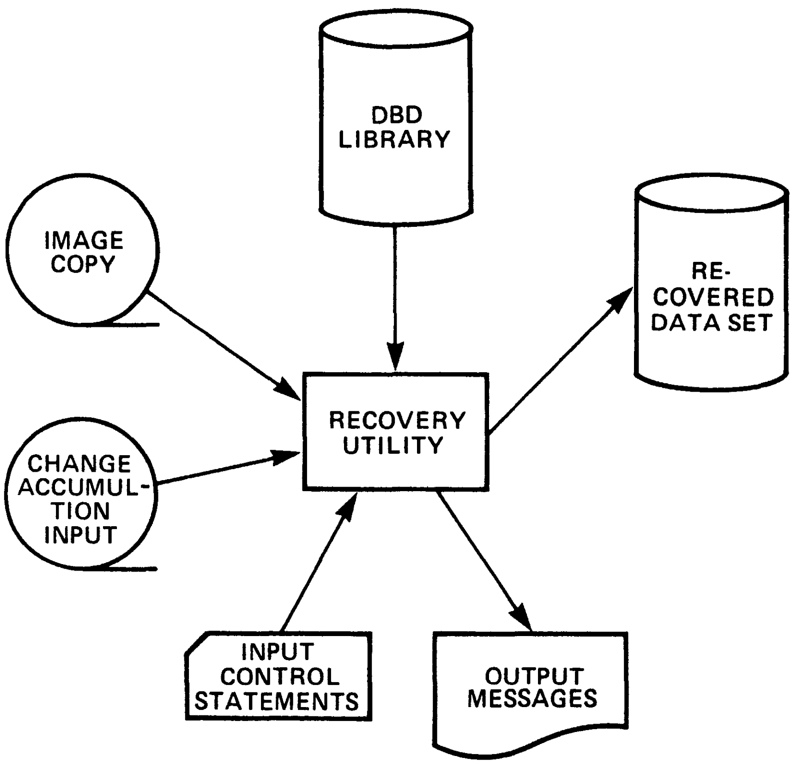


Figure 6-7. Data Base Recovery Utility

JCL Statements

The data base recovery utility is executed as a standard OS/VS job. Only one data set recovery is allowed for each execution. The following JCL statements are required: A JOB statement (defined by the using installation), an EXEC statement, and DD statements that define inputs and outputs are required.

EXEC	This statement must be in the following form: PGM=DFSRRC00,PARM='UDR,DFSURDB0,dbdname' where dbdname is the name of the DBD that includes the data set to be recovered.
IMS DD	Defines the library containing the DBD that describes the data base data set to be recovered. This is usually DSNAME=IMSVS.DBDLIB.
SYSPRINT DD	Defines the output message data set. If blocked it must be a multiple of 121.
DFSUDUMP DD	Defines the image copy input data set. It must be a data set created by the Data Base Image Copy utility. The data set can reside on tape or on a direct access volume. Standard labels must be used.
DFSUCUM DD	Defines the accumulated change input data set. If no accumulated change input is supplied, this statement must be coded DD DUMMY. This data set can reside on tape or a direct access volume. Standard labels must be used.
DFSULOG DD	This statement should be coded as DD DUMMY in our subset.
dataset1 DD	Defines the data set to be recovered. The ddname must be the same as the ddname in the DBD that describes this data set. It must also appear on the utility control statement. This data set must reside on a direct access volume.
SYSIN DD	Defines the input control data set.

__Utility Control Statement__

```
  1  4        13                                                  80
 ------------------------------------------------------------------
|                                                                  |
| S  BE1PARTS DE1PARTS                                             |
|                                                                  |
 ------------------------------------------------------------------
```

__Position__	__Description__
1	This must be the character 'S'. The 'S' defines this statement as a data base recovery utility control statement.
4	This must be the name of the DBD that describes the data base containing the data set to be recovered. This name must also appear in the PARM field of the EXEC statement.
13	This must be the ddname of the data set to be recovered. It must be the same as the ddname in the DBD and dataset1 DD statement.

__Note__: All other positions must be blank in our subset.

Return Codes

The data base recovery utility provides the following return codes:

Code	Meaning
0	Requested recovery successful.
4	Warning message issued
8	Serious error occurred, recovery terminated
16	Catastrophic error occurred; recovery unsuccessful

These return codes can be tested by the COND= parameter on the EXEC statement of a subsequent job step.

Examples

Job //SAMP182 in IMSVS.PRIMEJOB gives the JCL to recover the phase 1 Parts data base. Job //SAMP383 gives the JCL to recover all the phase 3 data bases using a change accumulation log data set.

DATA BASE BACKOUT UTILITY (DFSBBO00)

The data base backout utility removes changes in the data base which were made by a specific failing program. The following limitations apply:

- The log data set of the failing program must be on magnetic tape; the tape is read backwards.
- No other update programs should have been executed against the same data base(s) between the time of the failure and the backout.

The program operates as a normal DL/I batch job. It uses the PSB used by the program whose effects are to be backed out. All data bases updated by the program must be available to the backout utility.

A log tape is created during the backout process. This log tape, preceded by the log tape produced for the failing job, must be included in the next change accumulation run, as any other log tape. This tape must not be used as input to any subsequent backout attempt.

Note: For a multiple volume data set, the VOL parameter of the JCL statement specifies the volumes in ascending sequence.

Figure 6-8 presents a summary of conditions that terminate the processing of the data base backout utility.

Summary of Conditions Terminating the Processing of the Data Base Backout Utility	
CHKP-ID specified.	CHKP-ID not specified.
1. CHKP record requested.	1. First CHKP record encountered.*
2. The Accounting record for opening the log data set.	
*Note: The order of occurrence is referenced as from the end of the Log tape toward the start of the Log tape. (Read-backward processing.)	

Figure 6-8. Conditions That Terminate the Data Base Backout Utility

Notes:

- If checkpoint/restart is not used, then backout always backs out all the data base changes of the program.
- If checkpoint/restart is used (program uses XRST and CHKP calls), then backout will only do backout if the specified CHKP-ID is found on the log tape during read forward. If no CHKP-ID is specified then the last one on the log tape is used (the first one encountered during read backward).
- If, when using checkpoint/restart, you want to be able to completely back out a job (steps), you must issue a CHKP call immediately after the XRST call, that is, before any real data base activity. The CHKP-ID of this call can then be used for a full backout operation.

Figure 6-9 depicts the data set requirements for the data base backout utility.

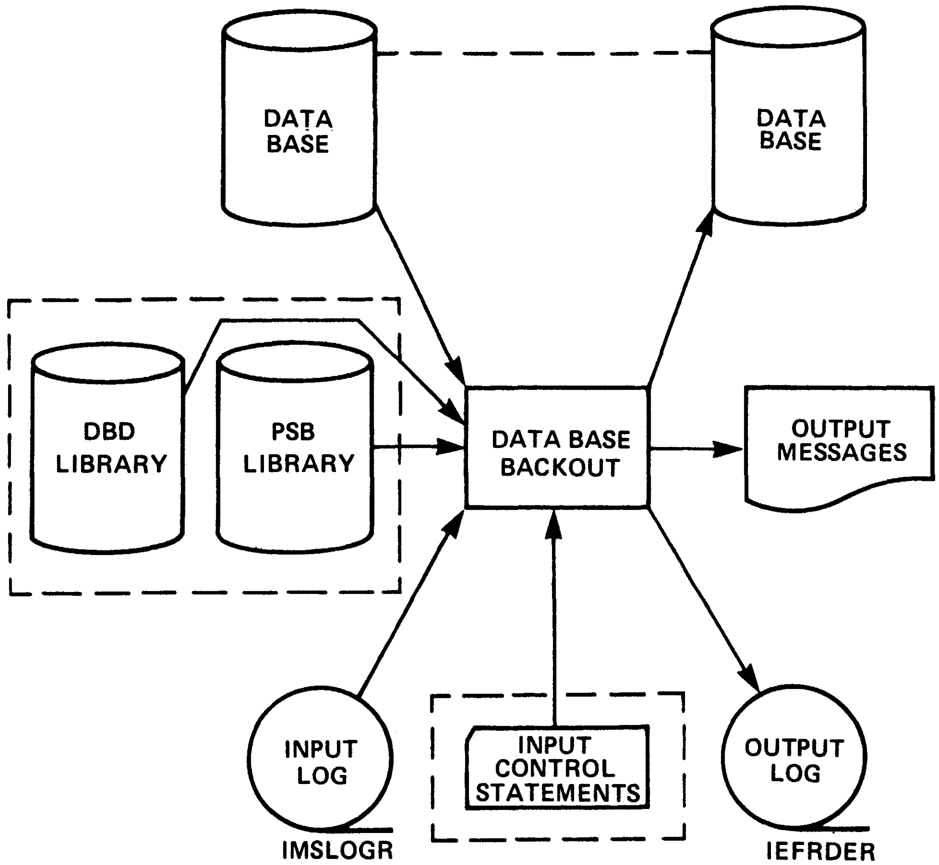


Figure 6-9. Data Set Requirements for the Data Base Backout Utility

JCL Statements

The data base backout utility is executed as a standard DL/I batch job. The following JCL statements are required:

EXEC — This statement must be in the following form:

PGM=DFSRRC00,PARM='DLI,DFSBB000,psbname'

where psbname is the name of the PSB used by the program to be backed out.

You can also use the DLIBATCH procedure to execute this utility. See Chapter 7, "Installing IMS/VS," for additional information on using the DLIBATCH procedure.

IMS DD — Concatenates the IMSVS.PSBLIB and IMSVS.DBDLIB libraries.

IMSLOGR DD — Describes the input log file. It must be a tape file; read-backwards is used.

IEFRDER DD — Describes the system log tape created during backout. The data set usually resides on tape. However, a direct access volume can be used.

dataset DD — These are the data base DD statements required by the PSB referenced in the EXEC statement. This data set must reside on a direct access volume. One DD statement is required for each data set of the referenced data bases.

SYSIN DD — Required only if a CHKPT control statement is supplied.

DFSVSAMP DD — Describes the data set that contains the buffer information required by the DL/I buffer handler. This DD statement is required if the data bases described by the dataset DD statements are VSAM data sets. For additional information, see the discussion on "Defining the IMS/VS Data Base Buffer Subpools" in Chapter 7, "Installing IMS/VS."

Utility Control Statement

One optional control statement can be used if the program uses the DL/I batch checkpoint/restart facility.

```
  1    7                                                         80
 ------------------------------------------------------------------
|                                                                  |
| CHKPT PPURC010                                                   |
|                                                                  |
 ------------------------------------------------------------------
```

Position	Description
1	Positions 1-5 must be the characters 'CHKPT'. These characters define the control statement.
7	This is the 8-character checkpoint ID supplied to DL/I with the 'CHKP' call. The ID is displayed as part of message DFS681I at the time the 'CHKP' call is made. If no ID is specified, the last checkpoint on the log tape will be used.

Note: All other positions should contain blanks.

Return Codes

The Data Base Backout utility provides the following return codes:

Code	Meaning
0	Backout successful. (DFS395I)
4	PSB incorrect. (DFS396I)
8	Unable to open data base. (DFS397I)
12 and above	Severe error condition; processing terminated.

These return codes can be tested by the COND= parameter on the EXEC statement of a subsequent job step. Each return code, however, causes a message to be printed.

Example

Jobs //SAMP177 and //SAMP364 in IMSVS.PRIMEJOB show the JCL for the backout of the FE1CPPUR program executions for phase 1 and phase 3, respectively.

SYSTEM LOG RECOVERY UTILITY (DFSULTR0)

This utility can be used to close a log data set when DL/I or OS/VS fails to do so. This will typically be required after an OS/VS, tape unit, CPU, or power failure. Note that when DL/I abends, the log data set will usually be closed by OS/VS. The OS/VS message IEF285I (data set KEPT) indicates that this has been done. Two steps are required to close a log data set with DFSULTR0. See Figure 6-10. Each step requires a separate execution of DFSULTR0.

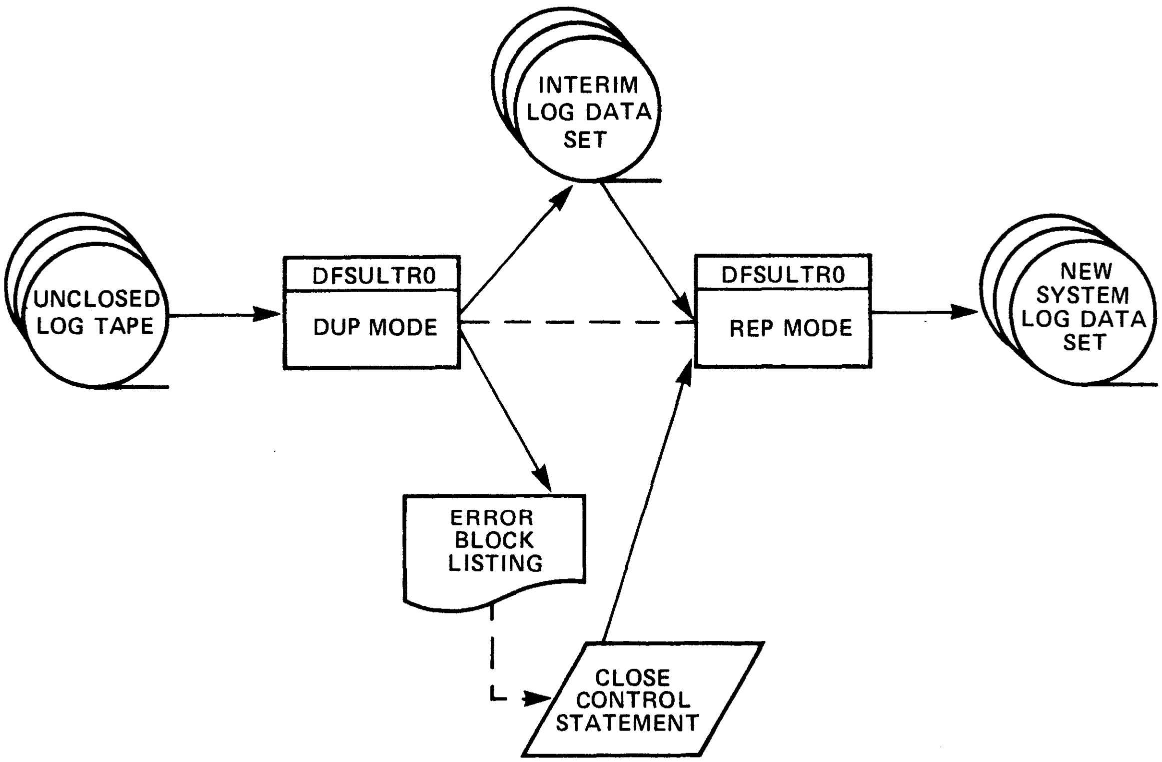


Figure 6-10. Closing the System Log with DFSULTR0

Step 1: DUP Mode

In the DUP mode, DFSULTR0 reads the log data set and copies it to a new interim log data set. A listing is produced of the error blocks. In the situation of the unclosed log tape, only the first error block is generally of interest. The sequence number of this error block (A00001) should be specified in the control statement of the second execution of DFSULTR0.

Notes:

1. If the log tape contains read errors before the end of the log data set, these would also be listed. In our subset we will not cover the correction of these errors. In that case, we will disregard that log tape data set and recover the data bases to the point of the start of the failing program. Following that, a full rerun of that program would be required.

2. Because the DFSULTR0 only checks the log record sequence number, old data from a previous (log) data set will be regarded as valid log data. This old data will therefore be copied to the interim log tape after signaling its initial sequence number break as an error block. Without checking the actual contents (that is, timestamps) in the log records you might not be able to distinguish this situation from read errors before the end of the current data set as discussed in note 1 above. To avoid this ambiguity you can use "clean" (that is, no data at all) log tapes or prewrite the log tapes with tape marks.

3. If the DUP mode of DFSULTR0 did not find an error block at the end of the current log data set, it will terminate normally, or abend. In that case, the log tape produced by the DUP mode is a valid log tape, and the REP mode is not necessary.

4. For more details on the IMS/VS log records, you should refer to the sections entitled "System Log Records" in Chapter 2, "System Maintenance/Tuning Facilities" of the *IMS/VS System Programming Reference Manual*.

Step 2: REP Mode

The REP mode of DFSULTR0 copies the correct blocks from the interim log data set, produced by the DUP mode execution, to a new log data set. At the end it will close that log data set, omitting the block in error as specified by the control statement.

JCL Statements

```
//RECOVER      JOB  (ACCTGINFORMATION)
//STEP         EXEC PGM=DFSULTR0
//SYSPRINT     DD   SYSOUT=A,DCB=(RECFM=FBA,LRECL=133)
//IEFRDER      DD   UNIT=3400,VOL=SER=LOG01,DSN=IMSLOG,DISP=OLD
//NEWRDER      DD   UNIT=3400,VOL=SER=NEWLOG,DSN=IMSLOG,
                    DISP=(NEW,KEEP),DCB=DSORG=PS
//NEWRDER2     DD   DUMMY,DCB=BLKSIZE=1460
//SYSIN        DD   *
```

Utility Control Statements

DUP Mode

The following control statement is required in our subset:

Beginning in Column	Format	Meaning
1	DUP	Indicates DUP mode.
5	ERRC=nnnnn	Indicates the maximum number of input I/O errors or sequence errors accepted before job termination. nnnnn must be a 5-digit numeric with leading zeros. Recommended value: ERRC=00010.

REP Mode

The following control statement is required in our subset:

Beginning in Column	Format	Meaning
1	REP	Indicates REP mode.
5	SEQ=AXXXXX	Indicates the identification number of the block in error. The identification consists of the letter "A" followed by a 5 digit integer, the error block sequence number (A00001 for the first error block). The number is provided in the listing output of the DUP step.
16	CLOSE	Close the output tape right *before* this error block.

Catalog Considerations

In general, log recovery is only required in case of hardware or OS/VS failure; that is, when the log tape is not closed. This normally implies that the jobstep termination processing did not occur. As a consequence the new log data set was not cataloged. Therefore we catalog the recovered log tape in the second phase of the log recovery process. However, this should be verified by comparing a list catalog with the manual log tape registration by the operator. Under no circumstance should the input tape for log recovery be used together with the ("duplicate") output tape for backout and data base recovery processing. Special care must be used if multi-volume log data sets are written and only the last volume is used for log tape recovery.

Examples

Jobs //SAMP190 and //SAMP191 in IMSVS.PRIMEJOB show the DUP and REP modes, respectively, of DFSULTRC.

BASIC RECOVERY PROCEDURES

The following guidelines should be observed when designing basic recovery procedures:

- The image copies of all data base data sets should be made at the same time. No intervening (update) programs should be executed against the data bases.
- A rigorous registration scheme should be established for the image copies and all program input and, optionally, output.
- All program input should be saved until the next image copies are made.
- In case correction of the error requires program changes, the old versions should be kept until the next image copies are made. Otherwise reruns, if necessary, could produce different results.

- The data bases must be restored immediately after any failing update job. The failure could be an application program, DL/I, OS/VS, or hardware error. Next, all data bases should be restored. Then, the application programs executed since that image copy was made should be reexecuted in the original sequence.

Note: As stated in the first part of this chapter under "Which One To Choose," you should realize the limitations of the above basic recovery procedures. In most cases the DL/I recovery procedures as outlined in the following section are far more desirable.

EXAMPLES

Job //SAMP180 in IMSVS.PRIMEJOB can be used to make an image copy of the phase 1 Parts data base. Job //SAMP182 can be used to restore this data base. These jobs show how the image copy and restore utilities are used in a basic recovery environment.

DL/I RECOVERY PROCEDURES

The following procedures can be used as a basis for the recovery procedures at your own installation. It is strongly recommended that you exercise and enforce such procedures before going into a production phase with your data bases.

ASSUMPTIONS AND RESTRICTIONS

1. Image copies of all data sets of all data bases are made at the same time, that is, no intervening data base processing.

   Note: The above restriction is based solely on the subset approach; it is not a DL/I requirement.

2. No other update program may have been executed after the failure involving the data base in error. If that should occur, you must restore all affected data bases and rerun the programs in proper sequence.

3. After each new image copy of your data bases, you must run the first change accumulation with a dummy old change accumulation data set; that is, never use a log tape or change accumulation data set of the previous period.

POSSIBLE FAILURES

The table in Figure 6-11 lists the most common failures, together with their symptoms, which can occur in the batch processing of DL/I data bases. For each failure, an error class is given. This error class determines the required recovery actions. See Figure 6-12.

FAILURE/SITUATION DESCRIPTION	SYMPTOMS	ERROR CLASS
1. OPERATING ERROR		
1.1 Job cancellation	X22 ABEND	A
1.2 Wrong input or wrong data base	INCONSISTENT RESULTS	A
2. APPLICATION PROGRAM ERROR		
2.1 Wrong logic/output	INCONSISTENT RESULTS	A
2.2 Abend	USER ABEND	A
3. DL/I ERROR		
3.1 Abend	DL/I ABEND	A
3.2 Loop or wait state	CANCELLED (X22)	A
4. OS/VS ERROR		
4.1 Abend	RE-IPL	C
4.2 Loop or non-dispatchable	RE-IPL	C
5. HARDWARE		
5.1 Read I/O error on a data base data set	A0 status code + DL/I message DFS451A	B
5.2 Write I/O error on a data base data set	DL/I message DFS451A	B
5.3 Power failure, machine check	RE-IPL	C

Figure 6-11. Possible Failures during Data Base Processing

Note: Upon receipt of message DFS451A, the OS/VS system console operator should (a) take action to stop the execution of subsequent DL/I jobs and (b) reply "ABEND".

CORRECTING THE CAUSE OF THE FAILURE

This activity is completely dependent on the type of failure. Often, the action to be taken is beyond the DL/I environment, for example, system/power failure. If the error is in DL/I, the IMS/VS Messages and Codes Reference Manual must be consulted. If it is a severe error condition, the IBM Field Engineering Program System Representative should be notified. Appendix A in the IMS/VS Messages and Codes Reference Manual provides guidelines for doing this.

RECOVERY TASKS

The subsequent recovery tasks to be performed for each defined error class (Figure 6-11) are listed in Figure 6-12.

ERROR CLASS	RECOVERY TASKS: LOG CLOSE (DFSULTR0)	CHANGE ACCUMULATION (DFSUCUM0): EXCLUDE CURRENT LOG	CHANGE ACCUMULATION (DFSUCUM0): INCLUDE CURRENT LOG	DB RECOVERY (DFSURDB0): ONLY AFFECTED DATA SETS	DB RECOVERY (DFSURDB0): ALL DATA SETS USED BY PROGRAM	BACKOUT (DFSBBO00): ALWAYS CURRENT LOG (must be a tape)	RESUME PROCESSING (APPLICATION PROGRAM): RESTART	RESUME PROCESSING (APPLICATION PROGRAM): RERUN	NOTES
A						*	*		1. current log + output log of back-out must both be included in next change accumulation run
B			*	*		*	*		1. see A1 2. if program was successfully completed, no restart required
C	*					*	*		1. see A1
D		*			*			*	1. the current, bad, log must not be used in any further back-out or change accumulation
SAMPLE JOB #	190 + 191	181,381		182,382	382+383	177,384	178	initial program executions	

Figure 6-12. Data Base Recovery Actions

The data base backout utility is not required for a retrieve-only program. The additional error class "D" would occur if for any reason the current log data set is unusable. The current log data set is the one being created when the failure occurred.

Possible causes for this could be:

- Log data set close (DFSULTR0) failed,
- Lost log data set due to operational error,
- I/O errors on the log data set during change accumulation.

The recovery tasks in Figure 6-12 must be executed in sequence from left to right. Whenever an error occurs during an A, B, or C error correction, you can fall back upon the error class D procedure.

If an error occurs during the recovery of a class D error, you have to fall back to previous image copies and log change accumulation data sets.

IMAGE COPY/LOG ADMINISTRATION

A rigorous administration of data base image copies, log data sets, and log accumulation data sets is a necessity for data base integrity. We will now discuss an administration scheme for this. In this scheme, we will use the generation data group facility of OS/VS. It would form the basis of your own scheme, adapted to your installation standards and requirements.

At a minimum you should set up a manual registration scheme for the log data sets, change accumulations, and image dumps.

Two sets of forms could be used. One form, Figure 6-13, is used to register all the DL/I jobs which produce log data sets.

DL/I LOG TAPE FORM		DATE:		PERIOD:
TIME	VOLUME(S)	JOB/STEP	IF FAILURE LAST CHECKPOINTID	REMARKS AND ERROR DESCRIPTION

Figure 6-13. Sample DL/I Log Tape Form

__Notes__:

1. All DL/I jobs should be listed, also the backout and recovery jobs.

2. If multiple log volumes are created in one job (step), they should be listed in time sequence.

3. A new period starts after each image copy.

4. Optionally, the data set name of the log data set should be registered. Normally this would always be the same or a generation data group, in which case only the generation number would be registered.

The second form is used to register the image copy and change accumulation jobs during each period (Figure 6-14).

```
+-----------------------------------------------------------------+
|DL/I Change Accumulation Form      DATE:            PERIOD:       |
|=================================================================|
|                            INPUT                 |     OUTPUT   |
|-----------------------------------------------------------------|
|        |        DBD          |      DATA SET      |DATA SET|VOLUME(S)|
|        |--------------------------------------------------------|
|        |                     |                    |        |         |
|IMAGE   |                     |                    |        |         |
| COPY   |                     |                    |        |         |
|        |                     |                    |        |         |
|=================================================================|
|        | OLD ACCUMULATION    |  LOG DATA SETS     | NEW ACCUMULATION |
|        |--------------------------------------------------------|
|        |DATA SET|VOLUME(S)   |DATA SET|VOLUME(S)  |DATA SET|VOLUME(S)|
|        |--------------------------------------------------------|
|        |        |            |        |           |        |         |
|CHANGE  |        |            |        |           |        |         |
|ACCUM.  |        |            |        |           |        |         |
|        |        |            |        |           |        |         |
|        |        |            |        |           |        |         |
|        |        |            |        |           |        |         |
|        |        |            |        |           |        |         |
|        |        |            |        |           |        |         |
+-----------------------------------------------------------------+
```

Figure 6-14. Registration of Image Copies and Change Accumulations

Notes:

1. Each period starts with an image copy of all data base data sets.

2. When generation data groups are used, only the generation number need be registered.

3. The first change accumulation run after the image copy should not have any old log tape or change accumulation tape as input (that is, //DFSUCUM0 DD DUMMY,DCB=BLKSIZE=100).

Examples

In our phase 3 sample jobs, we use generation data groups for the data set image copies, the log data sets, and the accumulation data sets.

FREQUENCY OF IMAGE COPIES AND CHANGE ACCUMULATIONS

The frequency of image copies is dependent on your installation environment. It is a trade-off between the necessary recovery costs in case of failure and the cost of taking the image copies.

The basic recommendation for taking image copies is:

- Immediately after initial load of the data bases.
- Immediately before data base reorganization, if the old space is deleted during reorganization.
- Immediately after data base reorganization.
- Once each week.

The basic recommendation for the change accumulation is once a day. Another approach would be to perform change accumulation as a second step, controlled via condition codes, in every DL/I update job.

Retention Period of Image Copy and Log Data Sets

To protect yourself against unusable image copies, logs, and change accumulation tapes, you should retain those tapes for at least two or three periods (a period is defined as the interval between two subsequent image copies). A suggested retention scheme, assuming a one week period, would be:

- Log tapes are retained two weeks or until the time the next image copy is made.
- Change accumulation tapes are made at least daily. The last one of the period is retained two extra periods.
- Image copies are retained three periods.

VSAM CATALOG CONSIDERATIONS

It is strongly recommended that you use a separate VSAM user catalog for your data base data sets. When your installation grows, you should consider a user catalog for each application or project.

In case of an error in the user catalog, you should first try to correct the problem with the OS/VS Access Method Services VERIFY function. If this fails, the following procedure can be followed (VSAM 2 only):

1. Perform Access Method Services: ALTER REMOVEVOLUMES

   This will delete all the data spaces owned by the user catalog and the user catalog itself. You should specify all the volumes owned by that user catalog.

2. Perform Access Method Services: DEFINE

   A new user catalog and new data base data space.

3. DL/I

   Recover all the affected data base data sets.

Warning: The above procedure completely erases (that is, overwrites with binary zeros) all VSAM data space, including the user catalog on the specified volumes. You should use this only if the VSAM user catalog has become inaccessible. For more details see also Perform Access Method Services: "VSAM Volume Cleanup."

DATA BASE RECOVERY IN AN ONLINE IMS/VS SYSTEM

Additional factors should be considered when setting up recovery procedures for data bases used by an online IMS/VS system.

As discussed in Chapter 3, "Data Communication Design," a dynamic log data set is used by the online system for recording data base changes, as well as the log tape. Abending online programs are automatically backed out by the online system using the dynamic log records. In addition, if the system should fail while an application program is active, any updates made by that program will be automatically backed out when the system is restarted. In our subset, if the program was a

BMP the updates are automatically backed out to its most recent checkpoint. Because of this automatic backout, the operator will usually need to run the recovery utilities only when there has been a major system failure, generally one which entails a re-IPL of OS/VS, or when there are I/O errors on a data base.

The recovery procedures outlined later in this chapter make use of the DL/I recovery procedures and utilities described earlier in this chapter as well as an additional log tape maintenance utility, the System Log Terminator utility.

SYSTEM LOG TERMINATOR UTILITY (DFSFLOTO)

At the time of a system failure, the System Log Terminator utility can be used to recover any log data that may have been lost as a result of the failure. A storage dump taken at the time of failure is required. This storage dump can be the SYS1.DUMP data set, or a stand-alone dump output tape. The log terminator program:

- Locates the log work area, buffers, and control blocks in the storage dump.
- Positions the log tape and writes the remaining buffers.
- Closes the log data set.

Detailed instructions on running the utility, and the possible error messages that may occur are described in the _IMS/VS Primer Master Terminal Operator's Guide_.

Figure 6-15 shows the data set requirements for running the System Log Terminator utility.

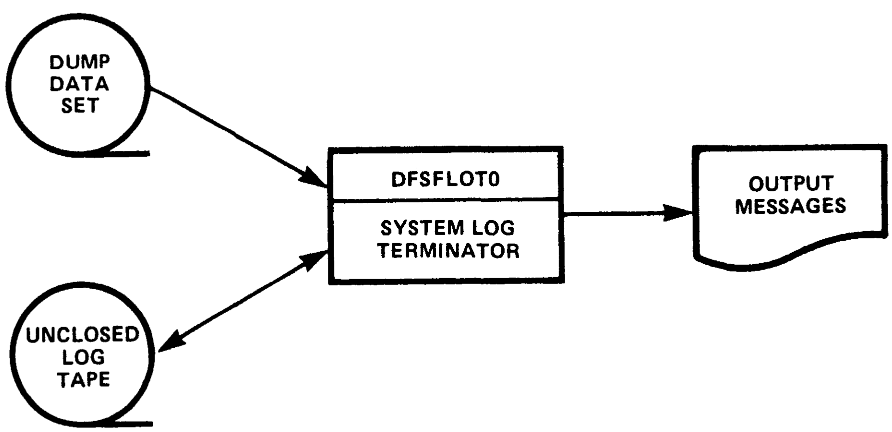


Figure 6-15. Running the System Log Terminator Utility

JCL Statements

The System Log Terminator utility is executed as a standard OS/VS job. The following JCL statements are required:

EXEC This statement must be in the following form:
PGM=DFSFLOTO

SYSPRINT DD Defines the output message data set.

LOGTAPE DD — Defines the log tape to be terminated. This data set must have had a disposition of (NEW,KEEP) at execution time.

DUMP DD — Defines the SYS1.DUMP data set, or the stand-alone dump data set. The DCB information for this data set should be obtained from the OS/VS system programmer in your installation.

__Examples__

```
//TERMIN    JOB  (ACCTINGINFO)
//STEP      EXEC PGM=DFSFICTO
//SYSPRINT DD   SYSOUT=A
//LOGTAPE   DD   DSN=IMSLOG,VOL=SER=LOG001,UNIT=3400-4,DISP=(,KEEP)
//DUMP      DD   DSN=SYS1.DUMP,VOL=SER=SYSDMP,UNIT=3400-4,DISP=OLD,
//               LABEL=(,NL),DCB=(RECFM=U,BLKSIZE=2056)
```

Job SAMP492 in IMSVS.PRIMEJCL also shows an example of the JCL for this utility.

__ONLINE RECOVERY PROCEDURES__

The following recovery procedures can be used as a basis for the recovery procedures in your installation. It is strongly recommended that you exercise and enforce such procedures before going into a production phase with your online system.

ASSUMPTIONS AND RESTRICTIONS

1. The same assumptions and restrictions described for the DL/I recovery procedures earlier in this chapter apply here as well.

2. The recovery procedures outlined here for handling I/O errors on data bases involve closing down the online system, running batch recovery procedures, and then restarting the online system.

   __Note__: The above restriction is based solely on the subset approach. It is not an IMS/VS requirement that the online system be closed down to perform data base recovery.

3. These procedures are designed to be used in conjunction with the __IMS/VS Primer Master Terminal Operator's Guide__. If you alter these procedures, you should ensure that the operating guide is changed accordingly.

POSSIBLE FAILURES

The table in Figure 6-16 lists the most common failures, together with their symptoms, which can occur in the online system. For each failure an error class is given. This error class determines the required recovery procedures, which are outlined in Figure 6-17.

FAILURE/SITUATION DESCRIPTION	SYMPTOMS	ERROR
1.OPERATING ERROR		
1.1 Cancel IMS control region	X22 or U0020 abend	E
1.2 Cancel BMP	X22 of BMP	A
2.APPLICATION PROGRAM ERROR		
2.1 MPP Abend	DFS555,DFS554,DFS980	-
2.2 BMP Abend	DFS555,DFS554,DFS980	A
3.IMS/VS ERROR		
3.1 Abend	IMS Abend code	B
3.2 Loop or wait state	Cancelled(X22 or U0020)	B
4.OS/VS ERROR		
4.1 Abend,loop or non-dispatchable (storage dump taken)	Re-IPL	C
4.3 Abend,loop or non-dispatchable (No storage dump taken)	Re-IPL	D
5.HARDWARE ERROR		
5.1 Machine check, power failure	Re-IPL	D
5.2 I/O Error on data base	DFS451	E
5.3 I/O Error on data base during backout	DFS981,DFS983	F

Note: The IMS/VS control region may be canceled, either by an OS/VS cancel command if it is running as a problem task, or by an OS/VS modify command if it is running as a system task.

Figure 6-16. Possible Failures During An Online Session

CORRECTING THE CAUSE OF THE FAILURE

This activity is completely dependent on the type of failure. The action, to be taken by the master terminal operator (MTO) is outlined in the IMS/VS Primer MTOs Guide.

RECOVERY TASKS

The subsequent recovery tasks to be performed for each defined error class are listed in Figure 6-17. The tasks must be executed from left to right. They are also described in flowchart form in the IMS/VS Primer MTO's Guide.

ERROR CLASS	RECOVERY TASKS								
	SHUT DOWN IMS	CLOSE LOG TAPE		CHANGE ACCUMULATION	DATABASE RECOVERY	BATCH BACKOUT	RESTART		NOTES
		DFSFLOT0	DFSULTR0	INCLUDE CURRENT LOG	ONLY AFFECTED DS	INCLUDE CURRENT LOG	IMS	BMP	
A								*	1.
B							*	*	2,3,4
C		*					*	*	2,3,4
D			*				*	*	2,3,4
E	*			*	*		*	*	2,3,4
F	*			*	*	*	*	*	2,3,4,5

Figure 6-17. Data Base Recovery Actions in an Online Environment

__Notes__:

1. Any updates done by an MPP or BMP which has been cancelled, or has abended, are automatically backed out by IMS/VS.

2. If a BMP or MPP is active at the time that the online system fails, the updates done by that application program are automatically backed out during the emergency restart of IMS/VS.

3. The BMP which was active at the time of failure should be restarted from its last checkpoint.

4. When the online system is restarted after a system failure, the terminal users should check the status of the transactions they entered immediately prior to the failure. Any special action they should take is documented in the operating procedures for that transaction in the IMS/VS Primer Remote Terminal Operator's Guide.

5. If during a system failure, the log buffer was lost (that is, log tape recovery was required) the subsequent restart may cause input messages to be lost (no data base updates done) or duplicate output messages (message retransmitted during restart).

6. The program named in the DFS983 message must be specified in the JCL for the backout utility.

It is important that these tasks be executed in the order shown. For example, if the system failed while a BMP was active, the online system must be restarted after the other indicated procedures have been followed. During restart, the BMP updates are backed out. Next the BMP should be restarted. If the BMP uses extended checkpoint/restart (XRST call), you must supply the log tape with the BMP checkpoint (the input log tape for the emergency restart) with the IMSLOGR DD statement to the restart JCL. Since OS/VS2 (MVS) enqueues on the volume serial information, you must first (MVS only) shut down the system in between. This must be done before any batch update programs are scheduled against the affected data bases. If desired, the system can be restarted, and then closed down as soon as the backout is complete.

__Note__: There is an additional error situation which could occur if for some reason it is necessary to recover a data base, __and__ the log tape from the most recent online session is unavailable. For example, there could be an I/O error on the data base and the log tape from the last session has been lost due to an operational error.

In this case, all data sets used in that online session must be recovered to the start of the online session during which the error occurred. All BMPs which update data bases, and which were run during that session must be re-run, and the remote terminal operators must re-submit all update transactions entered during that session. You can limit the above to a single application if the data base in error is only used by that single application. If so, you must ensure that you recover <u>all</u> data bases used by that application, and conversely that the BMPs and update transactions that are re-submitted affect only those data bases that are being recovered.

Although this situation should not occur frequently, if at all, as it is a result of a combination of errors, it is nevertheless possible, and should be taken into account when the application design is done. It may imply that remote terminal operators must keep a hard-copy record of the input to critical update transactions.

LOG TAPE ADMINISTRATION IN AN ONLINE ENVIRONMENT

The log tape produced by the online IMS/VS system is vital for data base integrity. In addition it is also necessary for restarting the online system, and for providing statistics for monitoring the performance of the system. We will now discuss an administration scheme for controlling the log tapes used by the online system. Although this method does not use generation data groups, we will show how it interfaces with the scheme for controlling batch log tapes that was described earlier in this chapter.

Log Tape Data Set Names

When a BMP is restarted, the log tape containing the checkpoint from which it is being restarted must be allocated via an //IMSLOGR DD card to the BMP partition. To avoid allocation conflicts some special tape handling techniques must be used. Although bypass label processing could be used in the BMP JCL, we recommend that standard labels be used for all jobs, but that the data set names be greater than seventeen characters. This will avoid OS/VS allocation conflicts.

The following JCL could be used:

- Control Region JCL
  //IEFRDER DD DSN=ONLINE.IMS.VS.PRIMER.LOG,...
- BMP Restart JCL
  //IMSLOGR DD DSN=RESTART.IMS.VS.PRIMER.LOG,....

This technique could be extended further to other jobs which you may wish to run on the previous day's log tapes, while the online system is active. For example:

- Log Tape Statistics
  //LOGIN DD DSN=STATS.IMS.VS.PRIMER.LOG,....
- Change Accumulation
  //DFSULOG DD DSN=ACCUM.IMS.VS.PRIMER.LOG,.....

<u>Note</u>: This technique is based on the fact that OS/VS enqueues on the full data set name in the DD card, but only the last seventeen characters are actually recorded on the tape label.

Examples of the use of this naming convention are shown in jobs SAMPI40, SAMP474, SAMP481 and SAMP495 in IMSVS.PRIMEJOB.

Log Tape Serial Numbers

To reduce the possibility of operator error, we suggest that a pool of tapes be allocated for use as online log tapes, and that they be sequentially numbered, for example, LOG001, LOG002, LOG003,.. etc. By using sequential numbering, and using the tapes in sequence, the restart and change accumulation procedures are simplified.

We also suggest that log tapes be clearly marked as such with external labels, possibly in a bright color. This is to minimize the possiblity of an online log tape being accidentally unloaded while the online system is active or being used by mistake as a scratch tape.

Log Tape Control Forms

Two forms are suggested. One is the IMS/VS Online Log Sheet, (Figure 6-18) which is discussed in more detail in Chapter 2 of the _IMS/VS Primer Master Terminal Operator's Guide_. The other is the Image Copy and Change Accumulation Form (Figure 6-14) described earlier in this chapter.

If you use generation data groups for maintaining your batch log tapes, two change accumulation jobs could be used: one for the log tapes produced by batch jobs, and one for those produced by the online system. All log tapes, batch or online must all be used in the _correct time order_ to produce the change accumulation data sets needed for data base recovery. Examples are shown in jobs //SAMP381 and //SAMP481 in IMSVS.PRIMEJOB.

FREQUENCY OF IMAGE COPIES AND CHANGE ACCUMULATIONS

The remarks made earlier in this chapter under this heading apply equally well to the online environment.

The basic recommendation for change accumulation of the online log tapes is once a day. Another approach could be to perform change accumulation whenever the online system is terminated. However this approach could delay the restarting of the system, if the shutdown was unscheduled.

Retention Period of Online Log Tapes

The remarks made earlier in this chapter relating to batch log data sets apply equally to online log tapes.

IMS/VS ONLINE LOG SHEET

DATE	
START TIME	
STOP TIME	
MTO NAME	

/NRE	BUILDQ
/ERE	FORMAT ALL
CHKPT	
SERIAL	
LOG BUFFER	

/CHE	DUMPQ
	FREEZE
OTHER (specify)	
LAST CHKPT-ID	

LOG TAPES		
SERIAL	CHANGE ACCUM	STATS

TIME	COMMENTS/INCIDENTS

Figure 6-18. IMS/VS Online Log Sheet

CHAPTER 7. INSTALLING IMS/VS

This chapter contains detailed information necessary to install and use IMS/VS.

Three different software installations are distinguished:

- IMS/VS-DB installation
- IMS/VS-BTAM installation
- IMS/VS-VTAM installation

In addition to the installation of IMS/VS itself, the generation of VTAM (level 2 only) and NCP/VS in our subset environment is also discussed.

Before reading this chapter, you should be familiar with OS/VS and its system generation, and the access methods used by IMS/VS. IMS/VS operates under OS/VS1 or OS/VS2. Very little difference is experienced by the IMS/VS user between OS/VS1 or OS/VS2. The application programs are particularly unaware of the operating system being used. At this point we will consider only OS/VS1 in our discussion and examples. At the end of the chapter, additional considerations and guidelines are presented for the OS/VS2 (MVS only) user.

The next section guides you through the installation process.

THE INSTALLATION PROCESS

The installation process for IMS/VS in an SNA environment consists of the following steps (see Figure 7-1).

1. OS/VS1 initial preparation.
2. Creation of the IMS/VS libraries.
3. Restoring the IMS/VS libraries.
4. IMS/VS Stage 1 system definition.
5. IMS/VS Stage 2 system definition.
6. OS/VS1 final preparation.
7. VTAM and NCP/VS generation.

   Step 7 is not necessary for an IMS/VS-DB installation or a BTAM-only installation.

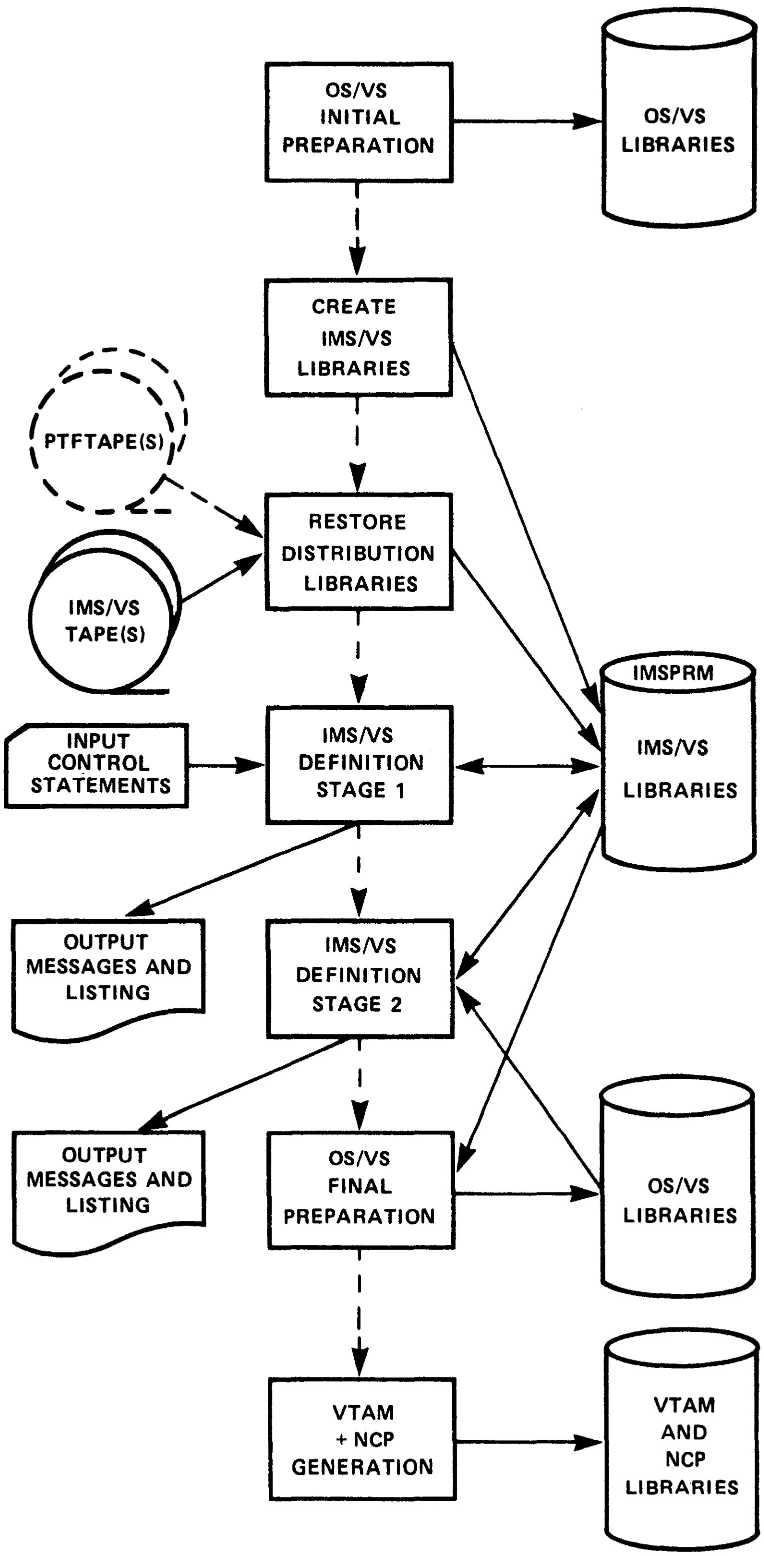


Figure 7-1. Installing IMS/VS

OS/VS1 PREPARATION

Some OS/VS1 optional facilities are required to support IMS/VS.

OS/VS1 VSAM Considerations

If VSAM is to be used for data bases, it must be included in OS/VS1 during system generation. Specify ACSMETH=(VSAM) in the OS/VS1 system generation DATAMGT macro instruction.

OS/VS1 VTAM Considerations (DC Only)

VTAM is incorporated into the operating system during operating system generation. In our subset, we will give an example of including VTAM in your operating system for use with IMS/VS. Details for planning and generating a complete OS/VS1 and VTAM appear in:

- *OS/VS System Generation Introduction*, GC26-3790
- *OS/VS1 System Generation Reference*, GC26-3791
- *OS/VS1 VTAM System Programmer's Guide*, GC27-6996

VTAM is specified for inclusion in the operating system by using the ACSMETH parameter of the DATAMGT macro instruction: ACSMETH=(VTAM). (The DATAMGT macro instruction is included in Stage 1 input to OS/VS1 system generation.)

VTAM should run in a low-numbered partition (for high priority, normally, P0). It should run at a higher priority than IMS/VS. The partition's size is determined by the user's needs. Information for estimating the size of the VTAM virtual storage partition appears in the OS/VS Storage Estimates manual.

GTF: The generalized trace facility (GTF) must be installed and active to use the VTAM trace facility as discussed in Chapter 9, "Optimization."

IMS/VS Supervisor Call Routine

One type 2 user SVC is required to execute IMS/VS, DB or DB/DC. This SVC must be defined in your OS/VS1 system during OS/VS1 system generation. See "OS/VS1 Systems Generation," SVCTABLE macro instruction for more details.

An example of the SVCTABLE macro in your OS/VS1 system generation follows:

```
    SVCTABLE       SVC-254-E2-S0
```

Normally, the SVC routine itself is not incorporated during the OS/VS1 generation. So you must include a "dummy" load module in the RESMCDS partitioned data set. This should be done prior to Stage 2 of the OS/VS1 system generation.

The format of the module is:

```
          IGCnnn  CSECT
                  BR   14
                  END
```

where nnn is the unique SVC number. This effectively NOPs the SVC number. The actual inclusion of the SVC routine in OS/VS1 will be done after the IMS/VS Stage 2 generation.

Optional Program Products

Additional program products are desirable but not required, such as:

- PL/I optimizer compiler
- OS/VS COBOL
- Sort/Merge
- Assembler H

Note: A Sort/Merge function is required if using logical relationships or secondary indices, or the DL/I log data set change accumulation utility.

INSTALLING A DB SYSTEM OR A DB/DC SYSTEM

In the following sections, we will discuss the IMS/VS-DB and the IMS/VS-DB/DC installation separately.

DB-only users should read the following section "Installing IMS/VS-DB" and then turn to the section entitled "Executing the Sample."

DB/DC users should turn now to the section entitled "Installing IMS/VS-DB/DC."

INSTALLING IMS/VS-DB

After the initial preparation of the OS/VS1 system described in the preceding section, the following steps should be performed.

CREATING THE IMS/VS-DB LIBRARIES

Operation of IMS/VS-DB requires three categories of libraries for storing modules, programs and control blocks.

The IMS/VS-DB Distribution Libraries

The distribution tape from the IBM program library contains three libraries.

- IMS.DBGENLIB contains the macros for the generation and execution of IMS/VS-DB.

- IMS.DBLOAD contains the IMS/VS-DB load modules.

- IMS.DBSOURCE contains the source code of the IMS/VS-DB modules.

You should pre-allocate space for these libraries on your IMS/VS system pack and use IEBCOPY to restore them from the distribution tape.

Notes:

1. The IMS.DBGENLIB (named IMSVS.GENLIB when restored) is used only during IMS/VS system definition, Stages 1 and 2. After this, it is required only for system maintenance.

2. The IMS.DBLOAD (named IMSVS.LOAD when restored) and IMS.DBSOURCE (named IMSVS.DBSOURCE when restored), are used only during Stage 2 of IMS/VS system definition. After this, they are required only for system maintenance.

The IMS/VS-DB System Libraries

For the execution of IMS/VS-DB, the following system libraries are needed:

- IMSVS.MACLIB contains the macros for the generation of DBDs and PSBs.
- IMSVS.RESLIB contains the IMS/VS-DB execution modules.
- IMSVS.PROCLIB contains the IMS/VS-DB job control procedures.

The MACLIB, RESLIB, and PROCLIB are established during IMS/VS-DB generation.

The IMS/VS-DB Application Libraries

The following application libraries are needed for the execution of IMS/VS-DB application programs:

- IMSVS.DBDLIB contains the DBDs.
- IMSVS.PSBLIB contains the PSBs.
- IMSVS.PGMLIB contains the application programs.

The DBDs and PSBs are stored as standard OS/VS load modules during DBD and PSB generation, respectively. The application programs are stored in the PGMLIB during link-editing.

The IMS/VS-DB Primer Function Sample Libraries

The following sample libraries are created after restoring the IMS/VS-DB distribution tape:

- IMSVS.PRIMESRC contains the sample source statement for the Primer function programs, DBD's, PSB's, data base data, etc.
- IMSVS.PRIMEJOB contains the JCL statements for the Primer function sample jobs.

RESTORING THE IMS/VS-DB DISTRIBUTION LIBRARIES

The IMS/VS-DB distribution tape contains three libraries which are unloaded partitioned data sets in IEBCOPY format.

Notes:

1. For the exact format you should check the distribution letter which accompanies the tape.
2. Optionally, you will also receive a "PTF tape". This tape contains updated members of the original libraries. This tape should be restored first and merged with the original tape.

IMS/VS-DB STAGE 1 SYSTEM DEFINITION

Three IMS/VS system definition macros are used for the definition of an IMS/VS-DB environment.

The coding conventions for these macros are the same as for the coding of OS/VS Assembler Language source statements. Processing of those macros by the OS/VS Assembler or the OS/VS program product Assembler H creates the input job stream for the execution of Stage 2. All the macros needed for the Stage 1 assembly are provided in IMSVS.GENLIB.

Coding the IMS/VS-DB System Definition Macros

The three system definition macros required for the batch system are IMSCTRL, IMSCTF and IMSGEN. They should be coded in that order as follows:

```
|          | IMSCTRL     | SYSTEM=({VS1V},BATCH,{6.0})
|          |             |         {VS/2}       {3.7}
|          |             |
|          |             | [,MCS=(number[,number,...])]
|          |             |
|          |             | [,DESC=number]
```

Optional operands:

SYSTEM= specifies the OS/VS version and release, and the type of IMS/VS system to be generated. VS1V with 6.0 specifies OS/VS1 Release 6.0. VS/2 with 3.7 specifies OS/VS2 Release 3.7. BATCH specifies the generation of an IMS/VS batch system.

MCS= specifies the VS routing code to be assigned to the IMS/VS system console if multiple console support (MCS) is included in the operating system. If MCS is not specified, no routing code is used. For a list of valid routing and descriptor codes see, <u>OS/VS Supervisor Services and Macro Instructions</u>, GC27-6979.

DESC= specifies the message descriptor code to be assigned to the IMS/VS system console messages if MCS support is included in the OS/VS generations. If DESC is not specified, no descriptor is assigned.

The MCS= and DESC= keywords should be defined as required for the ROUTCDE and DESC keywords of the OS/VS WTO macro. See the <u>OS/VS Supervisor Services and Macro Instructions</u>, GC27-6979, for a detailed description of the WTO macro keyword parameters.

```
|          | IMSCTF     | SVCNO=(,type2)
|          |            |
|          |            |  ,LOG=(SNGL,MONITOR)
```

Optional operands:

SVCNO= Specifies the operating system type 2 SVC number reserved for use by IMS/VS. Values entered may range from 128 to 255. The SVC number must be specified as the second parameter to be compatible with earlier releases of IMS/VS; therefore, the parentheses and the comma are required. The default is SVCNO=(,254).

LOG=(SNGL,MONITOR)

Should be coded as shown in our subset. It provides for one log data set per job (step) and the optional activation of the DB Monitor.

```
| IMSGEN | JCL=( [{IMSGEN }]     [,job accounting]
|        |       [{jobname}]
|        |
|        |     [,{IMS           }]  [,{A          }]
|        |     [ {programmername}]  [ {outputclass}]
|        |
|        |     [,(job miscellaneous)])
|        |
|        |     [,NODE=(IMSVS, IMSVS, IMSVS)]
|        |     [       node1  node2  node3 ]
|        |
|        |     [,OBJDSET=  {IMSVS.OBJDSET}]
|        |     [           {    name     }]
|        |
|        |     [,USERLIB=  {IMSVS.RESLIB}]
|        |     [           {    name    }]
|        |
|        |     [,ASM={VS}]
|        |     [     {H }]
```

JCL=

jobname specifies a maximum of six alphameric characters to be used as the first portion of the generated job names. The last two characters of the job names are the internally generated, sequentially incremented, numeric values representing the relative position of each job in the Stage 2 stream. The default is IMSGEN.

jobaccounting specifies job accounting data to be placed in the Stage 2 JCL. The length of the accounting data may not exceed 50 bytes.

programmername specifies the programmer name to be placed in the Stage 2 JCL. The default is IMS.

outputclass specifies the output class to be generated for the Stage 2 JCL. The default is A.

job miscellaneous specifies any additional parameters the user may desire to have placed in the Stage 2 JOB statements. Length of this parameter cannot exceed 50 bytes. Recommended: TYPRUN=HOLD.

Note: If job accounting, programmer name or job miscellaneous contains non-alphabetic characters then the parameter should be enclosed in double quotes: ''......''.

NODE=

node1 specifies the node to be assigned to all IMS/VS data set names to be used and generated by IMS/VS system definition. The specified node can consist of from 1 to 8 characters. The first character must be a letter or a national character (@, $, #). The default node generated is IMSVS.

node2 specifies the node to be assigned to the IMS/VS data set names MACLIB, PROCLIB, MATRIX, JOBS, and RESLIB. This node overrides the node1 assignment for these specific data sets.

node3 specifies the node to be assigned to the IMS/VS data set names DBSOURCE, GENLIB, and LOAD. This node overrides the node1 assignments for these specific data sets.

OBJDSET=

specifies the name (maximum of 24 characters) of a cataloged partitioned data set into which assembler object modules are placed during Stage 2 of IMS/VS system definition. The default is IMSVS.OBJDSET.

USERLIB=

specifies the name (maximum of 24 characters) of a library for user modules to be included in the system. This is not applicable to IMS/VS-DB. However, to avoid JCL errors in Stage 2, you should specify here the name of your RESLIB if it is not IMSVS.RESLIB.

ASM=

specifies whether the OS/VS Assembler (VS) or OS/VS program product Assembler H (H) JCL is to be produced for the Stage 2 assembly steps. The default is VS.

The IMS/VS Stage 1 system definition is a standard OS/VS assembly. The generated output is the Stage 2 job stream.

IMS/VS-DB STAGE 2 SYSTEM DEFINITION

This is the execution of the jobs generated by the Stage 1 system definition.

OS/VS1 FINAL PREPARATION

Some changes must be incorporated in OS/VS1 after the IMS/VS-DB Stage 2 system definition.

Relink the OS/VS Nucleus with the IMS/VS Type 2 SVC

The OS/VS nucleus must be re-linked to include the Type 2 IMS/VS SVC module. This module was placed in IMSVS.RESLIB during Stage 2 of the IMS/VS system definition.

Copy IMSRDR Procedure to SYS1.PROCLIB

To be able to use the IMS/VS supplied procedures in IMSVS.PROCLIB, the IMSRDR procedure should be copied from IMSVS.PROCLIB to SYS1.PROCLIB, or you should add IMSVS.PROCLIB to the IEFPDSI DD statement of your OS/VS reader procedure.

IMS/VS-DB INSTALLATION JOBS

This section presents all the jobs to install IMS/VS-DB. A listing of these jobs is provided in Chapter 2 of the *IMS/VS Primer Sample Listings*. All jobs for the installation are named "SAMPInn", except the jobs generated by the IMS/VS-DB Stage 1 system definition. These are named "SAMPGnn". Most jobs are re-executable to allow easy installation of a new release of IMS/VS. All the referenced jobs are distributed with the IMS/VS system. The first five jobs, SAMPI01, SAMPI02, SAMPI05, SAMPI07, and SAMPI08, must be initially punched because they format and reload the distribution libraries.

After the tape libraries are restored, all the sample jobs are contained in IMSVS.PRIMEJOB. The source code for programs, PSBs, DBDs, etc. are available in IMSVS.PRIMESRC.

Note: Unless otherwise stated, all these jobs should complete with a return code of zero for a proper IMS/VS-DB installation.

SAMPI01: PREPARE DISK VOLUME

This job creates a SYSCTLG on the IMSPRM disk volume and constructs an IMSVS CVOL pointer and index structure.

SAMPI02: ALLOCATE DISTRIBUTION LIBRARIES

This job allocates space for the IMS/VS-DB distribution libraries and the Primer function sample libraries.

SAMPI05: RESTORE PTF LIBRARIES

This optional job restores the libraries from a PTF tape, if any. A PTF tape contains updated versions of IMS/VS modules. If a PTF tape is available, it must be restored first.

SAMPI07: RESTORE IMS/VS-DB LIBRARIES

This job restores the libraries from the IMS/VS distribution tape.

SAMPI08: COPY PRIMER FUNCTION SAMPLE SOURCE AND JOBS

This job copies the Primer function sample source and JCL statements from the distribution libraries to their execution libraries.

The reader procedure (PRIME) in Figure 7-2 can be placed in SYS1.PROCLIB, to be used for reading in the sample jobs.

```
//PRIME       PROC  JOB=TEMPNAME,DSN='IMSVS.PRIMEJOB'
//IEFPROC     EXEC  PGM=IEFVMA,
//     PARM='C06003000050011E00011A00'
//IEFRDER DD        DSN=&DSN.(&JOB),DISP=SHR,DCB=BUFNO=1
//IEFPDSI DD        DSN=IMSVS.PROCLIB,DISP=SHR
//        DD        DSN=SYS1.PROCLIB,DISP=SHR
```

Figure 7-2. The PRIME Reader Procedure

The start command to be used with this reader is for example:

```
S  PRIME,JOB=SAMPI15
```

Note: The reader procedure of Figure 7-2 is for OS/VS1 Release 6. You should verify its parameters with the standard reader procedures in your SYS1.PROCLIB.

SAMPI15: ALLOCATE IMS/VS-DB APPLICATION LIBRARIES

This job allocates the IMS/VS-DB application libraries (DBDLIB, PSBLIB, and PGMLIB).

SAMPI17: ALLOCATE IMS/VS-DB SYSTEM LIBRARIES

This job allocates the libraries for the actual IMS/VS-DB system definition (RESLIB, PROCLIB, OBJDSET, and MACLIB.)

SAMPI21: EXECUTE IMS/VS-DB SYSTEM DEFINITION STAGE 1

This is an assembly job which generates the IMS/VS-DB Stage 2 system definition job stream. It needs only the macros in IMSVS.GENLIB. The output it produces can be punched into cards or placed on a direct access volume as a sequential data set or a member of a library. In our sample environment, we will place the generated job stream in the IMSVS.PRIMEJOB library, with a member name of STAGE2.

Note: This assembly requires a large virtual partition when using the OS/VS system assembler. 2M bytes should be sufficient.

SAMPG1 THROUGH SAMPG6: STAGE 2 JOBS

These jobs perform the actual IMS/VS-DB system definition. They must be executed in numerical sequence.

Notes:

1. These jobs are not listed in Chapter 2 of the

IMS/VS Primer Source Listings. They are created as one member (STAGE2) as a result of job SAMPI21.

2. Jobs SAMPG2, SAMPG3, and SAMPG5 need the OS/VS1 system generation macro library SYS1.AMODGEN. This library should have a blocksize not larger than SYS1.MACLIB, because it is concatenated in the assembler SYSLIB DD statement. It must be cataloged.

3. Control blocks and source modules processed during the execution of the Stage 2 job stream are assembled and link-edited into IMSVS.OBJDSET. Because these modules are link-edited individually, many will produce occurrences of the linkage editor message IEW046I (unresolved external reference) and set condition code 4. These references are resolved later during the linkage editor steps that create the load modules in IMSVS.RESLIB.

4. Job SAMPG6 can have a return code of 4 and message IEW046I will be issued for modules DFSIWAIT and DFSIOS40 when link-editing module DFSVC000 into IMSVS.RESLIB. This is a valid condition.

5. The Stage 2 jobs can be read in with the PRIME reader procedure of Figure 7-2.

SAMPI25: RELINK OS/VS NUCLEUS WITH IMS SVC

This job relinks the OS/VS nucleus to include the IMS/VS Type 2 SVC placed into IMSVS.RESLIB during Stage 2. Your OS/VS system programmer should check the linkage edit control cards to ensure that they comply with other installation requirements. Note that this job relinks the nucleus under the name IEANUlog. When you re-IPL the system after completing all the installation steps, you must specify this suffix (9) to load the alternate nucleus.

SAMPI35: RENAME THE IMS OS/VS NUCLEUS TO MAIN NUCLEUS

After you have re-IPLed your system with the alternate nucleus linked in SAMPI25, and tested it, you may wish to rename this nucleus to IEANUC01. This job will perform the rename, after saving your original nucleus under the name IEANUC0F.

You should now turn to the section "EXECUTING THE SAMPLE."

INSTALLING IMS/VS DB/DC

After the initial OS/VS preparation the following steps should be performed.

CREATING THE IMS/VS LIBRARIES

Operation of IMS/VS requires four categories of libraries for storing modules, programs, and control blocks.

The IMS/VS Distribution Libraries

Two distribution tapes are supplied to install IMS/VS DB/DC. The DB tape contains the following libraries:

- IMS.DBGENLIB contains the macros for the generation and execution of an IMS/VS-DB system.
- IMS.DBLOAD contains the IMS/VS-DB load modules.
- IMS.DBSOURCE contains the source code of the IMS/VS-DB modules.

The DC tape contains the following libraries:

- IMS.DCGENLIB contains the macros for the generation and execution of IMS/VS-DC.
- IMS.DCLOAD contains the IMS/VS-DC load modules.
- IMS.DCSOURCE contains the source code of the IMS/VS-DC modules.

You should pre-allocate space for the following four IMS/VS distribution libraries:

- IMSVS.GENLIB to contain DBGENLIB and DCGENLIB.
- IMSVS.LOAD to contain DBLOAD and DCLOAD.
- IMSVS.DBSOURCE to contain DBSOURCE.
- IMSVS.DCSOURCE to contain DCSOURCE.

The OS/VS utility IEBCOPY is then used to restore and merge the tapes.

Note: The above distribution libraries are needed only during IMS/VS system definition and subsequent system maintenance. They are not used during normal IMS/VS application processing.

The IMS/VS Sample Libraries

Two sample libraries are created after the reload of the distribution tape(s):

- IMSVS.PRIMESRC, contains all the sample sources used in this publication.
- IMSVS.PRIMEJOB, contains all the sample jobs used in this publication.

The IMS/VS System Libraries

For the execution of IMS/VS, the following system libraries are needed:

- IMSVS.MACLIB contains the macros for the generation of DBDs and PSBs.
- IMSVS.RESLIB contains the IMS/VS execution modules.
- IMSVS.PROCLIB contains the IMS/VS job control procedures.
- IMSVS.MATRIX contains the security tables and matrices.
- IMSVS.JOBS contains the JCL used to initiate the MPP region.

The MACLIB, RESLIB, MATRIX, and PROCLIB are established during IMS/VS generation. You should modify the JCL in the IMSVS.PROCLIB member named IMSMSG as necessary to meet the requirements of your installation and store it in IMSVS.JOBS.

The IMS/VS Application Libraries

The following application libraries are needed for the execution of IMS/VS application programs:

- IMSVS.DBDLIB contains the DBDs.
- IMSVS.PSBLIB contains the PSBs.
- IMSVS.PGMLIB contains the application programs.
- IMSVS.ACBLIB contains the ACBs.
- IMSVS.FORMAT contains the MFS control blocks.
- IMSVS.REFERAL contains intermediate text copies of MFS control blocks.

The DBDs and PSBs are stored as standard OS/VS load modules during DBD and PSB generation, respectively. The application programs are stored in the PGMLIB during link-editing. ACBLIB is built when the block builder utility invoked by the ACBGEN procedure is used to build application control blocks from your PSBs and DBDs. FORMAT and REFERAL are established by the MFS utility.

The IMS/VS Online Data Sets

The following data sets are essential for the execution of the online system:

- IMSVS.QBLKS contains message queue control blocks.
- IMSVS.SHMSG contains the short message queue.
- IMSVS.LGMSG contains the long message queue.
- IMSVS.RDS contains control records for IMS/VS restart.
- IMSVS.DBLLOG contains work records used for dynamic backout and restart.

QBLKS, SHMSG, LGMSG, RDS and DBLLOG are built and maintained by the online control program.

RESTORING THE IMS/VS DISTRIBUTION LIBRARIES

Each IMS/VS distribution tape contains three libraries which are unloaded partitioned data sets in IEBCOPY format.

Notes:

1. For the exact format you should check the distribution letter which accompanies the tapes.
2. Optionally, you will also receive two "PTF tapes." These tapes contain updated members of the original libraries. These tapes should be restored first and merged with the original tapes.

IMS/VS DB/DC STAGE 1 DEFINITION

IMS/VS system definition macros are used to describe the features and options you require for your system. These macros are provided in IMSVS.GENLIB.

The coding conventions for these macros are the same as for the OS/VS Assembler language. Processing of these macros by the OS/VS Assembler or the OS/VS program product Assembler H creates the job stream for the execution of Stage 2.

The macros can be divided into three categories:

- System Environment: To describe the basic control program options.
- Data Base and Application: To describe your online data bases, application programs, and transactions.
- Data Communication: To describe your data communications configuration.

We will now discuss these macros by category. Full details of coding the individual macros are given later in the chapter.

System Environment Macro Statements

This category of macro statements describes the basic IMS/VS control program options. You use them to describe such things as:

- Library and message queue data sets.
- Message processing region information.
- Number and size of message queue buffers.
- Sizes of various buffer pools and work areas.
- IMS/VS interface with the Operating System, for example SVC number.
- JOB and SYSOUT classes of the Stage 2 job stream.

The macros included in this category are:

IMSCTRL Defines the basic control program options.

MSGQUEUE Defines the characteristics of the three user message queue data sets.

SPAREA Defines the maximum number and size of scratchpad areas for conversational transactions to be maintained by IMS/VS.

BUFPOOLS Defines the main storage buffer pool sizes for use by the online control region.

IMSCTF Defines additional options and system parameters.

IMSGEN Defines the desired assembler and linkage editor data sets and output options.

Data Base and Application Macro Statements

The macro statements in this category describe the data bases, application programs, and transactions that will be processed by your online system.

The macros included in this category are:

DATABASE Defines a data base that is to be used in the online system.

APPLCTN Names the PSB of the application program that processes the transaction codes specified by the TRANSACT macros that follow.

TRANSACT Names the transaction codes that are to be processed by the application program described in the preceding APPLCTN macro.

You must specify the DBD name of each data base (except logical DBDs) to be used by the online system with the DATABASE macro statement.

One APPLCTN macro statement should be provided for each message processing program (MPP) that is to be used in the online system. The APPLCTN statement is followed by one or more TRANSACT macro statements defining the transactions processed by the MPP. You should also provide an APPLCTN statement for each batch program that is to run as a batch messsage processing program (BMP). In our subset, APPLCTN macros that define BMPs are not followed by TRANSACT macros.

Note: If a data base or batch program will never be used by the online system, it is not necessary to define it in the Stage 1 definition. However, you may include DATABASE, APPLCTN, and TRANSACT macro statements for resources that are not currently available. For example, you may define the statements for an application that is not yet implemented. This will mean that programs for the new application can be run online as soon as they are ready, without having to re-do the system definition. Warning messages for these 'dummy' definitions will be issued when the online control region is initiated.

Data Communications Macro Statements

These macro statements describe your IMS/VS data communication facilities. They define such things as:

- General communications requirements
- Communication line groups, lines, and terminals (BTAM only)
- VTAM nodes (VTAM only)
- IMS/VS logical terminals associated with the nodes and terminals

Macros included in this category are -- VTAM only:

COMM — Specifies general communications requirements that are not associated with any particular terminal type. Prepare one and only one per IMS/VS system definition.

TYPE — Describes a group of VTAM nodes of the same terminal type. Prepare one for each terminal type that is part of your IMS/VS system.

TERMINAL — Provides the characteristics of a terminal node of the type specified in the preceding TYPE statement. Prepare one for each terminal node.

NAME — Provides logical terminal names of the node specified by the preceding TERMINAL statement. Prepare at least one for each terminal node.

Macros included in this category are -- BTAM only:

COMM — Specifies general system attributes that are not associated with any particular terminal.

LINEGRP — Defines a group of lines of the same type over which the same type of terminal communicates.

LINE — Provides the address and/or characteristics of one line in the line group defined by the preceding LINEGRP statement.

CTLUNIT — Provides terminal control unit address and attributes. Prepare one for each control unit attached to the line specified by the preceding LINE statement.

TERMINAL — Provides physical terminal data. Prepare one for each physical terminal attached to the line specified by the preceding LINE statement.

NAME — Provides a logical terminal name for the physical terminal specified by the preceding TERMINAL statement. Prepare at least one for each physical terminal.

Resource Naming Rules

These rules and restrictions apply to all of the IMS/VS macro statements. Refer to them while naming your resources in each of the macro definitions.

- Names cannot include a blank, comma, period, hyphen, or equal sign.
- All PSB names must begin with an alphabetic character (A thru Z, #,$ and @.)
- Logical terminal names and transaction codes must begin with an alphameric character (A thru Z, #, $, and @, or 0 thru 9).
- IMS/VS null words cannot be used as a resource name: FOR, TO, ON, AFTER, SECURITY and MODE. Also, resource names should not begin with DFS or IWTOR.
- Each IMS/VS macro statement can appear in a system definition a limited number of times. Figure 7-3 shows for our subset, the maximum number of times for each statement.
- IMS/VS command keywords and their synonyms cannot be used as resource names. Figure 7-4 gives a list of command keywords and synonyms.

MACRO STATEMENT	MAXIMUM OCCURRENCES
IMSCTRL	1
IMSCTF	1
SPAREA	1
MSGQUEUE	1
BUFPOOLS	1
DATABASE	5000
APPLCTN	5000
TRANSACT	5000
COMM	1
LINEGRP	255
LINE	1000
CTLUNIT	1000
TYPE	5000
TERMINAL	5000
NAME	5000
IMSGEN	1

Figure 7-3. Number of macro statements per system definition.

Keyword	Synonym
ABDUMP	
ACTIVE	A
ALL	
AREA	
ASSIGNMENT	ASMT
BALGRP	BALG
BUILDQ	BLDQS,BLDQ,BUILDQS
CANCEL	
CHECKPOINT	CHKPT,CHECKPT,CHKPOINT
CLASS	CLS
CNS	
COMP	
COMPONENT	COMPT
CONVERSATION	CONV
CPRI	
DATABASE	DATABASES,DB,DBS
DBD	
DC	
DONE	
DUMPQ	DUMPQS
FORMAT	FMT
FPPROG	
FPREGION	FPRGN
FREEZE	
ICOMPT	
INPUT	
KEY	
LEVEL	
LINE	LINES
LINK	
LMCT	LCT
LOPEN	
LPRI	
LTERM	LTERMS
MODE	
MODULE	
MONITOR	MON
MSDBLOAD	
MSNAME	
MSPLINK	
NOBMP	
NOCOMP	
NODE	
NOFEOV	
NOPASSWORD	NOPSWD
NOSHUT	NOS
NOTERMINAL	NOTERM,NOTER
NOTRANCMDS	
NOTRDY	
NOUSER	
NPRI	
OPTION	
OUTPUT	
PARLIM	
PASSWORD	PASSWORDS,PSWD,PSWDS
PCH	
PDS	
PI	
PLMCT	PLCT

Figure 7-4 (Part 1 of 2). IMS/VS Command Keywords and Their Synonyms

Keyword	Synonym
POOL	
PRIORITY	PRTY
PROGRAM	PROGRAMS,PROG,PROGS,PGM,PGMS
PRT	
PSB	
PTERM	PTERMS
PURGE	
QUEUE	QUEUES,Q,QS
RDR	
READY	
REGION	REGIONS,REG,REGS,MSGREG,MSGREGS,MSGREGION, MSGREGIONS
RTCODE	RTC
SEGNO	
SEGSIZE	
SERIAL	SER,SERS,SERIALS
SET	
SHUTDOWN	
SNAPQ	
STATUS	
SYSID	
TDS	
TERMINAL	TERMINALS,TERM,TERMS,TER,TERS
TRANAUTH	
TRANCMDS	
TRANSACTION	TRANS,TRAN,TRANSACTIONS,TRANCODE,TRANCODES
UDS	
USER	
UVDL	
VID	
XKEY	

Figure 7-4 (Part 2 of 2). IMS/VS Command Keywords and Their Synonyms

Coding the IMS/VS System Definition Macros

This section contains the detailed coding instructions for the individual macros.
The macros in the system environment category are discussed first, followed by the data base and application macros, followed by the data communication macros.

IMSCTRL Macro

This statement describes the basic IMS/VS control program options, the OS/VS environment, and the type of IMS/VS system definition to be performed.

```
| IMSCTRL   | SYSTEM=({VS1V},{  ALL   },{6.0})
|           |         {VS/2} {NUCLEUS} {3.7}
|           |
|           |  ,MAXREGN=(2,128K,A,A)
|           |
|           | [,IMSID=imsid]
|           |
|           | [,MCS=(number[,number,...])]
|           |
|           | [,DESC=number]
```

Operands:

SYSTEM= Specifies the OS/VS system and release, and the type of IMS/VS system to be generated. VS1V,6.0 specifies OS/VS1 Release 6.0, VS/2,3.7 specifies OS/VS2 Release 3.7.

'ALL' should be specified for the first system definition that you do. This will generate a system definition for a complete DB/DC system.

If you subsequently wish to change the specifications of your system, for example by, adding data bases, application programs, transactions, or terminals, you should code the sub-parameter 'NUCLEUS'. This will generate a system definition for a new control program nucleus and control blocks.

MAXREGN= Specifies the maximum number of partitions supported by the IMS/VS online control program at any one time. The first sub-parameter indicates the total number of MPP and BMP regions that can be active. The second, third, and fourth sub-parameters specify region size, job class, and job message class, and are used for generating the JCL for the message region. In our subset the parameters should be coded as shown to comply with our recommendation for one MPP region and one BMP region.

IMSID= May specify a 1- to 4-character alphameric identifier for the IMS/VS system. This identifier will be used as the IMS/VS subsystem identifier; it must not conflict with any sub-system identifiers defined to the VS1 or MVS system, including other IMS/VS systems, batch or online. The default is IMSA. This identifier is also used to relate messages which are routed to the OS/VS system console to the corresponding IMS/VS system.

MCS= Specifies the VS routing code to be assigned to the IMS/VS system console if multiple console support (MCS) is included in the operating system. If MCS is not specified, no routing code is used. For a list of valid routing and descriptor codes see OS/VS Supervisor Services and Macro Instructions, GC27-6979.

DESC= Specifies the message descriptor code to be assigned to the IMS/VS system console messages if MCS support is included in the OS/VS generations. If DESC is not specified, no descriptor is assigned.

The MCS= and DESC= keywords should be defined as required for the ROUTCDE and DESC keywords of the OS/VS WTO macro. See OS/VS Supervisor Services and Macro Instructions, GC27-6979, for a detailed description of the WTO macro keyword parameters.

IMSCTF Macro

This statement defines certain control program options and system parameters.

```
 /---------------------------------------------------------------
/        |          |                                           |
|        |IMSCTF    | SVCNO=(,type2)                            |
|        |          |                                           |
|        |          | ,CORE=(2,16,2,2)                          |
|        |          |                                           |
|        |          |          ( 3330,2048 )                    |
|        |          | ,DYLOG=( { 3340,1540 } ,4)                |
|        |          |          ( 3350,2048 )                    |
|        |          |                                           |
|        |          |        ( 3330 )                           |
|        |          | ,RDS=( { 3340 } ,2048,2)                  |
|        |          |        ( 3350 )                           |
|        |          |                                           |
|        |          | ,LOG=(SNGL,MONITOR)                       |
|        |          |                                           |
----------------------------------------------------------------
```

Optional operands:

SVCNO= Specifies the operating system SVC number reserved for use by the generated IMS/VS system. Values entered may range from 128 to 255. The SVC number must be specified as the second parameter to be compatible with earlier releases of IMS/VS; therefore, the parentheses and the comma are required. Default is SVCNO=(,254).

CORE= Specifies the amount of dynamic storage used by the exclusive control ENQUEUE/DEQUEUE routines. In our subset the parameters should be coded as shown. This provides a minumum amount of 2K, which can be incremented by 2K to a maximum of 16K in subpool 2.

Note: You may need to increase these values if you intend to run BMPs with a high number of data base updates between checkpoint calls.

DYLOG= Specifies the device type, buffer size, and number of buffers (4) to be used by the dynamic logging facility. In our subset; if a 3350 device is used, code (3350,2048,4); if a 3340 device is used, code (3340,1540,4), if a 3330 device is used, code (3330,2048,4).

RDS= Specifies the device tape, buffer size, and numbers of buffers to be used for the restart data set. In our subset, a buffer size of 2048 and two buffers are recommended.

LOG= Specifies the type of logging to be done. In our subset the parameters should be coded as shown. It provides for one log data set per job (step) and the optional activation of the DC Monitor.

IMSGEN Macro

This specifies the JCL requirements and assembler and linkage editor options for the Stage 2 job stream. It must be the last macro statement in the input deck.

```
 IMSGEN    JCL=( [{IMSGEN }]   [,job accounting]
                 [{jobname}]

                 [,{ IMS            }]  [,{ A           }]
                 [ {programmername  }]  [ {outputclass  }]

                 [,(job miscellaneous) ])

                 [,NODE=(IMSVS, IMSVS, IMSVS]
                 [       node1  node2  node3]

                 [,OBJDSET=   {IMSVS.OBJDSET}]
                 [            {    name     }]

                 [,USERLIB=   {IMSVS.RESLIB}]
                 [            {    name    }]

                 [,ASM= {VS}]
                 [      {H }]
```

JCL=

jobname specifies a maximum of six alphameric characters to be used as the first portion of the generated job names. The last two characters of the job names are the internally generated, sequentially incremented, numeric values representing the relative position of each job in the Stage 2 stream. The default is IMSGEN.

job accounting specifies job accounting data to be placed in the Stage 2 JCL. The length of the accounting data may not exceed 50 bytes.

programmername specifies the programmer name to be placed in the Stage 2 JCL. The default is IMS.

outputclass specifies the output class to be generated for the Stage 2 JCL. The default is A.

job miscellaneous specifies any additional parameters the user may desire to have placed in the Stage 2 JOB statements. Length of this parameter cannot exceed 50 bytes. Recommended: TYPRUN=HOLD.

Note: If job accounting, programmer name or job miscellaneous contains non-alphabetic characters then the parameter should be enclosed in double quotes: ''....''.

NODE=

node1 specifies the node to be assigned to all IMS/VS data set names to be used and generated by IMS/VS system definition. The specified node can consist of from 1 to 8 characters. The first character must be a letter or a national character (@, $, #). The default node generated is IMSVS.

node2 specifies the node to be assigned to the IMS/VS data set names MACLIB, PROCLIB, MATRIX, JOBS, and RESLIB. This node overrides the node1 assignment for these specific data sets.

node3 specifies the node to be assigned to the IMS/VS data set names DBSOURCE, GENLIB, and LOAD. This node overrides the node1 assignments for these specific data sets.

OBJDSET=

specifies the name (maximum of 24 characters) of a cataloged partitioned data set into which assembler object modules are placed during Stage 2 of IMS/VS system definition. The default is IMSVS.OBJDSET.

USERLIB=

specified the name (maximum of 24 characters) of a library for user modules to be included in the system. This is not applicable to our subset. However, to avoid JCL errors in Stage 2, you should specify here the name of your RESLIB if it is not IMSVS.RESLIB.

ASM=

specifies whether the OS/VS Assembler (VS) or OS/VS program product Assembler H (H) JCL is to be produced for the Stage 2 assembly steps. The default is VS.

MSGQUEUE Macro

This defines the characteristics of the three message queue data sets.

```
  /-------------------------------------------------------------------┐
 /         |           |                                              |
|          |           |      (3330), (3330), (3330)                  |
|          |MSGQUEUE   | DSETS=(<3340 >,<3340 >,<3340 >)               |
|          |           |      (3350), (3350), (3350)                  |
|          |           |                                              |
|          |           | ,RECLNG=(250,1500)                           |
|          |           |                                              |
|          |           | ,BUFFERS=(10,1500)                           |
|          |           |                                              |
└---------------------------------------------------------------------┘
```

Operands:

DSETS= Specifies the device types on which the three message queue data sets will reside. (IMSVS.QBLKS, IMSVS.SHMSG, and IMSVS.LGMSG, respectively). The data sets need not all reside on the same device type.

RECLNG= Specifies the logical record lengths for the short and long message queue data sets, respectively. In our subset the operand should be coded as shown. It provides for our longest output message segment of 1388 bytes, control information and spare space.

BUFFERS= Specifies the number of buffers allocated for message queue management, and the block size used by the three message queue data sets. In our subset the operand should be coded as shown.

SPAREA Macro

This macro defines the maximum number and size of the scratchpad areas (SPAs) maintained by the system.

```
  /-------------------------------------------------------------------┐
 /         |           |                                              |
|          | SPAREA    | CORE=(number,1300)                           |
|          |           |                                              |
└---------------------------------------------------------------------┘
```

CORE= Specifies the number and size of the main storage SPAs. The first sub-parameter, 'number', indicates the maximum number of SPAs. This determines the number of conversations that can be active at any one time, and therefore the number of terminal users who can be using conversational transactions at any one time. The second sub-parameter indicates the maximum size of the SPA. In our subset, it should be specified as shown.

BUFPOOLS Macro

This macro specifies the default main storage buffer pool sizes. These sizes can be overridden at execution time via the PARM field of the IMS/VS control region procedure.

```
|          | BUFPOOLS| PSB=12000
|          |         |,PSBW=4000
|          |         |,DMB=8000
|          |         |,DBASE=7000
|          |         |,GENERAL=12000
|          |         |,FORMAT=18000
|          |         |,COMM=4000
|          |         |,FRE=40
```

Operands:

PSB= Specifies the size of the PSB control block pool.

PSBW= Specifies the size of the PSB work area pool.

DMB= Specifies the size of the DMB control block pool.

DBASE= Specifies the size of the common data base buffer pool. This pool supplies buffers for all the data bases used in the IMS/VS control region or partition.

GENERAL= Specifies the size of the general buffer pool used by the IMS/VS control program, for producing system messages in response to communication activity.

FORMAT= Specifies the size of the message format block pool.

COMM= Specifies any space to be added to the value calculated during system definition for the communication line buffer pool.

FRE= Specifies the number of fetch request elements used for loading MFS control blocks into the message format block pool.

Note: In our subset the operands should be coded as shown for the initial installation of your system. However these operands can be changed as your installation increases in size. Chapter 9, "Optimization," discusses how the usage of these pools should be monitored, and gives guidelines for optimizing their sizes. The values shown here will normally be sufficient for the first-time user.

DATABASE Macro

This macro is used to specify the data bases to be used by the online system. One DATABASE macro must be coded for every SHISAM and HDAM data base and two DATABASE macros should be coded for a HIDAM data base, one for the index DBD and one for the HIDAM DBD. One DATABASE macro should be included for each secondary index that refers to any data bases defined in other DATABASE macros. The DATABASE macro should not be used to describe logical DBDs.

```
 /----------------------------------------------------------------,
/          |          |                                            |
|          |DATABASE  | [INDEX,]                                   |
|          |          |                                            |
|          |          |  RESIDENT                                  |
|          |          |                                            |
|          |          | ,DBD=dbdname                               |
|          |          |                                            |
L-----------------------------------------------------------------J
```

Operands:

INDEX Indicates that this is a DATABASE statement for a HIDAM index or a secondary index. It is a positional parameter.

RESIDENT Indicates that the control block created for this DATABASE statement should be made resident at system initialization time. You should select this performance option for all your production data bases.

DBD= Specifies the name of a data base description block (DBD), as created by the DBDGEN utility.

APPLCTN Macro

The APPLCTN macro is used to describe the application programs that are to be run under the control of the online system. One APPLCTN macro must be specifed for each program that is to run in either an MPP or a BMP region.

```
 /----------------------------------------------------------------,
/          |          |                                            |
|          | APPLCTN  | [RESIDENT,]                                |
|          |          |                                            |
|          |          |  PSB=psbname                               |
|          |          |                                            |
|          |          | ,PGMTYPE={  TP  }                          |
|          |          |          {BATCH}                           |
|          |          |                                            |
L-----------------------------------------------------------------J
```

Operands:

RESIDENT Indicates that the PSB associated with this application program is to be made resident at system initialization time. This is a positional parameter, which you should select for all your production programs. The referenced PSB must have been processed by the ACBGEN utility before the program can be used online. If this is not done a warning message will be issued when the online system is initialized.

PSB= Specifies the name of the PSB associated with the application program being described. If PGMTYPE=TP, the PSB name must be the same as the program name.

PGMTYPE= Specifies the application program characteristics. TP indicates that the program being described is to run in an MPP region, and will be scheduled by the online system when there are messages which it can process. BATCH indicates that the program will run in a BMP region, and will be scheduled by the operator.

Note: Although the APPLCTN macro describes a program, the program name itself is never explicitly defined. This is because it can be determined from the PSB name if PGMTYPE=TP, or from the PARM field in the JCL for the BMP if PGMTYPE=BATCH. (See the IMSBATCH procedure described later in this chapter.)

TRANSACT Macro

The TRANSACT macro is used one or more times with an APPLCTN macro. Each one specifies a transaction code that can be processed by the application program defined in the immediately preceding APPLCTN macro.

```
         | TRANSACT | CODE=transaction code
         |          |
         |          |,MSGTYPE=(SNGLSEG,RESPONSE)
         |          |
         |          |,PROCLIM=(5,30)
         |          |
         |          | [,INQUIRY= {      NO          }]
         |          | [          {(YES,NORECOV)     }]
         |          |
         |          |,MODE=SNGL
         |          |
         |          |,SEGSIZE=1388
         |          |
         |          |,SEGNO=10
         |          |
         |          |[,SPA=(1300,CORE,FIXED)]
```

Operands:

CODE= Specifies the 1 to 8 character transaction code (alphameric). The code must begin with a letter or a number. Transaction codes and LTERM names must be unique. (See the NAME macro later in this section.)

MSGTYPE= Specifies number of segments in the input message for this transaction code, and whether it is a response-mode transaction. In our subset the operand should be coded as shown. This indicates that the input message contains only one segment, and that it is a response-mode transaction.

PROCLIM= Specifies the number of messages of this transaction code a program can process in a single scheduling, and the amount of CPU time (in seconds) allowed to process each message. In our subset the operand should be coded as shown. This means that an application program could process up to five messages for the same transaction code and would be allowed 30 seconds of CPU time to process each message. After processing the fifth message, the program would receive a 'QC' status code when it issues a GU to the message queue, even if there were more messages for that transaction code waiting to be processed. This is to prevent one program from occupying the message region for too long a period of time. The second sub-parameter is to ensure that a program that loops will be trapped after it has used 30 seconds of CPU time and cancelled by the control region.

INQUIRY= Specifies whether this is an inquiry transaction or not. If it is an inquiry-only transaction you should specify (YES,NORECOV). This means that the transaction will not be recovered during an emergency restart, that is, it must be re-entered after a restart. It also reduces the number of records written to the log data set. The default is NO, that is, an update transaction.

MODE= Specifies when data base buffers are to be written to direct access. MODE=SNGL means that the buffers will be flushed upon each request by the application program for a new message, and that only the last message processed by a program will be re-processed during emergency restart. In our subset the parameter should always be specified as shown.

SEGSIZE= Specifies the maximum size of an output segment inserted to the message queue by a program. In our subset the parameter should be specified as shown.

SEGNO= Specifies the maximum number of segments a program can insert to the output message queue per input message.

SPA= Indicates whether this transaction is a conversational one, and the size of the scratch pad area. If the transaction is not conversational, this parameter must be omitted.

Coding the Data Communication Statements - VTAM

COMM Statement

The COMM statement is used to specify general communication attributes that are not associated with any particular terminal type. COMM is always required for terminal types supported by VTAM.

```
|      | COMM  | RECANY=(number,size),
|      |       | APPLID=IMS,
|      |       | SECCNT=3,
|      |       | OPTIONS=(FORPSW,FORCTERM,TIMESTAMP,FMTMAST),
|      |       | COPYLOG=ALL
```

Operands:

RECANY= This parameter is required. It defines the VTAM RECEIVE ANY buffers.

number
: specifies the number of VTAM RECEIVE ANY buffers to be present in the IMS/VS system. A value of eight is normally sufficient for an entry installation.

size
: specifies the size of the largest RECEIVE ANY buffer. This size must be large enough to handle the maximum input that may be received from any VTAM-attached terminal. A value of 4000 would generally accommodate a full screen input from a 3278 Model 4 display terminal.

APPLID= Specifies the name through which VTAM identifies the IMS/VS system as a VTAM application program. This parameter should be coded as shown and must be identical to the application name coded on the APPL statement defining IMS/VS within the VTAM application major node.

SECCNT= Specifies the maximum number of terminal and/or password security violations that may occur per physical terminal before the master terminal operator is notified. In our subset the parameter should be coded as shown.

OPTIONS= Specifies certain system options, and in our subset should be coded as shown. This will mean that terminal and password security will always be used by the online system, that any system message whose number lies in the range DFS001 to DFS300 will have the time it was generated inserted in the message, and that IMS/VS-provided MFS support is to be used for the master terminal.

COPYLOG= Specifies hardcopy of all eligible commands and responses on the secondary master terminal. All subset commands are eligible for hardcopy.

TYPE Statement

The TYPE statement is used to define terminals attached to IMS/VS through VTAM. It defines the beginning of a set of one or more communication terminal and logical terminal description statements. All terminals must be of the same type.

```
|      | TYPE | UNITYPE=(3270[,LOCAL]),
|      |      | TYPE=3270-An,SIZE=(ll,cc),
|      |      | FEAT=IGNORE,
|      |      | OPTIONS=TRANRESP,
|      |      | PTRSIZE=IGNORE
```

Operand:

UNITYPE= Specifies the terminal device type contained in this communication description set. In our sample environment, just code either (3270, LOCAL) for the locally or (3270) for the remotely attached 3270 terminal groups.

In our subset, we limit ourselves to the following 3270 control units and their attached display/printers.

- 3271 Model 1, 2, 11, or 12
- 3272 Model 1 or 2
- 3274 Model 1B or 1C (BSC line protocol only)
- 3275 Model 1 or 2
- 3276 Model 1, 2, 3, or 4 (BSC line protocol only)

TYPE= Specifies the display screen size type in our subset. It should correspond with the SIZE= parameter as defined in the following table:

Screen Size	TYPE=	SIZE=
12x80	3270-A1	(12,80)
24x80	3270-A2	(24,80)
32x80	3270-A3	(32,80)
43x80	3270-A4	(43,80)
12x40	3270-A6	(12,40)
6x40	3270-A5	(6,40)

SIZE= Specifies the display screen size. See the preceding TYPE= parameter discussion.

FEAT= Specify as shown in our subset. It causes IMS/VS to ignore any special features of the display terminals.

OPTIONS= Specify as shown in our subset. It causes IMS/VS to place the terminal in response mode whenever the transaction is defined as such.

PTRSIZE= Specify as shown in our subset. It allows IMS/VS to be independent of printer terminal printline width.

Note: If UNITYPE=3270, the sequence of the TERMINAL statements defines which printer terminal will be used for a remote copy operation requested at a remote display terminal. When a copy function is requested, IMS/VS selects a printer terminal only from the set defined immediately after the definition of that display terminal. The printer selection process starts with the first subsequent printer terminal (must be a remote), and stops at the next non-printer terminal.

The printer selected must be ready, and not busy.

TERMINAL Statement

This statement defines terminal node characteristics. The NAME statements that follow a TERMINAL statement supply the logical terminal names that are associated with the node at system definition time.

	TERMINAL	NAME=nodename [,UNIT=3284] $\left[,\text{MODEL=}\begin{Bmatrix}1\\2\end{Bmatrix}\right]$

Operands:

NAME= The specified name must be the VTAM node name defined during VTAM/NCP generation.

It is through this name that IMS/VS addresses the VTAM node defined within VTAM/NCP. The name must be one of the node names defined on a LOCAL, TERMINAL or LU statement within the

VTAM 3270 local major node, the NCP major node BSC group and SDLC group definition, respectively.

UNIT= Specify for 3270 printer terminals only in our subset. May be specified as shown for any 3270 printer terminal type (that is, 3284, 3286, 3287, 3288, and 3289).

MODEL= Specify for 3270 printer terminals only in our subset. MODEL=1 applies only to the 3284/3286 printers. All other printers should be specified with MODEL=2.

NAME Statement

This statement defines a logical terminal name (LTERM) associated with a node. The presence of the keyword MASTER in the LTERM operand designates this logical terminal name as the primary master terminal. In our subset this must be a display terminal with a screen size of 1920 characters. The presence of the keyword SECONDARY in the LTERM operand designates this logical terminal name as the secondary master terminal. In our subset a secondary master terminal must always be specified and it must be a 3270 printer.

	NAME	{ lterm-name (MTC,MASTER) (MTOPRINT,SECONDARY) }

Operands:

lterm-name Specifies a 1- to 8-character name of a logical terminal to be associated with the previously defined physical terminal. WTOR is an invalid name because this is the default LTERM-name for the system console. In our subset, the names MTC and MTOPRINT are assumed for the primary and secondary master terminals, respectively, and should be coded as shown on the NAME statements following the TERMINAL statements for these terminals.

Note: A naming convention should be established in your installation for the names of logical terminals. For example it may be useful if a name indicates the department or person normally using that terminal. If printer terminals are used, there should be some way of determining in the MPP, the printer terminal that is associated with a group of display terminals. In our example the first character 'L' designates an LTERM name, the next four indicate the department name, the sixth character indicates whether it is a display or printer terminal, and the last two are used as a sequence number. Thus a program receiving an input transaction from logical terminal LDEP1D01 can determine that the associated printer in the same department is LDEP1P01, and can alter the destination of an alternate PCB to that name with a CHNG call.

Coding the Data Communication Statements -- BTAM

COMM Macro

This macro is used to specify general communication requirements that are not associated with any particular terminal type. In our subset it is used to specify additional system options.

```
  /---------------------------------------------------------------,
 /          |          |                                            |
|           | COMM     |SECCNT=3,                                   |
|           |          |OPTIONS=(FORPSW,FORCTERM,TIMESTAMP,FMTMAST),|
|           |          |COPYLOG=ALL                                 |
|           |          |                                            |
L-----------------------------------------------------------------J
```

Operands:

SECCNT= Specifies the maximum number of terminal and/or password security violations per physical terminal before the master terminal operator is notified. In our subset the parameter should be coded as shown.

OPTIONS= Specifies certain system options, and in our subset should be coded as shown. This will mean that terminal and password security will always be used by the online system, that any system message whose number lies in the range DFS001 to DFS300 will have the time it was generated inserted in the message, and that IMS/VS-provided MFS support is to be used for the master terminal.

COPYLOG= Specifies hard copy of all eligible commands and responses on the secondary master terminal. All subset commands are eligible for hardcopy.

LINEGRP Macro

This defines the beginning of a set of macros that describe one or more lines of the same type to which are attached the same type of terminal.

```
  /---------------------------------------------------------------,
 /          |          |                                            |
|           |LINEGRP   |DDNAME=ddname                               |
|           |          |                                            |
|           |          |,UNITYPE=(3270[,LOCAL])                     |
|           |          |                                            |
L-----------------------------------------------------------------J
```

Operands:

DDNAME= Specifies a 1 to 8 character name that associates the generated DCB for this line group with the DD statement generated by the Stage 1 system definition in the JCL for the control region. The name must begin with an alphabetic character. The following names cannot be used as LINEGRP ddnames: DFSRESLIB, DUMP, IEFRDER, IEFRDER2, IMSACB, IMSDBL, IMSDILIB, IMSLOG, IMSLOGR, IMSLOGR2, IMSLOG2, IMSMON, IMSRDS, IMSSPA, IMSTFMT, LGMSG, MSDBDUMP, MSDBINIT, MSDBCP1, MSDBCP2, PRINTDD, PROCLIB, MATRIX, JOBS, QBLKS, SHMSG and SYSUDUMP.

UNITYPE= Specifies the terminal device type attached to the lines in this line group. In our subset only the terminal types shown are considered. If UNITYPE=(3270,LOCAL) is coded, only one LINE macro can be in the line group. UNITYPE=3270 indicates remote 3270 terminals, and more than one LINE statement can be included in the line group. In our subset, we limit ourselves to the following 3270 control units and their attached display/printers.

- 3271 Model 1, 2, 11, or 12
- 3272 Model 1 or 2
- 3274 Model 1B or 1C (BSC line protocol only)
- 3275 Model 1 or 2
- 3276 Model 1, 2, 3, or 4 (BSC line protocol only)

LINE Macro

This macro describes the communication line itself. Each LINE macro must be followed by at least one TERMINAL macro. Only one LINE statement per line group is allowed if UNITYPE=(3270,LOCAL) was specified on the LINEGRP macro statement. If there is more than one terminal attached to the line, there should be multiple TERMINAL macros following the LINE macro.

```
|          | LINE     | [ADDR= cuu]
|          |          |
|          |          | [BUFSIZE=384]
```

Operands:

ADDR= Specifies the address of the communication line as defined in the transmission control unit. It must be of the form 'cuu'. It is used only to generate the DD statements in the procedure for the online control region that is generated by the Stage 1 definition. It must not be coded if UNITYPE=(3270,LOCAL) was specified on the LINEGRP macro for this LINE.

BUFSIZE Specifies the maximum size of an input message on this line. It is only required if this macro is defining a line containing local 3270 display terminals. In our subset the basic recommendation is 384.

CTLUNIT Macro

This macro specifies the 3271 remote control unit characteristics. This statement must not precede any 3275 terminal definition on that line.

```
/---------------------------------------------------------------------
/          |           |
|          | CTLUNIT   |  ADDR=hex byte
|          |           |
|          |           |  ,MODEL= {1}
|          |           |          {2}
|          |           |
----------------------------------------------------------------------
```

Operands:

ADDR= Specifies the two-digit hexadecimal polling address of the 3271, the 3274 Model 1C, or the 3276 Model 1, 2, 3, or 4. The address of the control unit is assigned by the IBM customer engineer upon installation. Note that the IBM customer engineer assigns the selection address which must be converted to the polling address for specification in this macro.

MODEL= Specifies the control unit model number for a 3271. For a 3274 or 3276, you must specify MODEL=2.

TERMINAL Macro

This macro defines physical and logical terminal characteristics. The NAME macro statements that follow a TERMINAL macro statement supply the logical terminal names that are associated with the physical terminal at system definition.

```
/---------------------------------------------------------------------
/          |           |
|          |           |        {cuu }
|          | TERMINAL  |  ADDR= { xx  }
|          |           |        {xxxx}
|          |           |
|          |           |  [,TYPE=3270-An,SIZE=(ll,cc)]
|          |           |
|          |           |  [,FEAT=IGNORE]
|          |           |
|          |           |  [,OPTIONS=TRANRESP]
|          |           |
|          |           |
|          |           |  [,MSGDEL=SYSINFO]
|          |           |
|          |           |
|          |           |  [,UNIT=3284]
|          |           |
|          |           |  [,PTRSIZE=IGNORE]
|          |           |
|          |           |
|          |           |  [,MODEL= {1}]
|          |           |  [        {2}]
|          |           |
----------------------------------------------------------------------
```

Operands:

ADDR= Specifies the physical terminal address.

For 3270 local terminals this address is used when generating the UNIT= parameter on the DD card in the JCL for the control region. It must be of the form 'cuu'.

Except for a 3275, the address must be specified as two hexadecimal digits specifying the terminal address of that terminal on its control unit.

For a 3275 the address must be specified as four hexadecimal digits. The first two specify the control unit polling address, while the last two specify the terminal address (for example, ADDR=4040).

Note that the IBM customer engineer assigns the selection address when installing the 3275, and this address must be converted to a polling addresss for specification in this macro.

Note: If 3275's are intermixed on the same line as 3270 control units, their TERMINAL definition must preceed the CTLUNIT statements within the same LINE.

TYPE= Specifies the display screen size type in our subset. It should correspond with the SIZE= parameter as defined in the following table:

Screen Size	TYPE=	SIZE=
12x80	3270-A1	(12,80)
24x80	3270-A2	(24,80)
32x80	3270-A3	(32,80)
43x80	3270-A4	(43,80)
12x40	3270-A6	(12,40)
6x40	3270-A5	(6,40)

This parameter may be specified for display terminals only.

SIZE= Specifies the display screen size. See the preceding TYPE= parameter discussion. May be specified for display terminals only.

FEAT= Specify as shown in our subset. It causes IMS/VS to ignore any special features of the display terminals. May be specified for display terminals only.

OPTIONS= Specify as shown in our subset. It causes IMS/VS to place the terminal in response mode whenever the transaction is defined as such. May be specified for display terminals only.

MSGDEL= Specifies which message types IMS/VS should discard for this terminal. In our subset this parameter should be coded as shown for printer terminals, and should be omitted for display terminals. When coded as shown, this will specify that DFS059 TERMINAL STARTED messages will not be sent to this terminal.

UNIT= Specify only for 3270 printer terminals in our subset. May be specified as shown for any 3270 printer terminal type (that is, 3284, 3286, 3287, 3288, and 3289).

MODEL= Specify only for 3270 printer terminals in our subset. MODEL=1 applies only to the 3284/3286 printers. All other printers should be specified with MODEL=2.

PTRSIZE= Specify as shown in our subset. It allows IMS/VS to be independent of printer terminal printline width. May be specified for printer terminals only.

Note: TERMINAL statements for local printer terminals may not be within the same LINE statement as local display terminals, that is, they must have their own LINEGRP and LINE statement.

NAME Macro

This macro defines a logical terminal name (LTERM) associated with a physical terminal. The presence of the keyword MASTER in the LTERM operand designates this logical terminal name as the primary master terminal. In our subset this must be a 3270 display terminal with a screen size of 1920 characters. The presence of the keyword SECONDARY in the LTERM operand designates this logical terminal name as the secondary master terminal. In our subset a secondary master terminal must always be specified and it must be a 3270 printer.

```
/--------------------------------------------------------------------.
/         |            |                                              |
|         | NAME       | ( lterm-name                )                |
|         |            | < (MTO,MASTER)              >                |
|         |            | ((MTOPRINT,SECONDARY)       )                |
|         |            |                                              |
L---------------------------------------------------------------------J
```

Operands:

lterm-name — Specifies a 1- to 8-character name of a logical terminal to be associated with the previously defined physical terminal. WTOR is an invalid name because this is the default LTERM-name for the system console. In our subset, the names MTO and MTOPRINT are assumed for the primary and secondary master terminals, respectively, and should be coded as shown on the NAME statements following the TERMINAL statements for these terminals.

Note: A naming convention should be established in your installation for the names of logical terminals. For example, it may be useful if the name indicates the department or person normally using that terminal. If printer terminals are used, there should be some way of determining in the MPP, the printer terminal that is associated with a group of display terminals. In our example the first character 'L' designates an LTERM name, the next four indicate the department name, the sixth character indicates whether it is a display or printer terminal, and the last two are used as a sequence number. Thus a program receiving an input transaction from logical terminal LDEP1D01 can determine that the associated printer in the same department is LDEP1P01, and can alter the destination of an alternate PCB to that name with a CHNG call.

STRUCTURE OF THE STAGE 1 INPUT DECK

The IMS/VS Stage 1 system definition is a standard OS/VS assembly. The generated output is the Stage 2 job stream.

Example:

```
//STAGE1  JOB   (ACCTING)
//        EXEC assembler-procname,PARM=DECK
//SYSLIB  DD    DSN=IMSVS.GENLIB,DISP=SHR
//SYSIN   DD    *
          IMSCTRL
          .
          .
          Stage 1 macro statements
          .
          IMSGEN
          END
/*
```

The sequence of macro statements in the input deck is as follows:

- System Environment Macros — IMSCTRL must be the first, and the others may follow in any sequence, except IMSGEN must be at the end of the input-deck.
- Data Base and Application Macros — As many sets of DATABASE and APPLCTN/TRANSACT statements as required.
- Data Communications Macros — The COMM macro must be the first of this category, followed by sets of LINEGRP, LINE, CTLUNIT, TERMINAL, and NAME statements if BTAM, or TYPE, TERMINAL, and NAME statements if VTAM.
- System Environment Macro — The IMSGEN statement must be the last macro in the deck. It should be followed by an Assembler END statement.

Jobs //SAMPI22 and //SAMPI23 in IMSVS.PRIMEJOB show examples of IMS/VS Stage 1 input decks for BTAM and VTAM systems, respectively.

IMS/VS STAGE 2 SYSTEM DEFINITION

This is the execution of the jobs generated by the Stage 1 system definition.

OS/VS1 FINAL PREPARATION

Some changes must be incorporated in your OS/VS1 system after the IMS/VS Stage 2 system definition.

Copy IMSRDR and IMS Procedures to SYS1.PROCLIB

To be able to use the IMS/VS-supplied procedures in IMSVS.PROCLIB, the IMSRDR procedure should be copied from IMSVS.PROCLIB to SYS1.PROCLIB. The IMS procedure should also be copied to SYS1.PROCLIB from IMSVS.PROCLIB so that the online control program can be started from the system console as a system task.

Relink the OS/VS Nucleus

The OS/VS nucleus must be re-linked to include the Type 2 SVC module that was placed in IMSVS.RESLIB by Stage 2.

Customize IMS Control Region Procedure

In order to be able to access your data bases via the online system you must include DD cards for them in the IMS procedure. If you are using VSAM data bases, you should also define which VSAM buffer pool specification and fixlist members in IMSVS.PROCLIB you wish to use. This is done by specifiying the VSPEC and FIX parameters on the EXEC statement.

Update DFSVSM00 Member in IMSVS.PROCLIB

To use LTWA and VSAM data bases online, you must update member DFSVSM00 in IMSVS.PROCLIB.

Create DFSFIX00 Member in IMSVS.PROCLIB

To assure a stable response time in an entry IMS/VS environment you should fix in real storage some of the IMS/VS control blocks, buffer pools, and nucleus. In our subset we include a basic recommendation for this fixlist in our sample job stream. This is done by adding the DFSFIX00 member to IMSVS.PROCLIB.

Update Initial System Security Tables

Before you can start up the IMS/VS CTL region for the first time, you must update the initial system security tables. See the IMS/VS Security Maintenance Utility, later in this chapter.

Update IMSMSG Procedure

Modify the JCL in the IMSMSG member of IMSVS.PROCLIB to meet your requirements and store it in IMSVS.JOBS. If you intend to use our sample status code error handling routine (DFSOAER) you must include the DD card for the error listing in the IMSMSG procedure.

PL/I Optimizer Considerations

Special care in organizing the PL/I modules will help to decrease response time for those IMS/VS MPPs which use the PL/I Optimizer. Some organization suggestions follow:

- Use several different program libraries, one for each region, putting only those modules required by the application in the library. Include in that library all supporting modules (such as the PL/I transient library modules).
- Concatenate the PL/I library into the message region STEPLIB.
- Put the required supporting modules in the link pack area. This is the recommended long-term solution for a virtual environment.

(Caution: Do not use the PL/I Optimizing Compiler for multi-tasking during link-editing. Do not use SYS1.PLITASK as a SYSLIB data set.)

After the OS/VS1 final preparation has been completed, you must re-IPL the system so that the changes you have made become effective. Note: If you are not using VTAM with IMS/VS, you can skip the following sections on VTAM and NCP/VS installation.

PREPARING VTAM

In the following sections, we will limit ourselves to a brief overview of VTAM Level 2 preparation using sample jobs. This is not intended as a replacement of the OS/VS VTAM System Programmer's Guide. To verify the exact level of VTAM or ACF/VTAM required to support your 3270 configuration, consult the IMS/VS Program Directory which accompanies your IMS/VS distribution tape.

Creating the VTAM Libraries

The operation of VTAM requires the following three libraries:

1. SYS1.VTAMLIB contains VTAM modules, tables and routines. This data set is initially created during OS/VS1 system generation, when ACSMETH=(VTAM) is specified in the DATAMGMT macro instruction.

2. SYS1.VTAMLST contains VTAM definition statements and start options.

3. SYS1.VTAMOBJ contains resource definition table (RDT) segments for each activated major node.

The first time a major node is activated at VTAM initialization or with the VARY command, the related definition statements in SYS1.VTAMLST are processed into SYS1.VTAMOBJ. If a major node is redefined, the member on SYS1.VTAMOBJ must be deleted prior to activating the changed major node. In our subset, we will assure this by always deleting and reallocating SYS1.VTAMOBJ before applying any changes to SYS1.VTAMLST.

Defining VTAM Start Options

Start parameters establish conditions and facilities that are to be effective when VTAM is started. They are entered into SYS1.VTAMLST as members of 80-character card-image records. These members are used by VTAM to determine which major nodes to activate at start-up time, which parameters to use concerning VTAM buffer sizes, etc.

There are five members in our SYS1.VTAMLST:

- ATCSTR00 contains start parameters plus a pointer to ATCCON00

- ATCCON00, a list of major nodes (VTAM application programs, local terminal sets, or NCPs) VTAM must activate when started. Each major node is defined as a member and listed in ATCCON00. Those in our subset are:

  - APPNODES, a list of VTAM application programs
  - LO3270, a list of local 3270s to be controlled by VTAM
  - NCP, the NCP source statements (only if 370X is used).

Note: Member ATCSTR00 specifies, in addition to general configuration parameters, the number, size, and threshold values of the various VTAM main storage pools. These values are based on our subset network. Chapter 9, "Optimization," provides guidelines for the optimization (in general, reduction) of these pools during initial operation. See the section "Monitoring VTAM Pools" in Chapter 9.

Defining IMS/VS to VTAM: At a minimum IMS/VS must be defined to VTAM as an application minor node. Additional applications can be defined together with IMS/VS as one major node called APPNODES in our sample.

Note: The label on the APPL statement for IMS/VS must be the same as that coded in the APPLID= parameter of the COMM macro statement within IMS/VS System Definition. See the section entitled "Coding the Data Communication statements-VTAM" earlier in this chapter.

Defining the Local Network to VTAM: The local 3270 display stations and/or printers are defined as minor nodes of the major node LC3270. Each local display station or printer is defined by a separate LOCAL statement as a minor node. These local minor nodes are then grouped into one local major node, LC3270.

Note: For a given terminal, the label on the LOCAL statement must be the same as the node name coded in the NAME parameter of the TERMINAL statement within IMS/VS Stage 1 system definition. See the section entitled "Coding the Data Communication Statements -- VTAM" earlier in this chapter.

Defining the Remote Network to VTAM: The remote network itself is defined in the network control program (NCP) in the 370X. To allow VTAM access to this definition, the NCP source deck is filed in SYS1.VTAMLST as member "NCP". This member name must be defined as a major node to VTAM.

Creating the VTAM START Cataloged Procedure

VTAM is started by naming a cataloged procedure in the OS/VS START command. The procedure must be filed into SYS1.PROCLIB by using IEBUPDTE. When starting VTAM, the first parameter of the START command must be the assigned procedure name concatenated with .Pnn, where nn specifies the numbered partition in which VTAM is to run, normally F0. For example, if the assigned procedure is NET and VTAM is to run in the partition numbered 00, the start command is: S NET.P0.

Note: Since other OS/VS commands used to operate VTAM, for example, the VARY command, always refer to NET instead of the cataloged procedure name, it is recommended that you name the procedure NET. This will also comply with our sample MTO guide.

GENERATING THE NETWORK CONTROL PROGRAM/VS (NCP/VS)

As for VTAM, we will give only a brief overview and a simple example of the NCP/VS generation. Since an NCP is largely hardware and application dependent, you should refer to the *IBM 3704 and 3705 Control Program Generation and Utilities* Guide and Reference Manual, GC30-3008 for the actual generation of your NCP.

Note: An NCP is not required if only a local network is used.

Overview

A network control program must be generated for each communication controller in the VTAM network. An NCP generated to control one communication controller will not work properly in another unless they and their remote terminal networks are identical. An NCP is defined in the form of a source program consisting entirely of NCP generation macro instructions. The information coded in these macro instructions includes characteristics of the remote terminals and options that affect the functions performed by the NCP. The resulting source program is assembled and link-edited to produce two load modules which constitute the NCP.

The following steps should be performed to create an NCP that is to run with VTAM in our subset environment.

Restoring the NCP Distribution Libraries

Prior to the NCP generation, the NCP support package for OS/VS, 5744-BA2 must be installed. The instructions in the Memorandum to Users describe the installation procedures for the basic material. The important files on the distribution tape for OS/VS1 are:

- An unloaded partitioned data set containing the Assembler, Loader, Dump, Dynamic Dump and Initial Test (SYS1.SSPLIB in our installation)
- An unloaded partitioned data set containing the NCP/VS Stage 1 System Generation macros which are required for NCP generation (SYS1.GEN3705 in our installation)
- An unloaded partitioned data set containing the NCP/VS Stage 2 Generation macros (SYS1.MAC3705 in our installation)
- An unloaded partitioned data set containing the object modules required for Stage II NCP System Generation (SYS1.OBJ3705 in our installation)

See the Memorandum to Users for a complete description of the installation procedure.

Creating the NCP Data Sets

The following libraries are required for the generation and execution of the network control program:

- SYS1.NCPMODS is the NCP load module library. Its name must match the QUALIFY= and LOADLIB= parameters of the NCP BUILD statement.
- SYS1.NCPDUMP will hold NCP dump records.
- SYS1.NCPOBJ will hold the NCP generation Stage 2 output as input to the link-edit of the NCP.

Defining the Remote Network to VTAM

The remote network is that part of the network which is maintained through a 370X. To define the remote network to VTAM, the NCP source deck is used. The information about NCP must be made available to VTAM. This is accomplished by filing the NCP source program in 80-byte card image form as a member of SYS1.VTAMLST.

Note: The name of the member must be the same as coded in the NEWNAME= parameter of the BUILD statement (NCP source deck).

File NCP Source Deck into SYS1.VTAMLST

The NCP source deck, which constitutes the input for Stage 1 of NCP generation, must be made available to VTAM.

The NCP is first stored in SYS1.VTAMLST under the member name "NCP". This same member name, NCP, is defined as a major node to VTAM via member ATCCON00 in SYS1.VTAMLST.

Note: The comments in our sample NCP identify NCP parameters which are likely to be different in your installation, and direct you to the appropriate reference manual.

Stage 1 of NCP Generation

Stage 1 is an assembly job using the communications controller assembler, CWAX00 located in SYS1.SSPLIB. Its output is the Stage 2 job stream.

Stage 2 of NCP Generation

Stage 2 uses the communications controller assembler to assemble the control tables and program modules that require conditional assemblies, and places the resultant object modules on SYS1.NCPOBJ. Stage 2 then link-edits these modules and other preassembled modules (located on SYS1.OBJ3705) into SYS1.NCPMODS. From this library the VTAM-provided loader loads the control program into the 370X.

IMS/VS DB/DC INSTALLATION JOBS

This section presents all the jobs to install IMS/VS DB/DC including VTAM and NCP/VS. A listing of these jobs is provided in Chapter 2 of the *IMS/VS Primer Sample Listings*. All jobs for the installation are named "SAMPInn," except the jobs generated by the NCP/VS and IMS/VS Stage 1 system definition. These are named "SAMPNnn" and "SAMPGnn", respectively. Most jobs are re-executable to allow easy installation of new releases of IMS/VS. These jobs can be executed from the IMSVS.PRIMEJCB library, except the initial jobs: //SAMPI01, //SAMPI03, //SAMPI04, //SAMPI05, //SAMPI06, //SAMPI07, //SAMPI08, and //SAMPI09. The two latter ones create the sample IMSVS.PRIMEJOB and IMSVS.PRIMESRC libraries. The source code for programs, PSBs, DBDs, etc., is available in a library called IMSVS.PRIMESRC, after above mentioned initial jobs.

Notes:

1. Unless otherwise stated, all these jobs should complete with a return code of zero for a proper IMS/VS installation. General exceptions are IEHPROGM and IEBUPDTE jobsteps, and the link editor steps of the PL/I compilations.

2. Jobs SAMPI6n should not be executed if only a local network is to be used.

3. Jobs SAMPI5n and SAMPI6n should not be executed if BTAM is to be used.

SAMPI01: PREPARE DISK VOLUME

This job creates a SYSCTLG on the IMSPRM disk volume and constructs an IMSVS CVOL pointer and index structure.

SAMPI03: ALLOCATE DISTRIBUTION LIBRARIES

This job allocates space for the IMS/VS distribution libraries.

SAMPI04: RESTORE LIBRARIES FROM DC PTF TAPE

This optional job restores the libraries from the DC PTF tape, if any. A PTF tape contains updated versions of IMS/VS modules. If PTF tapes are available, they must be restored first.

SAMPI05: RESTORE/MERGE LIBRARIES FROM DB PTF TAPE

This optional job restores the libraries from the DB PTF tape.

SAMPI06: RESTORE IMS/VS DC DISTRIBUTION LIBRARIES

This job restores the libraries from the IMS/VS DC distribution tape.

SAMPI07: RESTORE/MERGE IMS/VS DB DISTRIBUTION LIBRARIES

This job restores and merges the libraries from the IMS/VS DB distribution tape.

SAMPI08: COPY PRIMER FUNCTION DB SAMPLE SOURCE AND JOBS

This job copies the Primer function DB sample source and JCL statements from the distribution libraries to their execution libraries.

SAMPI09: COPY PRIMER FUNCTION DC SAMPLE SOURCE AND JOBS

This job copies the Primer function DC sample source and JCL statements from the distribution libraries to their execution libraries.

The reader procedure (PRIME), in Figure 7-5, can be placed in SYS1.PROCLIB, to be used for reading in the sample jobs.

```
//PRIME      PROC  JOB=TEMPNAME,DSN='IMSVS.PRIMEJOB'
//IEFPROC    EXEC  PGM=IEFVMA,
//    PARM='00600300005011E00011A00'
//IEFRDER DD       DSN=&DSN.(&JOB),DISP=SHR,DCB=BUFNO=1
//IEFPDSI DD       DSN=IMSVS.PROCLIB,DISP=SHR
//        DD       DSN=SYS1.PROCLIB,DISP=SHR
```

Figure 7-5. The PRIME Reader Procedure

The start command to be used with this reader is for example:

```
S  PRIME,JOB=SAMPI15
```

Note: The reader procedure of Figure 7-5 is for OS/VS1 Release 6. You should verify its parameters with the standard reader procedures in your SYS1.PROCLIB.

SAMPI15: ALLOCATE IMS/VS APPLICATION LIBRARIES-DB

This job allocates the IMS/VS application libraries (DBDLIB, PSBLIB, and PGMLIB). This job should be needed only the first time you install IMS/VS.

SAMPI16: ALLOCATE IMS/VS-DC APPLICATION LIBRARIES

This job allocates the application libraries used by the online IMS/VS control partition/region. This job should be needed only the first time you install the IMS/VS DC feature.

SAMPI17: ALLOCATE IMS/VS SYSTEM LIBRARIES

This job allocates the libraries for the IMS/VS system definition (PROCLIB, MACLIB, RESLIB, and OBJDSET).

SAMPI18: ALLOCATE IMS/VS ONLINE DATA SETS

This job allocates the IMS/VS system data sets used by the online system (QBLKS, SHMSG, LGMSG, RDS, MATRIX, JCBS, and DELLOG).

Warning: If you reallocate the IMS/VS message queues and you decrease the queue data set size, you must subsequently do a cold start of the IMS/VS control region. For a description of how to do a cold start, see Chart F-1 in the IMS/VS Primer Master Terminal Operator's Guide -- VTAM.

SAMPI23: EXECUTE IMS/VS SYSTEM DEFINITION STAGE 1 -- VTAM

This is an assembly job which generates the IMS/VS Stage 2 system definition job stream. It only needs the macros in IMSVS.GENLIB. The output it produces can be punched into cards or placed on a direct access volume as a sequential data set or a member of a library. In our sample environment, we will place the generated job stream in the IMSVS.PRIMEJOB library, with a member name of STAGE2.

Note: This assembly requires a large virtual partition, when using the OS/VS system assembler. 2M bytes should be sufficient.

Figure 7-6 gives an overview of our sample terminal network as specified in the IMS/VS Stage 1 input of SAMPI23 and the associated VTAM and NCP/VS jobs of the following sections.

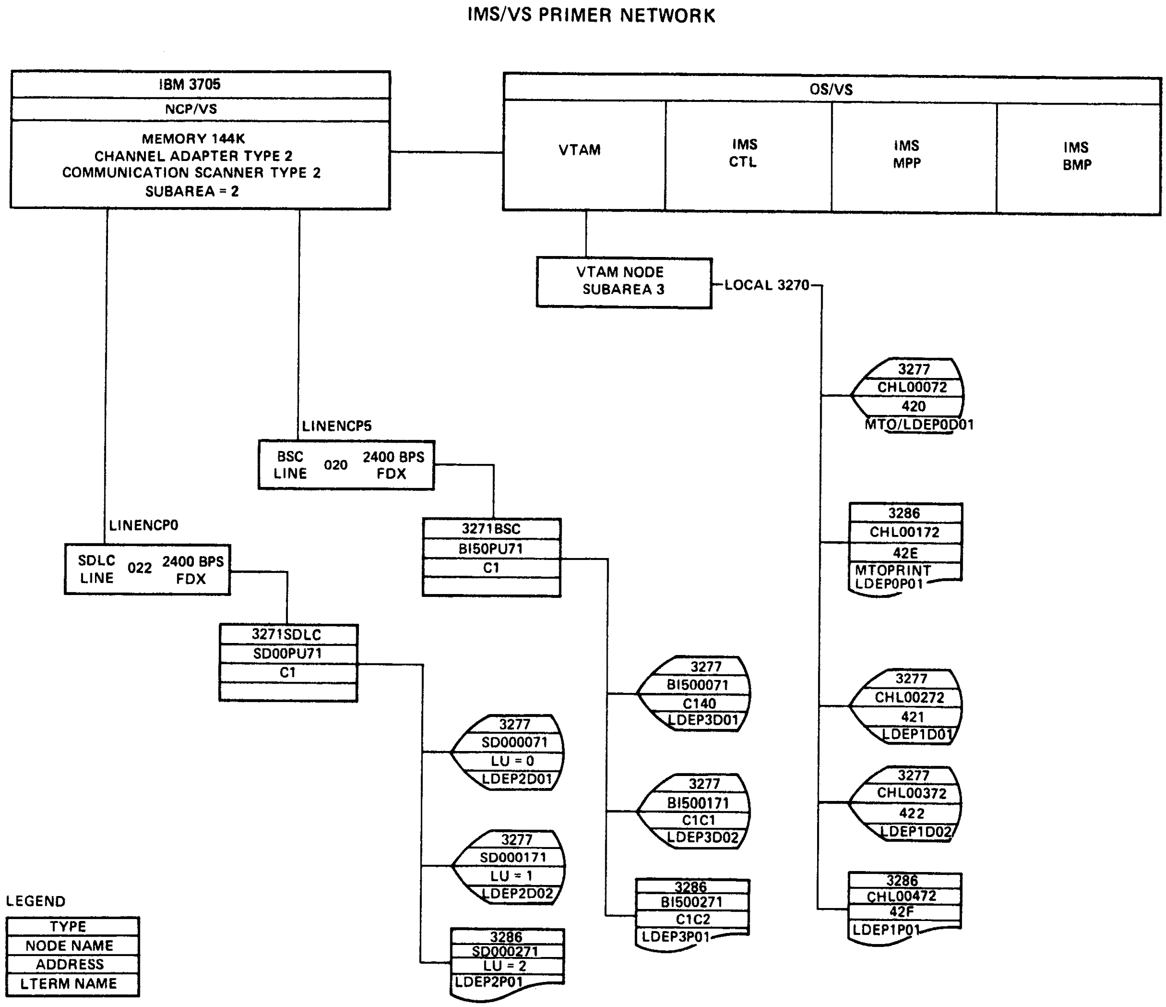


Figure 7-6. Sample IMS/VS-VTAM Network

SAMPI22: EXECUTE IMS/VS SYSTEM DEFINITION STAGE 1 -- BTAM

This is the IMS/VS system definition Stage 1 job to be used instead of the previous job if you are using BTAM instead of VTAM.

SAMPG1 THROUGH SAMPG19: STAGE 2 JOBS

These jobs perform the actual IMS/VS system definition. They should be executed in numerical sequence.

Note:

- These jobs are not listed in IMSVS.PRIMEJOB. They are created as one member (STAGE2) as a result of job SAMPI22 or SAMPI23.

- Jobs SAMPG4 through SAMPG11 need the OS/VS1 system generation macro library SYS1.AMODGEN. This library should have a blocksize not larger than SYS1.MACLIB, because it is concatenated in the assembler SYSLIB DD statement of some Stage 2 jobs. It must be cataloged.

- Control blocks and source modules processed during the execution of the Stage 2 job stream are assembled and link-edited into IMSVS.OBJDSET. Because these modules are link-edited individually, many will produce occurrences of the linkage editor message IEW046I (unresolved external reference) and set condition code 4. When these messages appear in listings, they should be considered normal for the following RESLIB modules:

```
DFSNOF10  control blocks module 1
DFSNOF30  control blocks module 1
DFSNEP10  control blocks module 2
DFSNEP30  control blocks module 2
DFSICPL0  features option list control blocks
DFSIBLK0  control blocks module 4
```

SAMPI24: COPY IMSRDR AND IMS PROCEDURES TO SYS1.PROCLIB

This job copies the IMSRDR procedure to SYS1.PROCLIB. The reader procedure provides access to the message region procedure in IMSVS.JOBS. You should adjust the parameters of the IMSRDR procedure to your installation standards. The IMS procedure is renamed IMSCTL and is the one used for executing the online control region.

SAMPI25: RELINK OS/VS NUCLEUS WITH IMS SVC

This job relinks the OS/VS nucleus to include the Type 2 SVC placed into IMSVS.RESLIB during Stage 2. Your OS/VS system programmer should check the linkage edit control cards to ensure that they comply with other installation requirements. Note that this job relinks the nucleus under the name IEANUC09. When you re-IPL the system after completing all the installation steps, you must specify this suffix to load the alternate nucleus.

SAMPI35: RENAME IMS OS/VS NUCLEUS TO MAIN NUCLEUS

After you have re-IPLed your system with the alternate nucleus linked in SAMPI25, and tested it, you may wish to rename this nucleus to IEANUC01. This job will perform the rename, after saving your original nucleus under the name IEANUC0F.

SAMPI40: UPDATE IMSCTL PROCEDURE WITH SAMPLE DATABASE JCL

This job updates IMSCTL procedure created in job SAMPI24 to include the DD statements for the data bases used by the sample programs. It also overrides the EXEC statement parameters with our subset values.

SAMPI41: UPDATE BUFFER POOL SPECIFICATION FOR THE ONLINE SYSTEM

This job updates member DFSVSM00 in IMSVS.PROCLIB, with buffer pool specifications suitable for running the sample programs.

SAMPI42: UPDATE IMSMSG PROCEDURE WITH USER DD CARD

This job updates IMSMSG procedure to include the DD card used by the status code error handling routine.

SAMPI43: CREATE FIXLIST CONTROL CARDS

This job creates the DFSFIXCC member in IMSVS.PROCLIB. This member contains control statements for fixing parts of the IMS/VS control region in real storage. These specifications should be sufficient for an entry system.

<u>Note</u>: No combination characters are allowed in the control cards of //SAMPI43.

SAMPI45: SYSTEM SECURITY TABLES -- VTAM

This job updates the initial system security tables. If necessary it can be modified to accommodate your own security requirements. At least you should change the password NONPRIME to your own password to secure the use of IMS/VS commands not included in our subset. This job must be executed prior to the initial start-up of the CTL region.

SAMPI44: SYSTEM SECURITY TABLES -- BTAM

This job is comparable to the previous one, SAMPI45, but it is suitable for a BTAM version of the IMS/VS system. The difference is in the logical terminal network and the command subset.

Note: After the OS/VS final preparation has been completed, you must re-IPL the system so that the changes you have made become effective. This should be done after job SAMPI25 and before the initial start-up of the CTL region.

SAMPI50: ALLOCATE SYS1.VTAMLST

This job allocates space for SYS1.VTAMLST.

SAMPI51: PROCEDURE FOR SYS1.VTAMOBJ

This job adds a cataloged procedure named SAMPI51A to SYS1.PROCLIB. This procedure will be used to scratch and reallocate SYS1.VTAMOBJ in every job which updates SYS1.VTAMLST. This ensures the synchronization of these libraries.

SAMPI52: DEFINE IMS/VS TO VTAM

This job adds the definition of IMS/VS as VTAM application to SYS1.VTAMLST via member APPNODES.

SAMPI53: DEFINE LOCAL NETWORK TO VTAM

This job adds the definition of the local 3270 terminals to VTAM via member L03270 in SYS1.VTAMLST. In the sample job, only three local 3277 Model 2 screens and two local 3286 Model 2 printers are defined. You can add additional ones by repeating the LOCAL statement.

SAMPI54: FILE VTAM START PARAMETERS

This job adds the VTAM start parameters, member ATCSTP00, and the major node list, member ATCCON00, to SYS1.VTAMLST.

Note: If only a local network is used, you should delete the NCP node name in the ATCCON00 list. And if only a remote network is used, you should delete the L03270 node name in the ATCCON00 list.

SAMPI55: FILE VTAM START PROCEDURE

This job adds to SYS1.PROCLIB the procedure to be executed during VTAM start-up.

Note: The NCPDUMP and NCPMODS DD statements are not required for a local-only network.

SAMPI56: STORE GTF PROCEDURE FOR VTAM TRACES

This job stores the GTF procedure for VTAM main storage pool traces into SYS1.PROCLIB.

SAMPI60: ALLOCATE NCP LIBRARIES

This job (re)allocates the libraries for NCP generation and execution. In addition the following NCP distribution libraries are required during NCP generation and must be cataloged:

- SYS1.SSPLIB
- SYS1.GEN3705
- SYS1.MAC3705
- SYS1.OBJ3705

SAMPI61: FILE NCP SOURCE DECK

This job files the NCP source deck into SYS1.VTAMLST as member NCP. This sample NCP must be adapted to your installation environment.

SAMPI62: LIST NCP JOBCARD MACRO

This job lists the macro JOBCARD from the NCP generation Stage 1 macro library SYS1.GEN3705. Stage 1 assembly provides job cards for the Stage 2 NCP generation job stream. You might have to change this macro statement for two reasons:

1. Special account information needed in your OS/VS1 installation.

2. The contents of library SYS1.SSPLIB as distributed with the NCP support package is not incorporated in SYS1.LINKLIB or concatenated to it as suggested.

SAMPI63: CHANGE JOBCARD MACRO FOR NCP STAGE 2

This job adds a JOB card to the macro JOBCARD and changes the account information. A JOBLIB DD statement can be added to refer to SYS1.SSPLIB containing the communication controller assembler used for Stage 2 assemblies.

SAMPI64: NCP GENERATION, STAGE 1

This job is the Stage 1 of the NCP generation. It creates the Stage 2 job stream and puts it into the partitioned data set IMSVS.PRIMEJOB under the name NCPSTG2.

Note: The MAXDATA=2468 value in the PCCU macro of the NCP deck is based on a screen size of 1920 characters. If your installation includes larger screen sizes, you might need to increase this value.

SAMPN1 THROUGH SAMPN16: NCP STAGE 2 JOBS

These jobs perform the actual NCP generation. They must be executed in numerical sequence. Several NCP generation jobsteps may cause a condition code of 4. Check the issued warning messages.

Note: These jobs are not listed in IMSVS.PRIMEJOB. They are created as one member (NCPSTG2) as a result of job SAMPI64.

SAMPI65: CREATE LOGON MODE TABLE

A logon mode table is required for 3270s using SDLC. This table should be constructed as shown in our subset.

SAMPI66: CREATE LOGON TABLE

A VTAM logon table is required for 3270s using SDLC. This table should be constructed as shown in our subset in order to comply with our operating procedures.

EXECUTING THE IMS/VS PRIMER SAMPLE JOBS

This section presents the sample jobs which can be executed after the installation of IMS/VS. These jobs are also included in IMSVS.PRIMEJOB, and listed in Chapter 2 of the *IMS/VS Primer Sample Listings* manual. The relevant output listings of selected sample jobs are contained in Chapter 3 of the *IMS/VS Primer Sample Listings* manual.

The following groups are distinguished:

- Initialization jobs for the creation of the sample environment, phase 0.
- Phase 1 jobs, used in the Phase 1 environment.
- Phase 2 jobs, used in the Phase 2 environment.
- Phase 3 jobs, used in the Phase 3 environment.
- Phase 4 jobs, used in the Phase 4 environment.

We recommend that you exercise these jobs in this sequence.

We will now briefly discuss the job in each of the above sections. A more detailed discussion of the function of each job can be found in its originating chapter.

Notes:

1. The sample jobs are designed so that they can be re-executed in most cases.
2. The jobs either contain all their input or refer to IMSVS.PRIMESRC for some input.
3. Guidelines to exercise the sample batch application programs are given in their source code.
4. The virtual storage requirement of each job(step) is less than 512K bytes unless specified in the region parameter of the execute statement. For more detailed information on IMS/VS storage requirements see: *IMS/VS System Programming Reference Manual*, Chapter 5, "IMS/VS Storage Estimates."

INITIALIZING THE SAMPLE ENVIRONMENT

The following jobs in IMSVS.PRIMEJOB can be used for the initialization of the sample environment.

SAMP005: Remove Sample VSAM User Catalog and Space

This job can be used to remove all the VSAM data space for the sample.

SAMP006: Define VSAM User Catalog

This job defines a VSAM user catalog (IMSPRIME) on volume IMSPRM, used by the sample system.

SAMP007: Define VSAM Data Space

This job defines the VSAM data space for the sample data bases.

SAMP009: Build Generation Data Groups

This job builds the generation data groups and model DSCBs to be used for the image copy, monitor, and log data sets.

PHASE 0 JOBS

The following jobs assemble and link edit programs and DBDs of general use:

- SAMP010 : GSAM DBDs
- SAMP030 : Linear randomizing module
- SAMP031 : HDAM sort exit routine
- SAMP032 : Sample status code error routine
- SAMP033 : Sample statistics print routine
- SAMP034 : Sample data base load program

Notes:

- Job SAMP010 uses the DBDGEN procedure in IMSVS.PROCLIB. It must therefore be read in with an appropriate reader such as IMSRDR or PRIME. (See the previous section on installing IMS/VS.) The same is true for PSBGENs.
- Jobs SAMP030 through SAMP034 use the standard ASMFCL procedure in SYS1.PROCLIB. Also, when later compiling a COBOL or PL/I program we will use the standard supplied procedures of the respective language program product.
- The linkage-edit phase of job SAMP033 will give warning messages for unresolved references. This is a valid but expected situation, since this module will later be linked with the sample programs.

PHASE 1 JOBS

SAMP100: PSBGEN for Phase 1 Data Base Load

This job generates the PSB which will be used to load the Phase 1 PARTS data base.

SAMP101: PSBGENs for Phase 1, COBOL

This job generates the COBOL version of the PSBs which will be used in our phase 1 environment.

SAMP102: PSBGENs for Phase 1, PL/I

This job generates the PL/I version of the PSBs which will be used in our phase 1 environment.

Note: The load module names of the comparable COBOL and PL/I PSBs are the same. The same is true for the load module names of the comparable COBOL and PL/I sample application programs. Therefore you cannot intermix the COBOL and PL/I versions of the sample applications unless you change these load module names and their execution jobs.

SAMP110: DBDGEN for Phase 1

This job generates the PARTS DBD which will be used in our Phase 1 environment.

SAMP140: Compile and Link-edit Parts Inventory COBOL Program

This job compiles and link-edits the parts inventory COBOL program, PE1CPINV (member DFS1CINV in IMSVS.PRIMESRC). Notice that in its link-edit, the DL/I language interface module (DFSLI000, alias CBLTDLI) is included. This is required for every COBOL application program. Also an ENTRY statement of DLITCBL is required. In addition the sample status code error routine (DFSOAER) and buffer pool statistics print routine (DFSOAST) are included, since they are used in the sample program.

SAMP141: Compile and Link-edit Purchase Order COBOL Program

This job compiles and link-edits the Parts Purchase Order COBOL program, PE1CPPUR (member DFS1CPUR in IMSVS.PRIMESRC).

SAMP150: Compile and Link-Edit Parts Inventory PL/I Program

This job compiles and link-edits the Parts Inventory PL/I program PE1PPINV (member DFS1PINV in IMSVS.PRIMESRC). Notice that in its link-edit, the PL/I language interface module DFSLI000, alias PLITDLI) is included with entry point PLICALLA. This is required for every PL/I optimizer application program. In addition, the sample station code error routine (DFSOAER) and buffer pool statistics print routine (DFSOAST) are included, since they are used in the sample program. The link-edit step may result in a condition code 4.

SAMP151: Compile and Link-edit Purchase Order PL/I Program

This job compiles and link-edits the Parts Purchase Order PL/I program, PE1PPPUR (member DFS1PPUR in IMSVS.PRIMESRC).

SAMP170: Load the Phase 1 Data Base

This job does the space (re)allocation for the phase 1 PARTS data base and its initial load.

The sort E61 exit routine (DFSOASRT) is used to sort the input data file in physical HDAM sequence. This sort is a performance option and is not necessary for the load process itself.

The IMS/VS supplied DLIBATCH procedure is used in this and the other DL/I batch jobs. The application program to be executed is specified in the PARM field, together with other parameters. For a detailed discussion of these parameters refer to the DLIBATCH procedure discussion later in this chapter.

SAMP171: Execute Parts Inventory Program

This job executes the Parts Inventory program (PE1CPINV).

This is a read-only job, so no log tape is used.

Note: If you are using the PL/I version of the sample programs, the PL/I transient modules are presumed to be available in one of the libraries in the system linklist (LNKLST00) or the link pack area (this is recommended). If not, you must add the PL/I transient library to the JOB/STEPLIB of the IMS/VS execution job. This can best be done by updating the appropriate IMS/VS procedure: DLIBATCH, IMSBATCH, and/or IMSMSG.

SAMP173: Execute Purchase Order Program

This job executes the Purchase Order program (PE1CPPUR). This program uses the DL/I batch checkpoint/restart facility. The sequential input (card) and output (print) files of this program are processed as GSAM data bases.

SAMP174: Execute Failing Purchase Order Program

This job is the same as the previous, except that it issues message DFS3125A, after which it can be abnormally terminated (reply ABEND), continued (reply CONT) or cancelled. This will set an environment for jobs SAMP177 and SAMP178. If cancelled, the GSAM print file can be printed with a separate job which consists of the LAST job step from SAMP174. The corresponding restart job, SAMP178 will do this in its first step.

SAMP177: Sample Backout Run

This job backs out the data base changes of the failing SAMP174 job. Before running this job you might wish to exercise log tape recovery, jobs SAMP190 and SAMP191, and data base recovery, jobs SAMP181 and SAMP182. (See Chapter 6, "Data Base Recovery," for more details.) In any case, the back out job must use a log tape (must be a tape) created by the failing job or the log tape recovery job (SAMP190 and SAMP191). Never use a change accumulation tape for back out.

Note: Message "DFS888, NO DATA BASE RECORDS FOUND FOR PSB=PEICPUR" may occur when backout is done from a (recovered) log tape after a (simulated) system failure during execution of sample job //SAMP174. This message occurs because the data base change records after the PPUR0010 checkpoint were not yet written to the log tape. This is a valid situation since these data base changes were also not yet written to the data base itself.

SAMP178: Restart Purchase Order Program

This job restarts the Purchase Order program from its latest successful checkpoint (PPUR0010) as specified in the PARM field of the EXEC statement.

Observe in sample output (see Chapter 3 of the IMS/VS Primer Sample Listings) that the checkpoint message output at restart (sample output job //SAMP178, stepname LAST, ddname SYSUT2, page 006) is not exactly the same as the old output of the abnormal terminated job (sample output job //SAMP178, stepname OLDPRINT, ddname SYSUT2, page 006). The reason is that the program prints out the textlines: "CHECKPOINT SUCCESSFULLY TAKEN" and "PURCH.AMOUNT: ....." after the actual checkpoint. During restart the checkpoint itself is not repeated; the program gets control after the XRST call and reads in the next input transaction.

SAMP180: Image Copy of the Phase 1 Data Base

This job creates an image copy (back up) of the Phase 1 PARTS data base. In addition it executes a dummy batch and change accumulation job. This is to stress the need for a new change accumulation period after an image dump. If this was not done, the next change accumulation would include the last accumulated log tape of the previous period.

SAMP181: Change Accumulation for Phase 1

This job performs the change accumulation of the Phase 1 log data sets. This job should be executed after each phase 1 job which creates a new log data set, that is, after SAMP173, SAMP174, SAMP177, etc.

Note: In a typical user situation, the DFSULOG DD statement can concatenate multiple subsequent log tapes since the previous data base change accumulation execution.

SAMP182: Unload Phase 1 PARTS Data Base

This job can be used to recover the Phase 1 PARTS data base using the image copy created by SAMP180 and the change accumulation data set created by SAMP181.

SAMP185: Unload Phase 1 PARTS Data Base

This job unloads the Phase 1 PARTS data base using the HD Reorganization Unload Utility.

SAMP186: Reload Phase 1 PARTS Data Base

This job reloads the Phase 1 PARTS data base using the output created by SAMP185.

SAMP190 and SAMP191: Log tape Recovery

These jobs can be used to recover an unclosed log tape. This can be exercised by physically replacing the log tape with a scratch tape before replying CONT on the WTOR of SAMP174. If no I/O errors occur before the end of the current log data set, the error block ID for job //SAMP191 is A00001, that is, the first (error) block after the current log data set. Note: If there is no data on the tape behind the current log data set, that is, a clean tape was used, then no error block will be listed. At termination (end of reel) the output log tape of //SAMP190 will be properly closed by OS/VS. If so, job //SAMP191 need not be executed, the output log tape of //SAMP190 can be used for back-out and change accumulation processing.

PHASE 2 JOBS

The Phase 2 jobs are related to the logical relationship function of DL/I. They should, therefore, only be exercised if you are planning to use this function.

SAMP200: Phase 2 PSBGENs

This job generates the PSBs which will be used in the Phase 2 sample environment.

SAMP210: Phase 2 DBDGENs

This job generates the DBDs which will be used in the Phase 2 sample environment.

SAMP243: Compile and Link-edit Customer Order Program -- COBOL

SAMP254: Compile and Link-edit Customer Order Program-PL/I

SAMP270: Load Phase 2 Data Bases

The prefix resolution step may issue a warning message for the existence of logical parents without logical children, quite a natural fact. As a result, the step will terminate with a condition code of 4, a valid situation.

SAMP271: Execute Sample PL/I Program

SAMP272: Execute Customer Orders Program

This job executes the Customer Order Program, PE2CORDR (member DFS2CCRD (COBOL) or DFS2PCRD (PL/I) in IMSVS.PRIMESRC). During the execution of this job, the DB Monitor will be activated. See Chapter 9, "Optimization," for more details on the DB Monitor operation and output.

SAMP293: Print DB Monitor Reports

This job generates the DB Monitor reports from the output generated by job SAMP272.

SAMP285: Unload Phase 2 Data Bases

The Phase 2 data bases PARTS and Customer Orders are unloaded as the first step in their reorganization.

SAMP286: Reload Phase 2 Data Bases

Using the output of SAMP285, the Phase 2 data bases are reloaded as the second step in their reorganization.

The prefix resolution step may issue a warning message for the existence of logical parents without logical children, quite a natural fact. As a result, the step will terminate with a condition code of 4, a valid situation.

SAMP287: Unload Phase 2 Primary Index Data Base

This job can be used to unload the phase 2 primary INDEX data base of the HIDAM customer orders data base as the first step in its separate reorganization.

SAMP288: Reload Phase 2 Primary Index Data Base

Using the output of the previous job, the primary Index data base of the HIDAM customer orders data base is reloaded as the second step of its separate reorganization.

SAMP289: Phase 1 to Phase 2 Transition

This job can be used to create the Phase 2 data bases using the Phase 1 data base as input. The result will be the same as after job SAMP270.

The prefix resolution step may issue a warning message for the existence of logical parents without logical children, quite a natural fact. As a result, the step will terminate with a condition code of 4, a valid situation.

PHASE 3 JOBS

SAMP300: Phase 3 PSBGENs

SAMP310: Phase 3 DBDGENs

SAMP341: Compile And Link-edit Purchase Order Program -- COBOL

This job compiles and link-edits the Phase 3 COBOL Purchase Order Program. This is an upgrade of the Phase 1 version to include secondary index processing.

SAMP351: Compile and Link-edit Purchase Order Program-PL/I

This job compiles and link-edits the Phase 3 PL/I version of the Purchase Order Program.

SAMP370: Load Phase 3 Data Bases

The prefix resolution step may issue a warning message for the existence of logical parents without logical children, quite a natural fact. As a result, the step will terminate with a condition code of 4, a valid situation.

SAMP373: Execute Phase 3 Purchase Order Program

SAMP380: Image Copy Phase_3 Data Bases

This job creates image copies of all the Phase 3 data bases. In addition, it creates an initial dummy change accumulation data set to emphasize the importance of image copy/change accumulation synchronization.

SAMP381: Phase 3 Change Accumulation

SAMP382: Recover Phase 3 PARTS Data Base

This job recovers the Phase 3 PARTS data base and its secondary index data base. It uses the PARTS image copy of SAMP380 and the change accumulation data set of SAMP381 or SAMP380.

SAMP383: Recover Phase 3 CUSTOMER ORDER Data Base

This job provides the same function as SAMP382 but for the CUSTOMER ORDERS data base and its primary index.

SAMP384: Backout Phase 3 Purchase Order Program

This job is analogous to SAMP177 of Phase 1.

SAMP389: Phase 2 to Phase 3 Transition

This job can be used to create the Phase 3 data bases from the Phase 2 data bases. Notice that no user application programs are needed. The result will be the same as after SAMP370.

The prefix resolution step may issue a warning message for the existence of logical parents without logical children, quite a natural fact. As a result, the step will terminate with a condition code of 4, a valid situation.

PHASE 4 JOBS

SAMP401: Phase 4 COBOL PSBGENs

This job generates the COBOL version of the PSBs for the MPPs and BMPs which will be used in our Phase 4 environment.

SAMP402: Phase 4 PL/I PSBGENs

This job generates the PL/I versions of the PSBs for the MPPs and BMPs which will be used in our Phase 4 environment.

SAMP410: Phase 4 DBDGENs

This job generates the DBDs which will be used in our Phase 4 environment. Note that the logical DBDs used in Phase 4 could have been introduced in Phase 3 (as suggested in Chapter 2).

SAMP420: Phase 4 ACBGEN

This job builds the ACBs from the DBDs and PSBs which will be used in our Phase 4 environment. This must be done before the CTL region can be started.

SAMP425: Phase 4 Formats

This job generates the MFS control blocks used for the message and screen formats in our Phase 4 environment. When initially executed, message DFS1014I may occur and the jobstep terminates with a condition code of 4. This is a valid condition.

SAMP441: Compile and Link-edit COBOL Customer Name Inquiry MPP

SAMP442: Compile and Link-edit COBOL Customer Order Inquiry MPP

SAMP443: Compile and Link-edit COBOL Customer Order Creation MPP

SAMP444: Compile and Link-edit COBOL Customer Order Update MPP

SAMP451: Compile and Link-edit PL/I Customer Name Inquiry MPP

SAMP452: Compile and Link-edit PL/I Customer Order Inquiry MPP

SAMP453: Compile and Link-edit PL/I Customer Order Creation MPP

SAMP454: Compile and Link-edit PL/I Customer Order Update MPP

SAMP471: Execute Parts Inventory BMP

This job executes the Phase 1 Parts Inventory program (PE1CPINV) as a BMP.

SAMP473: *Execute Purchase Order BMP*

This job executes the Phase 3 Purchase Order program (PE3CPPUR) as a BMP. The input provided to the program will cause message DFS3125A to be issued so that the online operating procedures may be tested. See the *IMS/VS Message and Codes Reference Manual* for a detailed description of message DFS3125A.

SAMP474: *Restart Purchase Order BMP*

This job will restart the Purchase Order BMP if it has been cancelled by the operator, or has failed for some other reason. Before execution you must verify that the checkpoint ID specified in the PARM field is the last one of the failing job.

This job needs the log tape from the IMS/VS CTL region in its IMSLOGR dd statement, see Chapter 8, "Operations" and the IMS/VS Primer Master Terminal Operator's Guide for more details.

SAMP481: *Phase 4 Change Accumulation*

SAMP490 and SAMP491: *Log Tape Recovery*

These jobs can be used to recover an unclosed log tape in the online environment. Chapter 8, "Operation," describes how these jobs can be tested.

SAMP492: *Log Tape Termination*

This job uses the System Log Terminator utility to close an unclosed log tape in the online environment. Chapter 8, "Operation," describes how this job should be tested.

SAMP494: *Print DC Monitor Reports*

This job generates the DC Monitor reports from a DC Monitor data set produced during an online session.

SAMP495: *Print Log Tape Statistics*

This job prints statistical reports from the log tape produced by the online system.

RECOMMENDED TEST SEQUENCE

The recommended sequence for exercising the sample jobs is (numbers only):

1. Sample initialization and phase 0: 5, 6, 7, 9, 10, 31, 32, 33, 34.

2. Phase 1 (COBOL): 100, 101, 110, 140, 141, 170, 180, 171, 173, 181, 182, 170, 180, 174, 190, 191, 181, 177, 181, 178, 181, 182, 185, 186.

   Phase 1 (PL/I): 100, 102, 110, 150, 151, 170, 180, 171, 173, 181, 182, 170, 180, 174, 190, 191, 181, 177, 181, 178, 181, 182, 185, 186.

   Note: For a successful exercise of the sample recovery jobs 174 and up, you must be familiar with the data base recovery utilities and procedures as presented in Chapter 6, "Data Base Recovery."

3. Phase 2 (COBOL): 200, 201, 210, 243, 270, 272, 293, 285, 286, 287, 288, 289.

   Phase 2 (PL/I): 200, 202, 210, 254, 270, 272, 293, 285, 286, 287, 288, 289.

4. Phase 3 (COBOL): 300, 301, 310, 341, 370, 380, 373, 381, 382, 383, 384, 389.

   Phase 3 (PL/I): 300, 302, 310, 351, 370, 380, 373, 381, 382, 383.

5. Phase 4 (COBOL): 401, 410, 420, 425, 441, 442, 443, 444.

   Phase 4 (PL/I): 402, 410, 420, 425, 451, 452, 453, 454. After these jobs have been run, the online system may be started as described in the IMS/VS Primer Master Terminal Operator's Guide. The online MPPs can then be exercised using the operating instructions in the IMS/VS Primer Remote Terminal Operator's Guide. Jobs 471, 473, and 474 should be run while the online system is up. Jobs 481, 490, 491, and 492 should be run when the MTO Guide and data base recovery procedures are tested, as described in Chapter 8, "Operations." Jobs 494 and 495 should use the tapes produced by the online system.

If you want to skip the Phase 1, Phase 2 and Phase 3 sample batch jobs and proceed directly to the sample online system, just exercise the following jobs (numbers only):

- Sample initialization: 10, 30, 31, 32, 33, 34
- Phase 1 jobs: 140 (COBOL) or 150 (PL/I).
- Phase 3 jobs: 300, 310, 370, 301 and 341 (COBOL), or 302 and 351 (PL/I).
- Then proceed with the Phase 4 jobs as described above.

Note: The jobs are listed in job number sequence in Chapter 2 of the *IMS/VS Primer Sample Listings* manual.

HDAM RANDOMIZING MODULES

The HDAM access method requires a randomizing module for root segment retrieval and insertion. Each HDAM data base has only one randomizing module, but several data bases can share the same module. The module name and its parameters are specified in the DBD.

The function of a randomizing module is to convert the root key of a data base record into an internal block and an anchor point address. This address is used by the HDAM access method for the storage and retrieval of data base records.

The randomizing module is loaded by IMS/VS when the data base is opened. The normal OS/VS program load facilities are used; therefore, the module must reside in SYS1.LINKLIB or IMSVS.RESLIB if it is to be used by the online control region. If a randomizing module is used for more than one HDAM data base, that is, is a general randomizing routine, it should be reenterable (RENT).

GENERAL RANDOMIZING MODULE

DL/I supplies a general randomizing module which is suitable for most key ranges. This module, DFSHDC40, is used with all our sample HDAM data bases.

You should write your own randomizing module only if you want to maintain rootkey sequence and your key range is suitable for that.

Note: For more details on DFSHDC40 you should consult its source listing, which can be obtained from IMSVS.DBSOURCE.

WRITING A RANDOMIZING MODULE

When an application program issues a Get Unique, Get Next with Qualification, or Insert call which operates on a root segment of an HDAM data base, the user-supplied randomizing module is invoked. (See also Chapter 4, "Processing Data Bases," for more details on DL/I calls.) The root key value is supplied to the randomizing module for conversion to a relative block number and anchor point number within the data base. In addition to the field value parameter supplied by an application program, parameters from the DBD are available to the randomizing module in a CSECT named RDMVTAB. The address of this CSECT is passed to the module each time a conversion is requested.

The following DSECT defines the format of this CSECT:

```
DMBDACS   DSECT
DMBDANME  DS    CL8  NAME OF ADDR ALGORITHM LOAD MODULE
DMBDAKL   DS    OCL1 EXECUTABLE KEY LENGTH OF ROOT
DMBDAEP   DS    A    EP OF ADDR LOAD MODULE
DMBDASZE  DS    H    SIZE OF THIS CSECT
DMBDARAP  DS    H    NUMBER OF ROOT ANCHOR POINTS/BLOCK
DMBDABLK  DS    F    NUMBER OF HIGHEST BLOCK DIRECTLY ADDRSD
DMBDABYM  DS    F    MAX NUMBER OF BYTES BEFORE OFLOW TO 2NDARY
DMBDABYC  DS    F    CUR NUM OF BYTES INSERTED UNDER ROOT
DMBDACP   DS    F    RESULT OF LAST ADDRESS CONVERSION
```

Randomizing Module Interfaces

Upon entry to any randomizing module, registers must be saved. Upon return to DL/I, registers must be restored. A save area address is provided in Register 13 upon entry for the purpose of saving the registers.

The following registers, on entry to a randomizing module, have the indicated meanings:

Register	Meaning or Content
0	Data Management Block address (DMB).
1	DMBDACS CSECT address.
7	Partition Specification Table address (PST). The first 8 bytes can be used as working storage.
9	Address of the first byte of the key field value supplied by an application program.

Register	Meaning or Content
13	Save area address. The first three words in the save area must not be changed.
14	Return to IMS/VS address.
15	Entry Point Address of randomizing module.

The result of a randomizing module conversion must be in the form BBBR where:

BBB is a three-byte binary number of the block into which a root segment is to be inserted, or from which it is to be retrieved.

R is a one-byte binary number of the appropriate anchor point, within a relative block, within an OSAM data set of the data base.

This result must be placed in the CSECT addressed by register 1 in the four-byte field named DMBDACP. If the result exceeds the content of the field DMBDABLK, the result is changed to the highest block and last anchor point of that block.

A SIMPLE KEY-SEQUENTIAL RANDOMIZING MODULE

This simple straightforward randomizing module does a linear conversion of the root key to block, anchor point address. This routine (DFSOALIN in IMSVS.PRIMESRC), can be used for numeric key ranges with an even distribution.

DL/I DATA BASE BUFFERING FACILITIES

The DL/I buffering services are controlled by three pools of control blocks and buffers: the OSAM buffer pool, the DL/I buffer handler pool, and the VSAM buffer pool. This section describes the structure, content, and use of these pools by DL/I.

The DL/I buffering services are the interface between the DL/I action modules (for example, Retrieve, Delete, Insert) and the data management access methods (VSAM and OSAM). Whenever an action module needs to inspect or change data in a data base, buffering services are called to perform whatever physical reading or writing is required. A separate pool of buffers is allocated for each type of data base: VSAM and OSAM. Data bases that use the VSAM access method share the use of buffers in the VSAM shared resource pool. Data bases that use the OSAM access method share the use of buffers in the OSAM buffer pool.

The concept of the buffer pool allows blocks of data to remain in main storage as long as possible, in order to avoid secondary storage reads and writes. Data in the buffer pool can be accessed and updated without causing 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, 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.

If an I/O error occurs while attempting to write a buffer of data, the buffer is marked as a permanent write error buffer and retained in the pool. No error indication is returned to the application program that encountered the error, but an I/O error message is written to the IMS/VS master terminal operator and/or OS/VS console operator, and an error log record is recorded on the IMS/VS log data set.

Whenever an I/O error occurs, the DL/I Data Base Recovery Utility program should be used to re-create the data base that was damaged. (See Chapter 6, "Data Base Recovery.")

DL/I maintains statistics on buffer pool utilization and access method requests. These statistics can be used to determine the optimum buffer pool sizes for a job. The DL/I statistics call (STAT) can be used in an application program to obtain these statistics. See Chapter 4, "Data Base Processing," for a description of the STAT call and Chapter 9, "Optimization," for the interpretation of these statistics.

LOG TAPE WRITE-AHEAD

The log tape write-ahead option is provided to ensure that a data base log record for a data change is physically written to the log device before the changed data is physically written to the data base storage device. This ensures that any change made to a data base is physically recorded on the log tape before the data base is changed. This allows recovery, even in the case of loss of main storage contents due to power failure.

The log tape write-ahead option is activated with the OPTIONS statement in the buffer pool initialization data set. See "Defining the IMS/VS Data Base Buffer Subpools" later in this chapter. In our subset, we will always select this option.

THE DL/I BUFFER HANDLER POOL

The buffer pool is the focal point for recording buffering services activity. The pool prefix (BFSP) contains pointers to the other elements of the pool, indicator flags, and some statistics. If VSAM data bases are used, a subpool statistics block (BFUS) exists for each VSAM buffer subpool defined. The subpool statistics block contains statistics on buffering services and VSAM request activity relevant to the associated subpool. These statistics can be requested with the STAT call. (See Chapter 4, "Data Base Processing.")

THE VSAM BUFFER POOL

The VSAM shared resource pool is used to buffer data for data bases that use VSAM. It is constructed by VSAM, as a shared resource pool, based on parameters provided by DL/I initialization. It contains buffers to be used for VSAM data sets (both index and data components) and the input/output control blocks necessary to perform VSAM requests. The buffers are combined in subpools. All buffers within a subpool are of equal length.

If VSAM is used, the minimum number of subpools is 1 and the maximum is 11. The minimum number of buffers in a subpool is 3, the maximum is 255. Buffer sizes range from 512 to 32768 bytes and must be a power of 2. For DL/I, VSAM control interval sizes may range from 512 to 30720 bytes and must be a multiple of 512 (or a multiple 2048 if greater than 8192). If no VSAM data bases are used, no VSAM subpools are required.

During DL/I data base open, a data set is assigned a specific buffer subpool based on the control interval (CI) size. The CI size must be equal to or less than the buffer size for the subpool assigned. The data and index components of a KSDS may be assigned to different subpools if their CI sizes are different, and corresponding subpools exists. A single subpool can be defined with buffers large enough to contain the longest CI, or several subpools can be defined which more nearly fit the different sized CIs by the programs.

THE OSAM BUFFER POOL

The OSAM buffer pool is used to buffer data for data bases that use the OSAM access method. The pool consists of one or more user defined subpools, comparable to the VSAM buffer pool. Each subpool consists of fixed-length buffers. When a data set is opened, it is assigned a buffer subpool which contains buffers at least as large as the data set block size. If the data set block size is smaller than the buffer size, a portion of the buffer space is not used. If data base data sets contain many different block sizes, many subpools must be defined to provide the best use of buffers. This can, in turn, restrict the number of buffers available to any given data set. Another consideration is the deliberate separation of certain data bases to a specific subpool: a data base with high activity may tend to monopolize a subpool. To avoid this, assign the data set a block size that causes it to be assigned a unique subpool.

DEFINING THE IMS/VS DATA BASE BUFFER SUBPOOLS

The size and structure of the VSAM and OSAM buffer pools and the IMS/VS buffer handler pool are determined by control statements processed during IMS/VS initialization.

In an IMS/VS batch system the control statements are in a data set with ddname DFSVSAMP.

In an online system, they are in a member of IMSVS.PROCLIB called DFSVSMnn. (nn is a user-defined suffix which is also specified as a sub-parameter in the PARM field when executing the online system.) Job //SAMPI41 shows how this member can be created using the OS/VS utility IEBGENER.

Three types of control statements are allowed: the VSAM subpool definition statements, the OSAM subpool definition statements, and the OPTIONS statement. The subpool definition statement is used to define the size and number of buffers in a subpool. The OPTIONS statement allows the user to influence the performance facilities of the DL/I buffering services.

VSAM Subpool Definition Statements

```
  /------------------------------------------------------------
 /                                                            |
 |   size, number                                             |
 |                                                            |
 L------------------------------------------------------------
```

size

A 3- to 5-digit number specifying the buffer size for this subpool. The permissible values are 512, 1024, 2048, 4096, 8192, 12288, 16384, 20480, 24576, 28672, and 32768.

number

A 1- to 3-digit number (3 to 255) specifying the number of buffers in this subpool. If the number of buffers is less than the minimum required, it is increased to the minimum and a warning message is issued.

Buffer size can start in position 1 or beyond and is separated from the number of buffers by a comma. Each statement defines one subpool. A blank must follow the number of buffers. The remaining portion of the statement is ignored.

If two or more subpool definition statements specify the same buffer size, the numbers of buffers from the statements are summed, and a single subpool with the total number of buffers is built.

Guidelines for Selecting Number of Buffers per VSAM Subpool: Following is a basic guideline for the number of VSAM buffers per subpool in an entry environment.

Number of buffers per VSAM subpool =

3

+ (number of ESDSs served from this subpool)*2

+ number of KSDSs index components served from this subpool

+ number of KSDS data components served from this subpool

Example:

A batch program which uses all the data sets of the phase 3 sample data bases should specify:

```
1024,6
2048,9
```

In the online environment the number of buffers should be calculated as shown above, but based on the MPP and BMP with the largest buffer pool requirement. The buffer pool requirements of the most demanding MPP should then be added to those of the most demanding BMP, and this total should be used as an initial estimate of the buffer pool requirements.

Note: The number of subpool buffers can be easily adjusted during production. Chapter 9, "Optimization," contains guidelines for monitoring and adjusting this performance parameter.

OSAM Subpool Definition Statements

IOBF= (l,n,fl,f2)

IOBF= is the required keyword for subpool definition. It must begin in the first position of the control statement. Only one subpool definition may appear on each control statement.

l

Specifies the length of the buffers in the subpool. This parameter is required. If it is invalid, the entire entry will be ignored. The parameter must be in the range 512-32000 bytes. The value that is specified is rounded up to the nearest power of 2, up to 4K. Thereafter, the value is rounded to a multiple of 2K.

n

Specifies the number of buffers in the subpool. This parameter is optional. If specified, it must be in the range 4-255. The default value is 4. If this parameter is invalid, the remainder of the entry will be ignored, and defaults will apply for all remaining parameters.

f1

Specifies the buffer long-term-page-fixing option. This parameter is optional. If Y is specified, all buffers and buffer prefixes associated with this subpool will be long-term-page-fixed at initialization of the subpool. If N is specified, no buffers associated with this subpool will be long-term-page-fixed at initialization of the subpool. The default is N. If this parameter is invalid, the remainder of the entry will be ignored, and defaults will apply for all remaining parameters. Y is recommended.

f2

Specifies the buffer prefix long-term-page-fixing option. This parameter is optional. If Y is specified, all buffer prefixes associated with this subpool and the subpool header will be long-term-page-fixed at initialization of the subpool. If N is specified, the subpool header and all buffer prefixes associated with this subpool will not be long-term-page-fixed at initialization of the subpool. The default is N. Y is recommended.

Guidelines for Selecting Number of Buffers per OSAM Subpool: Following is a basic guideline for the number of VSAM buffers per subpool in an entry environment.

Number of buffers per OSAM subpool =

3

\+ (number of OSAM data sets served from this subpool) *2

In the online environment the number of buffers should be calculated as shown above, but based on the MPP and BMP with the largest buffer pool requirement. The buffer pool requirements of the most demanding MPP should then be added to those of the most demanding BMP, and this total should be used as an initial estimate of the buffer pool requirements.

Note: The OSAM buffer pool definition statements need not necessarily be specified for a DL/I batch job. If not specified, a minimum of 4 buffers per subpool will be allocated in our subset.

Example: Job //SAMP174 shows the use of the OSAM subpool definition statements together with the LTWA option.

OPTIONS Statement

OPTIONS,LTWA=YES,VSAMFIX=(BFR,IOB),BHTRACE=0
OPTIONS,INSERT=SKP|SEQ

The word OPTIONS, starting in position 1 identifies the OPTIONS statement. The parameters can be specified in any sequence and must be separated by commas. A blank must follow the last parameter. The remaining portion of the statement is ignored. An OPTIONS statement cannot be continued on a subsequent statement, but several OPTIONS statements may be provided. If an OPTIONS parameter appears more than once, its setting is determined by the last occurrence.

LTWA=YES

Activates the log tape write-ahead function of DL/I. This should always be specified to concur with our subset recovery procedures.

VSAMFIX=(BFR,IOB)

This is a performance option. It normally should be selected.

BHTRACE=0

Suppresses the DL/I buffer handler trace, which is not part of our subset.

INSERT=SKP|SEQ

Specifies the insert mode that the buffer handler uses when inserting new KSDS logical records in a data base (SHISAM and INDEX). If a program inserts many new root segments in sequence by key, then specifying INSERT=SEQ causes the buffer handler to use VSAM sequential mode PUTs. VSAM leaves free space (if specified in the DEFINE) in CIs created for the new records that are inserted. (VSAM refers to this as "mass insert"). Specifying INSERT=SKP, or omitting the parameter, causes the buffer handler to use VSAM skip sequential mode PUTs. SEQ should be selected for initial load and mass insert jobs. It is automatically enforced in the reorganization reload utilities. INSERT=SEQ will be ignored in the online control region.

IMS/VS SECURITY MAINTENANCE UTILITY

The IMS/VS system security utility is used to define your terminal and password security. This utility must be initially executed after each Stage 2 of IMS/VS system definition, before the CTL region is started. It can be re-executed whenever you want to change your terminal/password security. The new security will become effective at the next start (cold or warm) of the CTL region.

EXECUTING THE SECURITY MAINTENANCE UTILITY

During IMS/VS system definition, a procedure named SECURITY is placed in IMSVS.PROCLIB. With this procedure you can create your own transaction and terminal security matrices which are stored in the IMSVS.MATRIX data set. Jobs //SAMPI44 and //SAMPI45 in IMSVS.PRIMEJOB show how to use the SECURITY procedure.

Note: Even if you don't intend to use the IMS/VS security facility, you must run job //SAMPI44 (BTAM) or //SAMPI45 (VTAM). If you fail to, you will be unable to start the CTL region because there would not be at least one password entry as required by our IMS/VS system definition.

Security Status Report

Each execution of the security maintenance utility produces a printed analysis of the IMS/VS system for which security is being maintained. If errors are encountered in processing the program's input control statements, no security block update functions are performed.

Instead, diagnostic error messages are produced for the entire input stream.

Note: The security status report should be properly maintained as it is vital to the security of the system.

TYPES OF SYSTEM SECURITY

The following types of security are distinguished in our subset:

- Command security: Certain commands are restricted to the master terminal.
- Terminal security: Certain transactions are restricted to selected logical terminals.
- Transaction security: Certain transactions will be accepted only when the terminal operator also enters a defined password.

COMMAND SECURITY

IMS/VS includes during system definition a default command security which we will use. With this default command security, the following commands are restricted to the master terminal: /ASSIGN, /CHANGE, /CHECKPOINT, /CLSDST, /COMPT, /DBDUMP, /DBRECOVERY, /DELETE, /DEQUEUE, /DISPLAY, /ERESTART, /IDLE, /MONITOR, /MSASSIGN, /MSVERIFY, /NRESTART, /OPNDST, /PSTOP, /PURGE, /RSTART, /SMCOPY, /START, /STOP, and /TRACE. As we will only use a subset of IMS/VS commands, it is recommended to protect the non-subset commands with a special password. This will prohibit the accidental use of those commands by an operator. The first part of the input of job //SAMPI44 and //SAMPI45 shows how to do this. The special password is defined by the statement:

```
    ) ( PASSWORD NONPRIME
```

The commands to be protected are specified via the immediately following COMMAND ... statements.

Notes:

1. The commands are abbreviated to their first 3 characters, a standard IMS/VS feature.
2. Obviously, you should change the password (NONPRIME) to your own.

TRANSACTION AND TERMINAL SECURITY

A set of input control statements is required for each transaction code is to be assigned password and/or terminal security.

TRANSACT Statement: One TRANSACT statement is required for each transaction code that requires security. Its format is:

```
   /------------------------------------------------------------
  /                                                             |
 |    ) (   TRANSACT       trancode                             |
 |                                                              |
 ---------------------------------------------------------------
```

Legend:

The)(must be coded in columns 1 and 2 and be followed by a blank. The TRANSACT operand must be followed by a blank and the transaction code (trancode).

PASSWORD Statement: This statement should be included only if you want the preceding transaction code to be protected by a password. Its format is:

```
   /------------------------------------------------------------------
  /                                                                   |
 |        PASSWORD        password                                    |
 |                                                                    |
 L--------------------------------------------------------------------J
```

Legend:

The PASSWORD operand must be preceded and followed by at least one blank. The specified password must be 1 to 8 alphanumeric characters.

TERMINAL Statement: One TERMINAL statement is required for each logical terminal authorized to enter this transaction. If no TERMINAL statement follows a TRANSACT statement, then that transaction code is accepted from any terminal. The format of the TERMINAL statement is:

```
   /------------------------------------------------------------------
  /                                                                   |
 |        TERMINAL        lterm-name                                  |
 |                                                                    |
 L--------------------------------------------------------------------J
```

Legend:

The TERMINAL operand must be preceded and followed by at least one blank. The specified lterm-name must be defined during IMS/VS system definition via a NAME statement.

IMS/VS CATALOGED PROCEDURES

Several procedures are placed in the IMSVS.PROCLIB by Stage 2 system definition.

The following are applicable to our subset environment and are used in our sample jobs:

- ACBGEN   builds online control blocks from your DBDs and PSBs
- DBDGEN   generates data base definition control blocks (DBDs).

- DLIBATCH executes a batch DL/I program.
- IMSBATCH executes a batch message processing program (Phase 4 only).
- IMS executes the online control program (Phase 4 only)
- IMSMSG executes the online message processing programs (Phase 4 only).
- IMSRDR is the reader procedure for using IMSVS.PROCLIB
- MFSRVC builds an index directory used by the online MFS pool manager.
- MFSUTL generates MFS control blocks (Phase 4 only)
- PSBGEN generates program specification blocks (PSBs).
- SECURITY generates the security blocks used by the online control program (Phase 4 only).

In the following overview of these procedures, we will discuss the execution parameters only of interest to our subset. For those parameters not discussed, you should use their default values.

ACBGEN PROCEDURE

```
//          PROC  SOUT=A,COMP=,RGN=160K                                   0000001
//G         EXEC  PGM=DFSRRC00,PARM='UPB,&COMP',REGION=&RGN               0000002
//SYSPRINT  DD    SYSOUT=&SOUT                                            0000003
//STEPLIB   DD    DSN=IMSVS.RESLIB,DISP=SHR                               0000004
//DFSRESLB  DD    DSN=IMSVS.RESLIB,DISP=SHR                               0000005
//IMS       DD    DSN=IMSVS.PSBLIB,DISP=SHR                               0000006
//          DD    DSN=IMSVS.DBDLIB,DISP=SHR                               0000007
//IMSACB    DD    DSN=IMSVS.ACBLIB,DISP=OLD                               0000008
//SYSUT3    DD    UNIT=SYSDA,SPACE=(80,(100,100))                         0000009
//SYSUT4    DD    UNIT=SYSDA,SPACE=(256,(100,100)),DCB=KEYLEN=8           0000010
//COMPCTL   DD    DSN=IMSVS.PROCLIB(DFSACBCP),DISP=SHR                    0000011
```

EXEC Statement Parameters for ACBGEN

SOUT=

specifies the SYSOUT class.

COMP=

specifies the library compression option. Specify POSTCOMP for in-place compression after new members are added for best performance.

RGN=

Specifies the region size for MVS users.

DBDGEN PROCEDURE

```
//        PROC  MBR=TEMPNAME,SOUT=A
//C       EXEC  PGM=IFOX00,REGION=256K,PARM='OBJ,NODECK'
//SYSLIB    DD  DSN=IMSVS.MACLIB,DISP=SHR
//SYSGO     DD  UNIT=SYSDA,DISP=(,PASS),
//              SPACE=(80,(100,100),RLSE),
//              DCB=(BLKSIZE=400,RECFM=FB,LRECL=80)
//SYSPRINT DD   SYSOUT=&SOUT,DCB=BLKSIZE=1089,
//              SPACE=(121,(300,300),RLSE,,ROUND)
//SYSUT1    DD  UNIT=SYSDA,DISP=(,DELETE),
//              SPACE=(1700,(100,50))
//SYSUT2    DD  UNIT=SYSDA,DISP=(,DELETE),
//              SPACE=(1700,(100,50))
//SYSUT3   DD  UNIT=(SYSDA,SEP=(SYSLIB,SYSUT1,SYSUT2)),
//             SPACE=(1700,(100,50))
//L       EXEC  PGM=DFSILNK0,PARM='XREF,LIST',COND=(0,LT,C),REGION=120K
//STEPLIB   DD  DSN=IMSVS.RESLIB,DISP=SHR
//SYSLIN    DD  DSN=*.C.SYSGO,DISP=(OLD,DELETE)
//SYSPRINT DD   SYSOUT=&SOUT,DCB=BLKSIZE=1089,
//              SPACE=(121,(90,90),RLSE)
//SYSLMOD   DD  DSN=IMSVS.DBDLIB(&MBR),DISP=SHR
//SYSUT1    DD  UNIT=(SYSDA,SEP=(SYSLMOD,SYSLIN)),
//            SPACE=(1024,(100,10),RLSE),DISP=(,DELETE)
```

See the section entitled "Execution of DBDGEN (JCL)" in Chapter 2, "Data Base Design" for details on using this procedure.

DLIBATCH PROCEDURE

```
//        PROC  MBR=TEMPNAME,SOUT=A,PSB=,BUF=7,
//              SPIE=0,TEST=0,EXCPVR=0,RST=0,LOGT=2400,
//              FRLD=,SRCH=0,CKPTID=,MON=N,LOGA=0,
//              FMTO=T,IMSID=
//G EXEC PGM=DFSRRC00,REGION=192K,
//              PARM=(DLI,&MBR,&PSB,&BUF,
//              &SPIE&TEST&EXCPVR&RST,&FRLD,
//              &SRCH,&CKPTID,&MON,&LOGA,&FMTO,
//              &IMSID)
//STEPLIB   DD  DSN=IMSVS.RESLIB,DISP=SHR
//          DD  DSN=IMSVS.PGMLIB,DISP=SHR
//DFSRESLB  DD  DSN=IMSVS.RESLIB,DISP=SHR
//IMS       DD  DSN=IMSVS.PSBLIB,DISP=SHR
//          DD  DSN=IMSVS.DBDLIB,DISP=SHR
//PROCLIB   DD  DSN=IMSVS.PROCLIB,DISP=SHR
//IEFRDER   DD  DSN=IMSLOG,DISP=(,KEEP),VOL=(,,,99),UNIT=(&LOGT,,DEFER),
// DCB=(RECFM=VB,BLKSIZE=1920,LRECL=1916,BUFNO=2)
//SYSUDUMP DD   SYSOUT=&SOUT,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=605),
// SPACE=(605,(500,500),RLSE,,ROUND)
//IMSUDUMP  DD  SYSOUT=&SOUT,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=605),
// SPACE=(605,(500,500),RLSE,,ROUND)
//IMSMON    DD  DUMMY
```

__Notes__:

1. The BLKSIZE and LRECL values shown in the IEFRDER dd statement are the default values. If the DCB parameters are changed, log initialization calculates the smallest value necessary for logical record length. If the JCL logical record length value is larger than the calculated value, the JCL value is used; otherwise, log initialization uses the calculated value for logical record length and adds 4 for the block size.

2. This statement describes the recording device to be used by the DB monitor. It is required only if MON=Y is specified in the PROC statement, and then only if a device other than the IMS/VS system log is to be used for monitor data. When a separate log device is used for DB monitor data, a //IMSMON DD statement must be included that specifies a sufficient BLKSIZE and LRECL (2048 and 2044 are suggested).

3. You must add the DD statements for all the data bases the job step uses.

4. A DFSVSAMP DD statement must be added for the LTWA option and the OSAM and/or VSAM subpool specifications.

EXEC Statement Parameters for DLIBATCH

MBR=

specifies the application program name.

SOUT=

specifies the class assigned to SYSOUT DD statements.

PSB=

is an optional parameter specifying a PSB name when the PSB name and application program name are different.

BUF=

specifies the data base buffer size. If not present, a default size of 7K will be used. Buffer size is specified in 1K multiples. Values may range from 1 through 999. Using IOBF= cards in the DFSVSAMP data set will override this parameter.

SPIE=

specifies the SPIE option: 0= allow user SPIE, if any, to remain in effect while processing the application program call. This option is recommended.

1= negate the user's SPIE while processing the application program call. Negated SPIEs are reinstated before control is returned to the application program.

TEST=

specifies whether (1) or not (0) the addresses in the user's call list should be checked for validity. Zero (0) is the recommended value.

EXCPVR=

specifies whether EXCP (0) or EXCPVR (1) is to be used for data sets processed by OSAM (OS/VS1 only). One is the recommended value if the program accesses OSAM data bases, otherwise zero should be specified.

RST=

must be zero (0) in our subset.

PRLD=

should be omitted in our subset.

SRCH=

is the module search indicator for directed load:
0= standard search.

1= search JPA and LPA before JOBLIB/STEPLIB (MVS only).

CKPTID=

specifies the checkpoint at which the program is to be restarted; specified as a 1-to 8-character extended checkpoint ID. If no checkpoint exists, this parameter should be omitted.

MON=

specifies whether (Y) or not (N) the DB monitor is to be active for this execution. See Chapter 9, "Optimization," for a description of the DB Monitor and its use.

LOGA=

specifies the use of BSAM (0) or OSAM (1) for the logging facility. 1 is the recommended value. 0 is the default.

LOGT=

specifies the tape device type where the log data set will be mounted. The default is device type 2400.

IMSID=

specifies a four character identifier that is a valid subsystem identifier to the operating system being used. This identifier will be used instead of the identifier specified at system generation of the IMS/VS system being executed. Multiple batch jobs with the same identifier are allowed to run concurrently. So you normally don't need to specify this parameter except when you want to separate a test and production version.

```
//       PROC  RGN=600K,SOUT=A,DPTY='(14,15)',
//             DUMPDS=DUMMY,DV=,DU=,LOGT=2400,
//             CTL=CT,RGSUF=L,RES=,FRE=,QBUF=,DYBN=,PST=,
//             SAV=,EXVR=,PRF=,SRCH=,FBP=,PSB=,DMB=,
//             DBB=,TPDP=,WKAP=,PSBW=,CWAP=,DBWP=,
//             MFS=,SUF=,FIX=,PRLD=,VSPEC=00,SOD=,
//             IOB=,VAUT=,LOGA=0,FMTO=T,
//             AUTO=N,RDBN=2,TRN=N,
//             SGN=N,RCF=N,IMSID=IMSA,ISIS=0,
//             LOGD=0
//IEFPROC EXEC PGM=DFSRRC00,REGION=&RGN,DPRTY=&DPTY,
// PARM=(&CTL&RGSUF,
// &RES,&FRE,&QBUF,&DYBN,&PST,&SAV,
// &EXVR,&PRF,&SRCH,&FBP,&PSB,&DMB,&DBB,
// &TPDP,&WKAP,&PSBW,&CWAP,&DBWP,&MFS,
// &SUF,&FIX,&PRLD,&VSPEC,&SOD,&IOB,
// &VAUT,,,,,,
// &LOGA,&FMTO,&AUTO,&RDBN,&TRN,&SGN,
// &RCF,&IMSID,&ISIS,,&LOGD)
//*
//*
//* THE MEANING AND MAXIMUM SIZE OF EACH PARAMETER
//* IS AS FOLLOWS:
//*
//******** CONTROL REGION SPECIFICATIONS ********
//*   RGSUF  X     EXEC PARM DEFAULT BLOCK SUFFIX
//*   RES    X     BLOCK RESIDENT (N = NO, Y = YES)
//*   FRE    XXX   NUMBER OF FORMAT REQUEST ELEMENTS
//*   QBUF   XXX   NUMBER OF MESSAGE QUEUE BUFFERS
//*   DYBN   XXX   NUMBER OF DYNAMIC LOG BFFRS FOR PI
//*   PST    XX    NUMBER OF PST'S
//*   SAV    XXX   NUMBER OF DYNAMIC SAVE AREA SETS
//*   EXVR   X     PAGEFIX DYN LOG AND QMGR BUFFER POOLS (1=YES, 0=NO)
//*   PRF    X     PREFETCH OPTION (Y = YES, N = NO)
//*   SRCH   X     MODULE SEARCH INDICATOR FOR DIRECTED LOAD
//*                0 = STANDARD SEARCH
//*                1 = SEARCH JPA AND LPA BEFORE PDS
//*   SOD    X     1 CHARACTER SYSOUT CLASS
//*   IOB    XXX   NUMBER OF OSAM I/O REQUESTS
//*   VAUT   X     VTAM AUTH PATH OPTION (1=YES, 0=NO)
//*   LOGA   X     0 = BSAM FOR LOGGING (DEFAULT)
//*                1 = OSAM FOR LOGGING
//*   FMTO   X     T = FORMATTED DUMP WITH
//*                    STORAGE IMAGE DELETIONS
//*                P = FULL FORMATTED DUMP
//*                N = NO FORMATTED DUMP
//*   AUTO   X     Y = AUTOMATIC RESTART DESIRED
//*                N = NO AUTOMATIC RESTART
//*   RDBN   XXXXX NUMBER OF RESTART DATA SET BUFFERS
//*   TRN    X     TRANSACTION AUTHORIZATION CHECKING
//*                F = FORCED, Y = YES, N = NO
//*   SGN    X     SIGNON AUTHORIZATION CHECKING
//*                F = FORCED, Y = YES, N = NO
//*   RCF    X     RACF USED FOR TRANS. AND SIGNON
//*                Y = YES, N = NO
//*   IMSID  XXXX  IMS/VS SUBSYSTEM IDENTIFIER
//*   ISIS   X     0 = NO RESOURCE ACCESS SECURITY
//*                1 = RACF RESOURCE ACCESS SECURITY
//*                2 = USER RESOURCE ACCESS SECURITY
//*   LOGD   X     0 = NO LOG VOLUME DEQUEUE (DEFAULT)
//*                1 = DEQUEUE LOG VOLS AT EOV (MVS)
```

```
//*
//******** STORAGE POOL SIZES IN 1K BLOCKS ******
//*
//*    FBP     XXX   MESSAGE BUFFER POOL
//*    PSB     XXX   PSB POOL
//*    DMB.    XXX   DMB POOL
//*    DBB     XXX   DATA BASE BUFFER POOL
//*    TPDP    XXX   TP DEVICE I/O POOL
//*    WKAP    XXX   WORKING STORAGE BUFFER POOL
//*    PSBW    XXX   PSB WORK POOL
//*    CWAP    XXX   SPA POOL
//*    DBWP    XXX   DATABASE WORK POOL
//*    MFS     XXX   MAXIMUM MFSTEST SPACE
//*
//******** MEMBER SUFFIXES *********************
//*
//*    SUF     X     LAST CHARACTER OF CTL PROGRAM LOAD MODULE MEMBER NAME
//*    FIX     XX    2 CHARACTER FIX PROCEDURE MODULE SUFFIX
//*    PRLD    XX    2 CHARACTER PROCLIB MEMBER SUFFIX FOR PRELOAD
//*    VSPEC   XX    2 CHARACTER BUFFER POOL SPEC MODULE SUFFIX
//**********************************************
//*
//DFSRESLB DD  DSN=IMSVS.RESLIB,DISP=SHR
//PROCLIB  DD  DSN=IMSVS.PROCLIB,DISP=SHR
//IEFRDER  DD  DSN=IMSVS.IMSLOG,DISP=(,KEEP),
// VOL=(,,,99),UNIT=(&LOGT,,DEFER),
// DCB=(RECFM=VB,BLKSIZE=3992,LRECL=3988,BUFNO=2)
//IMSLOGR  DD  DSN=IMSVS.IMSLOG,
// DISP=(MOD,KEEP),UNIT=AFF=IEFRDER
//IMSMON   DD  DSN=IMSVS.IMSMON,DISP=(,KEEP),
// VOL=(,,,99),UNIT=(&LOGT,,DEFER,SEP=IEFRDER)
//QBLKS    DD  DSN=IMSVS.QBLKS,DISP=OLD
//SHMSG    DD  DSN=IMSVS.SHMSG,DISP=OLD
//LGMSG    DD  DSN=IMSVS.LGMSG,DISP=OLD
//IMSACB   DD  DSN=IMSVS.ACBLIB,DISP=SHR
//IMSDILIB DD  DSN=IMSVS.FORMAT,DISP=OLD
//SYSUDUMP DD  SYSOUT=&SOUT,
// DCB=(LRECL=125,RECFM=FBA,BLKSIZE=3129),
// SPACE=(6050,300,,,ROUND)
//IMSRDS   DD  DSN=IMSVS.RDS,DISP=OLD
//DUMP     DD  &DUMPDS,
// DISP=OLD,LABEL=(,NL)&DV&DU
//IMSUDUMP DD  SYSOUT=&SOUT,
// DCB=(LRECL=125,RECFM=FBA,BLKSIZE=3129),
// SPACE=(6050,300,,,ROUND)
//MATRIX   DD  DSN=IMSVS.MATRIX,DISP=SHR
//PRINTDD  DD  SYSOUT=&SOUT
//IMSDBL   DD  DSN=IMSVS.DBLLOG,DISP=SHR
//*
//* USER MUST SUPPLY THE DD STATEMENTS
//* FOR THE ON-LINE DATA BASES TO BE
//* INSERTED HERE PRIOR TO ATTEMPTING
//* AN ON-LINE SYSTEM EXECUTION USING
//* THIS PROCEDURE.
```

Notes:

1. The program name specified on the EXEC statement is DFSRRC00 for OS/VS1 and DFSMVRC0 for MVS.

2. The BLKSIZE and LRECL values shown in the IEFRDER dd statement are the default values. If the DCB parameters are changed, log initialization calculates the smallest value necessary for logical record length. If the JCL logical record length value is larger than the calculated value, the JCL value is used; otherwise log initialization uses the calculated value for logical record length and adds 4 for the block size.

3. The DD statement called IMSMON describes the recording device to be used by the DC Monitor and is required only if you wish to invoke the monitor during an online session.

4. The above listed IMS procedure is produced for the VTAM only system. The procedure for the BTAM only system is the same, but will contain DD statements for the communication lines.

5. In our subset, you must add the DD statements for the data base data sets to be used by the control region to this procedure. See job //SAMPI4C in IMSVS.PRIMEJOB.

EXEC Statement Subset Parameters for IMS Procedure

RGN=

specifies the size of the region in which the control program is to run, and has meaning only in an MVS system.

SOUT=

specifies the class to be assigned to SYSOUT DD statements.

DPTY=

specifies the OS/VS dispatching priority at which the IMS/VS control region should operate. See the OS/VS JCL documentation for details of DPTY. The IMS/VS control region must not be executed at priority zero, or be scheduled into an OS/VS1 partition whose priority falls within JES1 DDG, or into an MVS region whose priority falls within a JES2 APG. A general rule to follow is that the IMS control region dispatching priority must always be higher than the dispatching priority of an IMS/VS dependent region.

CTL=

specifies whether IMS/VS should operate as a system task. CTL=CTL indicates that it should run as a system task; CTL=CTX indicates that it should run as a problem program. CTL=CTL is recommended.

VSPEC=

specifies the two-digit suffix of the DFSVSMnn member of IMSVS.PROCLIB that contains the VSAM and OSAM buffer pool specifications to be used.

FIX=

specifies the two-digit suffix of the DFSFIXnn member of IMSVS.PROCLIB which controls the fixing in real storage of selected parts of the CTL region.

VAUT=

specifies whether (1) or not (0) IMS/VS is to use the VTAM authorized path facility. The recommended option is 1.

LOGA=

specifies which logging facility, BSAM or OSAM, IMS/VS is to use. Specify 1 for OSAM, the recommended option.

The other symbolic parameters need not be specified because the default values calculated during system definition should be sufficient for our entry environment.

LOGT=

specifies the tape device type. The default is device type 2400.

LOGD=

specifies whether (1) or not (0) IMS/VS output log volumes are to be dequeued at EOV. This parameter is valid for MVS only. The recommended value is 1 especially if you consider restarts of BMP, which use the XRST call.

IMSBATCH PROCEDURE

```
//         PROC  MBR=TEMPNAME,SOUT=A,OPT=N,SPIE=0,TEST=0,
//               PSB=,PRLD=,STIMER=,CKPTID=,IN=,OUT=,DIRCA=000,
//               PARDLI=,CPUTIME=,NBA=,OBA=,IMSID=,AGN=
//G        EXEC  PGM=DFSRRC00,REGION=128K,
//               PARM=(BMP,&MBR,&PSB,&IN,&OUT,
//               &OPT&SPIE&TEST&DIRCA,&PRLD,&STIMER,&CKPTID,
//               &PARDLI,&CPUTIME,&NBA,&OBA,&IMSID,&AGN)
//STEPLIB   DD   DSN=IMSVS.RESLIB,DISP=SHR
//          DD   DSN=IMSVS.PGMLIB,DISP=SHR
//PROCLIB   DD   DSN=IMSVS.PROCLIB,DISP=SHR
//SYSUDUMP  DD   SYSOUT=&SOUT,DCB=(LRECL=121,RECFM=VBA,BLKSIZE=3129),
//               SPACE=(125,(2500,100),RLSE,,ROUND)
```

Note: You must add a DD statement for the log tape containing the checkpoint data when you are restarting a BMP which uses the XRST call. This DD statement has the DDname IMSLOGR.

EXEC Statement Parameters for IMSBATCH

MBR=

specifies the application program name.

SOUT=

specifies the class assigned to SYSOUT DD statements.

PSB=

is an optional parameter specifying a PSB name when the PSB name and application program name are different. The PSB name must be defined as PGMTYPE=BATCH with an APPLCTN macro in your IMS/VS system definition.

SPIE=

specifies the SPIE option: 0= allow user SPIE, if any, to remain in effect while processing the application program call. This option is recommended.

1= negate the user's SPIE while processing the application program call. Negated SPIEs are reinstated before returning to the application program.

TEST=

specifies whether (1) or not (0) the addresses in the user's call list should be checked for validity. Zero (0) is the recommended value.

PRLD=

should be omitted in our subset.

CKPTID=

specifies the checkpoint at which the program is to be restarted, specified as a 1-to 8-character extended checkpoint ID. If this is not a restart run, this parameter should be omitted.

CPT=

specifies the action to be taken if the batch message region starts and there is no control program active.

- N -- ask operator for decision. This is the default.
- W -- wait for a control region.
- C -- cancel the batch message region automatically.

N is the recommended value.

IN=

should be omitted in our subset.

OUT=

should be omitted in our subset.

DIRCA=

should be omitted in our subset.

IMSMSG PROCEDURE

```
//MESSAGE JOB 1,IMS,MSGLEVEL=1,PRTY=11,CLASS=A,MSGCLASS=A,REGION=128K
//REGION EXEC  PGM=DFSRRC00,REGION=128K,
//             TIME=1440,DPRTY=(12,0),
//             PARM='MSG,00100000000'
//STEPLIB  DD  DSN=IMSVS.RESLIB,DISP=SHR
//         DD  DSN=IMSVS.PGMLIB,DISP=SHR
//PROCLIB  DD  DSN=IMSVS.PROCLIB,DISP=SHR
//SYSUDUMP DD  SYSOUT=A,DCB=(LRECL=121,BLKSIZE=3129,RECFM=VBA),
//             SPACE=(125,(2500,100),RLSE,,ROUND)
```

This procedure must be copied to IMSVS.JOBS. See job //SAMPI42 in IMSVS.PRIMEJOB.

IMSRDR PROCEDURE

```
//         PROC  MBR=IMSMSG
//IEFPROC EXEC PGM=IEFVMA,          READER FIRST LOAD
//              PARM='00100300005210E00011AXX'  DEFAULT OPTIONS
//*             BPPTTTTSSCCCRLAAAAEFHXX    PARM FIELD
//*            B    PROGRAMMER NAME AND ACCOUNT NUMBER NOT NEEDED
//*            PP   PRIORITY=01
//*        TTTTSS   JOB STEP INTERVAL=30 MINUTES
//*           CCC   JOB STEP DEFAULT REGION=52K
//*            R    DISPLAY AND EXECUTE COMMANDS=1
//*            L    BYPASS LABEL=0
//*           AAAA COMMAND AUTHORITY FOR MCS=E000
//*                  - ALL COMMANDS MUST BE AUTHORIZED
//*            E    JCL MESSAGE LEVEL DEFAULT=1 -ALL MESSAGES
//*            F    ALLOC/TERM MESSAGE LEVEL DEFAULT=1 -ALL MESSAGES
//*            H    DEFAULT MSGCLASS=A
//*            XX   PARTITION,RDR WILL HAVE DISPATCHING
//*                 PRIORITY 1 LOWER THAN PARTITION
//*                 SPECIFIED. XX=SYSTEM TASK PRIORITY
//IEFRDER DD DSN=IMSVS.JOBS(&MBR),DISP=SHR,DCB=BUFNO=1
//IEFPDSI DD DSN=IMSVS.PROCLIB,DISP=SHR
//         DD DSN=SYS1.PROCLIB,DISP=SHR
```

This procedure must be copied to SYS1.PROCLIB. See job //SAMPI24 in IMSVS.PRIMEJOB.

PSBGEN PROCEDURE

```
//         PROC  MBR=TEMPNAME,SOUT=A
//C        EXEC  PGM=IFOX00,REGION=200K,PARM='OBJ,NODECK'
//SYSLIB    DD   DSN=IMSVS.MACLIB,DISP=SHR
//SYSGO     DD   UNIT=SYSDA,DISP=(,PASS),
//               SPACE=(80,(100,100),RLSE),
//               DCB=(BLKSIZE=400,RECFM=FB,LRECL=80)
//SYSPRINT DD    SYSOUT=&SOUT,DCB=BLKSIZE=1089,
//               SPACE=(121,(300,300),RLSE,,ROUND)
//SYSUT1    DD   UNIT=SYSDA,DISP=(,DELETE),
//               SPACE=(1700,(100,50))
//SYSUT2    DD   UNIT=SYSDA,DISP=(,DELETE),
//               SPACE=(1700,(100,50))
//SYSUT3   DD   UNIT=(SYSDA,SEP=(SYSLIB,SYSUT1,SYSUT2)),
//              SPACE=(1700,(100,50))
//L        EXEC  PGM=DFSILNK0,PARM='XREF,LIST',COND=(0,LT,C),REGION=120K
//STEPLIB   DD   DSN=IMSVS.RESLIB,DISP=SHR
//SYSLIN    DD   DSN=*.C.SYSGO,DISP=(OLD,DELETE)
//SYSPRINT DD    SYSOUT=&SOUT,DCB=BLKSIZE=1089,
//               SPACE=(121,(90,90),RLSE)
//SYSLMOD   DD   DSN=IMSVS.PSBLIB(&MBR),DISP=SHR
//SYSUT1    DD   UNIT=(SYSDA,SEP=(SYSLMOD,SYSLIN)),
//             SPACE=(1024,(100,10),RLSE),DISP=(,DELETE)
```

SECURITY PROCEDURE

```
//         PROC  OPTN=UPDATE,IMS=',0',SOUT=A
//S        EXEC  PGM=DFSISMP0,PARM='&OPTN.&IMS.'
//STEPLIB   DD   DSN=IMSVS.RESLIB,DISP=SHR
//SYSPRINT DD    SYSOUT=&SOUT,DCB=(RECFM=VBA,BLKSIZE=129,LRECL=125)
//SYSPUNCH DD UNIT=SYSDA,SPACE=(80,(800,400),,,ROUND),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),
//              DISP=(NEW,PASS)
//SYSLIN    DD   UNIT=SYSDA,SPACE=(TRK,(1,1)),
//               DCB=(RECFM=F,BLKSIZE=80),
//               DISP=(NEW,PASS)
//SYSUT1    DD   UNIT=SYSDA,DCB=(BLKSIZE=500,RECFM=FB),
//               SPACE=(100,(400,400),,,ROUND)
//SYSUT2    DD   UNIT=(SYSDA,SEP=SYSUT1),DCB=*.S.SYSUT1,
//               SPACE=(100,(400,400),,,ROUND)
//SYSIN     DD   DSN=NO.SYSIN.DD.ASTERISK
//C        EXEC  PGM=IFOX00,PARM='OBJ,NODECK',COND=(12,LT,S),REGION=128K
//SYSPRINT DD    SYSOUT=&SOUT,DCB=BLKSIZE=1089
//SYSGO     DD   UNIT=(SYSDA,SEP=SYSPRINT),DISP=(,PASS),
//               SPACE=(80,(400,400),,,ROUND),
//               DCB=*.S.SYSPUNCH
//SYSUT1    DD   UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSUT2    DD   UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSUT3    DD   UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2)),
//               SPACE=(CYL,(5,1))
//SYSIN     DD   DSN=*.S.SYSPUNCH,DISP=(OLD,DELETE)
//L        EXEC  PGM=DFSILNK0,PARM='LIST,NE,OL',REGION=110K,COND=(4,LT,S)
//STEPLIB   DD   DSN=IMSVS.MATRIX,DISP=SHR
//SYSPRINT DD    SYSOUT=&SOUT,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=605)
//SYSLMOD   DD   DSN=IMSVS.MATRIX,DISP=SHR
//INPUT     DD   DSN=*.C.SYSGO,DISP=(OLD,DELETE)
//SYSUT1  DD   UNIT=(SYSDA,SEP=INPUT),SPACE=(CYL,(5,1))
//SYSLIN  DD   DSN=*.S.SYSLIN,DISP=(OLD,DELETE)
```

MFSRVC PROCEDURE

```
//          PROC DEVCHAR=0
//MFSRVC   EXEC PGM=DFSUTSA0,REGION=250K,PARM='DEVCHAR=&DEVCHAR'
//*
//*          PRINT FILES
//*
//SYSPRINT DD   SYSOUT=A
//*             DCB=(RECFM=VBA,LRECL=137)
//SYSSNAP   DD  SYSOUT=A
//*             DCB=(RECFM=VBA,LRECL=125,BLKSIZE=1632)
//SYSUDUMP DD   SYSOUT=A
//*
//*          REFERAL LIBRARY
//*
//REFIN      DD  DSN=IMSVS.REFERAL,DISP=OLD
//*
//*          ON-LINE FORMAT LIBRARY
//*
//FORMAT     DD  DSN=IMSVS.FORMAT,DISP=OLD
//*
//*
//*           //SYSIN DD * MUST BE SUPPLIED BY
//*           USER WITH INPUT CONTROL CARD STREAM
//*
//*           ALL DISP=OLD SPECIFICATIONS OF THIS
//*           PROCEDURE ARE REQUIRED .......
//*
//*
```

MFSUTL PROCEDURE

```
//          PROC RGN=360K,SOUT=A,SNODE=IMSVS,
//               SOR=NOLIB,MBR=NOMBR,PXREF=NOXREF,
//               PCOMP=NOCOMP,PSUBS=NOSUBS,PDIAG=NODIAG,
//               COMPR=NOCOMPRESS,COMPR2=COMPRESS,
//               LN=55,SN=8,DEVCHAR=0
//S1        EXEC PGM=DFSUPAA0,REGION=&RGN,
// PARM='&PXREF,&PCOMP,&PSUBS,&PDIAG,&COMPR,
// LINECNT=&LN,STOPRC=&SN,DEVCHAR=&DEVCHAR'
//*SYSLIB - USER OPTION
//SYSIN     DD   DSN=&SNODE..&SOR.(&MBR),DISP=SHR
//REFIN     DD   DSN=IMSVS.REFERAL,DISP=OLD
//REFOUT    DD   DSN=IMSVS.REFERAL,DISP=OLD
//REFRD     DD   DSN=IMSVS.REFERAL,DISP=OLD
//SYSTEXT   DD   DSN=&&TXTPASS,UNIT=SYSDA,
//               SPACE=(CYL,(1,1)),DCB=BLKSIZE=800
//SYSUT3    DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4    DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//DUMMY     DD   DSN=IMSVS.PROCLIB(REFCPY),DISP=SHR
//UTPRINT   DD   SYSOUT=&SOUT
//SYSPRINT DD    SYSOUT=&SOUT,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=1330)
//SYSUDUMP DD    SYSOUT=&SOUT
//SEQBLKS   DD   DSN=&&BLKS,DISP=(NEW,PASS),
//               UNIT=SYSDA,SPACE=(CYL,(1,1))
//S2        EXEC PGM=DFSUNUB0,REGION=&RGN,
//               PARM='&COMPR2,DEVCHAR=&DEVCHAR',
//               COND=(8,LT,S1)
//SEQBLKS   DD   DSN=&&BLKS,DISP=(OLD,DELETE)
//UTPRINT   DD   SYSOUT=&SOUT,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=1330)
//SYSUDUMP DD    SYSOUT=&SOUT
//FORMAT    DD   DSN=IMSVS.FORMAT,DISP=OLD
//DUMMY     DD   DSN=IMSVS.PROCLIB(FMTCPY),DISP=SHR
//SYSPRINT DD    SYSOUT=&SOUT
//SYSUT3    DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4    DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//*
//*
```

GROWING FROM DB TO DB/DC

Although this is not an IMS/VS requirement we recommend that DB-only users wishing to upgrade to a DB/DC system re-do the entire IMS/VS installation, following the steps outlined in the section "INSTALLING IMS/VS DB/DC." The only step that should be omitted is the allocation and construction of the application libraries DBDLIB, PSBLIB and PGMLIB. The sample job stream in IMSVS.PRIMEJOB is constructed in such a way that the scratching and allocating of these libraries is done in a separate job (SAMPI15) which can be omitted when doing the installation.

INSTALLING IMS/VS UNDER OS/VS2-MVS

The installation of IMS/VS under OS/VS2-MVS is much the same as described for OS/VS1 in the first part of this chapter.

THE INSTALLATION JOBS

The jobs necessary to install IMS/VS under OS/VS2 MVS are, in general, the same as for OS/VS1. The differences are listed below and/or included in IMSVS.PRIMEJOB with a prefix of SMVS.

The following exceptions/additions apply:

1. The IMSCTRL macro of the Stage 1 definition should specify:

   SYSTEM=(VS/2,BATCH,3.7) for a DB installation (//SAMPI21)
   or
   SYSTEM=(VS/2,ALL,3.7) for a DB/DC installation (//SAMPI22, //SAMPI23)

2. Both VTAM and the IMS/VS online control region run as authorized subsystems under MVS. You must include the libraries from which IMS/VS, VTAM and NCP/VS are loaded and executed in the appropriate authorization tables. Note that you should not concatenate IMSVS.RESLIB with unauthorized libraries such as PGMLIB on the STEPLIB or JOBLIB DD statement of the IMS online control region procedure, as this will cause the job step to become unauthorized. The DL/I, MPP, and BMP regions do not require IMS/VS to run as an authorized subsystem.

3. If you choose to concatenate IMSVS.RESLIB to SYS1.LINKLIB in LNKLST00, the node IMSVS may not be used as a CVOL pointer. If you wish to use it as a CVOL pointer you should consider renaming the RESLIB. In our example the equivalent of job //SAMPI01 in the OS/VS1 installation, job //SMVSI01 builds an IMSVS CVOL pointer using Access Method Services. This job requires selectable unit 8 (SU8) to be installed in your OS/VS2(MVS) system. If you don't have SU8 installed, you cannot build an index structure for node IMSVS in the CVOL on IMSPRM. Instead you should catalog the IMSVS data sets in a VSAM catalog.

4. The resource clean-up module DFSMRCL0 must be link-edited into SYS1.LPALIB, and the IEAVTRML CSECT of module IGC0001C in SYS1.LPALIB must be updated. Jobs //SMVSI27 and //SMVSI33 show an example of the JCL to do this. The actual CSECT offset is in general 00. For more details see "Clean-up Routines" in the "OS/VS2 System Programming Library: Supervisor." After this job has been executed the system must be re-IPLed with the CLPA option.

5. The abend formatting module DFSAFMD0 must be link-edited into SYS1.LPALIB under the name IGC0905A. See job //SMVSI27.

6. If your MVS system uses JES2, you must add IMSVS.PROCLIB to the concatenation for PROC00 in the JES2 reader procedure. A job for this update is not provided in our sample jobs because of the critical nature of the JES2 procedure. A JCL error in this procedure leaves MVS without any readers, printers, or initiators. Another system must be used to correct the error. Therefore, it is recommended that this update be performed by the MVS system programmer.

   Certain considerations are involved when concatenating the IMSVS.PROCLIB to the PROC00 DD statement in the JES2 procedure:

   - the volume with the named data set must be available at every IPL.

   - the data set referenced first must have the largest block size or a 'DCB=BLKSIZE=' override parameter on the DD statement. Some procedures generated by IMS/VS system definition reference IMSVS.PROCLIB members as input to the linkage editor, which might have a blocksize restriction in your installation.

   - the named PROCLIB data set must be cataloged on the master catalog or must be referenced by the 'UNIT=' and 'VOL=SER=' keyword parameters in its DD statement. If cataloged in the master catalog, you cannot use the node name as a CVOL pointer.

   - the most elegant solution is to copy the IMS/VS procedures to SYS1.PROCLIB, or to copy the IMSVS.PROCLIB under a different name to one of the system resident volumes and catalog it in the master catalog.

7. The VTAM storage pool specification on //SAMPI54 should be adapted to your installation and VTAM level. Notice that the recommended IOBUF and PPBUF buffersize of 336 must be the same as the UNITSZ= value on the HOST macro of the NCP (job //SAMPI61).

8. The program name on the execution statement of the VTAM start procedure should be changed from ISTINA01 to ISTINM01 (job //SAMPI55).

9. The program name on the execution statement of the GTF procedure should be changed from HHLGTF to AHLGTF (job //SAMPI56).

10. The UNITSZ value on the HOST macro in job //SAMPI61 must be equal to the buffer size of the IOBUF and PPBUF storage pools in job //SAMPI54. See also note 7 above.

11. Whether or not the IMS/VS CTL region is executed as a system task or a problem task is dependent upon its starting as a system task via the OS/VS console or its starting as a problem task via JES.

THE SAMPLE JOBS

The sample jobs are the same as for OS/VS1. If you don't have SU8 installed, you must build the generation data groups in the VSAM user catalog (job //SAMP009). In addition, all data set names of the generation data groups (log and image copy data sets) in the sample jobs should use the node IMSPRIME instead of IMSVS. This is due to the difference in the VSAM catalog mechanism for generation data groups for OS/VS1 and OS/VS2 without SU8.

Executing the Sample Jobs with OS/VS2 MVS

Executing of the OS/VS 2 MVS sample jobs can be best done by submitting them from IMSVS.PRIMEJOB. The following job can be used to read those jobs from their job library and submit them to an internal reader:

```
//SUBMIT    JOB     A,'IMS/VS-PRIMER'
//SUBMIT    PROC    JOB=TEMPNAME
//STEP1     EXEC    PGM=IEBGENER
//SYSPRINT  DD      SYSOUT=A
//SYSIN     DD      DUMMY
//SYSUT1    DD      DSN=IMSVS.PRIMEJOB(&JOB),DISP=SHR
//SYSUT2    DD      SYSOUT=(A,INTRDR),DCB=BLKSIZE=80
//          PEND
//SUBMIT    EXEC    SUBMIT,JOB=jobname
```

MAINTENANCE CONSIDERATIONS

To provide for the continuing exapansion of hardware and software functions, IBM provides at regular intervals new releases of IMS/VS. Before using a new release and/or function, you might want to test them in your environment, but isolated from your existing (production) system.

It is recommended that you maintain separate production and test version of the following system libraries: LOAD, GENLIB, DBSOURCE, OBJDSET, PROCLIB, MACLIB, MATRIX, JOBS, and RESLIB. From an application development point of view, you might want to maintain separate versions of DBDLIB, PSBLIB, FORMAT, REFERAL, AND PGMLIB.

System Modification Program (SMP)

The OS/VS System Modification Program (SMP) is available to IMS/VS users as an option. SMP is a facility that allows you to apply program temporary fixes (PTFs) or user modifications either selectively or as a group to VS1 or VS2 or the distribution libraries (DLIBs) associated with them. See the *OS/VS System Modification Program (SMP) System Programmer's Guide* for a detailed description of SMP. A sample SMP job stream is provided to demonstrate what must be done to maintain your IMS/VS libraries using SMP.

This sample SMP job stream is on file 4 of your Data Base System tape. It may require minor changes depending on your system configuration. A detailed description of this job stream and its use is included in the *Program Directory* which accompanies the IMS/VS distribution tape.

Regression Testing of New IMS/VS Releases

When you are installing a new IMS/VS release, it is recommended to perform some kind of regression test before you use this new IMS/VS release as you production system. The IMS/VS Primer function sample jobs, although not explicitly designed for this purpose, can be used as an initial test vehicle. When your installation grows, you might complement this with a subset of your production jobs and procedures.

Quite often, initial test cases used during development are also very useful for regression testing. Therefore they should be maintained even after the application goes into production.

CHAPTER 8. OPERATIONS

This chapter discusses the factors involved in operating IMS/VS online. It should be read in conjunction with the IMS/VS Primer operator's guides:

- *IMS/VS Primer Master Terminal Operator's Guide-VTAM*
- *IMS/VS Primer Master Terminal Operator's Guide-BTAM*
- *IMS/VS Primer Remote Terminal Operator's Guide*

These guides are examples of a Master Terminal Operator's guide (MTO guide) and a Remote Terminal Operator's guide (RTO guide) in either a VTAM or BTAM environment. The MTO Guide also has a detailed discussion of the IMS/VS (and VTAM) commands used in our subset.

WHAT'S NEEDED TO OPERATE ONLINE IMS/VS

An online system puts a greater demand on an operations staff than a pure batch system. We have categorized the extra work into four groups, called *functions*, which are described below. Because each organization's policies and structure will determine how the functions will be implemented, we have limited ourselves to a discussion of the characteristics of each. However, it is important that these functions be recognized and the responsibilities assigned to specific individuals in the organization.

Two of these functions, the Master Terminal Operator and the User Liaison Group, are also discussed in greater detail later in this chapter.

THE MASTER TERMINAL OPERATOR FUNCTION

This function has the responsibility for the operation of the IMS/VS online system, including:

- Starting and stopping the system and its resources.
- Displaying system information.
- Carrying out emergency recovery procedures as outlined by DB and/or DC Administration.

THE NETWORK CONTROL FUNCTION

This function has the responsibilty for the physical maintenance of the terminals and associated equipment in the network including:

- Interfacing with the suppliers of the communication lines and any other equipment in the network.
- Co-ordinating the installation of new terminals and associated equipment.

THE APPLICATION SUPERVISOR FUNCTION

This function has the responsibility for maintenance of all programs and transactions in specific applications. Persons performing this function should have a detailed knowledge of applications, and ideally should be involved in the original design and analysis phases for them. This responsibility includes:

- Handling all queries relating to that application which are routed to them by the Master Terminal Operator or the User Liaison group (see below).
- Handling problems such as an application program abend, or a request by a remote terminal operator for clarification of a user procedure.

THE USER LIAISON FUNCTION

This function provides the first point of contact for all remote terminal operators who experience problems with, or who have queries about, the online system. As such, they:

- Provide assistance in the analysis of such problems, whether they are related to hardware, software, or a specific application.
- Route problems or queries which they cannot resolve to the appropriate function: Network Control, Master Terminal Operator, or Application Supervisor.

Note: We recommend that the Master Terminal Operator never be contacted directly by remote terminal operators, but that all queries be routed through the liaison function.

THE MASTER TERMINAL OPERATOR (MTO)

As stated earlier, the person assigned to this function is responsible for the operation of the online IMS/VS system. We recommend that, during any one shift, one specific operator be designated as the MTO, and that he be the only one who uses the master terminal during that period. However you should ensure that more than one person in your installation is trained as an MTO to provide backup.

It is important that the MTO be familiar with all the operating procedures he may be called upon to use. It is also important that formal reporting procedures are established, so that he can document any problems he encounters. Examples of forms to be used for this purpose are shown in the sample MTO Guide.

The interface between the MTO and the other functions in the organization must be clearly defined. He should be given a list showing whom he is to contact in DC Administration, Network Control, User Liaison, and the Application Supervisor group.

THE MASTER TERMINAL OPERATOR'S GUIDE

This sample guide can be used easily by an operator. It is also a learning tool. It includes only those procedures used by an MTO, and does not cover procedures for error situations to be handled by a support group such as DC Administration. The sample guide is designed in such a way that it can be easily maintained and extended with additional procedures as the network is increased and new applications are added. This document is not meant to replace the *IMS/VS Messages and Codes Reference Manual*, but it is to be used in conjunction with it.

MODIFICATIONS TO THE SAMPLE MTO GUIDE

If you wish to use the sample guide in your installation you will need to make some modifications to tailor it to your own needs.

Functional Titles

The functions described above are referenced in the guide with the names: DC Administration, Application Supervisor, Network Control, and User Liaison. If you do not use these titles in your organization you should replace them with the appropriate titles. You should also file with the guide a list of the names and telephone numbers of the people responsible for each function.

OS/VS1 Installations

The sample guide is designed for use in an OS/VS1 installation. It assumes that you are running the IMS/VS control region in partition P1. If this is not correct for your installation, you should modify Chart B-2, "Initializing the IMS/VS Control Region."

MVS Installations

If you use MVS in your installation, you will have to modify the chart described above in the OS/VS1 section, and also Chart E-7 (OS/VS abends). In addition, you have to use bypass label processing (BLP) in the IMSLOGR DD statement of the BMP restart JCL (//SAMP474) if the BMP restart is across a CTL region restart. This is because OS/VS2 MVS restores its volume serial enqueue on the input log tape in the CTL region. This can also be avoided by stopping and restarting the CTL region immediately after the emergency restart, and before the BMP restart.

Subset Limitations

The MTO guide is designed upon the IMS/VS Primer Function subset as defined in Chapter 1, "Introduction." As a result, certain IMS/VS functions are not included in our MTO guide. This is particularly true for the enhanced disk restart. The rationale behind this is that we feel that you should first gain experience with log tape restart before using disk restart. However, once you are familiar with log tape restart, we encourage you to switch to disk restart and adapt your MTO guide to that respect.

Forms and Tables

The sample MTO Guide includes in Chapter 4 tables which describe the network in our sample system, and details of the sample programs, transactions and data bases. These tables should be changed to reflect the configuration of your installation.

Restart and Recovery JCL

In the sample MTO guide, the operator is told to run certain batch recovery/restart jobs. Figure 8-1 shows a table of these jobs, where they are referenced, and the corresponding sample jobs in IMSVS.PRIMEJOB.

JOB DESCRIPTION	CHART	SAMPLE
BMP Restart	A-1,E-5	SAMP474
System Log Terminator	H-2	SAMP492
Log Tape Recovery -- Part 1	H-3	SAMP490
Log Tape Recovery -- Part 2	H-3	SAMP491
Change Accumulation	I-1	SAMP481
Data Base Recovery	I-1	SAMP382
Batch Backout	I-1	SAMP384

Figure 8-1. Jobs requiring JCL modification

These charts should be updated to reflect the actual job names used in your installation, and should include descriptions of any JCL changes the operator has to make before running them; for example, how he specifies the log tape serial number.

We recommend that all these jobs be set up and tested before you go into production mode. This setup would include preparing restart JCL for <u>all</u> BMPs, and data base recovery jobs for <u>all</u> online data sets.

Log Tape Administration

The sample MTO guide assumes that the method of online log tape administration described in Chapter 6: "Recovery," is used in the installation. If you use a different scheme, you should modify the guide accordingly.

Application Operating Procedures

Chart J-1 of the sample guide is an index to operating procedures for application programs. You should extend this section to include operating procedures for all application programs.

TESTING THE MTO GUIDE

Once you have prepared the MTO guide for your installation, all the operators who will be master terminal operators should be given an opportunity to familiarize themselves with it. After that, all the procedures in the guide should be thoroughly tested. Even if you use the sample guide, you should carry out the tests, to ensure that the procedures are accurate for your environment, and the operators know how to use them.

These tests should be carried out in a controlled fashion. DC Administration in conjunction with the Operations Department should prepare a detailed schedule, showing what tests are to be done, how they are to be done, and what procedures in the guide they test. The tests should be designed in such a way that all the operating procedures are checked out. All operators should have an opportunity to perform all tests. In order to test some of the recovery procedures, certain types of system failures have to be simulated. Figure 8-2 shows a table of possible failures, and how they may be simulated.

During the tests, the operators should write down which procedures they use, any deviations they were forced to make from the standard procedures, and any error messages they received that were not documented. After the tests have been run, a meeting should be held to discuss the results. Any corrections should be made, and the procedures re-tested if necessary.

After the initial testing of the guide, the recovery procedures should be re-tested on a regular basis, say once a month. This is to ensure that the operators remain familiar with the procedures, and that no changes need to be made.

MAINTAINING THE MTO GUIDE

It is vitally important to the successful running of the online system that the MTO guide be kept up to date, and that any errors or omissions in it are corrected.

After the initial tests have been completed, a procedure should be set up whereby an operator who finds an error can document it to alert DC Administration. The progress of the error correction should be followed up on a regular basis.

As new applications, data bases, or terminals are added to the system, the configuration tables in the guide should be updated. The procedures should be re-tested and the guide updated if necessary, after a new release of IMS/VS or OS/VS is installed.

SYSTEM FAILURE	SIMULATED BY:
1. BMP abended or cancelled in error	Use MTO command /STOP REGION n ABDUMP
2. CTL region abended or cancelled in error	Use OS/VS modify or cancel command
3. MPP abended	Run TE4CONEW in test mode* reply 'ABEND' to DFS3125A message
4. MPP looping or in wait state	Run TE4CONEW in test mode* reply 'LOOP' to DFS3125A message
5. I/O error on log tape	Use a log tape that has previously been mutilated
6. OS/VS error (loop or abend)	Unplug/Switch off OS/VS system residence drive
7. Hardware error, no loss of main storage	Unplug/Switch off OS/VS system residence drive
8. Power failure, or hardware error with loss of main storage	Press System Reset, and re-IPL

* Transaction TE4CONEW in the sample system includes a testing option which can be invoked by entering 'T' in the CNG-FUNC field. Message DFS3125A will be issued by the MPP. See the IMS/VS Messages and Codes Reference Manual for more details on this message and its allowed replies.

Figure 8-2. Simulating System Failures.

Planning for IMS/VS Disk Restart

Once you are familiar with the IMS/VS log tape restart operation, you should consider implementing IMS/VS disk restart. The reason we did not include it in our subset is that even with disk restart, proper handling of log tapes and log tape restart is essential to a problem-free IMS/VS operation. The benefit of disk restart is that it reduces significantly restart time and operator tape handling. For more information on disk restart, you should refer to the base IMS/VS publications.

If you implement disk restart, you should at least:

- Update your MTC procedures
- Increase the space allocation of the disk restart data set IMSVS.RDS

USER LIAISON GROUP

Depending on the number of users of the system, and the organization, the user liaison group may consist of only one person acting as a buffer between the remote terminal operators and the MTO or it may consist of a number of people, who resolve most user queries themselves.

People performing this function should have a good knowledge of terminal operating procedures, and a broad overview of all the applications. They should normally be able to diagnose a user's problem to the extent of knowing whether to route the query to the MTO, Network Control, or the appropriate Application Supervisor. In a large installation, members of this group might have their own terminals, and be authorized to use a subset of the master terminal commands, such as START, STOP, DISPLAY, and ASSIGN.

REMOTE TERMINAL OPERATORS

The success of an online system depends largely on its acceptance by the users. To make it acceptable to these people, you must provide good training and documentation, so that their interface to the system is as smooth as possible. The term "Remote Terminal Operator" or "RTO" is used to describe users of the online system, who may be operating local or remote terminals. The word "remote" is used to distinguish them from the Master Terminal Operator.

TRAINING REMOTE TERMINAL OPERATORS

Generally, an RTO Guide, no matter how comprehensive, is not sufficient to train new terminal operators who may not be familiar with computer concepts, and, in some cases, may not know the application either. We recommend that, as terminals are installed in departments for the first time, a training team be sent to provide initial user education.

This team should consist of a person, or persons, who can give an introductory talk on computers, who know how to operate the terminals, and who understand the applications and the transactions that will be used in that department. This team should remain in close contact with these users until their initial problems have been overcome.

A training program should also be set up within the department, so that new users can be trained by those already familiar with the system. This training program should be formalized to ensure that the education is done thoroughly. It may be possible to set up dummy data base records on which terminal operators can practise. If this is the case, the procedures for them to access these records should be documented in a training guide.

THE RTO GUIDE

This document should be supplied to all terminal users. The manual entitled *IMS/VS Primer Remote Terminal Operator's Guide* is a sample of such a guide. The aim of this document is to provide a guide to using the online IMS/VS system for a terminal user who is unfamiliar with computers. However, it is not intended to be a self-sufficient education document for such a user, although it could be used as part of a training program.

MODIFICATIONS TO THE SAMPLE RTO GUIDE

If you wish to use this sample guide in your installation, you will probably need to make some modifications to suit your environment.

Functional Titles

The guide refers to the function "User Liaison" as the contact point for any problems. If you do not use this title in your organization you should replace the references with the appropriate name. A list of the names and telephone numbers of people in the User Liaison group should be filed with the guide.

Use of the Subset

The guide refers to the fact that a subset of IMS/VS is being used. It is assumed that the standards we have recommended for screen design have been adopted. If you do not follow these standards, the guide should be changed accordingly.

Conversational Processing

If you do not use conversational transactions in your system, you may wish to delete the references in the sample guide. This includes the description of conversational transactions in Chapter 1 and the operating procedures for conversational processing in Chapter 3.

Terminal Operating Procedures

In Chapter 2 of the sample RTO guide, the operator is told to ask the supervisor for a copy of the IBM manual, *Operator's Guide for IBM 3270 Information Display Systems*, GA27-2742. You may wish to select from this manual the appropriate sections for the type of terminal and keyboard being used, and include them in the guide itself.

Application Operating Procedures

Chapter 5 of the guide describes the operating procedures for the sample programs. You should extend this section to include the operating procedures for the transactions used in your installation. We suggest that each terminal user be given only the procedures for the transactions that she/he is authorized to use. If possible, you should include with the operating procedures samples of the screen layouts. These can be produced by using the copy feature on a remote 3277 screen if you have one in your installation, or by using the output of the MFS generation.

Problem Reporting Procedures

In Chapter 4 of the sample RTO guide, the operator is told to ask the supervisor for a copy of the IBM manual, IBM 3270 IDS -- Problem Determination Guide, GA27-2750, if the operator suspects that there is a hardware problem on the terminal. You may wish to select the appropriate sections of this manual for the type of terminal in use, and include them in the guide.

Chapter 4 also describes the procedure to be used if the user has a problem. Generally they are told to ask the supervisor to contact the user liaison group if they cannot overcome the problem themselves. Depending on your organization, you may wish to redefine the problem reporting procedure. However where many users are physically located close to each other, one person should be designated as the interface to the user liaison group. This is to avoid the possibility of all users of the system trying to contact user liaison simultaneously after a total system failure.

A supervisor requires additional information that is not in the sample RTO guide. All supervisors should be given operating instructions for any additional equipment, such as datasets or modems. Depending on their location, and the organization of the company, they may need to know how to call for IBM Customer Engineering support and any other suppliers involved in case of hardware errors.

MAINTAINING THE RTO GUIDE

In order to achieve and maintain an acceptance of the online system by the users, the RTO Guide must be accurate and up to date. This implies that any errors in the guide reported by the users should be carefully investigated and corrected in all copies of the guide. If a new release of IMS/VS is installed, the guide should be thoroughly checked, especially the section on IMS/VS error messages.

VTAM AND IMS/VS OPERATION

In our subset we consider the VTAM operation as an integral part of the IMS/VS operations. This implies that the MTO is also responsible for the VTAM operation. This assumption might not be valid for your own installation. Especially if multiple subsystems are using VTAM, you may prefer to assign the VTAM operation to the OS/VS system console operation or to a special VTAM operation group. In that case, you should adapt this chapter and the operating guides to your own environment. In any case, proper communication procedures between VTAM and IMS/VS operations must be established.

CHAPTER 9. OPTIMIZATION

In this chapter we will present basic guidelines for the monitoring and optimizing of IMS/VS applications. The optimization we are concerned about is the performance optimization, that is, the optimal use of computer resources.

This chapter consists of two parts. Part 1 deals with the optimization of IMS/VS batch applications. Part 2 covers the optimization of the online IMS/VS system.

IMS/VS BATCH PERFORMANCE MONITORING

There are numerous areas for optimization of batch applications using IMS/VS. The most important areas are:

- Data base structure, that is, data base design optimization.
- Physical data set attributes, that is, data set location and internal storage utilization.
- Data base usage by the application programs, that is, DL/I call sequences.

The first part of this chapter briefly addresses the above three areas. But before doing so, we will take a closer look at the available tools for performance monitoring.

The ultimate measure of performance is cost. This includes manpower and system cost. We will consider only the system cost. The most important performance factors for DL/I applications are the number of physical I/Os and number of DL/I calls per transaction. DL/I provides two facilities to monitor these:

- The DL/I buffer pool statistics
- The DB monitor

In addition, the standard OS/VS facilities such as SMF, GTF, etc., are often very useful.

THE DL/I BUFFER POOL STATISTICS

DL/I maintains statistics on the use of its VSAM and OSAM buffer pools. These statistics can be obtained by your application program via the STAT call, as is done by subroutine DFSOAST in IMSVS.PRIMESRC. This is normally done at the end of each DL/I application program. For more details on DFSOAST and its use, see "The Stat Call" in Chapter 4. For a description of the VSAM and OSAM buffer pools, see "DL/I Data Base Buffering Facilities" in Chapter 7.

THE VSAM BUFFER POOL STATISTICS

For each VSAM subpool, DFSOAST prints 4 lines: 3 heading lines and the statistics. The format of the data is:

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


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

BSIZ = the size of the buffers in this subpool. In final total, this is the total size of all subpools.
NBUF = the number of buffers in this subpool. In final total this is the total number of buffers in all subpools.
RET RBA = the number of retrieve by RBA calls received by the buffer handler
RET KEY = the number of retrieve by key calls received by the buffer handler
ISRT ES = the number of logical records inserted into ESDSs
ISRT KS = the number of logical records inserted into KSDSs
BFR ALT = the number of logical records altered in this subpool
BGWRT = the number of times the Background Write function was invoked by the buffer handler
SYN PTS = the number of synchronization calls received by the buffer handler
GETS = the number of VSAM GET calls issued by the buffer handler
SCHBFR = the number of VSAM SCHBFR calls issued by the buffer handler
FOUND = the number of times VSAM found the control interval requested already in the subpool
READS = the number of times VSAM read a control interval from external storage
USR WTS = the number of VSAM writes initiated by IMS/VS
NUR WTS = the number of VSAM writes initiated in order to make space in the subpool
ERRORS = the number of write error buffers currently in the subpool/the largest number of write errors in the subpool during this execution

Following are guidelines to the interpretation of the most important fields:

- Normally, 50% - 90% of the buffer handler requests (RET RBA + RET KEY) would be satisfied from the pool (FOUND). This parameter can be used to initially optimize the pool size.

- The number of I/Os (READS + USR WTS + NUR WTS) should be related to the number of transactions processed by the job. An increase in this during production could be a signal for reorganization.

- ERRORS should be zero. If not, insure the data base is recovered. See Chapter 6, "Data Base Recovery."

THE OSAM BUFFER POOL STATISTICS

If an OSAM data base used by the program, DFSOAST will also print the OSAM buffer pool statistics.

The format of the data is as follows:

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

   WRITTEN LOGICAL    PURGE RELEASE     RET    ISAM     ISAM
   AS NEW  CYL FMT     REQ.    REQ.  BY KEY  GT NXT    SETLS  ERRORS
   nnnnnnn nnnnnnn  nnnnnnn nnnnnnn   nnnnn   nnnnn    nnnnn   nn/nn
```

BLOCK REQ	=	number of block requests received
FOUND IN POOL	=	number of times the block requested was found in the buffer pool
READS ISSUED	=	number of OSAM reads issued
BUFF ALTS	=	number of buffers altered in the pool
OSAM WRITES	=	number of OSAM writes issued
BLOCKS WRITTEN	=	number of blocks written from the pool
NEW BLOCKS	=	number of new blocks created in the pool
CHAIN WRITES	=	number of chained OSAM writes issued
WRITTEN AS NEW	=	number of blocks written on OSAM chains
LOGICAL CYL FMT	=	number of logical cylinder formats
PURGE REQ.	=	number of buffer purge requests
RELEASE REQ.	=	number of buffer release requests
RET BY KEY	=	number of ISAM records retrieved by key
ISAM GT NXT	=	number of ISAM get next calls received by the buffer handler
ISAM SETLS	=	number of ISAM SETLs issued by the buffer handler
ERRORS	=	number of write error buffers currently in the pool / the largest number of errors in the pool during this execution

<u>Note</u>: Because ISAM is never used in our subset, its corresponding statistics should all be zero.

Following are guidelines to interpreting the most important fields:

- Normally, 50% - 90% of the block requests received (BLOCK REQ) would be served from the pool (FOUND IN POOL). Also notice that, on the average, multiple block requests are required for a single DL/I call. This parameter can be used to optimize the buffer pool size for the job.
- The number of OSAM reads (READS ISSUED) and OSAM WRITES should be related to the number of transactions processed by the job. An increase in these during production could be a signal for reorganization.
- ERRORS should be zero. If not, insure that the data base is recovered. See Chapter 6, "Data Base Recovery."

<u>THE IMS/VS DB MONITOR</u>

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 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 in the partition/region, 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. 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, assume 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.

USING THE IMS/VS DB MONITOR

The DB monitor formats and records performance-related data during the execution of the IMS/VS batch system. The DB monitor can be active during the entire execution of the IMS/VS batch job or it can be started and stopped from the system console. Typically, activating the DB Monitor for 15 to 30 minutes is sufficient to collect representative data.

Activation and Control

Including the parameter MON=Y in the PARM parameter of the JCL execute statement in the batch procedure makes the DB monitor active when batch system execution begins. (See "The DLIBATCH Procedure" in Chapter 7, "Installing IMS/VS.")

"DFS2216A MONITOR ACTIVE, MODIFY TO STOP MONITOR" prints on the system console whenever the DB monitor is initialized or started. To stop and restart the DB monitor, the system console operator can, at any time, enter:

```
    MODIFY jobname,STOP       (or F jobname,STOP)
```

"DFS2215A MONITOR INACTIVE, MODIFY TO START MONITOR" prints on the system console when the IMS/VS DB monitor receives and acts upon the modify command.

To reactivate the DB monitor, the console operator enters:

```
    MODIFY jobname,START          (or F jobname,START)
```

"DFS2216A MONITOR ACTIVE, MODIFY TO STOP MONITOR" prints on the console and indicates that the modify command was accepted.

DB Monitor Data Recording

The data produced by the DB monitor is recorded on either the IMS/VS system log or on a separate DB monitor log. The presence or absence of a DD card named //IMSMON in the batch procedure determines which log is used. If a //IMSMON DD card is included (and does not specify DUMMY), it specifies a separate DB monitor log on which the DB monitor records are to be written. If there is no //IMSMON DD card (or if the //IMSMON DD card specifies DUMMY), the DB monitor records are written on the IMS/VS system log.

When a separate DB monitor log is used, the system console operator may want to force an end-of-volume when stopping the monitor from the console. The modify command can be used to accomplish this.

```
    MODIFY jobname,STOPEOV        (or F jobname,STOPEOV)
```

If, for any reason, the DB monitor log data set specified on the //IMSMON DD card cannot be opened, the message "DFS2217I UNABLE TO OPEN MONLOG, MONITOR UNAVAILABLE" is displayed on the system console. The batch execution continues, but the DB monitor is inactive.

If I/O errors are encountered on the DB monitor log device, the message "DFS2219I I/O ERROR ON MONITOR LOG, MONITOR TERMINATED" is displayed on the system console. The batch execution continues, but the DB monitor is inactive.

Note: When STOPEOV is used, execution of the batch region does not continue until the succeeding OS/VS mount message is satisfied.

MODIFY Command Errors

If the jobname is entered incorrectly when entering the MODIFY command, an OS message informs the operator of the error. If some other error is made while entering the modify command, the message "DFS2218I MONITOR MODIFY SPECIFICATION INCORRECT" is displayed on the system console, followed by either DFS2215A or DFS2216A (described above).

DB MONITOR REPORT PRINT PROGRAM, DFSUTR30

The DB Monitor Report Print program (DFSUTR30) is an offline utility that organizes, formats and prints performance related data collected by the DB Monitor during execution of a IMS/VS batch job. The reports printed by this program are:

- Statistics from the VSAM and OSAM pools
- Program I/O
- DL/I Call Summary
- VSAM Statistics
- DB Monitor Overhead

Note: The DFSUTR30 program is dependent upon the data records on the data set produced by the DB Monitor. Records of various events are expected in pairs -- a start-event record and an end-event record; events are not counted and reported unless both are received.

Definition of Terms Used in the Reports

The following are explanations of key terms used in the reports to describe activities or subtasks in the IMS/VS partition/region.

ELAPSED TIME: Time recorded by the time of day clock, from the start of the activity or subtask until the end of the activity or subtask.

IWAIT: A wait for an I/O or another resource which occurred during the processing of a DL/I call.

IWAIT TIME: Elapsed time, during which an IMS/VS subtask was inactive, waiting for a resource or the completion of an event. An exception to this definition occurs when the IWAIT time is related to VSAM activity. In this case, the IWAIT time is defined as the elapsed time between the entry to and the exit from the VSAM routines. During this VSAM time, an I/O access and wait may or may not have occurred.

NOT IWAIT (ELAPSED TIME -- IWAIT TIME): Elapsed time minus all IWAIT time identified for the subtask or activity. It includes any time spent by OS/VS, or by any other higher priority tasks running in the systems when the IMS/VS region was interrupted and dispatchable, and when the subtask to which the CPU time refers had been executing at the time of the interrupt. Note that this may approximate total CPU time if the IMS/VS-DB region is the high priority task and if no low priority tasks are causing interrupts.

CPUTIME: Actual CPU time used by an application program.

SCHEDULE TO 1ST DL/I CALL: Elapsed time accumulated for the following actions to occur: the region to gain control after being scheduled, plus the program either to be located in the region by contents supervision of OS/VS, or to be brought in by program load, plus the program to issue the first DL/I call.

ELAPSED EXECUTION TIME: Elapsed time from IMS/VS dispatch of the first DL/I call by a program until the IMS/VS termination of that program.

MAXIMUM TIME: Longest single time duration noted for an event.

TOTAL TIME: Sum of all the time durations noted for a group of events.

MEAN TIME: Quotient of the total time (above) divided by the number of occurrences of a certain event.

ISRT KSDS: A count of the root segments inserted into a SHISAM data base or index data base. (This count should not be confused with the ISRT totals in the DL/I summary report.)

ISRT ESDS: The number of times the insertion of a root or dependent segment required a new logical record for the new segment (SHISAM, HIDAM). (This count should not be confused with the ISRT totals in the DL/I call summary report.)

Note: All the above times are given in microseconds.

How to Execute the DB Monitor Report Print Program

Job //SAMP293 in IMSVS.PRIMEJOB shows the JCL to execute the DB Monitor Report Print Program, DFSUTR30. The ANALYSIS DD statement should specify DUMMY,DCB=BLKSIZE=80 in our subset.

This job prints the monitor output collected during execution of the customer order processing program with job //SAMP272.

The output, which is listed in Chapter 3 of the *IMS/VS Primer Sample Listings*, will be referred to in the following discussion of the various generated reports.

Statistics from the VSAM and OSAM Buffer Pools

These summary reports are formatted displays of the contents of selected statistics of the OSAM buffer pool and VSAM buffer subpool(s) that were collected for batch activity over the entire run of the DB Monitor.

The pool ending values and the difference between starting and ending values cannot be computed for these summary reports unless there are pool ending statistics on the trace tape. The OSAM buffer pool ending values are recorded only if the DB monitor is ended before the IMS/VS batch job is terminated.

The following message is printed if the summary reports are not printed:

```
NO DATA BASE BUFFER DATA AT END TIME ON MONITOR LOG TAPE:
****DATA BASE BUFFER POOL CANCELLED****
```

The VSAM Buffer Subpool Summary report is not produced if the VSAM facility is not invoked through the IMS/VS system definition or if the ending values are not written on the trace tape. In either instance, the following information message is printed:

```
NO VSAM BUFFER POOL TRACES ON MONITOR LOG TAPE:
****VSAM BUFFER POOL REPORT CANCELLED****
```

For an example of these statistics, see the sample output (pages 1,2 and 3) of job //SAMP293 in Chapter 3 of the *IMS/VS Primer Sample Listings*. For a description of these statistics, refer to the previous section "DL/I Buffer Pool Statistics."

Program I/O Report

This is a summary report of the total and mean IWAIT intervals recorded for I/O IWAITs caused by DL/I calls by the program during the trace.

The data is arranged by PCB name, ddname, and module identification of the module that IWAITed. The data under the column heading "IWAITS" indicates the number of times that DL/I calls for the associated PCB were required to wait for I/O activity to complete. The data set for which the I/O took place is indicated by the ddname. Entries under the heading "Module" are abbreviated identifications for IMS/VS-DB modules. The cross-reference is:

Abbreviated ID	Module Name
DBH	DFSDBHR0
DLE	DFSDDLE0
VBH	DFSDVSM0

PCB subtotals and a batch total are provided.

For an example of this report, see the sample output (page 4) of job //SAMP293, in Chapter 3 of the _IMS/VS Primer Sample Listings_.

The two main effects to be noted from this report are:

- If the number of IWAITS per transaction increases, it may be necessary to reorganize the data set in question.

- If two or more data sets with high activity are on the same disk drive, there may be a contention problem.

DL/I Call Summary Report

This is a compact DL/I call summary report for the trace. All DL/I calls issued by the program during the trace are arranged as follows:

- PCB name.
- For each PCB, the call function employed.
- For each call function, the segment accessed and its level number.
- For each segment, the return code obtained.

For each line in this report, the number of DL/I calls recorded, the IWAITs per call, and both the average and maximum elapsed time and Not-IWAIT times are given. A batch total of DL/I calls is provided at the end of the report.

For an example of this report, see the sample (page 5) output of job //SAMP293 in Chapter 3 of the _IMS/VS Primer Sample Output Listings_. The column entries, from left to right, are:

- PCB NAME — The 8-character PCB name.
- CALL FUNC. — The 4-character DL/I call function.
- LEV NO. — The data base level number reached in this call, or blank.
- SEGMENT — The 8-character segment name accessed by this call.
- STAT CODE — The status code returned by this call.
- DL/I CALLS — The number of calls noted having the unique combination of the above five attributes.
- IWAITS — The number of I/O WAITs observed for the calls.
- IWAITS/CALL — Quotient of the above two items.
- ELAPSED TIME or NOT IWAIT — For an explanation of these terms, see "Definition of Terms Used in Reports" earlier in this chapter.

The main effects to be noted from this report are:

- If the number of IWAITs per DL/I call increases, this may signal the requirement for data base reorganization.

- A relatively high number of IWAITS per DL/I call may indicate a small data base CI/blocksize, or buffer pools that are too small.

- Unnecessary calls issued by the application program can be traced by checking the report with the program specifications and detailed flow.

- Calls with very high IWAIT counts may indicate insufficiently qualified calls, which result in data base scans.

  Note: Frequent IWAITs with a very long elapsed time may be a result of excessive paging or frequent de-activations. This should be discussed with the OS/VS system programmer.

VSAM Statistics Report

This report (page 6) provides statistics on a per call basis for changes in up to 13 selected subpool statistics between the start and end of VSAM activity. The statistics reported are:

RET RBA	Number of retrieves by RBA calls received by the buffer handler.
RET KEY	Number of retrieves by key calls received by the buffer handler.
ISRT ES	Number of logical records inserted into ESDSs.
ISRT KS	Number of logical records inserted into KSDSs.
BFR ALT	Number of logical records altered.
BKG WTS	Number of times background write function invoked.
SYN PTS	Number of synchronization calls received by the buffer handler.
GETS	Number of VSAM GET calls issued by the buffer handler.
SCHBFR	Number of VSAM SCHFR calls issued by the buffer handler.
FOUND	Number of times VSAM found the control interval requested was already in the subpool.
READS	Number of times VSAM read a control interval from external storage.
USR WTS	Number of VSAM writes initiated by IMS/VS.
NUR WTS	Number of VSAM writes initiated in the subpool.

The report contains a set of the above statistics for each combination of PCB, call function and ddname detected in the trace. An occurrence count is printed. Each set of statistics is a summation of the changes in all subpools divided by the number of occurrences. Summary lines show totals for each PCB, for each ddname under the PCB, and for the complete trace.

DB Monitor Overhead Report

This report (page 7) provides the total elapsed time during which the DB Monitor was active and the total of the time intervals between entry to and exit from the DB Monitor module. The report also includes the number of DB Monitor records that were written and the average DB Monitor time per entry.

DATA BASE DESIGN OPTIMIZATION

Because the data base design optimization should be started before the physical implementation, the previously discussed tools are not applicable. Instead, we will introduce a simple paper and pencil technique to evaluate alternate designs. This technique will concentrate on the number of physical I/Cs, and the number and complexity of DL/I calls per transaction. This is because, as stated before, these two factors are the most important performance factors for data base processing.

DATA BASE LOAD FACTORS PER TRANSACTION

For all transactions, data base load factors can be established. A transaction load factor represents the CPU power needed for the processing of that particular transaction. It is a relative factor, not an absolute one. Its sole role is to provide a base for comparison between alternative designs.

Note: If more accurate performance prediction in the design phase is required, then design tools such as DBPROTOTYPE/VS should be considered.

Transaction Load Factor Units

The table in Figure 9-1 gives basic estimates of transaction load factor units for DL/I calls or data base I/O.

COMPONENT	UNITS
A GU CALL	1.1
A GN CALL	.9
A REPL CALL	.5
A ISRT CALL	1.7
A DLET CALL	1.8
RETRIEVE OF ONE SEGMENT	.5
INSERT OF ONE SEGMENT	2.4
REPLACE OF ONE SEGMENT	1.6
DELETE OF ONE SEGMENT	2.1
A DATA BASE I/O	5

Figure 9-1. Transaction Load Factor Units

The following considerations apply:

- A single DL/I call can incur multiple segment accesses, that is, to follow a twin chain.
- If HDAM, each access of a synonym on the anchor point chain is one segment retrieve.
- If HIDAM, the access of the primary index data base should be counted as one additional segment access.

- For replace, insert and delete each segment occurrence to be processed must be counted separately.

Example

As an example we use the logical CUSTOMER ORDERS data base (Figure 2-26). A GU call, for instance, to the third SE2OPART occurrence for a given customer order would cost:

Segment accesses:

		Units	
GU call		1.1	
Retrieve of index pointer segment		.5	
Retrieve of root segment		.5	
Retrieve of first, second + third ORDER LINE segment		1.5	
Retrieve of logical parent, PART segment		.5	
	subtotal	4.1	

I/Os:

KSDS index component		5	
KSDS data component		5	
ESDS for root segment + dependents		5	
ESDS of logical parent		5	
	subtotal	20	
	gross total		24.1

Assumptions:

1. The ORDER LINE segments are in the same CI as their root.

2. None of the requested CIs is in the buffer pool. Quite often the I/O for the index component is not necessary. Also, for the get next call, most retrieves are satisfied from the pool.

Once again, it should be realized that the above method gives only a rough estimate of the cost of a particular call. Its main use is to evaluate possible alternative designs.

DATA BASE DESIGN CHECKLIST

The following checklist gives an overview of the most important considerations/guidelines for data base design optimization. These considerations/guidelines are oriented towards performance. Sometimes, they will contradict application requirements. In such cases, a compromise must be made based on a cost/function analysis.

- Use a structure no more complex than necessary.

- Keep frequently accessed segments near the top and to the left of the hierarchy.

- Avoid widely varying segment sizes for volatile segments in the same data space.

- Check the requirement for any segment type whose relative frequency under its parent is one, or whose prefix length is greater than or equal to its data length.
- Oversegmentation results in many DL/I calls and longer reorganization times.
- Undersegmentation results in less security and less data independence.
- Avoid movement of data from one data base to another.
- Avoid secondary indexing on highly volatile source segments.
- Use secondary indexing for alternate entry, not sequential processing.
- If logical relationships exist, place the real logical child so that the physical path is the most active path. Also consider placing the real logical child on the longest twin chain.
- Sequencing of the logical twin chain is expensive on insert and delete processing.
- Avoid long twin chains, particularly logical twin chains.

OPTIMIZATION OF PHYSICAL IMPLEMENTATION

Because DL/I data bases are stored in OS/VS data sets and/or VSAM data spaces, the normal performance guidelines for these apply. In addition, the following considerations/guidelines should be observed.

- Keep segments to be accessed in the same block as the entry segment.
- Use HDAM whenever possible.
- Process HDAM data bases sequentially by inserting the randomizing routine into a sort exit and sorting into a root anchor point sequence.
- GN processing at the root level in HDAM proceeds in physical root anchor point sequence with synonyms maintained in logical key sequence off their root anchor point.
- If root key sequence is required in HDAM, consider a secondary index (for limited use) or a randomizing routine which assigns root anchor point addresses in key sequence.
- The expected I/Os required to access a HDAM root with a general randomizing module, is between 1.1 and 1.2 if the number of roots per block is 5 or more and the block and the RAP densities are less than 80%.
- Do not specify twin backward pointers for HDAM roots.
- Specify PTR=TB or none (NT) for the HIDAM root segment. GN processing at the root level in HIDAM proceeds along the physical twin chain if TB has been specified, or via the index if not. Note that a GN root with key qualification always proceeds via the index if the call cannot be satisfied at the current position.
- Specify TB pointers on segments to be deleted to improve delete performance for long twin chains (that is, more than 3 to 5). This is particularly important for the logical child segment.

- Specify LCL in the logical parent and LTB in the logical child if, on average, there are more than 2 logical children per logical parent.

- Do not define a sequence field for the virtual logical child, unless really needed.

- Check report from data base unload to identify long twin chains.

- For insert activity against a HIDAM data base, specify free space in the DBD.

- The HIDAM primary index should be reorganized frequently to minimize I/O and regain space from deleted entries.

- Leave sufficient free space for anticipated inserts prior to reorganization.

- HD free space within the block should be large enough for the largest segment type.

- When defining KSDS data sets, select the IMBED and replicate options and provide free space for insert activity.

- For initial load select speed option in VSAM define. Specify INSERT=SEQ in the DFSVSAMP DD * data set for initial load or any mass insert after initial load to maintain KSDS free space.

- To insure that KSDS index control intervals remain in main storage, provide a unique control interval size (1K is a good number) and provide enough buffers to hold all index set CIs plus at least one sequence set CI for each KSDS. Remember that sequence set and index set CIs contend for buffers in the same shared resource subpool on a demand basis.

- VSAM buffer pools and/or control blocks can be page fixed in main storage by specifying VSAMFIX=(BFR,IOB) in DFSVSAMP data set.

OPTIMIZATION OF APPLICATION PROGRAMS

In general, the number and complexity of DL/I calls determine to a large extent the performance of a DL/I application program. The following considerations/guidelines should be observed:

- Reduce the total number of calls to DL/I by using path calls and more fully qualified SSAs.

- Save data in virtual storage rather than reissue calls.

- Do not issue a get call prior to ISRT to check for prior existence.

- Use multiple PCBs when referencing data in two parts of the data base or data base record.

- Sort batch transactions to match the physical order of the data base.

OPTIMIZING THE IMS/VS ONLINE SYSTEM

The means to optimize an IMS/VS online system are essentially the same as discussed for the batch-only system in the first part of this chapter. The two key performance factors introduced in that part, the number of DL/I data base calls and the number of physical I/Os, retain their significance. They are expanded here to include DL/I message calls and physical I/Os for the data communication lines, the message queues, and other data sets used by the online system.

ONLINE PERFORMANCE MONITORING

The online IMS/VS system contains several tools to monitor its performance. Those used most often are:

- The online buffer pool statistics, which can be requested at regular time intervals by the master terminal operator (MTO).
- The log data set statistical analysis utility.
- The DC monitor and its DC monitor report print program.

In addition, the standard OS/VS facilities such as SMF, GTF, etc., are often very useful.

THE ONLINE BUFFER POOL STATISTICS

The online buffer pool statistics provide information on the usage of the IMS/VS data and control block buffer pools. These statistics are displayed as a result of the /DIS POOL ALL command on the master terminal. See the MTO Guide for more details on how to use this command. All displayed counts are relative to the last (re)start of the IMS/VS control region; all counters are reset to zero during restart processing. In general all counts should be related to the number of messages or transactions processed.

Note: Performance interpretation, especially of the number of physical I/Os, should not be based on a short session. The number of I/Os will be relatively high during the beginning of a session because initially all needed control blocks must be read in. It is therefore recommended that you disregard the first half hour of a session.

The following sections briefly discuss the statistics for each pool and give guidance for their initial interpretation. Figure 9-2 shows the format of these statistics as they appear on the 3270 display screen.

```
MESSAGE QUEUE POOL:   BFRS/SIZE  10/1500
ENQ      95  DEQ      90  CAN      55  WAIT      0  I/O      8  ERR   0
MESSAGE FORMAT POOL:  SIZE 20480  SPACE 17192
REQ1    193  I/O      13  DIR      5  WAIT     13  FREE  13184  ERR   0
DATA BASE BUFFER POOL: BSIZE    1024
  RRBA      0 RKEY      0 BFALT     0 NREC      0 SYN PTS    11
  NMBUFS  12 VRDS      6 FOUND    20 VWTS      0  ERRORS 00/00
DATA BASE BUFFER POOL: BSIZE    2048
  RRBA    988 RKEY      8 BFALT    45 NREC     12 SYN PTS    11
  NMBUFS  20 VRDS    171 FOUND   302 VWTS     13  ERRORS 00/00
                        DISPLAY LINES WAITING     PASSWORD:
/dis pool all
```

```
DATA BASE BUFFER POOL: BSIZE   53248
  RRBA    988 RKEY      8 BFALT    45 NREC     12 SYN PTS    11
  NMBUFS  32 VRDS    177 FOUND   322 VWTS     13  ERRORS 00/00
 DMBP BUFFER POOL:
 SIZE    8192   FREE    5936  HIGH    2256
 PSBP BUFFER POOL:
 SIZE   20480   FREE    9648  HIGH   10832
 CIOP BUFFER POOL:
 SIZE   40960   FREE    7568  HIGH   37440
 MAIN BUFFER POOL:
                        DISPLAY LINES WAITING     PASSWORD:
/dis pool all
```

```
SIZE   12288   FREE    7768   HIGH    4616
CWAP BUFFER POOL:
SIZE   12288   FREE   12048   HIGH    2632
PSBW BUFFER POOL:
SIZE    4096   FREE    4096   HIGH    1576
DBWP BUFFER POOL:
SIZE    4096   FREE    4096   HIGH     280
*78180/213051*



                                          PASSWORD:
/dis pool all
```

Figure 9-2. Online Pool Statistics Display Format

MESSAGE QUEUE POOL

The message queue pool statistics provide the number of messages received and sent. The following parameters are displayed:

BFRS/SIZE: The first value is the number of buffers. The second value is the size of a single buffer, as defined in the IMS/VS system definition.

ENQ: Is the number of messages enqueued in the input/output message queue.

DEQ: Is the number of messages dequeued from the message queue after their processing or transmission to their destination.

CAN: Is the number of messages cancelled. A message is cancelled if rejected as an invalid transaction or during processing of some commands. This figure is typically very low.

WAIT: Is the number of I/O waits issued. This figure should be low (<10% of ENQ). If not, you may want to increase the number of queue buffers.

I/O: Is the number of physical I/Os against the message queue data sets. This figure should be typically less than or equal to the ENQ figure. If higher, you may want to increase the number of queue buffers.

Notes:

1. The number of I/Os includes the I/Os needed to format/restore the queues during IMS/VS (re)start.

2. The number of queue buffers as defined at system definition can be overridden via the QBUF= parameter of the IMS online procedure.

ERR: Is the number of I/O errors for the message queue data set. This should be zero. If not, it is recommended to do an emergency restart with BUILDQ or a cold start as soon as possible.

MESSAGE FORMAT POOL

The following parameters for the MFS buffer pool are displayed:

SIZE: Is the total size in bytes of the MFS buffer pool.

SPACE: Is the available space in the MFS buffer pool. This is the SIZE minus the space needed for the directory index, DCBs, pool control blocks, and the fetch request elements (FREs). One FRE of 40 bytes is required to store a MFS format block in the pool.

REQ1: Is the number of block requests from the pool. Such requests are made for MIDs, MODs, DIFs, and DOFs needed by MFS processing.

I/O: Is the number of physical I/Os to the IMSVS.FORMAT data set. If more than 50% of REQ1, you should consider increasing the MFS buffer pool size.

DIR: Is the number of directory I/O operations. This should be very low in the sample system because we will make the directory index resident during MFS processing. This is done with the MFS service utility. See "MFS Control Block Generation" in Chapter 3, "Data Communication Design," and job //SAMP425, last step, in IMSVS.PRIMEJOB.

WAIT: Is the number of immediate fetch I/Os for the IMSVS.FORMAT data set. If more than 50% of REQ1, you should:

1. Check if your MIDs refer to their corresponding MODs (NXT=parameter in the MSG statement) whenever possible.

2. Consider increasing the size of the MFS buffer pool.

FREE: Is the amount of free space in the pool. If this amount is constantly above, say 2K, in your production system, you may want to decrease the MFS buffer pool size. However, a constant high value may also indicate too few FREs, which means the available space cannot be used.

ERR: Is the number of I/O errors for the IMSVS.FORMAT data set. This should be zero. If not, you should restore the MFS library.

Note: Because the IMSVS.FORMAT and IMSVS.REFERAL data sets are standard OS/VS partitioned data sets, normal library back-up and restore procedures can be used. However, they must be dumped and restored at the same time. Two sample procedures, MFSBACK and MFSREST, are provided in IMSVS.PROCLIB.

Adjusting MFS Buffer Pool Specifications

During IMS/VS system definition, we specified a MFS pool size of 18K and a number of FREs of 40. You can change these values in the IMS/VS control region procedure. See the IMS procedure in Chapter 7, "Installing IMS/VS." The parameter FRE specifies the number of fetch request elements. The parameter FBP specifies the number of 1K blocks in subpool 0 to be allocated to the MFS buffer pool.

DATA BASE BUFFER POOLS

Separate statistics are displayed for:

- The OSAM buffer pool (not shown in Figure 9-2)
- Each VSAM subpool
- The combined VSAM pool, that is, the total of the VSAM subpools

SIZE: Is the total pool size in bytes for the OSAM buffer pool.

REQ1: Is the number of internal DL/I requests to the pool.

REQ2: Is the number of above requests satisfied from the pool, plus number of new blocks created.

READ: Is the number of OSAM reads.

BISAM: Should be zero, because we are not using ISAM.

WRITES: Is the number of OSAM writes.

KEYC: Should be zero, because we are not using ISAM.

LCYL: Is the number of OSAM logical cylinder formats.

PURG: Is the number of synchronization calls received.

OWNRR: Is the number of release ownership requests.

ERRORS: Number of permanent errors (that is, blocks subject to a write error) now in the pool/total of these since last IMS/VS start-up. This should be zero. If not, shut the system down as soon as possible and recover the affected data base.

RRBA: Is the number of retrieve by RBA calls (direct retrieves) received by the buffer handler.

RKEY: Same as above for retrieve by key.

BFALT: Is the number of logical records altered.

NREC: Is the number of logical records inserted into ESDS/KSDS.

SYN_PTS: Is the number of synchronization points that involved this (sub)pool.

NMBUFS: Is the total number of buffers in this (sub)pool.

VRDS: Is the number of VSAM reads.

FOUND: In the number of requests (RRBA+RKEY) satisfied from the pool.

VWTS: Total number of VSAM writes.

ERRORS: Same as before.

DMBP BUFFER POOL

This pool contains the DMBs, which are the expanded DBDs of the data bases used by the control region. DMBs are loaded from the IMSVS.ACBLIB during CTL region initialization when defined as resident during IMS/VS system definition or during data base open processing. Data bases are opened during the processing of the first DL/I call against a PCB which references the data base.

SIZE: Is the size in bytes of the pool.

FREE: Is the available free space in the pool.

HIGH: Is the maximum amount of space used in the pool since the last CTL region start-up.

Adjusting the DMBP Pool Size

The need to increase this pool can best be interpreted from the DC Monitor report. See its description later in this chapter. If the HIGH value in your production system is constantly 2K or more below the SIZE value, consider decreasing the DMBP pool size. This pool size is specified in 1K blocks (rounded to the OS/VS page size) in the DMB parameter of the IMS/VS control region procedure. See the IMS/VS procedure in Chapter 7, "Installing IMS/VS."

PSBP BUFFER POOL

This pool plays the same role for the PSBs as the DMBP pool does for the DMBs. The same considerations regarding its size apply. However, because PSBs are typically between 2 and 8K you should be careful in adjusting the size downwards based on the difference between the SIZE and HIGH values. For instance, if this difference is 4K, you might still be swapping two 6K PSBs.

Note: The size of the PSBs is listed by the ACBGEN utility in message DFS940I. See the output of job //SAMP420 in Chapter 3 of the *IMS/VS Primer Sample Listings* manual.

The DC Monitor REGION IWAIT report lists the number of PSB loads. The corresponding parameter in the IMS/VS control region procedure is PSB. See the IMS/VS procedure in Chapter 7, "Installing IMS/VS."

CIOP BUFFER POOL

This is the communication line buffer pool. The SIZE, FREE, and HIGH numbers have the same meaning as for the DMBP pool. Also, the same considerations regarding adjusting its size apply. The corresponding IMS/VS control region procedure parameter is TPDP. See the IMS/VS procedure in Chapter 7, "Installing IMS/VS."

MAIN BUFFER POOL

This is the working storage pool, defined as WKAP in the IMS/VS procedure. Monitoring and adjusting its size is as previously discussed for the DMBP.

CWAP BUFFER POOL

This pool is used to buffer the SPAs of *active* conversations. Monitoring and adjusting its size is as previously discussed for the DMBP. Its size can be changed via the CWAP parameter in the IMS/VS procedure.

Note: The system definition value of the GENERAL parameter in the BUFPOOLS statement is used as the default value for both the MAIN and the CWAP buffer pool.

PSBW BUFFER POOL

This pool is used mainly to process segments between the CTI and dependent regions. Monitoring and adjusting its size is as previously discussed for the DMBP. Its size can be changed via the PSBW parameter in the IMS procedure.

DBWP BUFFER POOL

This pool is used for data base OPEN/CLOSE processing. Monitoring and adjusting its size is as previously discussed for the DMBP. Its size can be changed via the DBWP parameter in the IMS procedure.

STATISTICAL ANALYSIS UTILITY

The IMS/VS Statistical Analysis Utility provides statistical information about online IMS/VS operation. Its information is obtained from the online IMS/VS log data set. The utility consists of three basic modules, DFSISTS0, DFSIST20, and DFSIST30, and two intermediate sort steps. A sample job stream is listed as job //SAMP495 in IMSVS.PRIMEJOB. The sample output is listed in Chapter 3 of the *IMS/VS Primer Sample Listings* and is discussed later in this section.

JCL CONSIDERATIONS

The following should be observed when using the JCL of //SAMP495. The numbers refer to the JCL statement numbers in the job listing.

8. The SORT/MERGE program is used in all three steps with an entry point of SORT.

12. LOGIN defines the IMS/VS system log. Multiple volumes and data sets can be concatenated if desired.

14. The space parameter of this data set may need to be increased if the log data set reflects a large number of transactions.

32,50. The estimated number of sort records in the SIZE parameter may need to be increased, depending on the number of actual input log records.

50. The LINECNT parameter can be used to adjust the number of print lines per output page.

REPORT OUTPUT AND INTERPRETATION

Six types of reports are printed by the statistical analysis utility. Refer to the output of job //SAMP495 as listed in Chapter 3 of the _IMS/VS Primer Sample Listings_ when reading the following sections.

Messages Queued But Not Sent (by destination)

The number of not-yet-sent output messages per logical terminal (if known) is listed.

Line and Terminal Report

This report lists the message traffic received (R) or sent (S) by IMS/VS for each line, physical terminal, and logical terminal. An hourly distribution is given.

Messages Queued But Not Sent (by transaction code)

The number of not-yet-sent output messages per transaction code is listed. A transaction code of IMSSYS is listed if the output was generated by IMS/VS itself.

Transaction Report

This report lists the number and distribution of messages per transaction code.

Transaction Response Report

This report lists the response distribution per transaction code. Two response times are measured. The first line is the response time from complete receipt of the input message until the response message to the terminal is completely received by the terminal. The second line is the response time from complete receipt of the input message until the sending of the response message to the terminal is started. n% response means that n% of the messages had a response time less than or equal to the listed number.

Note: The actual figures of the sample output in Chapter 3 of the *IMS/VS Primer Sample Listings* should not be used for performance analysis and prediction. This is because the sample output was collected during a test run under VM/370.

Application Accounting Report

This report lists for each program and transaction code the number of messages and the average number of DL/I calls per message. The "CC or RC" column lists the number of times an application program terminated abnormally, or returned with other than zero in register 15.

The two last columns list the total and average CPU task time of the MPP region/partition. This includes the majority of the data base call processing.

THE DC MONITOR

The IMS/VS DC Monitor formats and records performance related data during the execution of the IMS/VS DB/DC system.

The DC Monitor can be available but inactive and cause no increase in CPU utilization. Including a DD card, described below, in the procedure makes the DC Monitor available. The Monitor remains inactive, however, until started from the master terminal by a /TRACE command.

The DC Monitor requires its own recording data set. Therefore a DD card (DDNAME=IMSMON) must be included in the IMS procedure to describe and specify the monitor log data set. If this card is not included, the Monitor will be unavailable.

USING THE DC MONITOR

It is, in general, not necessary to have the DC monitor active all day. A useful monitoring technique is to obtain "snapshots" on the Monitor log. Normally one to three hours of monitoring collects sufficient information for an entry system.

I/O Error: If a permanent I/O error occurs on the DC Monitor log data set, the Monitor stops and the message "DFS2202 UNCORRECTABLE I/O ERROR ON IMSMON" is displayed at the master terminal. In this situation the Monitor cannot be restarted until IMS/VS is restarted, and the DC Monitor log data set is only closed when IMS/VS is shut down.

If the problem that caused the error has not been corrected when IMS/VS is to be restarted, a different volume and/or unit should be specified.

Starting and Stopping the DC Monitor

Once the IMS/VS CTL region is started, the DC Monitor can be activated with the following command from the master terminal:

```
/TRACE SET ON MONITOR ALL
```

To stop the Monitor, enter:

```
/TRACE SET OFF MONITOR ALL
```

DC MONITOR REPORT PRINT PROGRAM, DFSUTR20

The DC Monitor Report Print program (DFSUTR20) is a batch program that takes the data collected by the DC Monitor (DFSMNTRO) and prints summary reports and distribution displays of the data. The report formats, and the nature of information in the reports are identical or similar to those printed by DFSUTR3C, the Monitor report program. The values shown in the report samples are not intended to reflect actual values that are received by a user's execution of this utility.

The following types of reports produced by DFSUTR20 are of interest to the first-time user:

- System configuration (data about OS/VS and IMS/VS systems used)
- Statistics from buffer pools (data collected at beginning and end of trace)
- Region (data on timing, IWAITs, and DL/I calls presented by region)
    - Region Summary -Region IWAIT
- Program (data on timing, IWAITs, DL/I calls, scheduling and dequeueing that is presented by an application program)
    - Programs by Region -Program Summary -Program I/O
- Communication (data on communication subtask timing, IWAITs, transmitted and received blocksizes, intersystem traffic and queueing)
    - Communication Summary -Line Functions -Communication IWAIT
- Transaction queueing (data on queue lengths and scheduling occurrences presented by transaction type)
- DL/I call summary (statistics on all DL/I calls issued by every program)
- Run profile, an over-all picture of IMS/VS performance during the DC monitor trace interval

For a description of the terms used in the report, see "Definition of Terms Used in the Reports" earlier in this chapter.

Note: The reports contain a rich variety of data, much of which can be interpreted only with detailed IMS/VS knowledge.

Therefore, we will only discuss those of immediate importance to the first-time user of our subset.

How to Execute the DC Monitor Report Print Program

Job //SAMP494 in IMSVS.PRIMEJOB shows the JCL to execute the DC Monitor Report Print Program, DFSUTR30. This job prints the monitor output collected during the execution of the IMS/VS CTL region.

The output listed in Chapter 3 of the *IMS/VS Primer Sample Listings*, will be referenced in the following discussion of the various generated reports.

If the DC Monitor does not collect certain types of information usually found in a particular report, that report, or the section of that report that would normally contain the information, is not produced. For example, if no checkpoints occur, only the headings for checkpoint are printed.

Note: The page numbers listed in the heading of the following sections refer to the page numbers in the sample output of job //SAMP494 in Chapter 3 of the *IMS/VS Primer Sample Listings*.

STATISTICS FROM BUFFER POOLS REPORT

These summary reports are a formatted display of the contents of selected items of the message queue, data base buffer pool, VSAM buffer pool and message format buffer pool that were collected at the beginning and end of trace. These reports are preceded by a system configuration report (Page 1), which gives information about OS/VS and IMS/VS systems used.

Since the pool ending values and the difference between starting and ending values cannot be computed if no pool ending statistics are written on the trace tape, this summary is produced only if the Monitor is ended before termination of the IMS/VS control region.

In the case where only the ending values of one buffer pool are not written on the trace tape, the corresponding summary report is not computed and the following information message is printed:

```
NO QUEUE BUFFER POOL TRACES AT END TIME ON MONITOR LOG TAPE
****QUEUE BUFFER POOL REPORT CANCELLED****
```

The VSAM buffer pool summary report and/or the message format buffer pool report will not be produced if the corresponding facility is not invoked through IMS/VS system definition. In this case, the following information message is printed:

```
NO VSAM BUFFER POOL TRACES ON MONITOR LOG TAPE
****VSAM BUFFER POOL REPORT CANCELLED****
```

Message Queue Pool, Page 2

The final number of this report (QUOTIENT) is the most interesting. Generally, in an entry IMS/VS system this number should be less than 1. If higher than 1.5, you should consider increasing the number of online queue buffers via the QBUF parameter in the IMS control region procedure.

Data Base Buffer (Sub)pools, Pages 3, 4 and 5

The statistics of the data base buffer (sub)pools and their interpretation are the same as discussed in the first part of this chapter for a batch application.

Message Format Buffer Pool, Page 6

The final number of this report (QUOTIENT) is the most interesting. Ideally, in an entry IMS/VS system, this number should be less than 1. If higher than 4, you should consider increasing the size of the online MFS pool, via the FBP parameter in the IMS control region procedure.

Region Summary Report, Page 7

Region timing information is printed for every MPP/BMP active during the trace. This summary report distinguishes the following activities per region:

- Scheduling and termination
- Schedule to first DL/I call
- Elapsed execution
- DL/I call
- Idle for intent
- Checkpoint
- Region occupancy

It should be noted that some of the values shown for region timing overlap in the timeframe of the trace period. Elapsed time for scheduling and termination are included in idle-for-intent time. The elapsed time for execution includes the elapsed time for DL/I calls. In general, the trace time period is slightly greater than the sum of scheduling and termination, schedule to first DL/I call, elapsed execution time, and idle-for-intent time. Differences between this sum and the trace time can be attributed to transactions active at the startup and shutdown of the tracing, or idle regions at the start or end of a trace.

Scheduling and Termination: Lines under this heading, for each region, show the number of occurrences of scheduling and termination in the region, and both the elapsed execution time and not IWAIT time associated with scheduling and termination. The total of all intervals, the maximum single interval, and the mean interval values are shown.

The elapsed time during which the scheduler is internally waiting is not included in the elapsed time shown for scheduling and termination.

This line of the report does not include data for a message region that was not scheduled but was executing at the start of the trace.

Schedule to First DL/I Call: The lines under this subheading show the elapsed time for the following to occur: the region to gain control after being scheduled; the program either to be located in the region or to be brought into the region; or the program to issue the first DL/I call requiring dispatching of the IMS/VS Call Analyzer Module.

This section does not appear for a message region or a batch message region that was not scheduled during the trace; it does not appear for one that was scheduled but did not issue a DL/I call following the scheduling. The number of program loads is one less than the number of schedulings, if the trace was ended after the scheduling but before the first DL/I call of the last scheduling.

Elapsed Execution: Lines under this subheading give the number of executions of programs in each region and the elapsed time for each execution.

The number of executions may be one less than the number of schedulings and schedule to first DL/I calls if the program that was scheduled had an outstanding DL/I call when the trace was ended.

DL/I Call: Lines under this subheading give the total number of DL/I calls from each region during the trace, the total, mean and maximum elapsed execution time intervals to complete those calls, and the total, mean and maximum non-IWAIT intervals used to complete those calls. The number of IWAITs per call is computed and displayed for each region under the heading "IWT/CALL."

Idle for Intent: Lines under this subheading give the number of times a region was in the idle state because of an intent conflict. An intent conflict occurs during scheduling when the program to be scheduled needs data base resources held by another active program. See the section "Data Base Processing Intent" in Chapter 3, "Data Communication Design."

Checkpoint: This line is produced if a checkpoint occurs during the trace.

The line gives the number of times checkpoint was dispatched, the total elapsed time of the checkpoints, the mean elapsed time for a checkpoint, and the maximum of those elapsed times.

The line also gives the total non-IWAIT time for the checkpoints, the average non-IWAIT time for a checkpoint, and the maximum of those non-IWAIT times.

Region Occupancy

Lines under this sub-heading indicate the percentage of time that the region was occupied. This value is determined from the formula:

$$\text{Region Occupancy} = \frac{\text{scheduling + termination + schedule to 1st DL/I call + program elapsed + idle-for-intent}}{\text{trace elapsed time}}$$

The value of trace elapsed time is the difference between the time recorded for the first traced IMS/VS event and the last traced IMS/VS event.

Region IWAIT, Pages 8, 9

This report lists the occurrences and duration of IWAITs for each dependent region during:

- Scheduling and termination processing
- DL/I call processing
- Checkpoint processing

OCCURRENCES: Lists the number of IWAITs

TOTAL, MEAN, MAXIMUM: Lists the total, mean and maximum elapsed time of IWAITs.

FUNCTION: Lists the cause of the IWAIT. PSB/DMB defines the PSB/DBD to be loaded, DD defines the data set via its DDNAME which needed an I/O. If the number of PSB/DMB loads (IWAITs) is high, you might consider increasing the PSB and DMB pool sizes.

MODULE: Defines the module involved in the IWAIT. BLR = block loader for PSBs, DMBs. VBH = VSAM buffer handler. QMG = queue manager.

Program Summary, Page 11

This report lists an overview of the most important factors for each MPP/BMP active during the period of monitoring. The time figures are given in microseconds. The columns and their meaning are:

NO.SCHEDS: The number of schedulings.

TRANS.DEQ: The number of input message dequeued after their successful processing by the program.

DL/I CALLS: The total number of both message and data base calls issued by the program.

DL/I CALLS/TRAN: Same as above, but per input message.

I/O IWAITS: The number of I/Os required by IMS/VS to process these DL/I calls. An average below two is preferable. However, higher values occur if the processing of a call involves scans of long physical twin chains or insert/deletes of segments involved in logical relationships and/or secondary indexes.

TRAN DEQD/SCH: The average number of input messages (same transaction code) processed in one scheduling of a MPP. This number in our subset is between 1 and 5.

CPU TIME/SCHED: The average CPU task time for the MPP/BMP region per schedule. This will be typically very high for BMPs because the full processing of a BMP constitutes one scheduling.

ELAPSED TIME/SCHED: The average program elapsed executing time per scheduling. This number is largely dependent of the processing performed by the program, especially its number of DL/I calls and associated I/Os. Ideally, for a simple application, it should be below 500 milliseconds.

SCHED. TO 1ST DLI/SCHED: The average elapsed time between the scheduling of the MPP/BMP and its first DL/I call. Typically, the major contributing factor in this is program load time. Typically, this value should be between 200 and 500 milliseconds. Basic steps to improve this figure are:

- Maintain a compact (compressed) IMSVS.PGMLIB containing only the MPPs used by the online system.
- Specify this IMSVS.PGMLIB as the first step/job library in the MPP region JCL.
- Use the COBOL options of NORES, NODYNAM and NOENDJOB.

ELAPSED TIME/TRANS: Same as ELAPSED TIME/SCHED, but now per transaction. If TRAN.DEQ./SCH. is 1, they are the same.

DL/I Call Summary, Pages 19, 20

This report gives a summary of the number of DL/I calls per PSB (that is, program), per segment returned, and the number of IWAITs for these calls. For a discussion refer to the DL/I Call Summary Report section of the DB Monitor in the first part of this chapter.

Run Profile, Page 22

This report gives a compact, over-all picture of IMS/VS performance during the period of the DC monitor trace interval. The headings and contents of this report are self-explanatory. Definitions of terms used are included in the discussion of previous reports.

USING THE VTAM STORAGE POOL TRACE

This section describes how to use the VTAM storage pool trace to optimize the VTAM main storage pool in your installation.

This VTAM trace facility is dependent upon the OS/VS

General Trace Facility (GTF).

The VTAM storage pool trace records the usage of all VTAM storage pools. When both GTF (USR trace option) and VTAM storage pool trace are active, the storage pool trace information is collected on every 1000th VTAM request to obtain a storage pool element.

The VTAM storage pool trace collects the following information for each of the VTAM storage pools:

- Storage pool name
- The maximum number of elements allocated from the pool at any one time since the last trace record was written
- The maximum number of queued requests for buffers at any one time since the last trace record was written
- Number of currently unallocated elements in the pool
- Date and time, if TIME=YES was specified in the GTF START command

OPERATING THE TRACE

To be able to activate the VTAM storage pool trace, GTF must be started first. If you are using our sample cataloged procedure VTAMGTF as generated by job //SAMPI56 in IMSVS.PRIMEJCB, you should use the following procedure.

Starting the Trace

enter: S VTAMGTF.P1 (This starts GTF)

response: nn HHL100A SPECIFY TRACE OPTIONS

enter: nn,TRACE=RNIO,USR

response: mm HHL125 RESPECIFY TRACE OPTION OR REPLY U

enter: mm,U

enter: F P0,TRACE,TYPE=SMS,ID=VTAMBUF

This starts the VTAM storage pool trace

Note: In the above it is presumed that VTAM runs in P0 and GTF in P1.

Stopping the Trace

enter: F PO,NOTRACE,TYPE=SMS,ID=VTAMBUF

enter: P VTAMGTF.F1

Printing the Trace Output

Job //SAMP496 in IMSVS.PRIMEJCL can be used to print the VTAM trace output.

OPTIMIZING VTAM STORAGE POOL PARAMETERS

VTAM has eleven storage pools to control the buffering of data. VTAM dynamically allocates and deallocates space in these pools for the VTAM control blocks, I/O buffers, and channel programs that control the transmission of this data.

The basic procedure for tailoring the VTAM storage pool values is to initially operate VTAM using the worst-case storage pool values as described in CS/VS Storage Estimates. Then adjust the storage pool values by using the VTAM storage pool trace facility. To tailor the VTAM storage pools, determine the following values for each VTAM storage pool:

- bno, the maximum number of elements in the pool
- bth, a threshold number of elements for a pool
- bsz, the size of each element (in bytes) in a pool (specify for IOBUF and PPBUF only)

Storage Pool (SMS) Trace Description

The VTAM storage pool (SMS) trace records information on the use and availability of VTAM's buffer pools. The trace records are written at regular intervals, after every 1,000 requests for buffers. Each set of records contains the maximum number of buffers in use, the maximum number of requests for buffers queued, and the current number of available buffers for each of VTAM's eleven buffer pools. An example of VTAM storage pool trace output is shown in Figure 9-3.

```
*** DATE   DAY 080    YEAR 1978             TIME 20.56.20.407148
USRFD FFO      VTAM BUFFERS       MAXU MAXQ AVNO     MAXU MAXQ AVNO     MAXU MAXQ AVNO     MAXU MAXQ AVNO
                               IO 0023 0000 00B2 PP 0002 0000 002C LP 000E 0000 0024 WP 0003 0000 0011
                               NP 0007 0000 0025 LF 0008 0000 002A CR 0006 0000 0037 UE 0002 0000 002C
                               SF 0001 0000 001A SP 0000 0000 0005 AP 0005 0000 0028
        TIME     75380.399270
```

Figure 9-3. Sample VTAM Trace Output

Legend:

POOLID

Identifies which buffer pool the trace entry is for. Pool IDs are:

IO Fixed I/O pool (IOBUF)

PP Pageable I/O pool (PPBUF)

LP Large pageable pool (LPBUF)

WP Working set pageable pool (WPBUF)

NP Nonworking set pageable pool (NPBUF)

LF Large Fixed pool (LFBUF)

CR Copy RPL pool (CRPLBUF)

UE UECB pool (UECBUF)

SF Small fixed pool (SFBUF)

SP Small pageable pool (SPBUF)

AP ACE pool (APBUF)

MAXU
: Indicates the maximum number of buffers in the pool that were in use at any one time in the last interval. A MAXU value of 0, however, does not mean that all buffers in the pool were released, but that there were no additional requests for buffers during this interval.

MAXQ
: Indicates the maximum number of requests for buffers that were queued at any one time since the last interval.

AVNO
: Indicates the average number of buffers in the pool that were available during the interval.

Adjusting the VTAM Storage Pools

Based on a VTAM trace output of several hours of regular production, use the following guidelines in adjusting the VTAM storage pool parameters as specified in job //SAMPI54.

For every pool find the following values:

- Average of all MAXUs related to pool
- Average of all MAXQs related to pool
- Average of all AVNOs related to pool
- Highest MAXU related to pool
- MAXQ related to highest MAXU
- AVNO related to highest MAXU

Decrease bth and bno if:

- highest MAXU is always at least 10% lower than bth.

Increase bth and bno if:

- Average and/or highest MAXU value is larger than or equal to bth.
- Average and/or highest MAXU value is close to bth and average and/or highest AVNO is close to zero.

Notes:

1. The increase should be at least as high as the highest MAXQ value.

2. The relationship between bth (threshold value) and bno (number of buffers inpool) is described in *OS/VS System Programming Library (MVS): Storage Estimates*, GC28-0604.

DATA COMMUNICATION DESIGN OPTIMIZATION

The importance of a good data base and program design for the performance of an IMS/VS online system is even more apparent than for a batch system. The guidelines for data base design and DL/I call used by programs, as given in the first part of this chapter, still fully apply.

Another important factor of general interest in data communication design is the transaction response time.

As discussed in the section "Transaction Response Time Considerations" of Chapter 3, "Data Communication Design," the response time consists of two components:

1. Network response time.

2. IMS/VS response time.

NETWORK RESPONSE TIME FACTORS

The following parameters influence the network response time:

- Length of input data character stream
- Length of output data stream
- Line mode operation, that is, half or full duplex
- Line speed and line length
- Modem turnaround time
- Number of clusters and number of terminals per cluster
- Communication controller delay time
- Number of transactions per terminal, arrival rate, and distribution

Based on the above factors, an assessment can be made for the network response time.

IMS/VS RESPONSE TIME FACTORS

One of the most important factors which influence the performance of an online IMS/VS system is the *MPP service time*.

The MPP service time is the elapsed time between scheduling of the transaction and the completion of its processing by the MPP.

The basic components of the MPP service time are:

1. Program loading of the MPP.
2. Retrieval of input message and associated physical I/Os.
3. Data base calls and associated I/Os.
4. Application program processing time.
5. Insert of output message into the message queue and its associated physical I/Os (if no free queue buffer is available).
6. OS/VS paging during any of the above activities.

The two most important components from the above list are usually the program load time and the data base calls with their associated I/Os. Typically, program load time is between 200 and 500 milliseconds elapsed time, and a direct I/O is between 30 and 50 milliseconds elapsed time, assuming 3330 disk drives.

Note: IMS/VS provides for preloading of selected application programs and PL/I modules. This preload option is not included in our subset, but you might consider its use as the next step on improving systems performance. More information on the preload option is included in Chapter 2 of the IMS/VS Installation Guide under the topics "Making High-Use Program Modules Resident" and "Organizing PL/I Modules for Use with the PL/I Optimizer."

Sample IMS/VS Response Time Estimate

We will now discuss a very simple IMS/VS response time estimation, based solely on MPP service time considerations.

Assumptions:

- Lightly loaded system, that is, ample available CPU time for IMS/VS activities.
- One MPP region.
- All MPPs have roughly same service time.
- MPP is loaded for each transaction; program load time is 300 milliseconds.
- Average of 10 DL/I calls with average of 1 I/O per DL/I call of 40 milliseconds average elapsed time.
- The message inter-arrival time is 2 seconds (one message every 2 seconds) with an exponential distribution.
- Basic queueing theory is applicable.

__Estimates__:

Based on above assumptions, the following parameters can be derived/calculated:

MPP Service Time: TS = 300+10*40=700 milliseconds.

Arrival Rate: A = 0.5 messages/second

MPP Utilization: U = TS*A=0.35

MPP Wait Time: TW = U*TS/(1-U)=380 milliseconds

MPP Response Time: TR = TS+TW=1.08 seconds

__Where__:

- The MPP wait time is the average time a message must wait in the queue before the MPP region can process it.
- The MPP response time is the average interval response time of a transaction, that is, the time between the enqueue of the input message in the queue and the enqueue of the output message in the queue.

__Notes__:

1. The formula for TW normally applies only for utilizations below 60% ($U \le 0.6$).
2. Many other factors can influence the total IMS/VS response time, such as:
    - Loading of DBDs, PSBs and MID/MODs, DIF/DOFs.
    - Data base open processing (required after DBD (re)load).
    - CPU, channel, and disk drive utilizations.
    - Dispatching priority of CTL and MPP regions.
    - Location of IMS/VS system data sets, noticeably the queue, format, and program library data sets.

APPENDIX A. IMS/VS STATUS CODES QUICK REFERENCE

STATUS CODE	DATA BASE CALLS					MSG CALLS				SYSTEM CALLS		CALL STATUS			DESCRIPTION
	GU GHU	GN GHN	DLET REPL	ISRT (LOAD)	ISRT (ADD)	GU	GN	ISRT	CHNG	CHKP	XRST	CALL COMPLETED	ERROR IN CALL	I/O OR SYST ERROR	
AB	X	X	X	X	X	X	X	X	X	X			X		SEGMENT I/O AREA REQUIRED. NONE SPECIFIED IN CALL
AC	X	X		X	X								X		HIERARCHICAL ERROR IN SSAs
AD	X	X				X	+						X		INVALID FUNCTION PARAMETER
AH	X	X		X	X								X		CALL REQUIRES SSAs. NONE PROVIDED
AI	X	X	X	X	X									X	DATA MANAGEMENT OPEN ERROR
AJ	X	X		X	X								X		INVALID SSA QUALIFICATION FORMAT
AK	X	X		X	X								X		INVALID FIELD NAME IN CALL
AL	X	X	X	X	X								X		CALL USING LT PCB IN BATCH PGM
AM	X	X	X	X	X								X		CALL FUNCTION NOT COMPATIVLE W/PROCESSING OPTION OR SGMT SENSITIVITY
AO	X	X	X	X	X									X	I/O ERROR OSAM, BSAM, OR VSAM
AP						X	X	X	X	X			X		MORE THAN 4 CALL PARAMETERS INVALID FOR DC PCB
AT			X	X	X			X					X	X	USER I/O AREA TOO LONG
AY								X					X		RESPONSE ALTERNATE PCB REFERENCED BY ISRT CALL HAS MORE THAN ONE PHYSICAL TERMINAL ASSIGNED FOR INPUT PURPOSES. NOTIFY MASTER TERMINAL
A1									X				X		CALL ATTEMPTED WITH 8-CHAR LOGICAL TERMINAL NAME NOT KNOWN TO SYSTEM
A2									X				X		CHANGE ATTEMPTED WITH INVALID PCB
A3								X					X		INSERT ATTEMPTED TO A MOD TP PCB WITH NO DESTINATION SET
A5								X					X		FORMAT NAME SPECIFIED ON 2ND OR SUBSEQUENT MSG ISRT CALL
A6								X					X		OUTPUT SEGMENT SIZE LIMIT EXCEEDED ON ISRT CALL
A7								X					X		NUMBER OF OUTPUT SEGMENTS INSERTED EXCEEDED THE LIMIT BY ONE
DA			X										X		SEGMENT KEY FIELD HAS BEEN CHANGED
DJ			X										X		NO PRECEDING SUCCESSFUL GET HOLD CALL
DX			X										X		VIOLATED DELETE RULE
GA		X										X			CROSSED HIERARCHICAL BOUNDARY INTO HIGHER LEVEL (RETURNED ON UNQUALIFIED CALLS ONLY)
GB		X													END OF DATA SET, LAST SEGMENT REACHED
GE	X	X			X										SEGMENT NOT FOUND
GK		X										X			DIFFERENT SEGMENT TYPE AT SAME LEVEL RETURNED (RETURNED ON UNQUALIFIED CALLS ONLY)
II					X										SEGMENT TO INSERT ALREADY EXISTS IN DATA BASE
IX					X								X		VIOLATED INSERT RULE
LB				X											SEGMENT TO INSERT ALREADY EXISTS IN DATA BASE
LC				X											KEY FIELD OF SEGMENTS OUT OF SEQUENCE
LD				X											NO PARENT FOR THIS SEGMENT HAS BEEN LOADED
LE				X											SEQUENCE OF SIBLING SEGMENTS NOT THE SAME AS DBD SEQUENCE
NE			X											X	DL/I CALL ISSUED BY INDEX MAINTENANCE CANNOT FIND SEGMENT
NO			X	X	X									X	I/O ERROR OSAM, BSAM OR VSAM
QC						X				X					NO MORE INPUT MESSAGES
QD							X								NO MORE SEGMENTS FOR THIS MESSAGE
QE							X						X		GET NEXT REQUEST BEFORE GET UNIQUE
QF								X		X			X		SEGMENT LESS THAN FIVE CHARACTERS (SEG LENGTH IS MSG TEXT LENGTH PLUS FOUR CONTROL CHARACTERS)
QH								X					X		TERMINAL SYMBOLIC ERROR. OUTPUT DESIGNATION UNKNOWN TO IMS/VS (LOGICAL TERMINALS OR TRAN CODE)
RX			X										X		VIOLATED REPLACE RULE
X3								X					X		INVALID SPA
X7								X					X		LENGTH OF SPA IS INCORRECT (USER MODIFIED FIRST SIX BYTES)
XC								X					X		PGM INSERTED MSG WITH Z1 FLD BITS SET RESERVED FOR SYSTEM USE
XD										X		X			IMS IS TERMINATING. FURTHER DL/I CALLS MUST NOT BE ISSUED. NO MESSAGE RETURNED
XX	X	X		X	X									X	INTERNAL GSAM ERROR
bb	X	X	X	X	X	X	X	X	X	X	X	X			GOOD. NO STATUS CODE RETURNED, PROCEED

NOTE: bb indicated blanks

APPENDIX E. IMS/VS STATUS CODES AND POSSIBLE CAUSES

The following listing of IMS/VS status codes and possible causes is divided into two parts. The first part lists the status codes which are, in general, caused by application program errors. The second part lists the status codes which are, in general, caused by system errors.

Note: A more detailed discussion of these and other status codes can be found in Appendix B of the *IMS/VS Application Programming Reference Manual*.

APPLICTION PROGRAM ERROR STATUS CODES

The following status codes are the most common ones caused by application program errors (error in call) in our subset.

AB: Segment I/O area is required but was not specified in the call.

AC: SSA(s) contains an error in hierarchical sequence.

Possible causes:

1. No segment name equal to that specified in SSA found within scope of this PCB.
2. Level at which this SSA appears is out of sequence with that specified by the PCB.
3. Two segments of the same level are specified in the same call.

AD: An invalid function parameter was supplied.

Possible causes:

1. Invalid function string
2. A GU or GN was requested for a terminal PCB other than the I/O PCB.
3. Invalid request type to a DC-PBC.
4. A call has been issued to the message queues with a DB-PCB.

AH: No SSA(s) was specified in call. Call required at least one SSA, and none was specified.

AJ: SSA qualification format was invalid.

Possible causes:

1. Invalid command codes.
2. Invalid relational operators.
3. Missing right parenthesis of Boolean connector.

4. DLET call has nultiple SSAs or qualified SSAs.

5. REPL call has qualified SSAs.

6. ISRT call has the last SSA qualified.

7. A path insert call into an existing data base involves a logical child segment.

AK: An invalid field name was supplied in the call.

Possible causes:

1. Unable to find the specified field name.

2. When accessing a logical child, a field name from the othe (paired) logical child is used for the destination parent concatenated key portion.

AL: The call is using a terminal PCB in a DL/I program.

AM: Call function not compatible with processing option, segment sensitivity, or transaction-code definition.

Possible causes:

1. Command code D used for path retrieval call without path sensitivity

2. Processing option of I and call function is not insert

3. DLET, REPL, or ISRT call without corresponding segment sensitivity

4. A DLET, REPL, or ISRT call was issued by a program while a transaction defined as inquiry was being processed.

   A GET call was attempted for a segment with KEY sensitivity. Correct the error by specifying DATA sensitivity.

5. This status code occurs for a checkpoint (not restart) call if a GSAM/VSAM data set is opened for output.

6. An invalid request was included in a GSAM call.

7. Any call to a GSAM dummy data set is invalid.

AT: Error in call. The length of the user's I/O area data is greater than the area reserved for it in the control region. The length of the area reserved was determined by the ACB utility program, DFSUACB0, and printed as part of its output.

<u>Action</u>: Correct the PSB or the program (message segment length field) in error.

AY: Insert call ignored because the logical terminal referenced by the response alternate PCB currently has more than one physical terminal assigned to it for input purposes.

Action: Ask the master terminal operator to determine (use /DISPLAY ASSIGNMENT ITERM X) which physical terminals (2 or more) refer to this logical terminal. Use the /ASSIGN command to correct the problem.

A1: The CHNG call was attempted with an eight-character logical terminal name which was unknown to the system.

Action: Correct program.

A2: The CHNG call was attempted with an invalid PCB. It was either not an alternate PCB, was not defined as modifiable, or had a message in process but incomplete.

Action: Correct program.

A3: An INSERT call was attempted to a modifiable alternate PCB which had no destination set.

Action: Issue a CHNG call to set the PCB destination, and reissue the INSERT call.

A5 An invalid call list was supplied. A fourth parameter (MOD name) was supplied, but the function was not ISRT for the first segment of an output message.

Action: Correct the ISRT call and retry the application program.

A6: Insert call ignored because output segment exceeded specified limit.

Action: Correct the application program.

A7: Insert call ignored because number of output message segments inserted exceeded specified limit by one. If another attempt is made to insert too many segments before the program issued another GU, the program is abended.

Action: Correct the application program.

DA: A segment sequence field has been changed; no action in data base.

DJ: No previous successful get hold call; no action in data base.

DX: Violated delete rule; tried to delete across a logical relationship. Check RULES = parameter on DBD.

GA: Call is completed.

Explanation: Crossed hierarchical boundary into higher level. This status code is returned on unqualified calls only.

Action: User determined.

GB: Call is not completed.

Explanation: This is the end of the data set; last segment is reached. If GSAM, the data set will have been closed.

Action: User determined.

GE: Call is not completed.

Explanation: Segment has not been found.

Action: User determined.

GK: Call is completed.

Explanation: Different segment type at same level returned. This status code is returned on unqualified calls only.

Action: User determined.

II: Call is not completed.

Explanation: The segment that the user tried to insert already exists in the data base.

Possible causes:

1. Segment with equal physical twin sequence field already exists for parent.

2. Segment with equal logical twin sequence already exists for parent.

3. Logical parent has logical child pointer, logical child does not have logical twin pointer, and segment being inserted is second logical child for logical parent.

4. Segment type does not have physical twin forward pointer, and segment being inserted is second segment of this type for parent or is second HDAM root for one anchor point.

5. The segment being inserted is in an inverted structure; that is, the immediate parent of this segment in the logical structure is actually its physical child in the physical structure.

Action: User determined.

IX: Violated insert rule.

Possible causes:

1. Insert of logical child and logical or physical parent does not exist, or wrong DBCK.

2. Insert of logical or physical parent via its logical path.

3. ISRT request after previous Open, Close or I/O error status code.

4. A GSAM ISRT call was issued after a previous AI or AC status code was returned.

Action: Correct program.

LB: Call is not completed.

Explanation: The segment user tried to load already exists in the data base.

Possible causes are:

1. A segment with an equal physical-twin-sequence field already exists for the parent.

2. A segment type does not have a physical-twin-forward pointer (PTR=NT in SEGM statement in DBD) and the segment being inserted is either the second segment of this segment type for the parent or the second HDAM root for one anchor point.

3. An application program inserted a key of X'FF'..FF' into a SHISAM or HIDAM data base.

Action: User determined.

LC: Call is not completed.

Explanation: Key field of segments is out of sequence.

Action: Check and correct.

LD: Call is not completed.

Explanation: No parent has been loaded for this segment.

Action: Check and correct.

LE: Call is not completed.

Explanation: Sequence of sibling segments is not the same as the sequence in the DBD.

Action: Check and correct.

NE: Call is not completed.

Explanation: Index maintenance issued a DL/I call, and the segment has not been found.

Action: User determined.

QC: There are no more input messages. If CHKP call, call was successful.

Action: The program should terminate.

QD: There are no more segments for this message.

<u>Action</u>: As appropriate.

QE: A GET NEXT call was issued before a GET UNIQUE.

<u>Action</u>: Check and correct.

QF: Length of segment is less than five characters. Allowable segment length is length of message text plus four control characters.)

<u>Action</u>: Check and correct.

QH: The output designation, the LTERM, is unknown to IMS/VS.

<u>Action</u>: Check LTERM name specification in PCB or CHNG call.

RX: Violated replace rule. Review the RULES= parameter in the DBDs.

<u>Action</u>: Correct program/DBD.

X3: Invalid SPA (user modified the first six bytes).

<u>Action</u>: Correct the program.

X7: The length of the SPA is incorrect (user-modified first six bytes).

<u>Action</u>: Correct the program.

XC: Program has inserted a message which has some Z1 field bits set which are reserved for IMS/VS use.

<u>Action</u>: Correct the program to prevent it from setting those bits.

XD: IMS/VS is terminating by a CHECKPOINT FREEZE or DUMPQ. This code is returned only from a CHKP call issued by a batch-message application program. The checkpoint itself was successful.

<u>Action</u>: Any subsequent DL/I call will result in an abend. The BMP should terminate.

<u>SYSTEM ERROR STATUS CODES</u>

The following status codes represent the most common errors in our subset:

AI: I/O, system, or user error

<u>Explanation</u>: Data management open error.

Possible causes:

1. Error in DD cards.

2. The data set was opened for something other than load mode, but it is not loaded.

3. Buffer too small to hold record read at open time. See Chapter 7 for minimum buffer pool size.

4. DD cards for logically related data bases not supplied.

5. For an OSAM data set, the DSORG field of the OSAM DCB, DSCB, or JFCB does not specify PS or DA.

6. For an old OSAM data set, the BUFI or BLKSIZE field in the DSCB is zero.

7. The data set is being opened for load, and the processing option for one or more segments in other than L or LS.

8. The allocation of the OSAM data set is invalid; allocation is probably (1,,1) rather than (1,1), and this causes the DSORG to be PO.

9. Processing options is L, the OSAM data set is old, and the DSCB LRECL and/or BLKSIZE does not match the DBD LRECL and/or BLKSIZE.

10. Incorrect or missing information prevented computation of blocksize or the determination of the logical record length.

11. A catalog was not available for accessing a VSAM data base that was requested.

12. OS/VS could not perform an OPEN, but the I/O request is valid. Information is either missing, or data definition information is incorrect.

<u>Action</u>: Check DD cards; ensure ddname is name specified on DATASET card of DBD. Segment name area in PCB has ddname of data set which could not be opened.

AO: There is a physical I/O error. When issued from GSAM, this status code means that the error occurred when:

1. A data set was accessed, or

2. The CLOSE SYNAD routine was entered. The error occurred when the last block or records was written prior to closing of the data set.

<u>Action</u>: Determine whether the error occurred during input or output, and correct the problem. Recover the data set.

NO: I/O error

<u>Explanation</u>: There was a BSAM, VSAM, or OSAM physical I/O error during a DL/I call issued by indexing maintenance.

<u>Action</u>: Check and correct (recover data base).

XX After initialization, the XX status code indicates an IMS/VS error--probably GSAM.

An XX status code from initialization itself (prior to the first DL/I call) may be either a system, IMS/VS, or user error.

Explanation: When the XX status code is issued from initialization, the cause may be:

- Insufficient storage
- Invalid DBD
- Invalid blocksize
- Invalid option
- GSAM error

Action: Any subsequent GSAM call will result in an abend. The application should terminate.

INDEX

IMS/VS Version 1 Primer
SH20-9145-0

Reader's Comment Form

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this 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.*

List TNLs here:

If you have applied any technical newsletters (TNLs) to this book, please list them here:

Last TNL ____________________

Previous TNL ________________

Previous TNL ________________

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.

SH20-9145-0

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

IBM®

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 Primer Printed in U.S.A. SH20-9145-0

SH20-9145-0

IMS/VS Version 1 Primer Printed in U.S.A. SH20-9145-0

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