IBM Data Language/I Disk Operating System/
Virtual Storage (DL/I DOS/VS)

Data Base Administration

Version 1 Release 7

SH24-5011-01

IBM Data Language/I Disk Operating System/
Virtual Storage (DL/I DOS/VS)

Data Base Administration

Version 1 Release 7

SH24-5011-01

Note !

Before using this information and the product it supports, be sure to read the general information under “Notices” on page xiii.

Second Edition (December 2002)

This is a major revision of SH24-5011-0 including Technical Newsletter SN24-5740. This edition, SH24-5011-1, applies to Version 1, Release 7 (Version 1.7), of Data Language/I Disk Operating System/Virtual Storage (DL/I DOS/VS), Program Number 5746-XX1, and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the addresses given below.

A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Deutschland Entwicklung GmbH
Department 3248
Schoenaicher Strasse 220
D-71032 Boeblingen
Federal Republic of Germany

You may also send your comments by FAX or via the Internet:

Internet: s390id@de.ibm.com
FAX (Germany): 07031-16-3456
FAX (other countries): (+49)+7031-16-3456

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Part 3. Implementing the Data Base Design

Part 4. Using Data Bases

Part 5. Controlling Data Base Operation

Part 6. Appendixes

Figures

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785, U.S.A.

Any pointers in this publication to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement. IBM accepts no responsibility for the content or use of non-IBM Web sites specifically mentioned in this publication or accessed through an IBM Web site that is mentioned in this publication.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

```
IBM Deutschland Informationssysteme GmbH
Department 0215
Pascal Str. 100
70569 Stuttgart
Germany
```

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks and Service Marks.

The following terms are trademarks of International Business Machine Corporation in the United States, or other countries, or both:

```
CICS
DB2
IBM
```

Part 1. Introduction

Chapter 1. Introduction

Data Base Administration

Data base administration is a function that is unique to data processing installations that use data bases for information storage. A data base management system such as DL/I is required to create, maintain, and use data bases. In order to use and control the data bases and data management system efficiently, a data base administration function must be established within the installation.

This book describes the tasks performed under data base administration. It also provides information about DL/I that is needed to perform these tasks.

The major data base administration tasks described in this book are:

- Installation planning
- Implementing the data base design
- Using data bases
- Controlling data base operation.

These can be broken down into many smaller tasks that are summarized in the next section. In Figure 1-1, the tasks are cross-referenced to the chapters that describe them in detail.

Data Base Administration Task Descriptions

Installation Planning

Pre-Installation Planning

You should be deeply involved in the planning that is necessary to prepare for the installation of DL/I. This involves making decisions about machine (hardware) requirements, programming system (software) requirements, personnel requirements, and other similar considerations.

Planning for Installing Additional Applications

You will also be involved in making similar types of decisions when new applications are added to an existing DL/I system.

Implementing the Data Base Design

Preparing to Implement the Data Base Design

A conceptual data base design and requirements it must meet were developed as part of application design (see *DL/I DOS/VS Application and Data Base Design*). One of your tasks is to prepare for implementing this conceptual design as a physical data base. This involves:

- Learning about DL/I data base organizations and access methods, the various capabilities of DL/I, and how the capabilities can affect data base performance
- Deciding which DL/I capabilities are required for implementing the data base
- Choosing the best organization and access method for the data base

- Making decisions about the physical implementation of the data base that will ensure good performance and space utilization.

Implementing the Design

The actual physical implementation of a data base, and its preparation for use with application programs and DL/I, consists of a number of steps:

- Coding and generating data base descriptions (DBDs) for all data bases
- Coding and generating program specification blocks (PSBs) for all programs
- Using the application control blocks creation and maintenance utility to create data management blocks (DMBs) from DBDs, and expanded PSBs from PSBs
- Determining data base space requirements and allocating data sets
- Making DL/I virtual and real storage estimates
- For online operation: determining online nucleus and CICS system generation requirements
- Preparing for MPS batch operation if it is used.

Using Data Bases

DL/I Program Execution

A number of steps have to be performed before application programs can be executed under DL/I. You have to learn what these steps are, and see that they are completed. You need to become familiar with the three DL/I operating environments and how execution occurs in each one. You also need to develop program testing requirements and learn how to create test data bases.

Loading Data Bases

You are responsible for ensuring that an application program, called an initial load program, is available for each data base. This program is used to load the initial data into the data base. After the initial load program has loaded data into the data base, application programs can be run against it.

Modifying Data Base Structure, Organization, and Environment

As new applications are developed or the needs of your users change, you may need to make changes to data bases. You may, for example, need to change data base hierarchies or the segments and fields within them, or you may want to add or delete DL/I functions. The task of modifying data bases is an ongoing one.

Controlling Data Base Operation

Data Administration and Security

You are responsible for establishing and maintaining the integrity and security of the data in the data bases used in your installation. You can control security at three levels:

- You can limit access to data at the segment and field level
- You can limit the types of processing a given program can perform
- You can limit access to data base data sets by non-DL/I programs.

A data dictionary can be used to check data usage and to help in administering security. You can create your own, or use the DOS/VS DB/DC Data Dictionary Program Product (Program Number 5746-XXC). The DL/I Documentation Aid facility can also be used to help maintain data integrity.

Monitoring the Data Base

Once a data base is in use, you will need to monitor its performance. A variety of tools for monitoring the DL/I system is available. Monitoring data base performance is an ongoing task.

Improving Data Base Performance

When monitoring shows that data base performance or use of external storage is not optimum, you will need to correct that situation. There are a number of tools that are available for use in improving data base performance. Like monitoring, the task of improving data base performance is an ongoing one.

Data Base Recovery

Though it shouldn't happen often, a data base can be damaged. Data can be lost or become inaccurate. Data base recovery is restoring a data base to its original condition after it's made invalid by some failure. The task of developing recovery procedures and performing recovery is an important one. DL/I provides tools that help in recovery.

Data Base Administration Task Summary Chart

Figure 1-1 is a chart that associates the tasks just described with the chapters in which they are discussed.

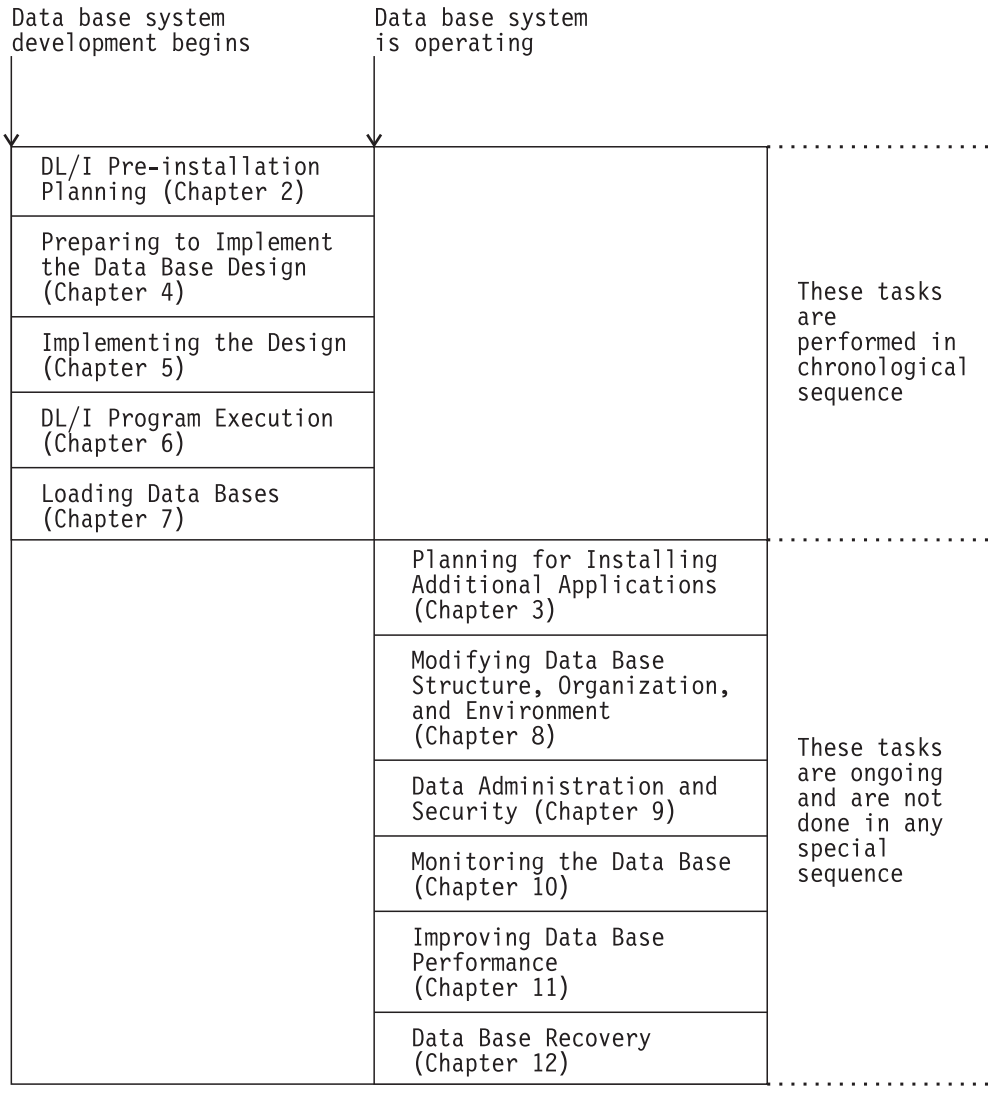


Figure 1-1. Tasks in the Data Base Development Process

Concepts and Terminology

There are a few concepts and terms you need to be familiar with before going on with the rest of this book. They are explained in this section. It is assumed that you already know:

- What a data base is and the benefits of storing data in a data base rather than in one or more conventional files (see *DL/I DOS/VS General Information*).
- What DL/I HLPI commands or DL/I CALL statements are, and how they are coded. (See *DL/I DOS/VS Application Programming: High Level Programming Interface* or *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces*.)

General Data Base Terminology

The data stored in a data base is grouped into a series of *data base records*. Each data base record is made up of smaller groups of data called *segments*. A segment is the smallest unit of data stored and retrieved by DL/I. The data in segments can be divided into one or more smaller units called *fields*.

Figure 1-2 represents a data base record of an education application data base. Each of the boxes in the figure represents a segment in that data base record. The segments contain the following information:

- COURSE—the name of the course
- INSTR—the name of the instructor for the course
- REPORT—a report the instructor prepares at the end of the course
- STUDENT—the name of a student taking the course
- GRADE—the grade the student earned in the course
- PLACE—the room in which the course is taught

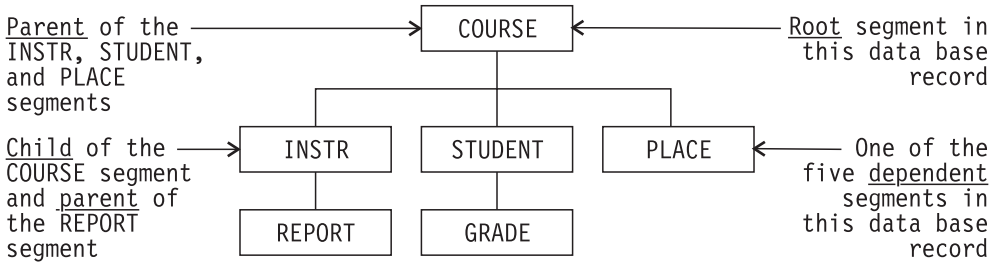


Figure 1-2. A Data Base Record in an Education Data Base

The segments within a data base record are arranged in what is called a *hierarchy*. The order of the segments in the hierarchy is significant. In this data base, all of the stored data has something to do with the courses that are taught in that school. The COURSE segment is at the top of the hierarchy. The data in the other segments of the data base record would be largely meaningless if they didn't apply to a particular course. The COURSE segment is called the *root segment*. There can be only one root segment in a data base record.

All other segments in the data base record (INSTR, REPORT, STUDENT, GRADE, PLACE) are called *dependent segments*. Their existence depends on the existence of the root segment. Without the root segment COURSE, there would be no reason for having, for example, a PLACE segment saying what room the course was held in.

Notice that the third level of dependent segments (REPORT and GRADE) depends on the existence of second level segments INSTR and STUDENT. Without the second level segment STUDENT, for example, there would be no reason for having a GRADE segment saying what grade the student earned in the course.

The way in which segments relate to each other in a hierarchy can also be thought of in different terms: parent and child. A *parent segment* is any segment that has a dependent segment directly beneath it in the hierarchy. COURSE is the parent of INSTR. INSTR is the parent of REPORT. A *child segment* is any segment that's a dependent of another segment immediately above it in the hierarchy. REPORT is the child of INSTR, and INSTR is the child of COURSE. Notice that INSTR is a parent segment in its relationship to REPORT and a child segment in its relationship to COURSE.

The terms used so far to describe segments (root, dependent, parent, and child) actually describe the *relationship* between segments. There are other terms used for describing segments in a different sense: type and occurrence. The term *segment type* is used when speaking of segments in the data base record without reference to the data stored in the segments (for example, the COURSE segment or the INSTR segment) The term *segment occurrence* is used when speaking of one particular occurrence of a particular segment type (the occurrence of the COURSE segment that contains MATH 1001, for example).

Figure 1-2 shows the segment types in the education data base record.
Figure 1-3 shows the data base record with segment occurrences. The contents (BAKER and COE) of two occurrences of the STUDENT segment type are shown. We can speak of BAKER and COE as multiple occurrences of the STUDENT segment type. MATH is an occurrence of the COURSE root segment type.

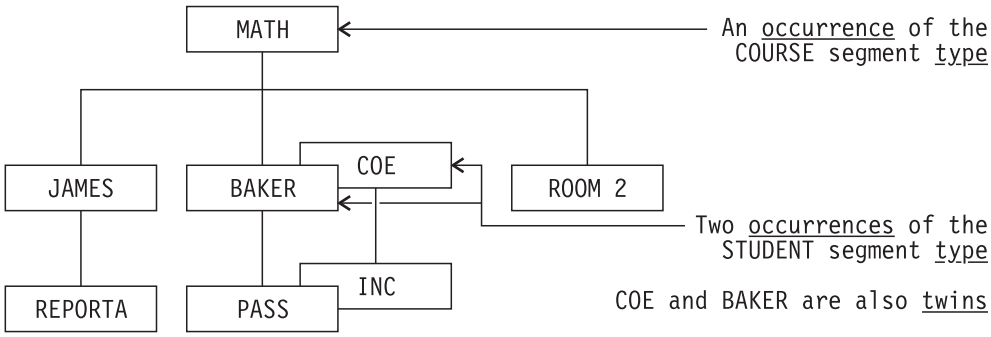


Figure 1-3. Occurrences in a Data Base Record

There is another term used for multiple occurrences of a segment type. It is *twin*. Twin (like root, dependent, parent, and child) describes a relationship between segments. Twin segments are multiple occurrences of the same segment type under a single parent. In Figure 1-3, BAKER and COE are twins. They have the same parent (MATH), and BAKER and COE are the same segment type (STUDENT). PASS and INC are *not* twins. Although they're the same segment type, they don't have the same parent occurrence.

We have been looking at some general data base terminology. Now, we will look, in more detail, at the concepts involved with some of these terms.

Hierarchy

We've said that a data base is composed of a series of data base records, that the records contain segments, and that the segments are arranged in a hierarchy within the data base record.

When a data base record is stored in the data base, the segments are processed in a definite order that is directly related to the hierarchy defined for that data base. Starting at the top of the data base record (at the root segment), segments are processed in the sequence illustrated in Figure 1-4. The numbers in the lower right of the boxes indicate the sequence. (Notice that a new segment type (TOPICS) has been added under INSTR.)

The sequence goes from the top of the hierarchy to the bottom in the first (leftmost) *path* or leg of the hierarchy. When the lowest segment type in the leftmost path is reached, the sequence continues from left to right. When all segments in that hierarchical path have been processed, the sequencing moves back up to the first common node (the point between COURSE and STUDENT, in our example). The next path to the right is selected (STUDENT to GRADE), and sequencing proceeds again from top to bottom and then from left to right. (In this second leg of the hierarchy there's nothing to the right at the lowest level, so sequencing goes back up to the common node and back down to PLACE).

This kind of sequence is called *top-to-bottom*, *left-to-right*.

Figure 1-4 shows the sequencing of segment *types*. Figure 1-5 shows the same data base record, with segment *occurrences* rather than types.

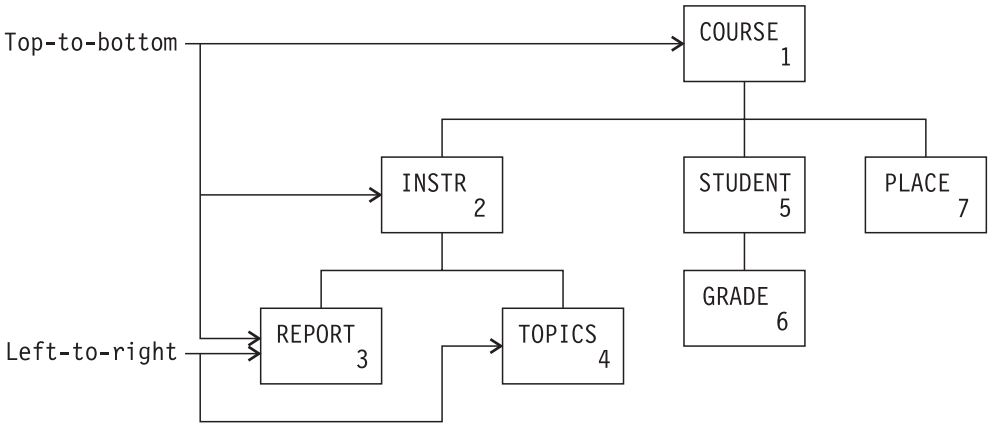


Figure 1-4. Sequence in a Hierarchy (Showing Segment Types Only)

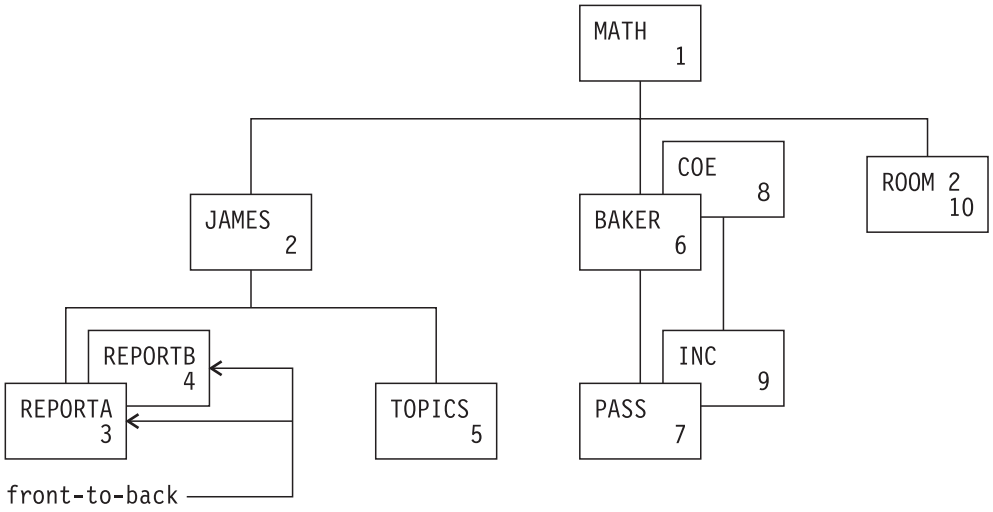


Figure 1-5. Sequence in a Hierarchy (Showing Segment Types and Occurrences)

Something is added in the processing sequence here. Notice that the numbering sequence is still from top to bottom in the leftmost path, but when we reach the bottom of the hierarchy there are two occurrences of the REPORT segment. Because we're at the bottom of the hierarchy, both segment occurrences are processed before we move to the right in this path. As you can see, both reports relate to the instructor segment JAMES, so it makes sense to process them together in the data base. In the second path of the hierarchy, there are also two segment occurrences, this time of the student segment. However, we're not at the bottom of the hierarchical path until we reach the grade segment PASS. The second student segment occurrence (COE) is not processed until after everything under BAKER has been handled. This makes sense as you want to keep student and grade (BAKER and PASS) together. Notice that the grade (INC) under student COE isn't considered another occurrence under BAKER. COE and INC become a separate path in the hierarchy. Only when the bottom of a hierarchical path is reached is the top-to-bottom, left-to-right sequencing interrupted to pick up multiple segment occurrences. Sequencing in the hierarchy can actually be referred to as *top-to-bottom*, *front-to-back*, *left-to-right*, but you must remember that “front-to-back” only occurs at the bottom of the hierarchy. Multiple occurrences of a segment at any other level are sequenced as separate paths in the hierarchy.

If an application program requested all segments in a data base record in hierarchical sequence, this is the order in which segments would be presented to the application program.

One more term associated with hierarchies: level. *Level* is the position of a segment in the hierarchy in relation to the root segment. The root segment is always on level one. Figure 1-6 illustrates the concept of levels.

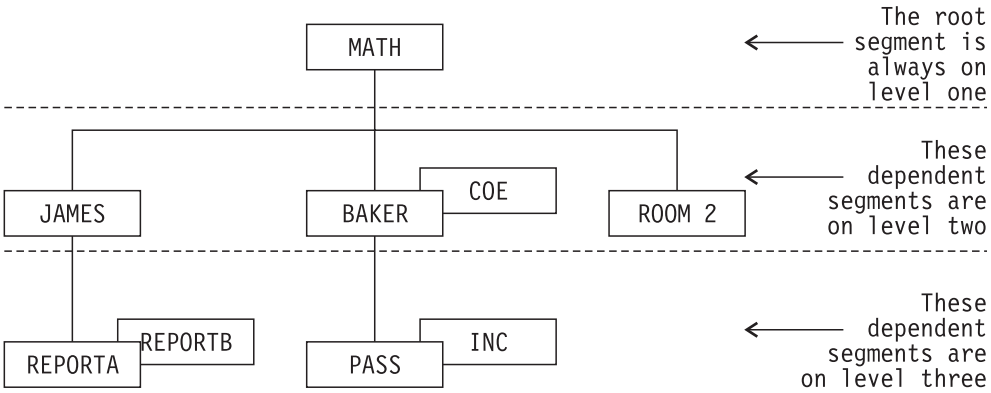
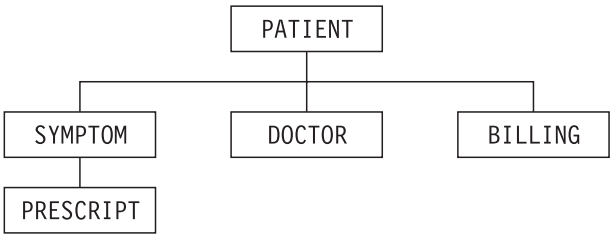


Figure 1-6. Levels in the Data Base

Data Base

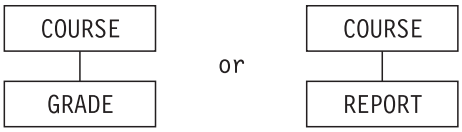
A DL/I *data base* is the entire stored collection of all occurrences of one type of data base record. One specific data base can only contain one kind of data base record. That is, each data base record must have the same root segment type. The data base containing the education data base record we have been using as an example could contain any number of occurrences of that record. It could not contain any occurrences of this medical data base record, for instance.



Data Base Record

We have said that a data base record is made up of segments. It is important to understand that a specific data base record, when stored in the data base, doesn't have to contain occurrences of all of the segment types possible in that record. To exist in a data base, a data base record need only contain an occurrence of the root segment. For example, in our education data base, all four of the records shown in Figure 1-7 could be stored.

However, no segment can be stored unless its parent is also stored. We could *not*, for example, store:



Occurrences of any of the segment types possible in a data base record can be added to or deleted from the data base at a later time.

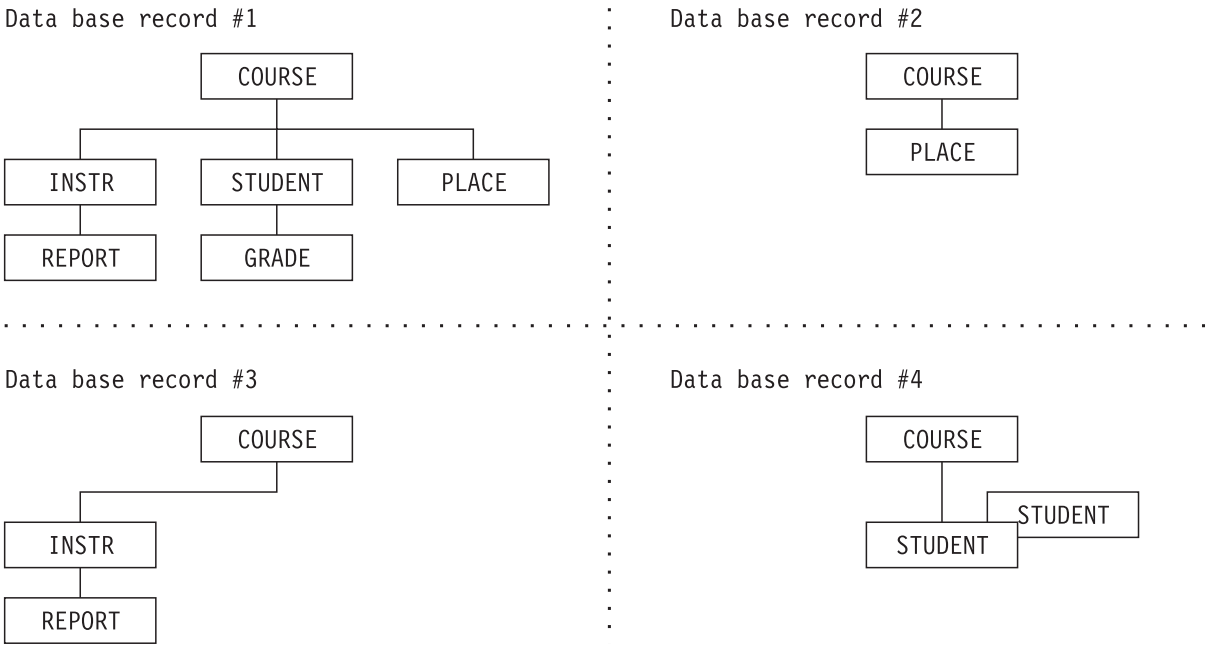


Figure 1-7. Possible Records in the Education Data Base

Segment

We've seen that a segment is the primary unit of data in a DL/I data base. Here are some additional facts about segments:

- There can be only one root segment type in a data base record, and it can have only one occurrence in that data base record.
- A data base record can contain a maximum of 255 segment types; each, other than the root segment, having zero to n occurrences (unless you specify that there is to be no more than one occurrence). The number of segment occurrences is limited only by the amount of space allocated for the data base.
- The length of a segment is specified by you. The maximum size of a segment varies from 4086 bytes to 32766 bytes, depending on the DL/I access method used. In any case, a segment can't be larger than the physical record length of the device on which it's stored.
- The root segment usually contains a *sequence* field. The value in the sequence field controls the position of the data base record within the data base. The sequence field is also called a *key* field.
- Segments lower in level in the hierarchy can also have sequence fields. These keys can be used to sequence multiple occurrences of the same segment type within a data base record.

Now we are ready to go on to a discussion of the tasks involved in planning for the addition of DL/I to your data processing installation.

Part 2. Installation Planning

Chapter 2. DL/I Pre-Installation Planning

This chapter describes the planning that must be done before DL/I is installed. There are four major topics:

1. User Responsibilities in Installing DL/I
2. Application Design
3. Data Base Design
4. Project Approach to Planning

The first section describes the items that are your responsibility in preparing for a DL/I installation. They include determining hardware, programming system, and personnel requirements; data base design; user-written programming; system implementation; operational procedures; and scheduling. The second section describes the importance of application design to the success of a DL/I installation. The next section introduces data base design as important to the success of applications making use of DL/I data bases. The final section suggests the use of the project approach as a method of efficiently accomplishing the tasks described above, and illustrates the approach with an example.

User Responsibilities in Installing DL/I

This section describes items that are involved in the successful installation of the DL/I system. These areas are the responsibility of personnel in your data processing installation. Pre-installation planning is the task of understanding and meeting these responsibilities. The more successfully this task is completed, the more satisfactory your DL/I installation will be.

Machine Requirements

Minimum DL/I Configuration

DL/I DOS/VS operates on any IBM machine currently supported by VSE that has sufficient real storage to meet the combined requirements of DL/I DOS/VS, VSE, and other customer required applications.

The minimum machine requirements for DL/I DOS/VS are the same as those needed for conventional VSE operation. Because of virtual storage support, the minimum machine configuration is dependent on application characteristics and performance requirements. Central processing unit performance can be traded off against main storage size.

DL/I DOS/VS used in conjunction with CICS/DOS/VS in a VM/370 virtual machine (under control of VSE) is subject to the following considerations:

- CICS/VS operating in a virtual machine has the same requirements as CICS/VS operating in a real machine. Other software components (access methods, compilers, and the release of VSE under which CICS/VS runs) must be valid for that release of CICS/VS.

- The minimum hardware requirements of CICS/VS operating in a VM/370 virtual machine are the same as those for CICS/VS running in a real machine, and should be considered as additional to the minimum requirements for VM/370 itself, and any other virtual machines within the VM/370 environment.
- CPU utilization and possibly terminal response times will be greater when CICS/VS is running under VM/370 than when it is running in a real machine. The effect on performance will be most noticeable when VM/370 is introduced into an installation where CPU and main storage resources are already substantially committed to existing CICS/VS and other work. General guidance on the performance considerations associated with the running of CICS/VS under VM/370 is contained in the *CICS/VS Performance Guide*.

The minimum requirements for executing DL/I DOS/VS are outlined below. The real storage requirements of the DL/I application program are not included. Two environments are shown, a batch environment and an online environment in conjunction with CICS/VS. In both cases, a limited and a full set of DL/I functions are considered to show the resulting difference in the real storage requirements.

The estimates of real storage requirements are derived from the size of the modules in the DL/I system necessary to handle normal execution. Modules handling exceptional conditions such as errors or open/close, have not been included. Thus, if the specified real storage is available to DL/I and if no exceptional condition arises, then no paging will occur in the DL/I code. Of course, as for every system executing in a virtual storage environment, if performance is not critical, DL/I can execute in less real storage. Also, a trade-off may be considered between the available real storage in a central processing unit model and its instruction rate. For example, a 4341 processor may require less real storage than a 4331 processor to achieve the same performance, since its internal speed can make up for some additional paging overhead.

The minimum storage estimates provided here are based on the following application profile:

- One HISAM data base
- Three KSDS buffers and two ESDS buffers
- VSAM Control Interval size of 512 bytes
- No use of advanced functions (secondary indexing, logical relationships, variable length segments)
- No logging

For the online system, the profile includes:

- Two PSBs in use
- One active task
- Program Isolation not selected.

System Function	Units Permitted
Processing Unit	Any IBM system supported by VSE with a minimum real storage of 256K for a batch system (including VSE/VSAM) or 512K for an online system (including CICS/DOS/VS and VSE/VSAM). VM/370 storage requirements must be considered if CMS/DOS is used.

System Function	Units Permitted
Minimum Real Storage (including VSAM)	The practical minimum amount of real storage is: **Batch System** • Basic retrieve operations only 80K • All functions 92K **Online system** (excluding CICS/VS requirements*) • Retrieve operations only 100K • All Functions 112K **Multiple Partition Support (MPS)** • The real storage requirements are the same for an MPS system as for an online system plus 4K to 10K for each active MPS batch partition.
Recommended Minimum Virtual Storage	Virtual Storage must be: Batch System 512K Online System 1024K
Tape Units	At least one 9-track 2400, 3400, or 8809 series tape units and control if tape logging is used, or two if SHSAM or HSAM files are on tape.
Direct Access	For system libraries and working sotrage space, any devices supported by VSE. Minimum space for system use and maintenance: Batch system (75 cylinders 2316 or equivalent). Online system with CICS/VSE (150 cylinders 2316 or equivalent). For DL/I data base storage, within the capabilities and restrictions of VSE support by the virtual storage access method and sequential access method: 2314 Direct Access Storage Facility 2319 Disk Storage 3330-1/3330-11 Disk Storage 3340/3344 Direct Access Storage Facility 3310 Direct Access Storage 3350 Direct Access Storage 3370 Direct Access Storage 3375 Direct Access Storage
Telecommunication and Programmable Facilities	Any terminal, terminal control unit, or special feature supported by CICS/DOS/VS*; or equivalent product. IMF and IUG require a 3270-type terminal supported by ISPF.**

* See *CICS/VS General Information* for Information about CICS/VS storage requirements and CICS/VS-supported terminals and features.

** See *Interactive System Productivity Facility Installation and Customization VSE* for information about ISPF.

Typical DL/I Real Storage Requirements

The following is an example of the real storage requirements for a typical DL/I system that could serve as a usable production system. The figures are based on the following application profile:

- Five HDAM data bases
- One HIDAM data base
- Use of advanced functions (secondary indexing)
- Use of data base change logging
- One buffer subpool with 12 buffers of 2K each
- Five 1K buffers for each of two INDEX data bases (HIDAM primary index and one secondary index).

For the online system, the profile covers:

- 10 PSBs in use

- 3 tasks running concurrently

Typical real storage requirements are:

Real Storage	Batch system	
	• Retrieve operations only	114K
	• All functions	150K
	Online system (excluding typical CICS/VS requirements)	
	• Retrieve operations only	124K
	• All functions	160K
Virtual storage (including real storage)	Batch System	600K
	Online System	1024K

For a more detailed discussion of storage considerations and how to calculate your own virtual and real storage requirements, see hdref refid=virsto and refid=realsto.. i1.system requirements, programming

Programming System Requirements

DL/I DOS/VS (Version 1.7) operates under the VSE/Advanced Functions program product (Release 1.3.5 and later).

DL/I DOS/VS also operates in batch mode, with certain restrictions, on the level of the Virtual Machine/370 (VM/370) Conversational Monitor System DOS/VS simulator (CMS/DOS), which supports the equivalent DOS/VS or VSE/Advanced Functions release.

DL/I DOS/VS is designed to work with DOS/VSE (5745-020 and 5745-030) and the following Licensed Programs (or their equivalents):

- VSE/Advanced Functions, 5746-XE8, Release 1.3.5
- CICS/DOS/VS, 5746-XX3, Version 1 Release 6 (Release 1.6)
- VSE/VSAM, with Space Management for SAM feature, 5746-AM2, Release 2 or later
- Structured Query Language/Data System (SQL/DS), 5748-XXJ, Release 1 or later
- DOS/VS Sort/Merge, 5746-SM2, Version 2 Release 3.0 or later
- VSE/POWER, 5746-XE3, Version 2 Release 1
- Interactive Productivity Facility, 5748-MS1, Release 3 (This facility can be used only with DL/I releases prior to Version 1.7)
- Interactive System Productivity Facility, 5668-960, Release 1 or later (This facility can be used only with DL/I Version 1.7)
- VSE/ICCF (VSE/Interactive Computing and Control Facility), 5746-TS1, Release 3.5

VM/370 SCP, 5749-010, supports IMF and IUG through CMS and the following Licensed programs (or their equivalents):

- Interactive Productivity Facility, Release 3, 5748-MS1 (This facility can be used only with DL/I releases prior to Version 1.7)
- Interactive System Productivity Facility, 5668-960, Release 1 or later (This facility can be used only with DL/I Version 1.7)
- VM/Basic Systems Extensions, 5748-XX8
- VM/Systems Extension, 5748-XE1
- VM/System Product, 5664-167

DL/I DOS/VS is supported for use with:

- DOS/VS COBOL Compiler and Library, 5746-CB1, or Library only, 5736-LM4
- Full ANS COBOL V3 Compiler, 5736-CB2 and Full ANS COBOL Library, 5736-LM2
- ANS Subset COBOL, 5736-CB1
- PL/I Optimizing Compiler and Libraries, 5736-PL3
- PL/I Optimizing Compiler, 5736-PL1
- PL/I Resident Library, 5736-LM4
- PL/I Transient Library, 5736-LM5
- RPG II - DOS/VS, 5746-RG1
- VM/370 SCP, 5749-010
- VM/Basic Systems Extensions, 5748-XX8
- VM/Systems Extensions, 5748-XE1
- VM/Systems Product, 5664-167.

DL/I DOS/VS is designed to run in batch or Multiple Partition Support (MPS) batch mode in VSE/Interactive Computing and Control Facility (VSE/ICCF), 5746-TS1, pseudo partitions. VSE/ICCF can also be the host environment for the Interactive System Productivity Facility (ISPF), 5748-MS1, under which the DL/I DOS/VS Interactive Macro Facility (IMF) and Interactive Utility Generation (IUG) run.

DL/I DOS/VS uses the Virtual Storage Access Method (VSAM), Sequential Disk IOCS, and Magnetic Tape IOCS data management facilities.

DOS/VS DB/DC Data Dictionary (5746-XXC) can be used with DL/I DOS/VS to help simplify and manage the creation, maintenance, and reporting of definitions or descriptions of the data in the system and the data of the application programs.

Development Management System/CICS/VS (5746-XC4) provides DL/I DOS/VS support for inquiry, insertion, update, and deletion of data base records. Access into the data base may be by way of either primary or secondary indexes.

DL/I DOS/VS Space Management Utility IUP (5746-PKF) is designed to help improve system performance and programmer productivity. These utilities assist the programmer by detecting and reporting DL/I hierarchical direct (HD) pointer discrepancies, providing statistics and information for HD tuning, and assisting with segment restructuring and reloading during data base reorganization.

Personnel

This section describes tasks that are unique to DL/I, in relation to each of the roles filled by personnel in your data processing installation. How these tasks are distributed is a function of the way your installation is organized. Some of the tasks will require additional training of some sort. Assignment of the tasks and the scheduling of the required training must be handled as part of pre-installation planning.

Roles

Application Design: Application design for an application making use of DL/I data bases is different from conventional application design only in respect to the inclusion of data bases. However, that inclusion does make a major difference in the process. Personnel doing application design have to:

- Understand the purposes and advantages of using data bases

- Learn how to collect, organize, and analyze the requirements that the application will fulfill
- Learn how to design efficient data structures that will not require frequent and expensive revision or redesign
- Learn enough about the physical design of data bases to be able to recognize and handle requirements and restrictions imposed by the physical design.

(These subjects are covered in *DL/I DOS/VS Application and Data Base Design*.) planning and design, data base

Data Base Planning and Design: Data base planning and design is a new role within your data processing organization. It is closely related to application design and may be performed by the same personnel, as discussed above. Because of the importance of good data base design to the success of the installation, prior experience or training is recommended for anyone performing this task. Part 2 of *DL/I DOS/VS Application and Data Base Design* describes the process of data base design.

Data Base Administration: Whether handled by one person, or by several, data base administration is a key function when you install DL/I. Data base administration is involved in or responsible for many tasks, including:

- Installing DL/I
- Designing data bases
- Establishing conventions and procedures
- Assuring the security of the stored data
- Monitoring the performance of the DL/I system and data bases
- Improving data base performance
- Planning and setting up procedures for recovery of the data base when failures occur.

System Programming: Adding DL/I to the data processing system affects the system programming function in such areas as:

- DL/I installation and maintenance
- CICS/VS installation and maintenance if DL/I is to operate online
- System performance
- Storage estimates
- System back-up and recovery procedures

System Operation: The persons responsible for handling the operation of DL/I will be required to initialize the system for DL/I operation (and CICS/VS if DL/I is to be used in the online or MPS environments). If logging is used, the tapes or disk extents must be set up and handled when filled. Procedures for back-up, recovery, and restart have to be learned and executed.

Application Programming: The High Level Programming Interface commands are used in application programs to load, retrieve, modify, add, and delete information stored in DL/I data bases (the DL/I CALL interface statements can be used instead, if desired). Persons writing application programs must learn how to use the commands, how to structure their programs to use DL/I most efficiently, and how to test the status codes DL/I returns after the execution of each command.

Training

The training required of the personnel in your installation should be completed before DL/I is installed, or during the installation period if that is not possible. In any case, you should not expect DL/I to be learned on the job.

It is possible that the necessary training could be accomplished through study of the various books in the DL/I library. However, they are not designed to be the primary source of training. IBM provides a number of courses and materials, such as those in the Independent Study Program, that can be used at your location. There are also a number of IBM courses that personnel could attend in person. Contact your IBM branch office for complete information on the IBM instructional offerings.

Data Base Design

Data base design is a task that is unique to installations using a data base management system such as DL/I. It is important that planning for data base design is carried out during the pre-installation phase. This includes the selection and training of personnel for the task.

Data base design can be considered in two phases. The first is associated with application design, and can be accomplished as part of that process. It involves the creation of “local views” of the data for each application program using the data base. These local views are then combined into a “system view” that represents the data base in a conceptual form. The second phase of data base design consists of converting the system view into a physical design of the data base, taking into consideration all requirements and restrictions involved with the application and with DL/I. The people who perform the two phases do not necessarily have to be the same people.

The first phase of data base design is covered in *DL/I DOS/VS Application and Data Base Design*. The second phase is covered in part 3 of this book.

User-Written Programming

In order to use DL/I and DL/I data bases, certain types of program must be written by programming personnel in your installation. Planning for the writing of these programs should be part of the pre-installation procedure. The various types are discussed separately in this section.

Data Base Loading Programs

Before any data base can be used, it must be loaded with the initial data that it is to contain. This loading is done by an application program using the DL/I LOAD command or ISRT call to tell DL/I where each data segment is to be placed in the data base. The load program must be completed, tested, and executed before any application program can run against the data base.

Application Programs

The application programs for the first applications to be put into production must be planned during the pre-installation period. The writing and testing of these programs should proceed as DL/I is being installed. They must be ready for use as soon as the appropriate data bases have been loaded.

Other User Programming

There are other programming tasks that may have to be planned during the pre-installation period, depending on the way your installation uses DL/I. These include:

- One or more randomizing routines for data bases that use the HD randomized access method
- User field exit routines
- Segment edit/compression routines to manipulate segments during storage and retrieval.

These are discussed in detail in chapter 4 and Appendix G, “Randomizing Modules and DL/I User Exit Routine Interfaces” on page G-1.

Use of Existing Programs and Program Packages

There are IBM programs and program packages available that will help in the design, documentation, and use of data bases (ELIAS, Data Dictionary, etc.). These should be discussed with your IBM representative during the pre-installation period so that you can choose those that will be of use to you. There may also be other programs already existing in your installation or within your industry that may be available to you. Some pre-installation time spent in looking for these could prove worthwhile.

System Implementation

Tasks that must be performed in the installation of DL/I and the generation of an operable system include:

- DL/I installation
- CICS/VS system generation if online or MPS environments are used
- Online nucleus generation if online or MPS environments are used
- Data base description generation for every data base.

Operational Procedures

An important task in assuring a smooth-running DL/I installation is the planning and setting up of operational procedures. These may include areas such as:

- Security

  You need to set up naming conventions for all named items associated with your data base system, and document the use of these items by every program, in order to be able to control which programs are to be allowed access to which data. You need to set up rules for determining what data is sensitive, and procedures for securing it. There may be a need for securing data from access by non-DL/I programs. All of these considerations should be included as part of an over-all security plan and procedures.

- Data base maintenance

  Procedures for maintaining your data bases need to be defined. They should specify how often the data bases should be backed-up, and the procedure for doing it. They should specify the criteria for scheduling data base reorganization and the procedures to be followed.

- Performance monitoring

  Procedures need to be established for monitoring the performance of the DL/I system and data bases. These should specify the tools to be used, the

intervals at which they should be used, and the procedure to follow when performance has dropped below acceptable levels.

- Data base recovery

  Occasionally a data base will be damaged in some way, causing data to be lost or made inaccurate. You need to be ready with a recovery plan that will cover every possible situation.

Scheduling

Scheduling the various tasks described in this chapter is extremely important in assuring a smooth DL/I installation and move into production. Because each installation involves so many variables, it is impossible to generate a generalized schedule that will cover each case realistically. However, we can give a few suggestions to help you in setting up your own schedule.

- *Training*

  The training of personnel who will be involved in data base administration and system programming functions should begin at the earliest possible time. This should be soon after the decision to install DL/I is made. People involved in application design and application programming can start their training somewhat later if necessary. System operation personnel can receive their training later; perhaps just before actual installation of DL/I begins.

- *Machine and programming system requirements*

  These must be determined at the earliest possible time, so that any new hardware and software can be ordered and will be available when needed.

- *DL/I system development*

  The overall design of the DL/I system should begin as soon as the personnel involved have received enough training to be able to handle the task. Application design for the first application to be implemented should be started as early as is practical. Data base design needed for that application should begin as soon as the requirements are available. Load programs and system type programs and routines (such as the HD randomizing routine and user exit routines) should be scheduled so that they are available for initial testing. Design and coding of the application programs can begin at about the same time.

- *Installation*

  Any new hardware should be installed and thoroughly tested before any attempt is made to make DL/I operational. The actual installation of DL/I and any associated software (such as CICS/VS if online operation is planned) can begin at any time after the hardware is available. Testing of DL/I can begin at once, using the sample application shipped with DL/I. This will give all of the personnel experience with the system before moving to your own application.

- *System test*

  System test can begin using the DL/I sample application, then move on to use of the programs of the first application to be implemented, using test data bases initially. Once the first portion of the application is running on a test data base, the first production data can be loaded and the entire system tested, before it is turned over for production runs.

Application Design

Application design is important to the success of any data processing application, but it is vital to the success of an application using data bases. Many of the advantages of data bases will be lost if they aren't taken into consideration during the design of the application. Since a data base may contain information used by several applications, it is important that its design be coordinated over all the applications.

Application design should be carried out in an orderly manner, using a procedure like the following:

- Make a preliminary analysis to determine whether or not the application should use data bases
- Obtain the application requirements. This is done primarily by interviewing end users of the application.
- Perform an application analysis on the requirements to determine the number and type of programs needed
- Create local views of the application for each program
- Combine the local views into a system view ready for implementation as a physical data base
- Implement the application.

These subjects are covered in *DL/I DOS/VS Application and Data Base Design*.

Data Base Design

Data base design is one of the most important tasks to be performed in a data processing installation that uses data bases. Use of data bases makes it possible for many areas of the company to access the same data. Thus, a large part of the company's operation can be affected if the data bases are not designed for efficiency, ease of upkeep, ease of recovery, and security. A poor data base design will negate the advantages of using data bases. It will result in user dissatisfaction, expense, loss of time, poor performance, resistance to further use of data bases, and a host of other ills.

The pre-installation planning period is the time to prepare for successful data base design by selecting the right people for the job and seeing that they are trained to do it. Part 3 of *DL/I DOS/VS Application and Data Base Design* gives a data base design procedure that helps assure good data base design.

Project Approach to Planning

The implementation of a DL/I application is most successfully done by using the project approach. This approach makes sure 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, you may want to try using a project team with a definition of the tasks and responsibilities of all parties involved.

Project Cycle

Like most other data processing projects, a DL/I project can generally be divided into the following phases:

- Feasibility study
- Planning
- Design
- Design review
- Implementation
- Testing
- Production

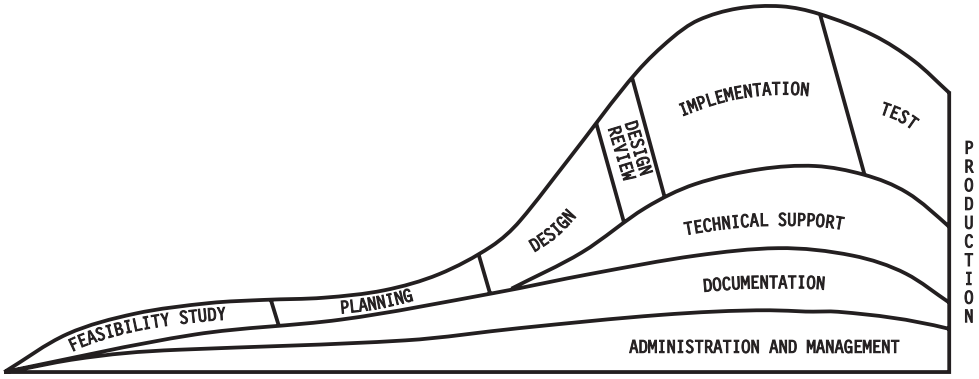


Figure 2-1. The Project Cycle

Figure 2-1 shows the relative manpower requirements for each of the phases.

The project cycle can be described in greater detail by breaking it down like this:

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

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

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

Design and Implementation: The system is designed, followed by a design and performance review. After design approval, detailed designs are worked out together with a test plan.

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

Production: Production is started. Any further changes to the system are controlled through maintenance procedures.

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

- Naming of data base items such as DBDs, PSBs, segments, fields, etc.
- 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 is under the control of the data base administration function.

Sample Project Plan

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

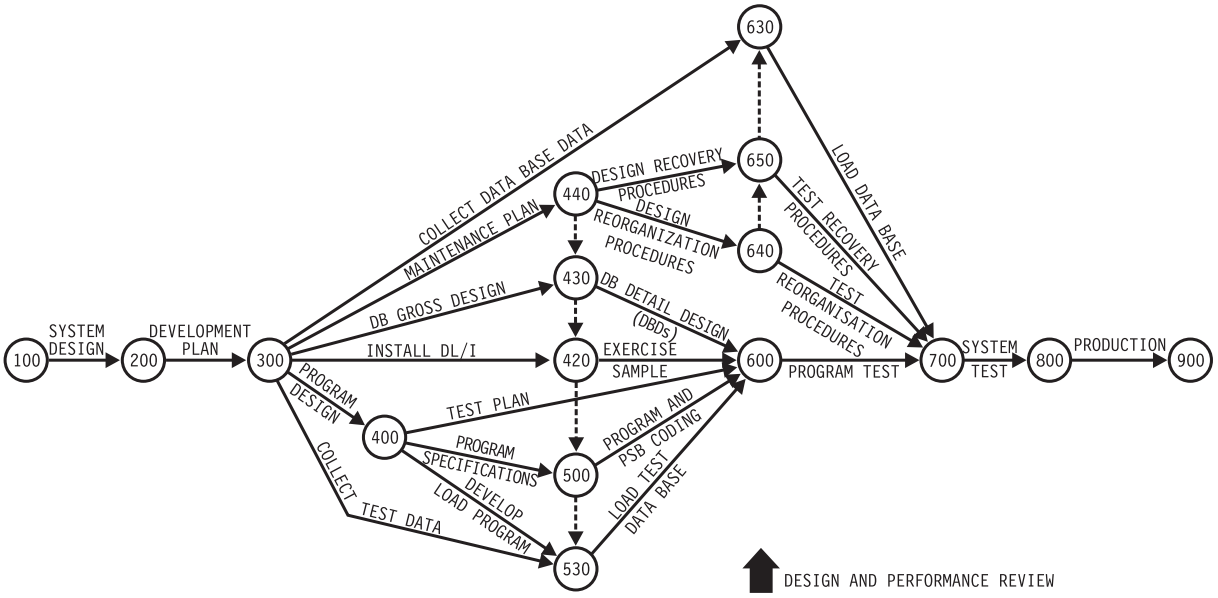


Figure 2-2. DL/I Installation Plan PERT Chart

Gross PERT Chart

Figure 2-2 shows a gross PERT chart for the implementation of a DL/I 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's environment. The following descriptions apply to the activities shown in the chart.

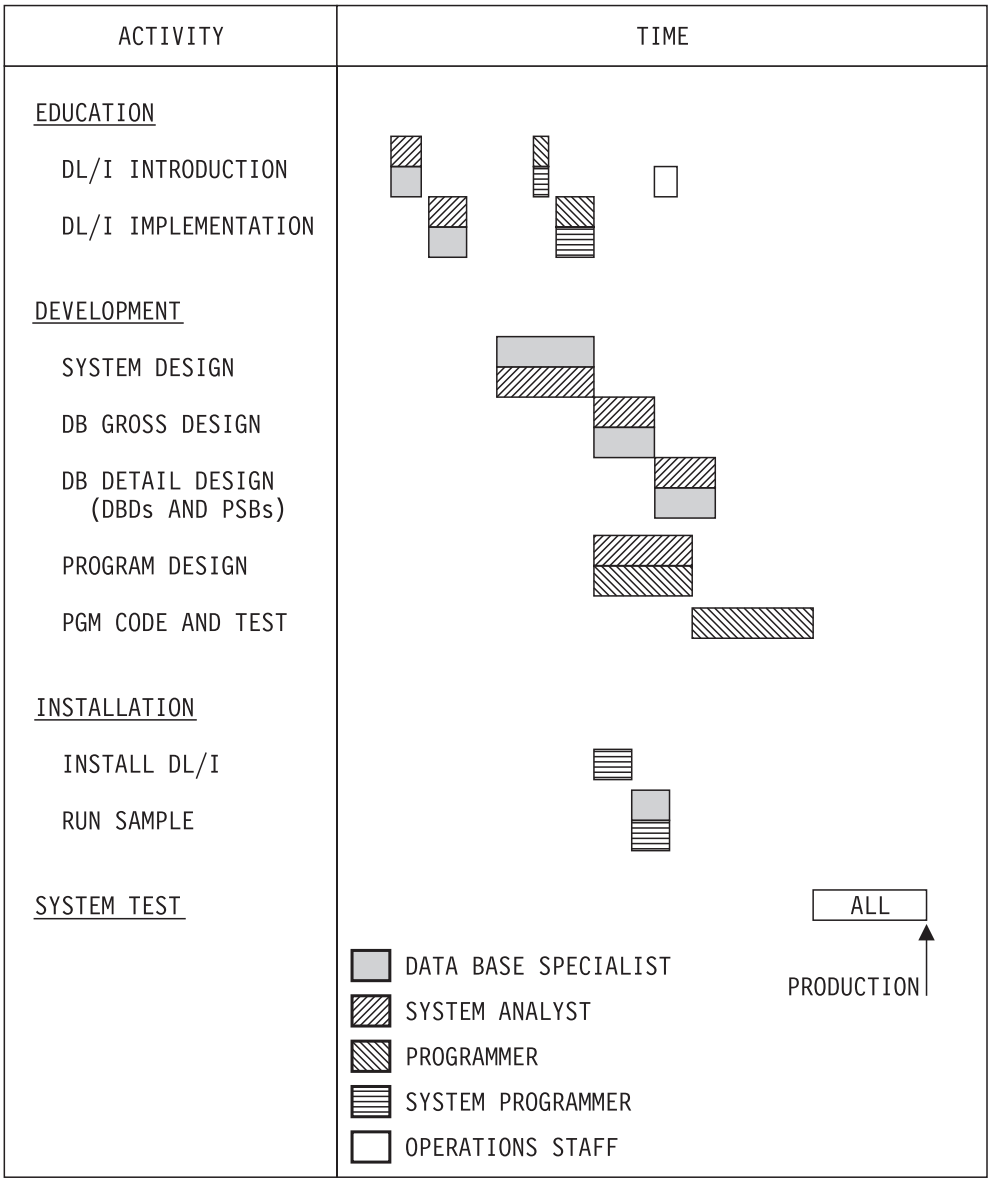


Figure 2-3. Sample Gantt Chart

System Planning (000-100): The sample PERT chart is adapted to your project. Manpower and machine time estimates are compiled. External references are defined. Elapsed time calculations are performed and the chart is extended with the proper time frame. The critical path is calculated. A *Gantt chart* can be constructed showing the duration of each activity and the people involved. Figure 2-3 shows an example of such a Gantt chart. Your chart should clearly show the actual time 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 is devised for the development of data bases and programs. All single activities and their dependencies are determined.

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 (300-400): Each individual application program is designed. Its input, processing, output, and data base accesses are defined. Common guidelines and routines are established. Often more than 50% of the data processing programs are reports. Using COBOL, PL/I, or RPG report writer features can significantly reduce the required manpower for program design.

Collect Data (300-530|300-630): Both test data and live data are collected, or procedures and programs are established for the conversion of existing data files.

Recovery and Reorganization (300-440-650|640-700): A plan for recovery and reorganization can avoid later redesigns and reprogramming. These procedures, although rarely needed, are vital to data base integrity and availability. Therefore, a thoroughly tested 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 recovery and reorganization procedures. An incomplete treatment of this topic is the most common source of problems with data base management systems.

Install DL/I and Run Sample Application(s) (300-420-600): The system programmer installs the DL/I data base system. The samples provided with the system are exercised to get practical experience with the system. Conventions and procedures are established for system maintenance.

Data Base Detail Design (430-600): The detailed logical and physical data base structures are defined. Access methods are selected and the DBDs are coded and tested.

Program Specification (400-500): Detailed flow charts are created. The data base call sequences are defined in a standard fashion.

Test Plan (400-600): A detailed test plan is made. Procedures for unit test and systems test are established.

Develop Load Programs--Load Test Data Bases (400-530-600): Load programs are designed, written, and tested with the test data; resulting in test data bases for program and recovery/reorganization tests.

Design Review (600): At this stage it is appropriate to conduct a design review. The basic aim of a 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 as expected?
- Are there any pitfalls in the data base and program design?

Program and PSB Coding and Test (500-600-700): Each application program is coded and tested, using the test data bases and the test procedures.

Load Live Data Bases (630-700): The data bases are loaded with the actual data. Backup copies are made immediately after initial load. The process at times exposes existing inconsistencies in data. You may need to include extra time to resolve these inconsistencies.

System Test (700-800): Integrated tests are executed on the live data bases. Reorganization and backup/recovery procedures are tested on those data bases.

Production (800-900): Production starts. The established monitoring and maintenance procedures are enforced. Final feedback 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 in future trouble shooting, application modification, and application extensions.

Chapter 3. Planning for Installing Additional Applications

This chapter describes the planning you must do when adding a new application after DL/I is already installed and operating. There are five major sections:

1. Data Base Considerations
2. Implementation Considerations
3. Personnel Requirements
4. Storage Requirements
5. Machine Requirements

The first section describes the considerations to be taken into account if the new application will use existing DL/I data bases, modify existing data bases, or add new ones. The second section reminds you of the many steps that may be needed to implement the application. The third section mentions possible personnel requirements. The fourth section calls attention to the fact that the new application may bring about a requirement for more storage--both real and virtual. The last section discusses the possible need for additional or different hardware because of storage or performance requirements.

Chapter 2 described the pre-installation planning tasks that must be done *before* DL/I is installed in your data processing system. It is just as important that you do careful planning before installing additional applications on a DL/I system that is already installed and running. This chapter calls your attention to some of the items you should consider in doing that planning.

Data Base Considerations

When you add an application to an existing DL/I system you have to plan according to the way in which the data base(s) needed for the new application are handled. Those data bases could fit into any of four categories:

- Existing data bases with no changes
- Completely new data bases, with no connection to existing data bases
- Existing data bases modified
- New data bases added and existing ones modified.

When using existing data bases unchanged, you should still re-evaluate usage paths and performance considerations. It could be advantageous to rearrange the data base structure to improve performance. Chapter 8 describes possible changes and how to make them.

When the new application uses completely new data bases, the planning is the same as for the initial data bases when DL/I was originally installed. See *DL/I DOS/VS Application and Data Base Design*.

When the new application requires modification of existing data bases, careful planning is required to prevent problems such as degradation of performance in existing applications. The best course is to do a complete re-evaluation and redesign of the data bases as described in *DL/I DOS/VS Application and Data*

Base Design. This will ensure that all applications can have an optimum view of the data.

When the new application requires changes to existing data bases and the addition of new ones, you should plan for a complete re-evaluation and redesign. This will ensure optimum data base designs and performance, and minimize storage requirements. See chapter 8 of this book and *DL/I DOS/VS Application and Data Base Design*.

Implementation Considerations

An important part of your planning is the scheduling of time and resources for some or all of the following:

- DBDGENs for all new and modified data bases
- PSBGENs for all new application programs and existing programs if changes to their views are necessary
- ACBGENs for each new or modified PSB
- Determination of new or modified VSAM space requirements and allocation of data sets
- If online or MPS operation is being added, CICS system generation and online nucleus generation as described in chapter 5
- If online or MPS is already in operation, evaluation of any new requirements. CICS system and online nucleus generations if needed. See chapter 5 for details.

Personnel Requirements

Determine whether additional programming and operational personnel will be needed to implement and execute the new application. If so, they may need training. Even if existing personnel are to handle the new application, they may need additional training; particularly if new function, such as online operation, is added.

Storage Requirements

Virtual

Determine whether or not the new application makes any changes in virtual storage requirements. See Appendix A, “DL/I Virtual Storage Estimates” on page A-1 for details.

Real

Determine whether or not the new application makes any changes in real storage requirements. See Appendix B, “DL/I Real Storage Estimates” on page B-1 for details.

Machine Requirements

Evaluate the effect on machine requirements of adding the new application:

- Will the existing processor be able to handle the added application with adequate performance?
- Will more disk space be required for added data?
- Will faster direct access storage be required?
- Will additional processor storage be required?
- Will terminals and associated hardware be required?

Part 3. Implementing the Data Base Design

Chapter 4. Preparing to Implement the Data Base Design

This chapter discusses items about which you may need to make decisions before implementing a data base design as a physical data base. There are five major sections:

1. Two Views of the Data
2. Choosing DL/I Definitional Capabilities
3. Choosing DL/I Operational Capabilities
4. Choosing the Access Method
5. Defining Data Base Physical Characteristics

The first section defines the two views of the data that must be described to DL/I. The second describes capabilities of DL/I that are associated with the definition of those two views. The third describes capabilities of DL/I that are associated with the operation of the data base when it is put to use. The fourth section gives guidance in choosing the DL/I access method for the data base. The last section gives information that helps in making decisions about how the data base is to be physically implemented.

This chapter describes the decisions you have to make in preparation for implementing the design of a data base. It gives you the information you need to make those decisions intelligently. Once they have been made, they will be the basis for the procedures used to actually implement the data base, as described in the next chapter.

Before you begin this task, someone in your installation must have completed the design of the data base as a conceptual structure or structures, called a system view. That structure represents the inherent properties of the data associated with it, with little consideration for the way in which you will physically implement it as a data base.

As input to your task, you will need from the designer of the data base:

- A description of the system view
- A listing of the requirements the application is expected to meet
- A naming convention that includes specifications for all of the DL/I data-base-related components. Although there are restrictions on DL/I names that vary from component to component, all names will be acceptable if they begin with an alphabetic character, contain only alphameric characters, and contain no at-signs (@). For the specific requirements of each component, see Appendix D, “A Recommended Naming Convention” on page D-1.
- A data dictionary that lists all the pertinent information about the data the application requires
- A list of implementation requirements such as:
  - Performance considerations.

    You need an estimate, for instance, of how many occurrences of each field and segment will be stored; how often each will be accessed; the relative frequency of retrieving, updating, inserting, and deleting them; and which are likely to have a great deal of update or change activity.

– Data access requirements.

  You need to know how each program in the application accesses the root segments and how it accesses segments within data base records.

– Structural considerations.

  You need to know when use of such DL/I capabilities as logical relationships, secondary indexing, and variable length segments are required; as well as any other structural details that were included in the system view for a specific reason.

– Security requirements.

  You need to know what segments and fields must be made secure from unauthorized access.

– Recovery requirements.

  You need to know of any special requirements that would influence the planning and scheduling of data base back-up, reorganization, and recovery.

A data base design procedure for generating the system view and the other information listed above is described in *DL/I DOS/VS Application and Data Base Design*. Your installation can, of course, develop its own design procedure; but whatever procedure is used, it should provide you with the types of information described above.

Two Views of the Data

In creating a DL/I data base, you have to define two different views of the data. Each type serves a different purpose and involves making different decisions. The purposes are described in this section. The decisions are described in the rest of the chapter.

System View: The system view represents a structure that contains all of the data needed by *every* application that accesses the data base. The system view has to be described to DL/I. You do this by coding statements called a data base description (DBD). The DBD is created when the statements are executed as a normal application program job called a DBD generation (DBDGEN).

Application View: An application view represents a structure that contains all of the data needed by *one* application program that accesses the data base. It could be the same as the system view, but usually is only a part of the structure in the system view. A particular program may access different data bases and must have an application view of each data base. The application view must be described to DL/I. You do this by coding statements that describe a program communication block (PCB). The PCB statements that describe every application view needed by one program are executed together as a normal application program job called a program specification block generation (PSBGEN).

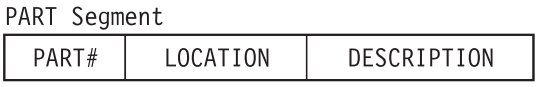
Choosing DL/I Definitional Capabilities

Data Base Structures

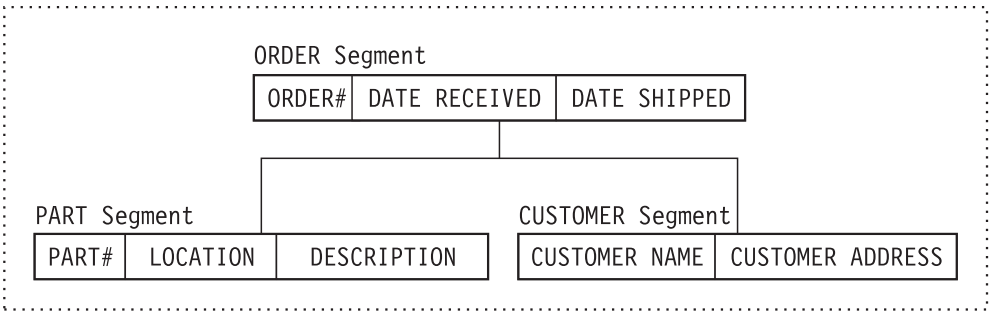
Structural Elements

Fields and Segments: When using DL/I you can organize your data into various kinds of structures. The basic building block of these structures is the *field*. Fields are single pieces of information, such as a part number or a shipping date.

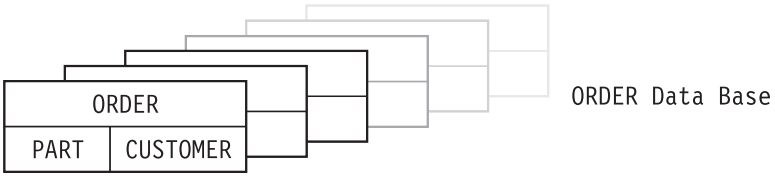
A set of related fields are grouped together into a *segment*. The segment is the unit of reference for DL/I. In order to access a field, the segment containing it must be requested.



A set of related segments makes up a *data base record*.



A set of data base records of the same type forms a *data base*.



Segment Types and Segment Occurrences: The structure of a data base record and the formats of the segments that are included in it are identified to DL/I by defining *segment types*. In defining a segment type, you define the types of fields that make it up, and its relationships to other segments.

An example of a segment type would be one that contains information about employees.

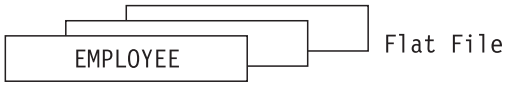
A *segment occurrence* is one instance of a segment type in a data base. An example of a segment occurrence would be the specific employee information segment for John Jones.

Structural Organizations

Segments in DL/I can be organized in one of three different ways: flat file, hierarchical, or extended structures.

Flat Files: If the data base being defined contains only a single type of segment, the structure is said to be a flat file. Using DL/I to access VSE SAM or VSAM files as flat files adds recovery capability to SAM or VSAM.

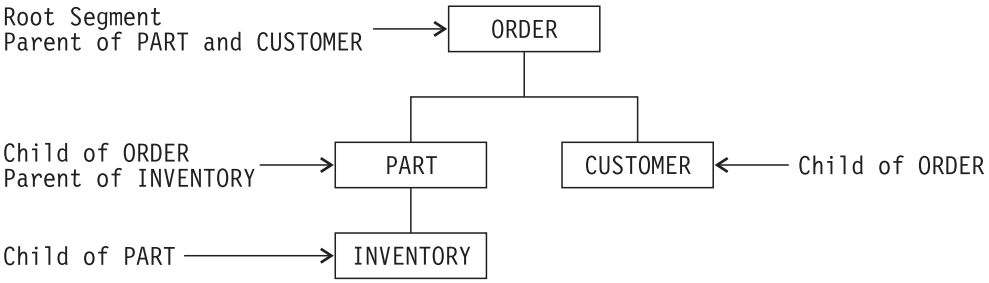
The DL/I access methods provided for flat file support are the Simple Hierarchical Access Method, *SHSAM* and the Simple Hierarchical Indexed Access Method, *SHISAM*.



Hierarchical Structures: For DL/I data base records containing more than one type of segment, DL/I provides hierarchical structure support.

In a hierarchical structure, one segment type (the *root* segment) is defined at the top of the structure. Additional segment types (dependents) are defined as required below the root segment. DL/I supports up to 255 different segment types for a data base and up to 14 levels of dependents under the root (a total of 15).

Each segment other than the root segment must have a segment above it in the hierarchy as its *parent*. The dependents immediately below a segment in a hierarchy are known as its *children*. Although each child segment may have only one parent, each parent may have many different child segment types.



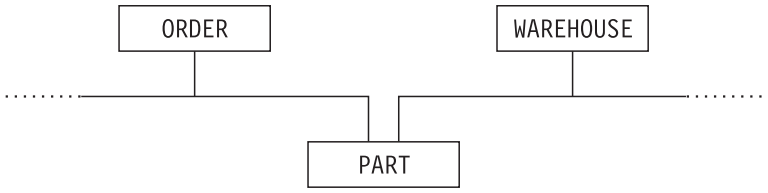
Multiple occurrences of a particular child type under one parent occurrence are called *twins*, or *physical twins*.

The DL/I access methods provided for hierarchical structure support are the Hierarchical Direct access method, *HD*; the Hierarchical Sequential Access Method, *HSAM*; and the Hierarchical Indexed Sequential Access Method, *HISAM*. HD provides two means of access: *HD indexed*, with access provided through an index, and *HD randomized*, with access provided through a randomizing routine.

Extended Structures: While all applications must view data bases as either flat files or hierarchies, DL/I supports and maintains system data structures of a more complex nature, on which these views are based. These structures are supported through a mechanism called logical relationships.

Logical relationships can be used for either of two purposes:

- To let a segment have more than one parent segment type
- To let a segment type that occurs elsewhere, either in this data base or another, be included as a dependent of a particular segment.



DL/I supports extended structures only for the HD access methods.

Details for Structure Definitions

Flat Files

There are no structural considerations for flat files, since there is only one segment type in this type of data base.

Hierarchical Structures

You need to make two decisions in relation to the definition of hierarchical structures:

1. The relationships between parents and children (the immediate dependents of a segment)
2. The left-to-right order of dependent segment types (the top-to-bottom order is fixed by the requirements of the application).

Techniques for Connecting Parent and Child Segments

DL/I provides two methods of connecting parent and child segment types: sequentially and by direct pointers.

In the sequential method, used by HSAM and HISAM, the segments are stored in a SAM file (in the case of HSAM) or in a VSAM data set (for HISAM) in top-to-bottom, left-to-right order. If a segment has any immediate dependents, the first segment sequentially following it in the file or data set will be the first existing dependent. (See the section “Defining Data Base Physical Characteristics” for a discussion of files and data sets.)

In the direct pointer method, used by the HD access methods, space is reserved in a segment *prefix* preceding each occurrence of a segment. This segment prefix contains a four-byte VSAM relative byte address (RBA) for each possible segment type defined as an immediate dependent of the segment. This pointer, called the *physical child first* pointer, contains the RBA of the first occurrence of the physical child of the corresponding segment type.

In addition, in the prefix of each dependent segment that can have more than one occurrence for a given parent segment occurrence, a four-byte *physical twin forward* RBA pointer is reserved.

Example:

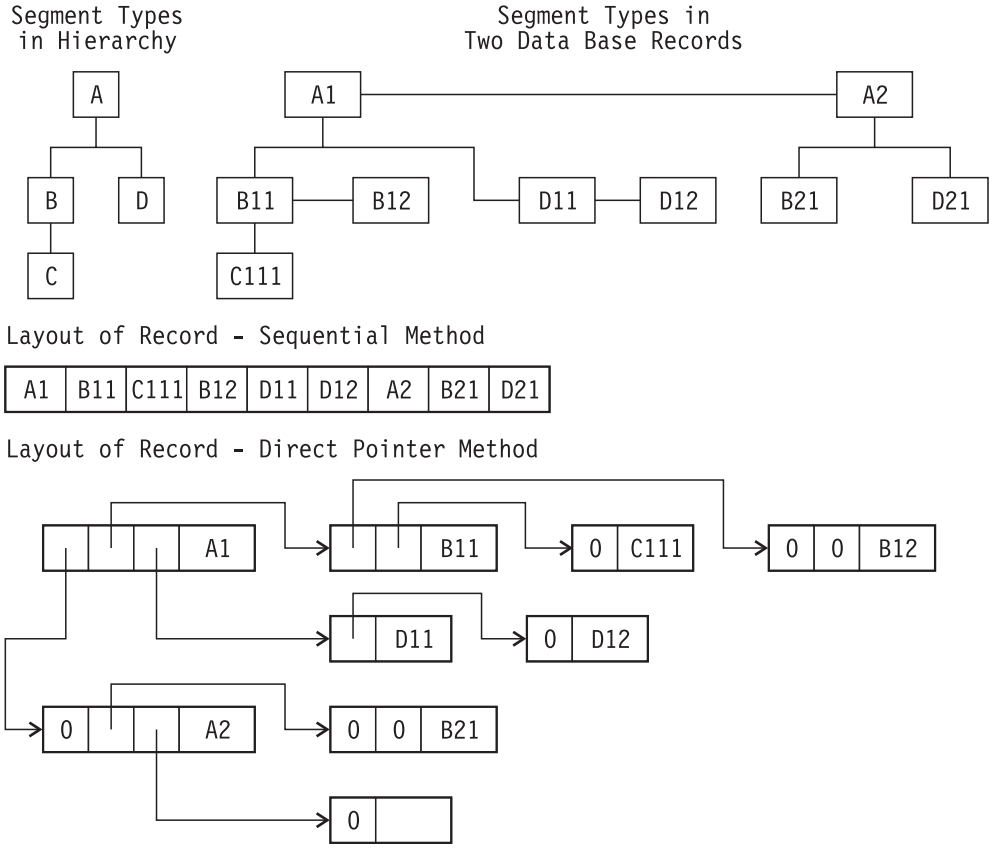
A data base having segment types A, B, C, and D is shown below. Segments B and D are immediate dependents of segment type A. Segment type C is a dependent of A and an immediate dependent of B. In the actual data base, there are two occurrences of segment type A: A1 and A2. (Note that each occurrence of the root segment A is equivalent to the occurrence of a data base record.)

Occurrence A1 has two occurrences of segment B (B11 and B12) and two occurrences of segment D (D11 and D12). One occurrence of segment C exists as a dependent of B11.

A2 has one occurrence of segment B (B21), and one of segment D (D21).

The upper part of the figure shows the relationship between the segment types in the hierarchy, and the occurrences of the segment types in the two data base records.

The center part of the figure shows the layout of a record using the sequential method. The lower part of the figure shows the layout using the direct pointer method.



Selecting a Technique for Connecting Parent and Child Segments

Assuming that no overriding requirements force a decision to use HD and thus the direct pointer method, selection of a connection technique may be based on the following considerations:

1. Alternate Views: In DL/I, unqualified sequential retrieval of dependent segments proceeds in a top-to-bottom, left-to-right order based on the application view of

the data. For the sequential method (HSAM or HISAM), the top-to-bottom and left-to-right order of segment types in the application view must be the same as the order in the system view. If you need a sequence other than that of the system view, you have to use the direct pointer method (HD).

2. Direct Versus Sequential Processing of Dependents: Location of a specific dependent segment in a data base structured using the sequential method involves a sequential search of all the segments in the data base record that precede the one you want. In the example shown, for instance, retrieval of segment D12 would require searching through A1, B11, C111, B12, and D11 before locating D12.

   Use of the direct pointer method requires searching of only the twin chains above the required segment. In this case, only segments A1 and D11 are examined before D12 is located. Direct pointers do use additional space in the data base, however, and this may adversely affect performance due to longer I/O times when attempting to locate data.

You should use the direct pointer method when there will be enough direct accesses to dependent segments that are not located on the left hand side of the physical data structure (segment types A, B, and C in the example) to offset the additional overhead involved due to the extra space for pointers.

Additional Considerations for the Direct Pointer Method

1. **Additional Pointer Types:**

   In addition to the *physical child first* and *physical twin forward* pointers, you can specify two other pointers: the *physical child last* and *physical twin backward* pointers.

   The physical child last pointer is optional. You can use it for performance enhancement, if no sequence field has been defined for the related dependent, when new dependent segments are to be placed at the end of the physical twin chain (see “Access Techniques” for a definition of sequence fields).

   The physical twin backward pointer is also optional. You can use it to improve performance related to deletes of segments located in relatively long physical twin chains.

2. **Suppression of Physical Twin Pointers:**

   In the case of a parent/child relationship where only one occurrence of the child is to be allowed, you can suppress allocation of the physical twin forward pointer. This should be done for two reasons: first, to save space in the data set and, second, to ensure that no additional child occurrences might be accidentally added.

3. **Inserting Segments:**

   When only physical twin forward pointers are specified, segments are inserted in chronological order. This eliminates the need to search through the entire chain again to find the previous segment to perform an insertion. However, when physical twin backward pointers are used, segments are inserted in sequential order. This is because the backward pointers identify the previous segment; thus, there is no execution time overhead involved in keeping the segments in this preferred order. There is, of course, the overhead of maintaining the additional pointer on all the segments of that type in the data base.

Left-to-Right Ordering of Segment Types:

With Sequential Connections

The primary consideration for this technique is the order in which segments are to be retrieved during sequential retrieval. Because the application views for this mechanism must show the segments in the same order as they appear in the physical view, the segment types must be defined in the order in which they are expected to appear when using sequential retrieval.

A secondary consideration, for HISAM only, is to locate those segments most often retrieved directly, rather than sequentially, on the left side of the hierarchy, if this can be done within the constraints set by sequential processing requirements.

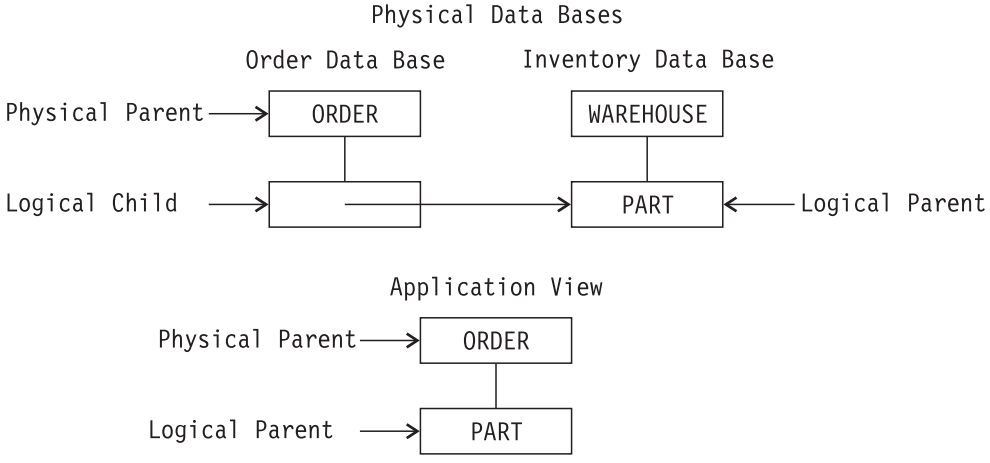
With Direct Pointer Connections

Retrieval performance is the primary consideration in this case. During initial load operations, segments on the right side of the hierarchy are most apt to be placed in a VSAM control interval that is different from that of their parent. For sequential processing, this will make no difference, as all segments will be retrieved in any case, but those segment types most apt to be retrieved directly should be placed on the left side of the hierarchy.

Extended Structures

Extended structures are used whenever application views require that a segment already defined in a hierarchy also appear in a different place in the same or a different hierarchy. For instance, an inventory data base already contains a “part description” segment. A customer order data base requires a part description for each part ordered. Rather than include the information as a segment in each place the part is referenced, the extended structure facility may be used. This makes it possible to reference the “part description” segment in the inventory data base and also as a dependent segment in the customer order data base.

This is done by using a connector segment called a *logical child*. The logical child segment is defined at the location in the hierarchy where the dependent segment is to be located, with the segment that is to be the parent in the relationship defined as its parent. The dependent segment is then identified as the *logical parent* of the logical child. The normal parent of the logical child in the hierarchy in which it is defined, is known as the *physical parent*.

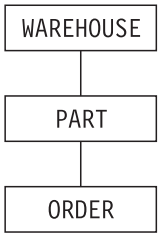


Features of Extended Structures:

Bidirectional Relationships

Logical children define two way connections, known as bidirectional relationships, between the physical parent and the logical parent. Application views may be developed where the logical parent is seen as a dependent of the physical parent; and other views with the physical parent a dependent of the logical parent.

Alternate Application View



Concatenated Segments

When a logical parent segment is accessed as a dependent of the physical parent in a logical relationship during an insert, replace, or delete operation, the segment as viewed by the application is made up of three parts:

- The concatenated key of the logical parent
- The logical child data (known as intersection data), if any
- The logical parent data.

This collection is known as a *concatenated segment*.

(The concatenated key of a segment is defined as: the sequence field of the segment preceded by the sequence field of its physical parent preceded by the sequence field of its physical parent's parent, etc.; up to and including the root segment.)

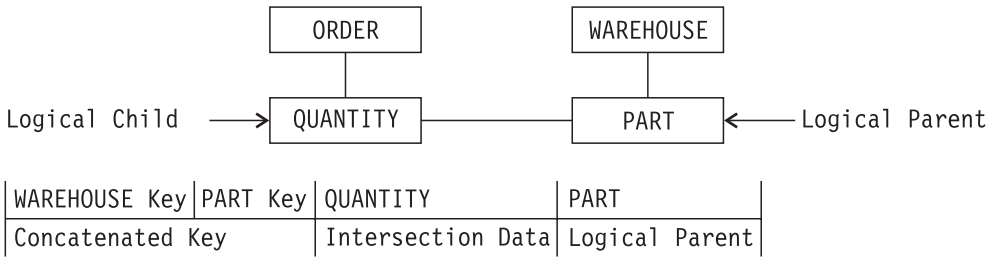
WAREHOUSE Key	PART Key	Logical Child Data	PART Data
Concatenated Key		Logical Child	Logical Parent Data

A physical parent accessed as a dependent of a logical parent has the same format. It contains the concatenated key of the physical parent, followed by the logical child data, if any, and the physical parent data.

Intersection Data

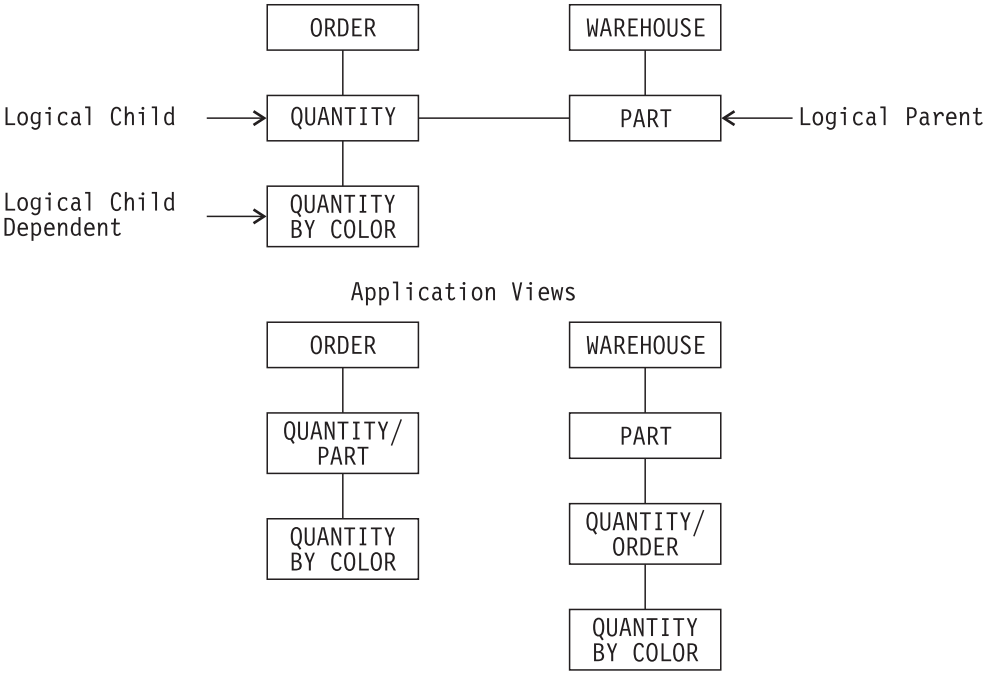
When a logical child is defined in the system view, it may be defined as containing data. This data is known as *intersection data*. It will be attached as a suffix to the logical parent when it is retrieved as a dependent of the physical parent, and as a suffix to the physical parent when it is retrieved as a dependent of the logical parent.

Intersection data may be used to contain data pertaining to the connection of two items (the number of PARTS required to fill a given ORDER), or it may be required to allow implicit sequencing of either the logical parents as dependents of the physical parent or the physical parents as dependents of the logical parent.



Dependents in Logical Relationships

Dependent segment types may be defined for a logical child segment the same as for normal segments. When the logical parent is considered as the dependent of the physical parent, the dependents of the logical child are viewed as dependents of the logical parent. When the physical parent is the dependent of the logical parent, they are seen as the dependents of the physical parent.



Logical Data Bases

In order to make use of logical relationships, you have to define a special type of system view called a *logical data base*.

A logical data base combines existing HD data bases (referred to as *physical data bases*), or parts of them, into a new hierarchical data structure without altering the physical position of the data involved. To application programs, this data structure appears as if it were a physical data base. It contains one root segment type and one or more dependent segment types.

Each segment of a logical data base corresponds to a segment (or segments for the dependent of a logical relationship) in the participating physical data base(s). The hierarchical parent/child relationship between two segments of the logical data base corresponds to certain relationships in the participating physical data base(s).

Allowable Structures in a Logical Data Base

The hierarchical structure of a logical data base is, in a way, predetermined by the structure of the participating physical data bases. Because of this, certain rules have to be followed when defining a logical data base.

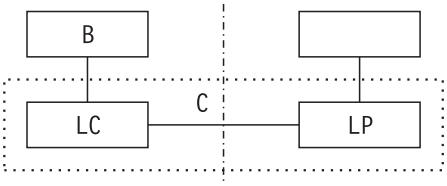
Logical Data Base

1. The root of a logical data base is always the root of one of the participating physical data bases.
2. Segment C may be a child of segment B in a logical data base only if:
   a. segment C is a child of segment B in the physical data base,

Physical Data Base

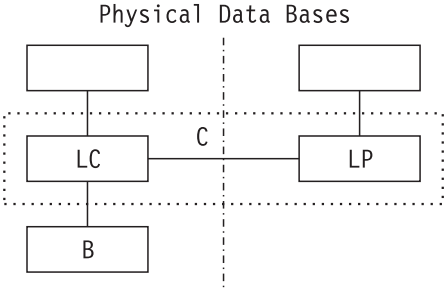
   b. segment C represents the logical parent as a dependent of the physical parent, and segment B is the physical parent,

Physical Data Bases

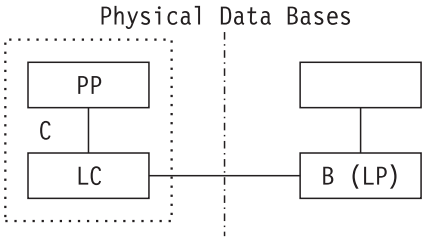


   c. segment C represents the logical parent as a dependent of the physical parent, and segment B is a child of the logical child,

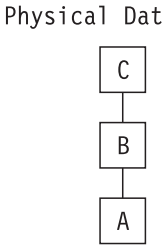
Physical Data Bases



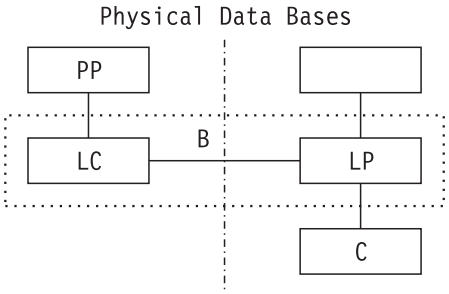
d. segment C represents the physical parent as a dependent of the logical parent, and segment B is the logical parent,



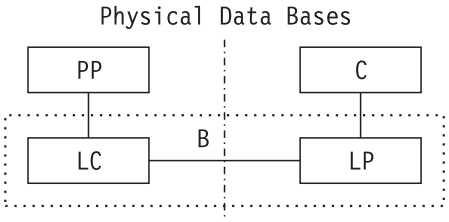
e. segment C is the parent of segment B and segment B is the parent of segment A in the physical data base,



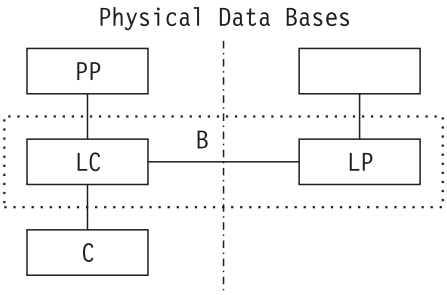
f. segment B represents the logical parent as a dependent of the physical parent in a logical relationship, and segment C is the child of the logical parent,



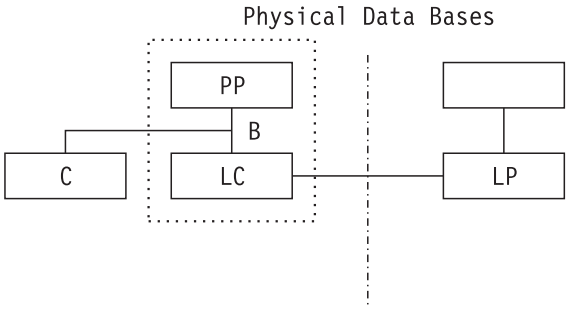
g. segment B represents the logical parent as a dependent of the physical parent in a logical relationship, and segment C is the parent of the logical parent,



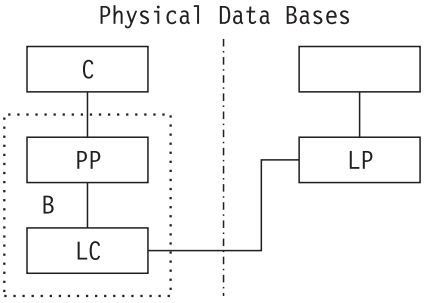
h. segment B represents the logical parent as a dependent of the physical parent in a logical relationship, and segment C is the child of the logical child,



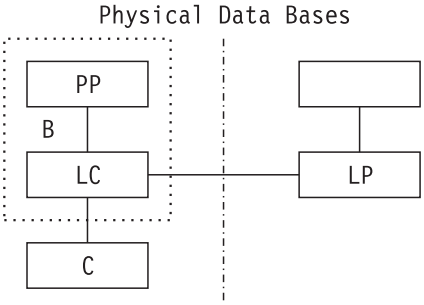
i. segment B represents the physical parent as a dependent of the logical parent in a logical relationship, and segment C is the child of the physical parent,



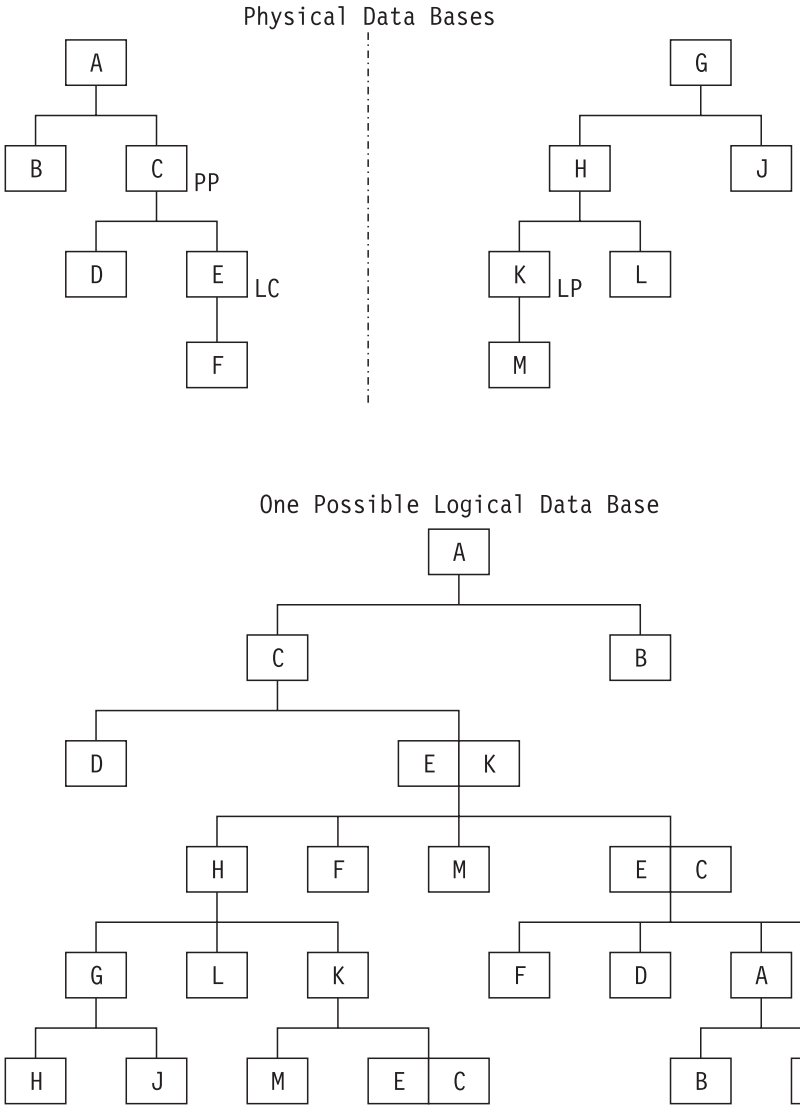
j. segment B represents the physical parent as a dependent of the logical parent in a logical relationship, and segment C is the parent of the physical parent,



k. segment B represents the physical parent as a dependent of the logical parent in a logical relationship, and segment C is the child of the logical child.



The example below shows two physical data bases connected by a logical relationship and a valid logical data base derived from them.



Application Views Based on the Physical Data Base

Although most applications making use of logical relationships will have views based on the logical data base, some, such as initial loading, require views based on the physical model.

Application views based on the physical model see logical relationships somewhat differently. The view of the physical parent as a dependent of the logical parent is not available in this case, and references to the logical parent as a dependent of the physical parent see the segment as the logical parent concatenated key followed by the intersection data only. The logical parent data isn't available.

Considerations in Defining Extended Structures

In defining extended structures, many different factors must be taken into consideration:

1. A one way (unidirectional) connection may be defined to enhance performance if one segment type is never considered the dependent of the other.
2. In a bidirectional relationship, the decision must be made as to which physical hierarchy will contain the logical child.
3. There are various optional pointers that may be selected to enhance performance.
4. The restrictions on – and results of – insert, replace, and delete operations on the connection and the segments involved must be defined.
5. Certain restrictions apply to the location of the logical child in the physical data base structure.
6. It may also be required to decide which of two connections is to be made using the normal hierarchical parent/child connection and which is to be made using logical relationships.

Bidirectional Versus Unidirectional Relationships

Logical relationships may be either bidirectional or unidirectional. Since a unidirectional (or one way) connection is just a special case of a two way connection, unidirectional relationships are useful only for performance reasons.

A unidirectional relationship is implemented using a normal parent/child connection, including optional pointers, between the physical parent segment and the logical child. The logical child segment contains a pointer to the logical parent. Each occurrence of the logical parent contains a *logical child counter* in its prefix that is a count of the total number of logical children (of all types) that are connected to it. An insert or delete affecting the logical child requires a reference to the logical parent to update this counter.

A bidirectional relationship adds a set of pointers to the logical relationship that are equivalent to those for a parent/child connection. A *logical child first* (and optional *logical child last*) pointer is added to the logical parent prefix, and a *logical twin forward* (and optional *logical twin backward*) pointer is added to the logical child prefix.

Unidirectional relationships should be used only if there are no requirements for a view of the connection where the physical parent is seen as a dependent of the logical parent, and the performance implications involved in the maintenance of the additional pointer(s) in the logical child segment are significant. The extra four bytes per occurrence of the logical child, and the overhead of maintaining the pointer during inserts, deletes, and data base reorganizations should be considered.

Placement of Logical Child in Bidirectional Relationships

The two directions of connection in a bidirectional relationship are identified as the *physical path* and the *logical path*. The physical path is the path through which the logical parent is considered a dependent of the physical parent. The logical path has the logical child's physical parent as the dependent of the logical parent.

The decision as to which segment is to be the physical parent and which is to be the logical parent involves both performance considerations and functional restrictions.

Performance Considerations

An attempt is made in the HD organization to physically store segments that are on the same physical twin chain as close as possible to each other. This is not true for segments on the same logical twin chain. The combination of how often a segment will be accessed by each path, and how long the average chain will be on each path, should be the determining factor as to which path will be the physical path.

Functional Restrictions

There are certain restrictions that apply to the definition of sequence fields (see the section “Sequential Processing in Data Base Records”) for the sequencing of logical parents on the physical path that do not apply to the sequencing of physical parents on the logical path. The field to be used in sequencing the logical parents must be defined as data in the logical child segment. The field to be used in sequencing physical parents may be made up of multiple fields taken from the logical child segment data and/or the sequence fields of the physical parent or its parents up to and including the root segment.

If sequencing of the latter type is required, the path must defined as the logical path.

Optional Pointers for Logical Relationships

In addition to the logical child first and logical twin forward pointers for bidirectional relationships, two other pointers, the *logical child last* and *logical twin backward* pointers can be defined.

The logical child last pointer is optional. It may be requested for performance enhancement, if no sequence field has been defined for the logical path, and new dependent segments are to be placed at the end of the logical twin chain (see the section “Access Techniques”).

The logical twin backward pointer is also optional. It may be specified to improve performance related to deletes of segments located in relatively long logical twin chains.

Suppression of Logical Twin Pointers

In the case of a logical relationship where only one occurrence of the child is allowed on the logical path, it is possible to suppress the allocation of the logical twin forward pointer. This should be done for two reasons: first, to save space in the data set, and second, to ensure that no additional occurrences might be accidentally added.

Insert Rules

The connection between an occurrence of a physical parent and an occurrence of a logical parent is made by “inserting” one as the dependent of the other. For bidirectional relationships, an “insert” on either path makes the connection in both directions.

The insert creates an occurrence of the logical child. It may create a logical or physical parent occurrence or it may just form a connection to one already in existence.

Insert rules can be specified to control how this occurs, and how the insert will affect the segments involved.

Insert rules can be specified for both the physical and logical parents. Insert rules for the logical parent control inserts of the logical parent as a dependent of the physical parent. Insert rules for the physical parent (bidirectional relationships only) control inserts of the physical parent as a dependent of the logical parent.

Logical Parent Insert Rules

One of three rules may be selected: P, L, or V.

If the P rule is specified, inserts of the logical parent as a dependent of the physical parent are allowed only if the logical parent already exists. Only the logical child part of the data passed will be inserted.

If the L rule is specified, all inserts are valid. If the logical parent already exists, it is not changed. If it does not, it is inserted using the data passed with the insert request.

If the V rule is specified, all inserts are valid. If the logical parent already exists, its data is replaced with the data passed as a result of the insert. If it does not exist, it is inserted using the data passed.

Note: The logical parent may only be inserted if *its* physical parent already exists.

Physical Parent Insert Rules (Bidirectional Relationships Only)

One of three rules may be selected: P, L, or V.

If the P rule is specified, inserts of the physical parent as a dependent of the logical parent are only allowed if the physical parent already exists. Only the logical child part of the data passed will be inserted.

If the L rule is specified, all inserts are valid. If the physical parent already exists, it is not changed. If it does not, it is inserted using the data passed with the insert request.

If the V rule is specified, all inserts are valid. If the physical parent already exists, its data is replaced with the data passed as a result of the insert. If it does not exist, it is inserted using the data passed.

Note: The physical parent may only be inserted if *its* physical parent already exists.

Replace Rules

The replace rules are used to control how a replace operation can occur, and how it will affect the segments involved.

Replace rules may be specified for both the physical and logical parents. Replace rules for the logical parent control replacement of the logical parent as a dependent of the physical parent. Replace rules for the physical parent (bidirectional relationships only) control replacement of the physical parent as a dependent of the logical parent.

Logical Parent Replace Rules

One of three rules may be selected: P, L, or V.

If the P rule is specified, replaces of the logical parent as a dependent of the physical parent are only allowed if only the logical child data is changing. If the data passed for the logical parent portion is different from the data that is currently in the data base, the request is rejected.

If the L rule is specified, only the logical child portion of the data will be replaced. The logical parent part is ignored.

If the V rule is specified, both the logical child and the logical parent data will be replaced.

Physical Parent Replace Rules (Bidirectional Relationships Only)

One of three rules may be selected: P, L, or V.

If the P rule is specified, replacement of the physical parent as a dependent of the physical parent is only allowed if only the logical child data is changing. If the data passed for the physical parent portion is different from the data that is currently in the data base, the request is rejected.

If the L rule is specified, only the logical child portion of the data will be replaced. The physical parent part is ignored.

If the V rule is specified, both the logical child and the physical parent data will be replaced.

Delete Rules

The delete rules are used to control how a delete operation can occur, and how it will affect the segments involved.

Deletes in a data base with no logical relationships make the segment being deleted and all of its dependents no longer available.

With the default delete rule (L), deletes involving logical relationships work the same way, but only for the path on which they are entered. A delete of the logical parent as a dependent of the physical parent, for instance, would still leave the physical parent accessible as a dependent of the logical parent.

Exceptions to the above processing may be invoked using the delete rule specifications of P or V for the logical child and/or logical parent.

Logical Child Delete Rules (Bidirectional Relationships Only)

The P delete rule, when specified for the logical child segment, adds a restriction to deletes involving bidirectional relationships: the logical parent may not be deleted as a dependent of the physical parent until the physical parent has been deleted as a dependent of the logical parent.

The V delete rule, when specified for the logical child segment, indicates that a delete in either path will cause a delete in the other.

Logical Parent Delete Rules

The logical parent is accessible either as a dependent of the physical parent or as an independent segment. The logical parent delete rules control the interrelationships of deletes involving these two paths.

The P delete rule, when specified for a logical parent segment, adds the restriction that the logical parent may not be deleted as an independent segment until it has been deleted in all logical relationships.

The V delete rule, when specified for a logical parent segment, causes it to be deleted as an independent segment automatically when it has been deleted in all logical relationships. (Note that this does not ensure that the logical parent will not exist as an independent segment without any logical relationship connections, as it may be inserted without any.)

Physical Structure Rules For Logical Children

1. A logical child cannot be a root segment.
2. A logical child cannot be the child of another logical child (but it may be a dependent).
3. A logical child cannot be a logical parent.
4. A logical child cannot be the source or target of a secondary index (see the section on “Access Techniques” for a discussion of secondary indexing).

Physical Versus Logical Relationships

It is possible, in designing a data base, to define a segment that requires more than one parent. In this case, a decision must be made concerning which parent will be its actual parent and which will be connected by logical relationships.

The following should be considered:

1. Accesses and Chain Lengths:

   As an attempt is made to store the children of physical relationships close together, retrievals of segments on a path from their parent segment are generally more efficient.
2. Insert, Replace, and Delete Controls:

   Certain rules and restrictions may be specified concerning logical relationships that do not apply to physical relationships. (See above discussion).
3. Application views of a segment through a logical relationship will include the concatenated key of the segment. This is not true for the same segment viewed through the physical relationship.

Access Techniques

The process of referencing a segment in a data base is divided into two steps:

1. The proper data base record must be located. (This is not necessary for every request, as some may rely on positioning from a prior request.)
2. Once the proper record has been located, the segment occurrence must be located within the record.

Access Techniques for Data Base Records

Processing of data base records may be either sequential, moving from record to record in some predetermined order; or direct, selecting each record through some specified “key.”

Sequential Processing of Data Base Records

Sequential processing of data base records is supported for all but the HD randomized access method. Records can be processed in a sequential manner when using HD randomized, but there is no way to define the sequence. If occasional sequential processing of an HD randomized data base is required, the use of a secondary index is suggested.

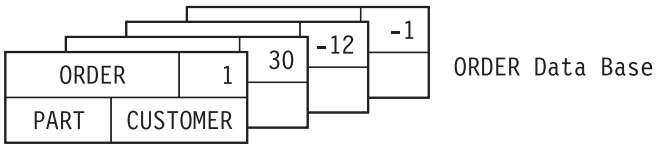
There are two aspects of sequential processing to be considered: how the sequence is specified, and how it is maintained.

Specification of Sequence: The sequence of data base records may be controlled by the application creating them, by the order in which they are created, or it may be based on the value of some field within the record.

Control by Field Value

The order in which data base records will be processed sequentially may be defined by the value of a field within the record. Such a field in the root segment is identified as the *sequence field*. Data base records will then be ordered based on the value of this field in each record.

The sequence is be based on the results of a logical compare (a binary -1 will be treated as greater than a +1), with the lower logical values appearing first.



If duplicate values exist for a sequence field, the order of the duplicates may be controlled by the application building the record, or may be based on the order of creation.

Sequential ordering by field value is required for HISAM, SHISAM, and the HD access methods. Duplicate values in sequence fields are allowed only for HSAM, SHSAM, and HD randomized access methods.

Control by Order of Creation

The order in which data base records are processed sequentially may be defined as depending on the order of creation in either chronological or reverse chronological order. In chronological order, the oldest record will appear first, and in reverse chronological order, the newest will appear first.

If this rule is specified in conjunction with a sequence field, it controls the order of appearance of records with duplicate sequence field values.

Sequencing by order of creation alone is allowed only for the SHSAM and HSAM access methods. All others require a sequence field.

Control by Application Program records, controlling by application program

The order in which data base records will be processed sequentially may be defined as being controlled by the application program creating the record. In this case, the application must have positioned itself to a point just after the point at which the record is to appear.

If this rule is specified in conjunction with a sequence field, it controls the order of appearance of records with duplicate sequence field values.

Sequencing under sole control of the application is allowed only for the SHSAM and HSAM access methods. All others require a sequence field.

Sequence Maintenance: The sequencing of data base records is accomplished in one of three ways: sequential, direct pointers, or indexes.

(The method to be used will more than likely be determined by which access method you choose, and this decision will probably be based on other factors. The following discussion is provided only to explain the different methods used by DL/I.)

Sequential Sequencing

In sequential sequencing, data base records appear physically in the data set or file in sequential order.

Data bases implemented using sequential sequencing of records cannot be updated and, as a consequence, are of use mainly for the collection of historical data.

Sequential sequencing of records is used in the HSAM and SHSAM access methods. Sequentially sequenced data bases are not supported for the online or MPS batch environments.

Direct Pointer Sequencing

In direct pointer sequencing, a pointer to the root segment of the next record in sequence is located in the prefix of each root.

Direct pointer sequencing is used in the HD indexed access method, and for sequencing of records with duplicate keys in the HD randomized access method.

Indexed Sequencing

In sequential processing using indexes, the next record is found by locating the index with the next greater key.

Sequential processing by index is used in the HISAM and SHISAM access methods.

Direct Processing of Data Base Records: There are three methods of direct access of data base records: by sequential search, by index, and by randomizing routine.

Sequential Search Direct Access: In direct access by sequential search, the data base is searched (either the entire data base, or forward from the current position), for the root segment with the specified key.

Direct access by sequential search is used by the HSAM and SHSAM access methods.

Indexed Direct Access: In direct access by index, an index is used to locate the record with the specified key.

Direct access by index is used by the HISAM, SHISAM, and HD indexed access methods.

Direct Access by Randomizing Routine: In direct access by randomizing routine, the key is passed to a randomizing routine, either written by you or one supplied by DL/I. The routine converts the key into an address in the data base that DL/I then uses to locate the record. All the records whose keys convert to the same address are chained together with direct pointers. DL/I searches the chain for the record with the right key.

Secondary Indexes: For the HD access methods only, DL/I provides the ability to define alternate processing sequences and alternate record entry points through the use of secondary indexes. This is in addition to the primary method of accessing data base records described above.

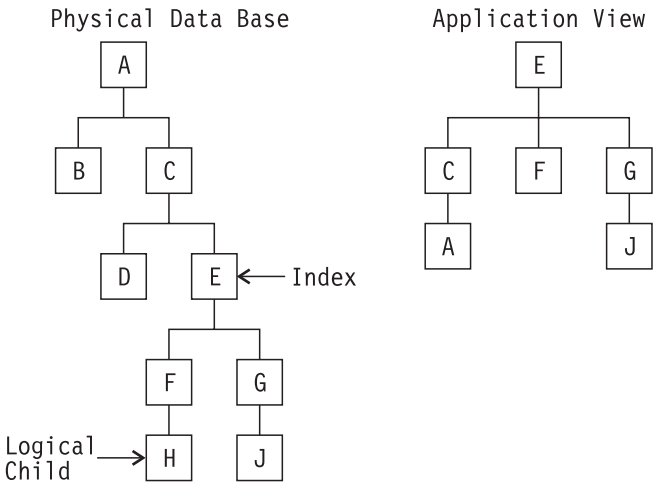
Secondary indexes can be used:

- To identify an alternate sequence in which data base records are to be processed, based on a field or fields in the root segment or in any other segment in the record. (Fields in a logical child segment cannot be used.)
- To specify sparse sequencing, where not all records in the data base will be represented, based on the existence of a dependent segment. Sparse sequencing may also be based on values of the key field, the results of an inspection by a user-written routine, or both.
- To add sequential processing capability to an HD randomized data base.
- To define a segment other than the root segment as an alternate entry point to the data base record.

Alternate Record Entry Point Using Secondary Indexing: The point at which the entry to the data base record is made using a secondary index is known as the *target segment* of the index. It can be defined as any segment in the record, with the exception of a logical child segment or any dependent of a logical child segment.

When the index target segment is not the root segment in the physical data base, the following rules apply:

1. The index target segment becomes the root segment in the application view.
2. The existing parent segment types of the target segment become the leftmost dependent segments in reverse order.
3. All dependent segments of the target segment can be included in the application view without change in their order.
4. Only parent and dependent segment types of the index target segment can be included.
5. If a logical child occurs in the physical data base, it must not be referenced in the application view.



Although it is possible to have an application view for this data base with fewer segments than are shown, no additional ones are allowed.

Note that, unlike the case of logical relationships, where the alternate hierarchy was defined as a system view (logical data base), the alternate hierarchy for a secondary index is defined in the application view only.

Sequential Processing Using Secondary Indexing

Sequential processing using secondary indexing uses the value of a sequence field to define the sequence. The sequence is maintained in an index just as it is for primary sequential access by index.

Secondary Index Sequence Fields

The field used to determine the sequential processing sequence for access with a secondary index can consist of up to five separate fields from the target segment or any dependent of the target segment. (Logical child segments cannot be used.) The segment from which the sequence information is taken is called the *sequence segment*.

Unlike primary access using an index, there is no requirement that the sequence values be unique. However, if they are not unique, the order in which records having duplicate sequence values are referenced is unpredictable.

Sparse Sequencing Using Secondary Indexing: Secondary indexing may be used to define a sequence that does not include all of the data base records in the data base.

Sparse Sequencing Based on Segment Occurrence

If the sequence segment is defined as other than the physical data base root segment, there is no requirement that there must be an occurrence of the sequence segment in every record. No index entries will be created referencing those records for which no occurrence of the sequence segment type exists.

Note also that multiple references to the record will occur if more than one occurrence of the sequence segment type exists for the record.

Sparse Sequencing Based on Sequence Field Value

It is possible to specify a “suppression value” to suppress the creation of indexes. If all of the fields making up the sequence field are equal to the specified suppression value, no index will be created.

The value specified is one byte of information. If the field being checked is longer than one byte, the byte is expanded to make the check.

If the field is packed decimal, only the last byte of the field is compared to the suppression value. All of the remaining bytes are compared to a one byte value constructed by placing the first four bits of the suppression value into both the first and last four bits of the new value.

If the field is zoned decimal, again only the last byte of the field is compared to the suppression value. All of the remaining bytes are compared to the suppression value with its top four bits set to ones.

If the field is any other type, all the bytes of the field are compared to the suppression value, one at a time.

This process is carried out for each of the fields making up the sequence field.

Sparse Sequencing Based on Inspection by User Routine

A “suppression routine” can be specified for each secondary index. This routine will be entered to determine if an index entry should exist for a given sequence segment.

The routine must be supplied by you, and must reside in the core image library.

Note that if both a suppression value and a suppression routine are specified, a suppress indication from either one is sufficient to suppress creation of the index entry. (The suppression routine will always be entered.)

See Appendix G, “Randomizing Modules and DL/I User Exit Routine Interfaces” on page G-1 for further information about the secondary indexing exit routine and its interface.

Considerations for Variable Length Index Source Segments: If a variable length segment type is used as an index source segment and an attempt is made to insert or replace an occurrence of the segment type whose length does not include some fields or portions of some fields specified for use in the search, subsequence or duplicate data fields of an index pointer segment, one of the following actions occurs:

- If the missing index source segment data is used in the search field of an index pointer segment, generation of the index pointer segment for that source segment is suppressed.
- If the missing index source segment data is used in the sub- sequence or duplicate data fields of an index pointer segment, the index pointer segment field will contain a default value for the missing data. This may result in an attempt to create a duplicate index (return status code NI).

Normal default values are:

TYPE	LAST	LAST CHAR
C	x'40'	x'40'
H	x'00'	x'00'
P	x'00'	x'0C'
Z	x'F0'	x'C0'

Default values if IMS compatibility is in effect (IMSCOMP=YES was specified in the DBD) are:

TYPE	LAST	LAST CHAR
C	x'F0'	x'F0'
H	x'00'	x'00'
P	x'00'	x'0F'
Z	x'F0'	x'C0'

The representation used will be the type specified on the FIELD statement that defined that index source segment field.

Extended Secondary Index Support: Certain extra functions involving secondary indexing are available in conjunction with the Hierarchical Indexed Direct Access Method, *HIDAM*, and the Hierarchical Direct Access Method, *HDAM*. (HIDAM provides special definitional capabilities for HD indexed and HDAM provides special definitional capabilities for HD randomized.) You should be aware that the definition process for secondary indexes becomes much more complex when using these access methods.

These extra capabilities are:

1. The ability to access the index as a separate data base. This data base is a flat file data base. The root (and only) segment type contains the sequence field and, optionally, a special field called the subsequence field, a duplicate data field, and any additional fields defined and maintained by the application.
2. The ability to control the order of sequence for index entries with duplicate sequence fields.

Definition of Index Data Base index data base, definition of

In order to use the extended facilities, a separate index data base must be defined. This data base is defined automatically by DL/I for normal primary and secondary indexes (when HD randomized or HD indexed is used). If either the HIDAM or HDAM access methods are used, separate data bases must be defined for all indexes.

Referencing the Index Data Base

The secondary index may be referenced as a separate data base. The access method is called “INDEX.” The data base has the same organization as a HISAM data base, but it may contain only one segment type.

When referencing the index as a separate data base, inserts and deletes are not allowed, and replace operations are not allowed to change any system maintained data, including the key, subsequence fields, or duplicate data fields.

Duplicate Data Fields

If the index is to be processed as a separate data base, up to five fields can be specified to be included in the index data base segment and maintained by DL/I. These fields can be any field from the sequence segment or the concatenated key of the sequence segment.

Additional Data in Index Root Segments

When processing the index as a separate data base, you can include any additional data desired in the index root segment. In order to do this, the index root segment length must be defined as large enough to contain the sequence field, any subsequence fields, and any duplicate data fields as well as the length of the additional data.

It is your responsibility to maintain this data. A reorganization of the primary data base will result in the destruction of any user data in the index segment.

Subsequence Fields

Although there is no requirement that sequence field values for secondary indexes be unique, VSAM requires that all its keys be unique. DL/I ensures this is true for normal secondary indexes by suffixing the sequence field with the VSAM RBA of the sequence segment if the sequence field has been identified as possibly being non-unique.

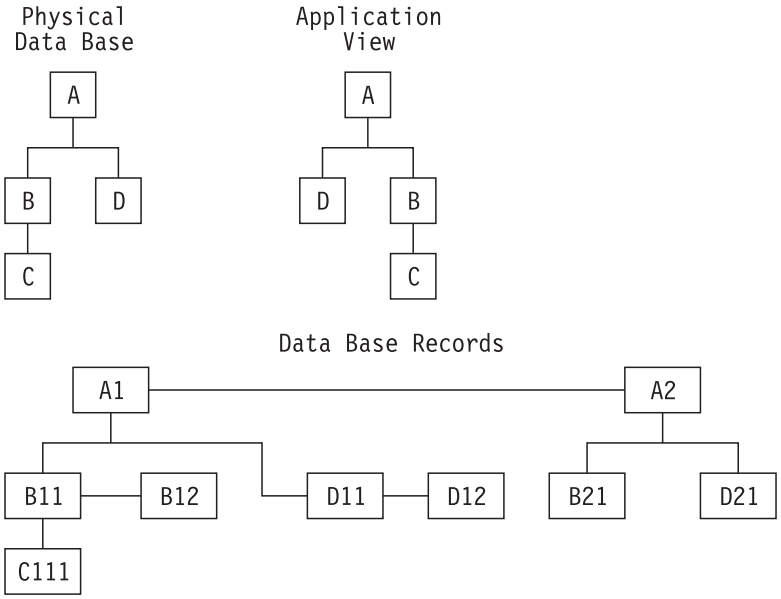
The extended definitional support allows the user to specify up to five fields to be appended to the sequence field to ensure uniqueness, and control the sequence of entries with duplicate sequence field values.

These fields may include fields from the sequence segment, from the sequence segment's concatenated key, or the four byte VSAM RBA of the sequence segment.

Access Techniques within a Data Base Record

Once the data base record has been entered, you may need to process segments other than the one at which entry was made. Processing of these segments may be either sequential, moving from occurrence to occurrence in some predetermined order; or direct, selecting each occurrence through some specified “key.”

Sequential Processing in Data Base Records: Within a data base record, sequential processing proceeds in a top-to-bottom, left-to-right manner, based on the order of segment types as defined in the application view. In the example below, sequential retrieval would access segments in the order A1, D11, D12, B11, C111, B12, A2, D21, and B21. (Note that an application view in which the dependents of a segment are defined in a different order than in the system view is only valid for HD or LOGICAL data bases.)



There are two aspects of sequential processing to be considered: how the sequence is specified, and how it is maintained.

Specification of Sequence: The sequence of segment occurrences among twins can be controlled by the application creating them, by the order in which they are created, or it may be based on the value of some field within the occurrence.

Control by Field Value

The order in which segment occurrences will be processed sequentially may be defined by the value of a field within the occurrence. A field within the segment is identified as the *sequence field*. Segment occurrences will then be ordered based on the value of this field in each occurrence.

The sequence will be based on the results of a logical compare (a binary -1 will be treated as greater than a +1), with the lower logical values appearing first.

If duplicate values exist for a sequence field, the order for duplicates can be specified as being controlled by the application building the record, or may be based on the order of creation.

Special Considerations for Logical Relations

The order of processing of logical parents seen as dependents of physical parents and of physical parents seen as dependents of logical parents can also be controlled by the value of a field as follows:

1. Logical parent as dependent of physical parent:

   The “sequence field” must occur in the data part of the logical child (not in the concatenated key or in the logical parent).

2. Physical parent as dependent of logical parent:

   May consist of up to five fields taken from the data part of the logical child or the concatenated key of the physical parent, or both (but not from the physical parent itself).

Control by Order of Creation

The order in which segment occurrences will be processed sequentially may be defined as depending on the order of creation in either chronological or reverse chronological order. In chronological order, the oldest occurrence will appear first, and in reverse chronological order, the newest will appear first.

If this rule is specified in conjunction with a sequence field, it controls the order of appearance of physical twins with duplicate sequence fields.

Control by Application Program

The order in which segment occurrences will be processed sequentially may be defined as being controlled by the application program creating the occurrence. In this case, the application must have positioned itself to a point on the physical twin chain just after the point at which the occurrence is to appear.

If this rule is specified in conjunction with a sequence field, it controls the order of appearance of physical twins with duplicate sequence field values.

Special Considerations for Logical Relations

Note that, as the application cannot have established a position for a logical relationship in the direction opposite to that for which the insert is taking place, care should be taken to ensure that this rule is specified only if all inserts or loads are to occur from the same side of the relationship and that the rule is specified for that direction only.

Maintenance of Sequence segment occurrences, sequence maintenance*:* The sequencing of segment occurrences is accomplished in one of two ways: sequential or direct pointers.

Sequential Sequencing

In sequential sequencing, segment occurrences appear physically in the data set or file in sequential order.

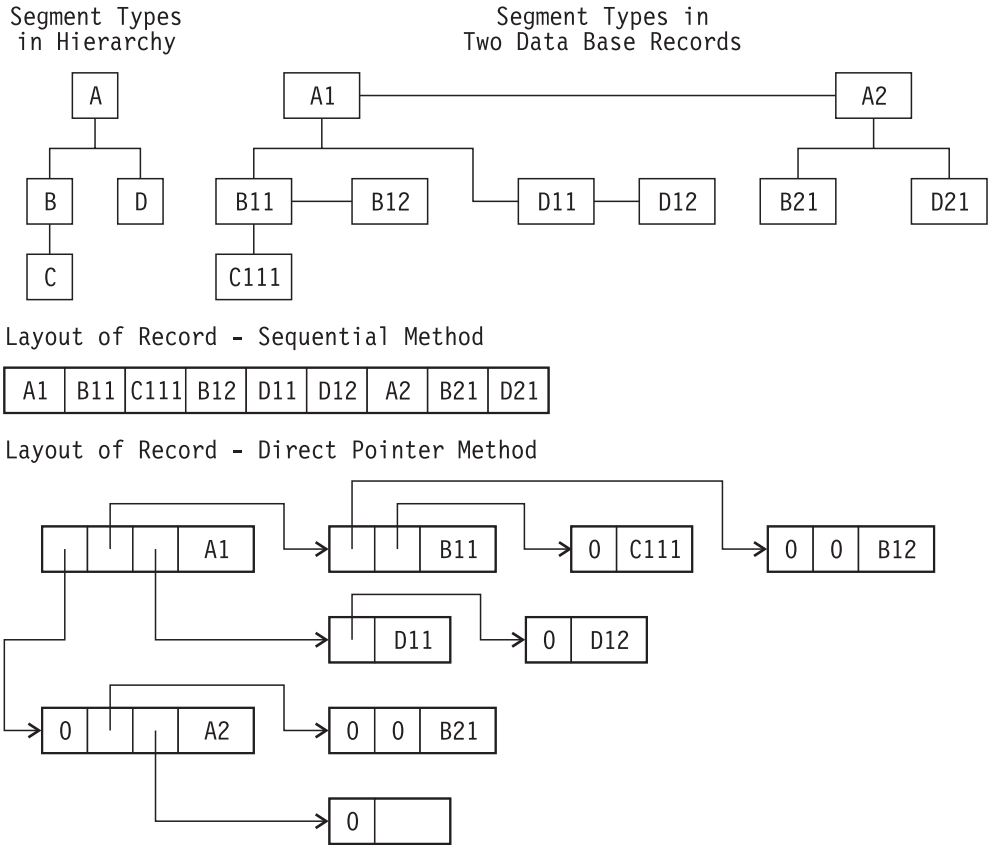
Sequential sequencing of segment occurrences is used in the HSAM and HISAM access methods.

Direct Pointers

In direct pointer sequencing, each segment occurrence contains, in its prefix, a pointer to the next occurrence of the same segment type under its parent, and a separate pointer to the first occurrence of each child segment type for which it is a parent.

Direct pointer sequencing is used in the HD access methods.

The example below shows two data base records and their layouts using the sequential and direct pointer methods.



Direct Processing in Data Base Records: For direct processing in a data base record, the application programmer identifies a particular segment by passing a "key" value for it and for all other segments between it and the record entry segment in the hierarchy. While these "key" fields are usually the segment sequence fields, they may be any field in the segment.

There are two methods of direct access in data base records: sequential search and a combination of sequential search and direct pointers.

Sequential Search Direct Access: In direct access by sequential search, the entire data base record is searched for the segment occurrences with the specified keys. In the example above, for instance, retrieval of segment D12 requires inspection of A1, B11, C111, B12, and D11.

Direct access by sequential search is used by the HSAM and HISAM access methods.

Direct Access by Direct Pointer/Sequential Search: In the combination method, movement from a parent to a child segment type is through a pointer located in the parent segment occurrence to the first occurrence of the particular child type. This pointer is called the *physical child first* pointer. The use of this pointer avoids the necessity for examining any other dependent segment occurrences whose types appear to the left of this one in the hierarchy.

In addition, each occurrence of a particular child segment type for a single parent occurrence contains a pointer in its prefix to the next occurrence of the same type for the same parent. This is called the *physical twin forward* pointer.

Once the first occurrence of the proper type has been located, the physical twin chain is searched sequentially to locate the specific segment.

Using this method in the example above, only segments A1 and D11 are examined before D12 is located.

Direct access by direct pointer/sequential search is used by the HD access methods.

Considerations for the Selection of Access Techniques

The access techniques for data base records and for segments within the records have already been specified as part of the data base design process. This should include the definition of sequencing rules and any required secondary indexes. As a consequence, no decisions need be made at this time.

Data Formats

There are several DL/I capabilities that are concerned with the way data is formatted. This section describes these capabilities so that you can make decisions about their use. Since some of them apply to the system view, and some apply to the application view, they will be discussed under those headings.

In System Views

Variable Length Segments: Variable length segments are segment types that may not contain the same number of characters in every occurrence of the segment type. You can specify that segments are of variable length if the access method used for your data base is one of the HD methods. Variable length segments cannot be used with the sequential access methods.

Variable length segments let you vary the amount of storage space used to store different occurrences of the same segment type. Their primary use is in application programs that process variable length text or descriptive data. They can also be used, in some cases, to make more efficient use of secondary storage space.

You can specify that the space used for each occurrence of the segment type may vary between a minimum and a maximum number of bytes. The actual number of bytes of data in each occurrence of the segment is contained in a two-byte size field included in the data portion of that segment. The value in the size field tells DL/I the length of the data in that occurrence of the segment. Since the size field is included in the data portion of the segment, the data length must include the two bytes of the size field itself. If a sequence field is defined for the segment type, the minimum length that can be specified in the size field must include at least the size field and all data to the end of the sequence field.

The format of a variable length segment is shown in Figure 4-1. Fixed length and variable length segment formats are the same except for the size field in the data portion of the variable length segment. The storage requirements of data bases using variable length segments are increased by two bytes for each occurrence because of the size field.

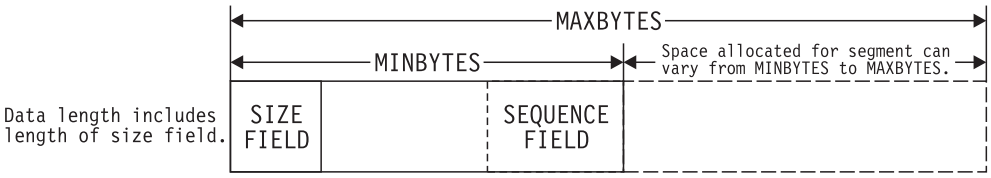


Figure 4-1. Variable Length Segment Format

When initially loading occurrences of a variable length segment type, the space used to store the data portion of an occurrence is the minimum length you specified, or the length specified in the size field, whichever is greater. When the minimum length specification is greater than the length specified in the size field of an occurrence, more space is allocated for the segment than is needed to store it. The additional space allocated is free space that can be used when existing data in the segment is replaced with data that is greater in length.

You can load segments initially with a given number of bytes of data, and then either increase or decrease the length of data in each by replacing the data. When the data in an existing segment is replaced with data that is greater in length and the space allocated for the existing segment is not sufficient for the new data, the prefix and data portions of the segment are separated to obtain sufficient space for the new data. When the prefix and data are separated, as shown in the lower part of Figure 4-2, a pointer is placed in the first four bytes after the prefix, which remains in its original position, to point to the new data portion of the segment. The remaining bytes at the original location, if any, are freed and made available for use by other data base updates. When the prefix and data are separated, and existing data is replaced with data that fits into the original space allocated for the data portion of the segment, the new data portion is placed in the original space (assuming it is still available), overlaying the data pointer. When the prefix and data portions of a segment are not separated, and data in an existing segment is replaced with data that is shorter in length, the new data, followed by free space, occupies the position of the original data. In addition, when the prefix and data portions of a segment are separated, a one-byte segment code and a one-byte delete flag are stored with the data portion of the segment. Performance can be affected when the prefix and data portions of a segment are separated and stored in different blocks in a file.

Note: A variable length segment must not be a logical child segment.

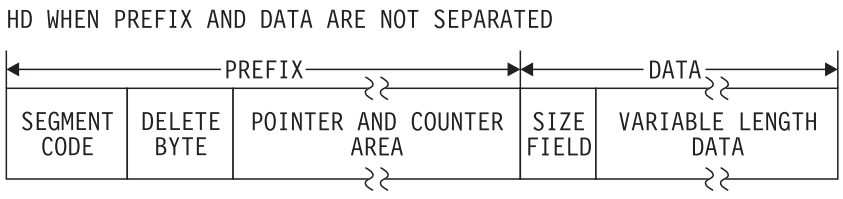


Figure 4-2 (Part 1 of 2). Separation of Prefix and Data in Variable Length Segments

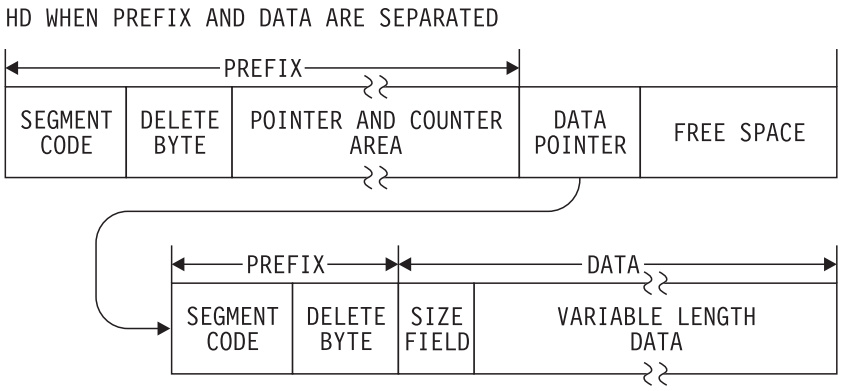


Figure 4-2 (Part 2 of 2). Separation of Prefix and Data in Variable Length Segments

Use variable length segments only when they are clearly required by the nature of the data or when the benefits of using them definitely outweigh the extra system resources needed to maintain them.

- If the data base includes variable length segments, try to design the segments so that the number of replacement requests made against them, which change the segment length, is minimized.
- Sometimes it is worthwhile to define a segment as a variable length segment and write segment compression/expansion routines to store it in a more compact form (see “Segment Edit/Compression” below). The benefits of doing this must be weighed against the additional CPU time required by DL/I to manage the variable length segment and the CPU time used by the compression and expansion routines. The amount of secondary storage space saved is usually not sufficient to justify this procedure. However, other benefits may result. For example, compressing one or more large segments may make it practical to fit the average data base record into one CI. The resulting reduction in data base I/O might easily justify the extra CPU overhead.

  Note: An example of a segment compression routine is documented in the *DL/I DOS/VS Guide for New Users*.

Segment Edit/Compression: The segment edit/compression exit facility of DL/I lets you supply a routine to edit a segment as it moves between the application program I/O area and the data base buffer pool. You can specify the use of a segment edit/compression exit if you use one of the HD access methods for your data base. The exit cannot be used with the sequential

access methods. Segments to be processed by an edit/compression routine must be specified as variable length segments.

You can use this facility to encode data for security purposes, to format data to be used by application programs, and to compress a segment by eliminating redundant characters. The primary purpose of segment compression is to decrease the amount of secondary storage space needed for the data base. Usually, compression is performed on data between the end of the sequence field and the end of the segment. The logic for the data encoding can be in the routine itself or it can be based on information from an external source, such as data provided in tables examined at execution time. An example might be a table of substitution characters maintained separately from the executable code. This table could reflect different combinations for different segment types, resulting in a

general purpose routine capable of processing different segment types in the same, or different, data bases.

A segment work area is constructed by DL/I at initialization time, and the edit routine is loaded when the data base is opened. When the application program requests a segment from that data base, its location in the buffer pool is obtained. If an edit routine has been specified, the address of the data portion of the segment and the starting address of the segment work area are supplied to the routine, and it is given control. On a retrieval operation, the edit routine is responsible for moving the data from the buffer pool to the segment work area. DL/I moves the data from the segment work area to the application program I/O area. For load, insertion, or replacement operations, data is moved from the application program I/O area to the segment work area by the edit routine, then to the buffer pool by DL/I. Figure 4-3 illustrates these operations.

You should consider the following items before deciding whether or not to use the edit/compression exit facility:

- Only variable length segments described in a physical data base can be handled by the exit routine
- The exit routine cannot be used with logical child segments
- Neither the relative position or the contents of the key field (if one exists) can be changed by the exit routine
- If an exit routine used in an online environment is designed to edit more than one segment type, in one or more physical data bases, the routine must be reentrant
- The size of the edit routine(s) should be considered when estimating main storage requirements for the DL/I system
- The exit routine cannot use such operating system macros as STXIT
- Edit/compression processing of each segment on its way to or from an application program involves added CPU time. In addition, the search time required to locate the requested segment may be increased, depending on the options selected.
- In an online environment, use of a table of substitution characters can put the entire DL/I partition in a wait state while the module containing the table is loaded into main storage
- In the case of segment compression, requests using qualified segment selection for data fields require that all segments in the path to the one being searched for must be expanded or decoded to allow examination of the appropriate field by the DL/I retrieval module.
- For a key field request, little more processing is required to locate the segment than that of non-compressed segments, since only the segment sequence field is used to determine if this segment occurrence satisfies the qualification.

See Appendix G, “Randomizing Modules and DL/I User Exit Routine Interfaces” on page G-1 for details of the segment edit/compression exit interface and how to use it.

In Application Views

Segment Sensitivity: Segment sensitivity is the DL/I facility for defining which of the segments in the system view are to make up the application view. The application program is able to access only those segments that it has been defined as being sensitive to.

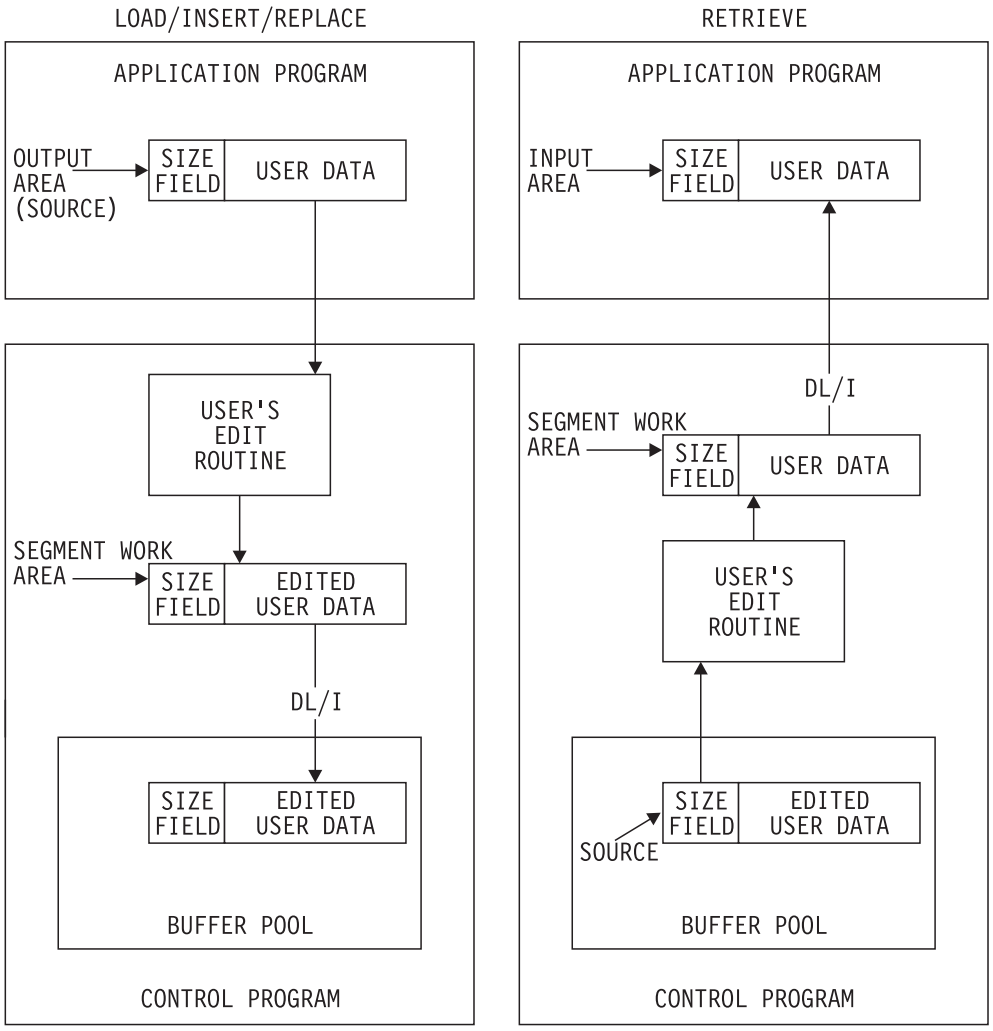


Figure 4-3. Segment Edit/Compression Operation

You can use segment sensitivity to provide data security. By not making an application program sensitive to a particular segment you prevent the program from gaining access to the data in that segment. Unless otherwise prohibited, an application program can retrieve, update, insert, or delete any segment it is sensitive to.

You can determine which segments an application program should be sensitive to from the documentation provided by the designer of the data base.

There are a couple of points to be aware of:

- The program must be sensitive to all segments in the path from a given segment upward through the hierarchy to the root segment
- When a segment is deleted by DL/I, all segments that are dependents of that segment are also deleted. This means that a program can inadvertently delete segments that it is not sensitive to if they are dependents of the sensitive segments being deleted.

Field Level Sensitivity: Field level sensitivity lets you specify which of the fields in the physical definition of a given segment are needed in the application program's view of that segment. You can also specify the locations of the chosen

fields in the application's view of the segment. These field locations can be the same or different from their locations in the physical definition. This makes it possible for different application programs to have entirely different views of the same segment. DL/I automatically maps the chosen fields from the physical segment into the application program's view during execution. Field level sensitivity also provides data security by preventing the application program from accessing fields it is not sensitive to.

Field Level Sensitivity provides other capabilities that are described in the following sections.

Virtual Fields: You can specify fields for the application program's view of a segment that do not exist in the physical segment. You can also specify an initial value to be assigned to the field and/or the name of a user-written routine that can be used to create the field. When you specify both an initial value and the name of a user-written routine, DL/I inserts the initial value in the application program's view of the field before the routine is called during a retrieve for the field. If a routine is specified, it is called for both retrieves and stores involving the field. See "User Field Exit Routine" below for further detail.

Automatic Data Format Conversion: If, during DBD generation, you define the type of data to be maintained in a given field, that data can be automatically converted to another type for a particular application program. You do this during PSB generation by specifying a different data type for the application program's view of the field. The data types are:

```
'X' hexadecimal
'H' halfword binary
'F' fullword binary
'P' packed decimal
'Z' zoned decimal
'C' character
'E' floating point (short)
'D' floating point (long)
'L' floating point (extended)
```

The automatic conversions supported are:

```
From     To
X        H, F, P, or Z
H        X, F, P, or Z
F        X, H, P, or Z
P        X, H, F, or Z
Z        X, H, F, or P
C        C (length conversion only)
```

Note:

- Conversion of data types E, D, and L is not supported.
- Data contained in a field specified as type 'C' is considered to be in an "as is" format, and no conversion is made when the field being moved into is specified as containing data of a different type. For example, if a field in a physical segment is specified as type 'C' and the field in the application's view is specified as type 'P', the data from the physical field is treated as though it is packed decimal. Only necessary length adjustments are made.

- Conversions that are not supported (such as: physical type 'Z' to user's type 'E') will pass through the ACB generation phase if, but only if, you specified a user-written exit routine for the field. Such a non-supported conversion causes a status code to be returned when encountered during an access of the field.

  If the status code is not corrected (reset) by a user exit routine, DL/I terminates the request. No more fields or segments are processed.

See the *DL/I DOS/VS Resource Definition and Utilities* for additional information about field type conversion.

User Field Exit Routine: You can specify the name of a user-written field exit routine to be given control whenever this field is retrieved or stored.

For retrieves, the routine is entered after the field has been moved (and converted, if necessary) from the physical segment to the application program's view. For virtual fields, it will occur after the field has been initialized with the null or initial value.

For stores, the routine is entered after the field has been moved (and converted, if necessary) to the physical view. If the field is virtual, the routine is entered immediately, because no conversion is done.

See Appendix G, “Randomizing Modules and DL/I User Exit Routine Interfaces” on page G-1 for details of the user field exit routine interface and how to use it.

Dynamic Segment Expansion: Fields can be added to a segment in the application program's view without unloading and reloading the data base, and without re-compiling other application programs that access the segment. To do this, use the following procedure:

1. During DBD generation, define the physical segment as variable length with the maximum and minimum lengths both set to the data length plus 2 (for the length field).

   Programs that utilize field level sensitivity always view these segments as fixed length. The two-byte length field is maintained by DL/I. The application program does not see the length field unless it is also defined as a sensitive field.

2. During PSB generation, define all fields to which the application program is sensitive by using SENFLD or VIRFLD statements.

3. To add a field to a segment, add a FIELD statement after the last currently existing field and increase the maximum length parameter for this segment. Re-run the DBD, PSB, and ACB generation for that data base.

   When a variable length segment is called by an application program that utilizes field level sensitivity, and the added field does not yet exist (contains no data), DL/I expands the segment with null values (for defined fields) or binary zeroes (for undefined areas) to fit the application program's view.

Note: To add dynamic segment expansion capability to fixed length segments in an existing data base, it will be necessary for you to unload the data base, add the two-byte length field to the desired segments, and reload the data base using a user-written initial load program.

Additional Field Level Sensitivity Considerations

- Segment Selection

  Any field to be used in segment selection in a segment defined by field level sensitivity must be defined as a sensitive field using either a SENFLD statement or a VIRFLD statement containing SENFLD subfields.

  Field information supplied in segment selection should be in the format of the application program's view of the field. The field identified in the segment selection, and any subfields that the application is sensitive to, are converted to the physical view before the compare is done. Any fields overlapping either end of the field identified in the segment selection are not converted.

  Note: DL/I does not take field type into consideration for compares for segment selection. For example, for binary segment selection, a negative number will be larger than a positive.

  Also, fields converted by DL/I to packed or zoned format will use the S/370 preferred sign.

- Insert

  If you specify insert activity for a segment containing fields the application is sensitive to, sensitivity must also be specified for any sequence fields in the segment. The field need not be identified by name, as long as its area is included in some field for which sensitivity has been specified.

  Note: DL/I DOS/VS users planning future migration to IMS/VS are cautioned that IMS/VS requires that sequence fields be identified by name.

  Insert sensitivity of bidirectional logical children requires sensitivity to both normal and logical twin sequence fields.

  If insert sensitivity is specified for a logical child, the application must be sensitive to the entire destination parent concatenated key. If the destination parent is to be inserted as part of the concatenated segment, the application must be sensitive to its sequence field.

- Fields and Subfields

  You may define a field for the application program's view that contains a number of other fields as subfields. This allows a set of separately processed fields to be referenced as a group and used in segment selection. For purposes of this discussion, we will call this field an "overfield". The following considerations apply:

  – Overfields must be completely defined for the application view by the subfields they contain. These subfields must be contiguous (no holes).
  – Overfields may be defined with the SENFLD statement only if there is a corresponding overfield defined in the physical view, and any non-virtual subfields in the application view appear in the physical view of the corresponding field.
  – Overfields for which there is no matching physical field that contains all the same physical subfields must be defined with a VIRFLD statement.
  – Field exit routines for overfields can do no conversion. The overfield is always processed before the subfields that make it up.
  – Two fields that overlap must both be completely defined in the application view by subfields. Their

intersections must be completely (no holes) and exactly (no overlap on ends) defined by subfields.
  - DL/I does no conversion on overfields.

Application View of Search Fields for Alternate Processing Sequence: An application's view of the search field to be used in segment selection for alternate sequence processing may be defined in one of two ways:

- Explicitly, by defining the application's view of the search field with SENFLD statements for the target segment.
- Implicitly, by defining the application's view of the search fields with SENFLD statements for the source segments.

Explicit definitions will override any implicit definitions.

If no sensitivity for the search fields is defined, either implicit or explicit, the physical view is assumed.

- Explicit Definition

  During PSBGEN, the application's view of the search field(s) for an alternate processing sequence may be explicitly defined with SENFLD statements following the SENSEG statement for the first (target) segment in the PCB for the alternate sequence.

  Following any SENFLD statements (optional) for fields in the target segment, a SENFLD statement should be inserted naming the indexed field for the secondary index.

  If only a single search field is defined, this SENFLD should describe the application's view of the search field.

  If the index search field is made up of more than one field from the source segment, the initial SENFLD defines the total field. Additional SENFLD statements must be provided for each search field.

- Reordering Search Fields

  Subfields within any sequence or search field cannot be reordered in the application's view.

  If a field appears in both the physical and application's view (real field), and the application's view contains subfields, at least one of the subfields must be included in the physical view.

- Implicit Definition

  If the application's view specifies sensitivity, by name, to the field (or fields) used as search fields in the index source segment, the definition of those fields with a SENFLD statement for the indexed field in the target segment may be omitted. If multiple search fields are present, their order will be as defined in the XDFLD statement SRCH operand (physical view).

  Note: If there is no sensitivity to the source segment, or if the application is sensitive, but not field sensitive, and the view of the segment selection fields was not explicitly defined, DL/I will assume that those fields will have the same format as the physical version.

  If the application is field sensitive to the source segment, the application view of the search fields must be either implicitly or explicitly defined.

Definition types may not be mixed (one search field in an index implicitly defined, one explicitly).

Processing Controls

Processing Options

There are several types of processing that application programs can perform on the data stored in a data base. These are called processing options (PROCOPTs). You can specify which of these types a particular program is allowed to do. They can be specified at the level of the application view (PCB) and/or at the level of individual segment types. You have to make decisions about which PROCOPTS are to be specified for each program and, in some cases, for each segment type. The types are:

- Get (G). If this option is specified, your program can retrieve (get) segments from the data base.
- Insert (I). Specifying this option lets your program insert segments in the data base.
- Replace (R). This option lets your program replace the values contained in segments in the data base with different values.
- Delete (D). You use this option to let your program delete segments from the data base.
- All (A). Specifying this option lets your program do all of the previous functions. (They can also be specified in any combination of two or three.)
- Path (P). You have to specify this option if the program uses DL/I requests that operate on multiple segments in a hierarchical path with one HLPI command or CALL statement. P can not be used with options L or LS. In the HLPI command, a "FROM" or "INTO" option is used for a parent segment. In the CALL statement, a "D" command code is used in an SSA.
- Exclusive use (E). This option causes every segment defined in this PCB to be marked “exclusive control.” This means that only PCBs that are not sensitive to these segments can be scheduled at the same time this PCB is scheduled. Use this option only when you wish to override program isolation in an MPS or online environment. You can override the effect of this option for individual segment types by coding a processing option in the individual specification statements for those segments. Specifying this option here does not force the option onto individual segment specification statements that also specify some other processing option. Because of this, you must specify processing option E in the specification statement of any segment that requires exclusive use, if that statement specifies any other option. This option can be used with options G, I, R, D, and A.
- Read-only (O). You use this option to inhibit locking (enqueueing) by the program isolation function during retrievals of the same segment types in a data base. It can only be used with the G or G and P options, which gives it the read-only function.

  Note: If this option is specified here, only the G and GP options can be specified on related segment specification statements. Consider always specifying G here, and overriding that whenever necessary on a related segment statement. Use processing option O with care, and only if the integrity of the data returned is not of critical importance. Do not perform updating based on data read with this option.

- Load (L). Use this option for the PCB of programs doing the original loading of data bases; except where the access method is HD indexed or HIDAM, or

HSAM or simple HSAM when a sequence field is defined in the root segment. This option cannot be used with G, I, R, D, or A.
- Load sequentially (LS). You must use this option for the PCB of programs doing the original loading of data bases that use the HD indexed or HIDAM access method, or HSAM or simple HSAM when a sequence field is defined in the root segment. This option cannot be used with G, I, R, D, or A.

Notes:

1. Processing option G is implied when an option of I, R, D, or A is used
2. Only options G, L, and LS are permitted when the access method is simple HSAM or HSAM.

Choosing DL/I Operational Capabilities

Online Operation

You can use DL/I in a CICS/VS online environment. This allows DL/I applications to access data bases concurrently. In this environment, DL/I executes within the CICS/VS partition. DL/I routines interact with CICS/VS routines during:

- CICS/VS and DL/I system initialization and termination
- DL/I scheduling of data bases
- Use of CICS/VS facilities including storage control, journalling, recovery, tracing, performance monitoring, and intersystem communication
- End of logical units of work including DL/I checkpoints and termination calls, CICS/VS syncpoint requests, and end of task.

Certain types of DL/I programs cannot be run in an online environment. They are:

- DL/I utilities
- Programs that access SHSAM or HSAM data bases
- Programs that load data bases.

In order for you to use the online environment, you have to perform a:

- CICS/VS system generation to tailor the CICS/VS system to your DL/I requirements
- DL/I online nucleus generation to tailor DL/I to the online environment.

These and other DL/I online requirements are discussed in chapter 5 and in Appendix A, “DL/I Virtual Storage Estimates” on page A-1 and refid=online.. Also see *DL/I DOS/VS Resource Definition and Utilities*.

Storage Layout Control (SLC) Option

DL/I modules are loaded into storage from a core image library during CICS/VS and DL/I system initialization. They are loaded in a predetermined sequence into storage from high available storage to the low available storage address.

Eligible DL/I action modules should be placed in the VSE shared virtual area (SVA) so that a single copy of the DL/I code can be shared by several partitions. This improves performance. See “Storage Layout Control (SLC) Option” in chapter 5 for information on which modules can be placed in the SVA.

You may want to control the allocation of virtual storage to modules and buffers. For instance, in an environment with high paging activity, DL/I action modules

should be placed in the CICS/VS partition with frequently used modules in low address space and infrequently used modules in high address space.

The storage layout control option lets you specify the sequence in which modules are loaded, whether each module is to be loaded into low or high storage within the partition, and whether on a page boundary.

During CICS/VS and DL/I initialization, the order of module, table, and buffer loading is:

1. Nucleus Load Table (NLT). CICS/VS loads CICS/VS management modules and tables based on the order and specification in the NLT. Included in the NLT is the DL/I online interface module (DLZNUC), which was created during ACT generation.
2. Storage Layout Control Table (SLC). DL/I loads its action modules and locates the DL/I buffer based on the order and specification in the SLC.
3. Application Load Table (ALT). CICS/VS loads its application modules and tables based on the order and specification in the ALT.

The CICS/VS partition looks like Figure 4-4 after loading.

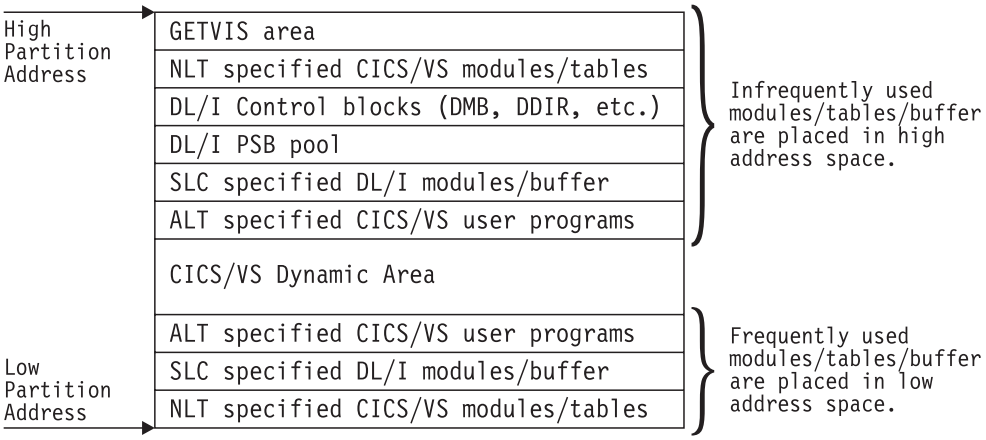


Figure 4-4. CICS/VS Partition

MPS Operation

You can use the multiple partition support (MPS) capability of DL/I to make it possible for multiple batch programs, executing in different partitions, and online application programs to access the same data base concurrently. For instance, an online application could issue inquiries to a data base while a batch program updates it.

MPS uses the DL/I resources and the multitasking facilities of DL/I and CICS/VS.

Certain types of DL/I programs cannot be run in an MPS environment. They are:

- DL/I utilities
- Programs that access SHSAM or HSAM data bases
- Programs that load data bases.

In addition, any batch DL/I programs that modify the contents of DL/I control blocks cannot run under MPS because the control blocks are in the online partition.

All data bases accessed by a program running under MPS control must be defined in the CICS/VS partition.

An MPS batch partition is defined as any VSE partition running under control of MPS. A maximum of twelve MPS batch partitions can be active at one time. Each MPS batch partition

is represented by one CICS/VS task; the MPS mirror task. All DL/I requests from a MPS batch partition are handled by this task, by passing the request to the DL/I action modules in the online partition and returning the results to the batch partition.

An online partition is defined as the partition that contains CICS/VS and the DL/I resources. MPS can be active in only one CICS/VS partition of the operating system at a time.

MPS Performance Considerations

Analyze the online applications that can run concurrently with each MPS job to see if any of them have update intent for the same segment types as the MPS job. If so, see if the processing options in either the batch or online application PSB can be changed to read-only intent for these conflicting segment types. If this is not possible, the MPS batch job should issue frequent checkpoint requests. You should use Program Isolation (PI) in the online partition to minimize scheduling conflict. If MPS is doing updating, frequent checkpoints will improve performance, free up PI resources, and minimize contention for updated segments.

MPS programs using Program Isolation with update intent for segments should issue frequent checkpoints to avoid using more storage than is necessary. Storage is obtained within the CICS/VS partition in blocks of 2K each time Program Isolation requires additional storage to maintain segment locks. Without the use of checkpoints, the longer an MPS program using Program Isolation runs, the more storage it needs. By taking frequent checkpoints, however, the storage currently used to lock segments is freed for reuse by Program Isolation.

Note: Storage, once allocated for Program Isolation usage, is never returned to VSE. The amount of storage allocated at any time represents the 'high water mark' for the total time that CICS/VS has been active. The effects of a long running MPS job could therefore be felt long after the job has completed.

Ordinarily, it is best to assign a lower priority to the MPS mirror task (transaction CSDC, program DLZBPC00) than to the CICS/VS tasks. This helps maintain online response time by ensuring that online tasks will be dispatched before the MPS mirror task.

In some cases you may find a marked increase in elapsed time for DL/I application programs run under MPS, as compared to running the same DL/I application programs as normal batch jobs. The difference in performance is due to a combination of the following reasons:

- Additional instructions are required by the interface between the MPS batch partition and the CICS/VS partition where the DL/I request is executed
- The MPS mirror task may have to wait to be dispatched.
- The fact that the online partition is normally of higher operating system dispatching priority than the MPS batch partition.

The sequence of events that occurs when an MPS batch partition issues a DL/I request is:

1. MPS Batch program issues DL/I request
2. DL/I in batch partition indicates it has a request to be processed by XPOSTing the MPS mirror task
3. The MPS mirror task, the batch partition controller, is dispatched by CICS/VS
4. The batch partition controller issues the DL/I request on behalf of the batch program
5. DL/I processes the request in the online partition
6. DL/I indicates the request was processed by XPOSTing DL/I in the batch partition
7. DL/I in the batch partition returns the result of the request to the application program.

The time required to process a DL/I request in an MPS environment is dependent on the activity of the CICS/VS system and the priority of the batch partition controller.

If the DL/I batch program is doing primarily sequential processing there is little I/O wait time per request. Consequently the additional time required to operate in an MPS environment becomes a significant part of the request processing time. In random processing, where I/O is normally associated with each request, the increased CPU time is a smaller percentage of the request processing time.

Between DL/I requests by the MPS batch program, CICS/VS will determine at regular intervals (called “short wait” intervals) if any events have completed, for which its active tasks are waiting. The length of the “short wait” interval is under control of the system operator (see “CSMT Short Wait Interval (SWT)” in *CICS/VS Operator's Guide*). Note that the “short wait” interrupt occurs for any CICS/VS system, not just when DL/I MPS is operational. When no tasks are active in the CICS/VS system except the DL/I MPC (master partition controller) task (CICS/VS is “idling”), CICS/VS interrupts the batch programs once every ICV interval (see *CICS/VS Performance Guide*) instead of at “short wait” intervals. The ICV interval is also under control of the system operator.

MPS System Requirements

In order to use MPS, CICS/VS and DL/I must be installed on the system. In addition, entries in CICS/VS and DL/I control tables are required. These are discussed in chapter 5. Also see *DL/I DOS/VS Resource Definition and Utilities*.

Processing Contention Control

Program Isolation and Intent Scheduling

DL/I provides two facilities that can be used to ensure the integrity of data bases concurrently accessed by multiple tasks in an online or MPS environment. These facilities are program isolation and intent scheduling. Intent scheduling requires less processing and real storage but generally permits fewer DL/I tasks to execute concurrently. Program isolation generally allows a greater degree of concurrency but requires additional processing and may require somewhat more real storage. Program isolation is the default choice and should be used whenever possible, especially when running MPS. Otherwise, online tasks in conflict with the MPS task would not be scheduled until the MPS task terminates. Intent scheduling is a better choice in a system that has severe real storage constraints and can

concurrently execute only two or three DL/I tasks. You should understand how each of these facilities operates in order to choose the proper facility for your environment. They are discussed in the following sections.

DL/I Scheduling: During PSB generation, you specify processing options on each sensitive segment in each data base PCB within the PSB. This is done by specifying the PROCOPT in either the PCB statement for the application view, or in the individual segment specification statements.

Scheduling of a task is by segment intent. That is, according to the manner in which a user intends to access segments in his logical data structure. The intents derived from the processing options specified by the user for certain segment types may implicitly affect other segments as well. For instance, delete intent for a certain segment also means delete intent for its dependents. This is called segment intent propagation and further details are discussed later in this section.

There are three scheduling intent types; read-only (RO), update (UP), and exclusive (EX). If a segment has more than one intent type as the result of multiple references or intent propagation, the most restrictive use applies. That is, exclusive intent applies in preference to update intent, with read-only intent being the least restrictive.

The following table shows the relationship between the processing options and intent.

PROCOPT	FUNCTION	INTENT
G	Get	RO
I	Insert	UP
R	Replace	UP
D	Delete	UP
A	All of the above	UP
E	Used in conjunction with any of any of the above. Specifies that an application program has an exclusive use of the data base or segment specified.	EX

Prior to accessing DL/I data bases, a CICS/VS program must issue a DL/I scheduling request to make a PSB available for use by the task. If, when a DL/I scheduling request is issued, a conflict in data base usage with a currently scheduled task arises, the task requesting scheduling must wait until the current task that is causing the conflict terminates. The following table shows how a conflicting action is defined.

	Segment Intent of Task Currently Executing			
Segment Intent of Task Being Scheduled	EX	UP	RO	NR
EX	X	X	X	
UP	X	X		
RO	X			
NR				

X indicates conflict when transaction scheduling is attempted. NR indicates no sensitivity; the segment type is not referenced.

Read-only intent with intent scheduling allows the task to be scheduled with any number of other read-only users and one update intent task. With program isolation (discussed later in this section), a read-only task may be scheduled with any number of read-only and update tasks.

Update intent allows any number of tasks that reference the same segment types for read only to be scheduled with the updating task. With intent scheduling, all tasks that reference the same segment type for update intent must be scheduled serially. Scheduling conflict does not occur between update versus update when program isolation is used.

Exclusive intent prohibits the concurrent scheduling of any tasks that reference the same segment type as the task that has specified exclusive use. There is no propagation of exclusive

intent. The effect of exclusive intent on scheduling logic is the same for both intent scheduling and program isolation.

During program and data base design, an analysis of the effect of intent on scheduling should be made. A matrix showing intent by data base, program, and segment is useful in this analysis. Figure 4-5 is an example of such a matrix.

DATA BASE NAME PART / SEGMENT NAME	PROGRAM NAME: RELPART	RELASSY	RELFAB	FINDPART	
PARTSEG	UP	RO	UP	RO	
ASSYSEG	RO	UP	RO	RO	
CONFIG	NR	RO	RO	RO	
FABSEG	RO	UP	EX	NR	
					PROPAGATED INTENT RO - Read Only UP - Update EX - Exclusive NR - No reference

Figure 4-5. Example of Matrix for Controlling Intent

Implications of Intent Propagation: The implications of intent propagation depend on many factors, some of which are: physical organization, segment rules, pointer combinations, processing options, and logical relationships. The following explains the effect these factors have on scheduling concurrency, as they relate to typical data base structures. Keep in mind that if a segment has more than one processing intent type (as the result of explicit or implicit processing options) the most restrictive intent is used.

- Get processing option

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

- Replace processing option

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

- Insert processing option

  Insert intent propagation is based on three rules:

  1. Programs that insert segments of the same segment type are not scheduled concurrently.
  2. Tasks that separately insert a physical parent segment and its physical child are not scheduled concurrently.
  3. Tasks that insert logical child/logical parent concatenated segments involving the same logical parent are not scheduled concurrently, if the insert rule of the logical parent is either virtual or logical. If the insert rule of the logical parent is physical, then only one program per logical child segment type can be scheduled. The physical insert rule prohibits inserting the logical parent by means of a concatenated segment. Only the logical child need be inserted.

  Update intent applies for the segment type designated by PROCOPT=I (rule 1). Update intent is propagated to all the immediate children (one level lower) from the designated segment because of rule 2. If the designated segment is a logical child, the update intent is propagated to the logical parent segment as specified by rule 3, and to the immediate children of the logical parent as specified by rule 2.

  The first variable that affects insertion intent is the data base organization. Since root segments in a HISAM data base are hierarchically related by being physically adjacent, a root segment insertion can cause other segments in the data base record to shift physical location. In this case all segments have update intent propagated to them and the root segment is marked as exclusive.

- Delete Processing Option

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

  When a segment type that is a logical parent is encountered in the downward scan, the delete rules of its logical children and their physical parents are determined. If the delete rule is virtual, then update intent is propagated to the logical child and to its physical dependents, and/or to the physical parent and its physical dependents. Since the propagation is downward, all segments in

the downward scan are inspected for logical relationships. As they are encountered, the logical child/logical parent/physical parent segment types are processed in the same manner as the original segment type. Deletion of the parent requires deletion of all physical dependents.

Scheduling Logic: When an online application program makes a PSB scheduling request, either of two immediate results can occur:

1. The task is scheduled. Control is returned to the program indicating this condition.
2. The task is suspended and must wait because of intent conflict. It will wait until there is no conflict. The scheduling request is considered unsuccessful until completed after the wait.

Since the goal of the scheduling request is the first result, the second should be avoided or its occurrences minimized.

Program isolation is the easiest way to eliminate intent conflicts among tasks that have update intent to the same segment type. When program isolation is not used, intent conflicts

between update tasks can be minimized by keeping the time between the PSB scheduling request and task termination (or TERMINATE request) as short as possible, and by issuing frequent checkpoint requests.

Application Programming Hints

- Attempt to group all data base update requests, preferably near the end of the transaction; issuing the scheduling request, the updating requests, and the TERMINATE request within the shortest possible time.
- Where grouping update requests is not possible, attempt to issue the scheduling request immediately before the first data base requests, and the TERMINATE request immediately after the last data base requests.
- PSB PROCOPT selection

  By carefully considering PROCOPT selection you can prevent unnecessary propagation of update or exclusive intent. By specifying PROCOPT=G on the PCB statement and then overriding it on the SENSEG statement as required, you can ensure that intent is not propagated unnecessarily.

  When specifying PROCOPT, use these guidelines:

  – Specify G whenever possible as there is no update intent with this processing option.
  – Specify R instead of I and D whenever possible as there is no intent propagation with this processing option.
  – Specify I and D only as required. If only insert capability is required, don't specify D also.
  – The use of PROCOPT=A is justified only when specifying all processing options (GIRD) and a P or E processing option is also required.
  – The use of PROCOPT=E should be well justified before it is used as it also prevents read-only tasks from running concurrently.
- If it is necessary to write conversational transactions, issue the TERMINATE request prior to a CICS/VS terminal write command requesting more transaction input. Failure to do this can result in a PSB being scheduled for an extended period, waiting for a terminal response from an operator.

- The shorter the time that a task has an update or exclusive PSB scheduled, the less time other tasks with intent conflict will have to wait. One way to minimize the time a task has an update or exclusive PSB scheduled is to use different PSBs during execution based on immediate task needs. Consider the following example of a CICS/VS-DL/I transaction:

```
PCB scheduling request (update PSB) ┐
Map input                           │
Edit input                          │
If input bad, write error message   │
  to terminal and return to CICS/VS │
Retrieve data base information and  ├─  Update PSB
  validate input                    │     scheduled
If input bad, write error message   │
  to terminal and return to CICS/VS │
Update data base                    │
Write acknowledgment message to     │
  terminal and return to CICS/VS    ┘
```

  Note that an update PSB is scheduled for the entire life of this transaction, even though it is needed only after the decision has been made to update the data base. The transaction could have been written as follows:

```
Map input
Edit input
If input bad, write error message to
  terminal and return to CICS/VS
PCB request (read-only PSB)
Retrieve data base information and
  validate input
TERMINATE request (read-only PSB)
PCB scheduling request (update PSB)'
Reposition in data base             ┐  Update PSB
Update data base                    ├─  scheduled
TERMINATE request (update PSB)      ┘
Write acknowledgment message to
  terminal and return to CICS/VS
```

  This revised version has an update PSB scheduled for a much shorter time, thus reducing wait time for other tasks with intent conflict. Note that the transaction must reposition within the data base following the first TERMINATE request. This is because a TERMINATE request causes position to be lost. The repositioning, while introducing some additional DL/I requests into the transaction logic, should not normally cause any additional I/O operations due to DL/I's I/O buffering techniques.

- These techniques can result in loss of data base positioning during a transaction, since issuing a TERMINATE request releases a program's position in a data base. Subsequent GET UNIQUE requests using segment selection with key values retained from previous requests can be used to regain positioning when needed. (See *DL/I DOS/VS Application Programming: High Level Programming Interface* or *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces* for a discussion of positioning.) Note that issuing a TERMINATE request automatically causes a CICS/VS synchronization point to be taken, which releases all currently owned recoverable CICS/VS resources.

CICS/VS Priority Assignment: The priority assigned to a CICS/VS task can also have an effect on its ability to be scheduled sooner rather than later.

For example, assume that two tasks, A and B, are scheduled, and that task X makes a scheduling request for a PSB that has update sensitivity for data bases 1 and 2. It must wait.

```
SCHEDULED        WAIT QUEUE

Task A - DB1     Task X - DB1
Task B - DB2               and DB2
```

Task A terminates and task X is dispatched by CICS/VS. DL/I performs segment intent checking against task B and, because of a conflict, task X must wait again. A new task C makes a scheduling request with update sensitivity to data base 1. It has the *same* priority as task X and therefore no intent conflict checks are made between these two tasks. No conflict is found between task C and the scheduled task B. The system now appears as:

```
SCHEDULED        WAIT QUEUE

Task B - DB2     Task X - DB1
Task C - DB1               and DB2
```

Task B now terminates, so task X is posted and is soon dispatched. It now conflicts with task C and must wait again.

Task X may be kept waiting for a long time. However, if task X had a higher priority than the other tasks, any new scheduling request would result in a segment intent conflict check against task X. In the above example, when task C made its scheduling request, it would have had to wait in this case because of intent conflict with task X. Task X would then have been scheduled when task B terminated.

Program Isolation: Program isolation reduces the amount of data base contention in the online and MPS environments. Program isolation performs two major functions: contention management and deadlock avoidance.

Contention Management: Contention management avoids conflicts in data usage by making contenders for a resource wait until the resource is available. This is functionally equivalent to intent scheduling processing. There are, however, some differences, which are explained in the following comparisons of the results of specifying each of the three processing intents (exclusive, update, and read-only) under both intent scheduling and program isolation.

Exclusive Intent:

Intent Scheduling	**Program Isolation**
Restrictions on Task	
Will not be scheduled until all other tasks with any specified intent for this segment type have issued a TERMINATE request.	Identical to intent scheduling.
Restrictions on Other Tasks	
Will not be scheduled if they have specified any intent for this segment type until this task has issued a TERMINATE request.	Identical to intent scheduling.
Comparison: Intent scheduling is used for exclusive intent under both procedures.	

Update Intent:

Update is defined for any segment for which the user has specified replace, delete, or insert intent; plus any additional segments so defined by the intent propagation rules.

Intent Scheduling	**Program Isolation**
Restrictions on Task	
Will not be scheduled until all other tasks with either exclusive or update intent specified for this segment type has issued a TERMINATE request.	1. Will not be scheduled until any other task with exclusive intent specified for this segment type has issued a TERMINATE request. 2. Will wait to read any segment updated by another task with update intent for that segment type until that task has issued a TERMINATE request. 3. Will wait to update any segment read by another task with the LOCKED option until that task has issued a TERMINATE request.
Restrictions on Other Tasks	
Will not be scheduled if they have specified update or exclusive intent for this segment type until this task has issued a TERMINATE request.	1. Any other task with exclusive intent for this segment type will wait to be scheduled until this task issues a TERMINATE request. 2. Any other task with update intent for this segment type will wait to update a segment read by this task with the LOCKED option until this task has issued a terminate request. 3. Any other task with either read-only or update intent for this segment type will wait to read segments updated by this task until this task has issued a TERMINATE request.
Comparison: Specification of update intent under program isolation provides the same protection from data usage conflicts as intent scheduling. Additionaly, contention is greatly reduced with program isolation because multiple tasks can concurrently update segments of the same type as long as they are in different data base records.	

Read-only Intent:

Intent Scheduling	Program Isolation
Restrictions on Task	
Will not be scheduled if any other scheduled task has specified exclusive intent for this segment type until that task has issued a TERMINATE request.	1. Identical to intent scheduling. 2. Will wait to read any segment updated by another scheduled task until that task has issued a TERMINATE request.
Restrictions on Other Tasks	
Any other task with exclusive intent for this segment type will wait to be scheduled until this task has issued a TERMINATE request.	Identical to intent scheduling.
Comparison: Specification of read-only intent under program isolation provides functionally superior support because, under intent scheduling, a task specifying read-only intent could read a segment subsequently backed out due to the failure of another task. This cannot occur under program isolation. Note that the CHECKPOINT request has the same effect in freeing resources enqueued for the issuing program as the TERMINATE request.	

Deadlock Avoidance: Deadlock avoidance recognizes and remedies the case where two or more tasks are interlocked on resources for which they are waiting. When a deadlock is recognized, it is avoided by terminating one of the involved tasks. The decision as to which task to terminate is made as follows:

1. Online tasks are terminated when a potential deadlock with an MPS batch task develops.
2. Within a class, the task with the fewest resources currently enqueued will be terminated.

CICS/VS dynamic transaction backout will be called for the terminated task if it is active for the transaction.

Program Isolation Operation: When program isolation is used, the intent scheduling logic is still invoked when a PSB request is processed. However, during system initialization of DL/I, if PI is to be used, all update intent bits in the PSB segment intent lists (PSILs) are translated to read-only bits as the PSBs are loaded. Therefore, tasks that would otherwise have intent conflicts due to update intent to the same segment type are allowed to execute concurrently. Tasks that have intent conflict due to exclusive intent are still suspended as if PI were not being used. To provide integrity for the concurrent update tasks, all data base requests made by tasks are monitored by DL/I so it can prevent simultaneous updating of the same segment occurrence by multiple tasks. By monitoring the requests DL/I is also able to detect and resolve deadlocks between tasks due to conflicting data base request patterns. To provide this integrity DL/I uses an internal queueing mechanism. DL/I enqueues two types of resources--data base records and segments.

Record Level Enqueueing: Record level enqueueing is used for HISAM and HD access methods. Regardless of access method, while a task is positioned within a data base record, no other task is

allowed to access that record. If another task attempts to access any segment in that record, it will be suspended. DL/I enqueues on the data base record whenever a task attempts to access any segment in the record. If that record is already enqueued by another task, then the task is suspended. When the record level enqueue is released another task may access the record. If another task had already requested some segment from the record and had been suspended, DL/I will resume that task.

If the data base uses HISAM and the task that “owns” the record has modified some segment within the data base record, then the record level enqueue is not released until the task commits the change. The change is committed whenever the task issues a CHECKPOINT (for MPS batch programs) or TERMINATE request, a CICS/VS sync point, or returns control to CICS/VS (or the operating system for an MPS batch program). If no segment was modified, the record level enqueue is released when the task positions within another data base record in the same data base (issues a data base request for a segment in another data base record).

If the data base uses an HD access method, the record level enqueue is released when the task positions within another data base record, regardless of any modifications made to segments. Integrity for modified segments is maintained through the use of segment level enqueueing, which is discussed next.

Segment Level Enqueueing: Segment level enqueueing is used only with HD access methods. If a task modifies a segment in an HD data base, DL/I enqueues that segment and no other task is allowed to access that segment occurrence until the “owning” task commits the change. If another task attempts to access the segment before the owning task commits the change, it will be suspended. Once the task that owns the segment has committed the change, the segment level enqueue is released and another task can then access the segment. If a task had tried to access the segment earlier and had been suspended, DL/I resumes that task.

The difference between the actions taken for a segment modified in a HISAM data base and an HD data base is in the ability of other tasks to access other segments in the data base record that contains the modified segment. Before the change is committed, if the modified segment is in a HISAM data base, no other tasks may access any segment in the record containing the modified segment. In an HD data base, other tasks may access other segments in the data base record containing the modified segment before the change is committed. Therefore, using an HD access method reduces the chances for long duration suspends due to PI enqueues.

Operational Considerations: When a deadlock occurs between two or more DL/I tasks because of conflicting request patterns, program isolation resolves the deadlock by abnormally terminating one of the tasks. Usually the task with the fewest DL/I records and segments enqueued is the one abended. The only exception to this is when an online task is deadlocked with an MPS batch program. In this case the online task is always terminated regardless of which has the fewest enqueues.

Because a deadlock occurs only between tasks, and is dependent on the time sequence of data base requests made by the various tasks, it is not possible to predict at what point in processing any particular program will be when a deadlock

abend occurs. The task that is abended may not be the one that issued the data base request that caused the deadlock.

If an MPS batch program is abended, the changes it made to the DL/I data bases are backed out by CICS/VS dynamic transaction backout if it is active for the MPS mirror transaction (CSDC). Any changes the MPS batch program made to other batch files are not backed out because they are not under control of DL/I or CICS/VS. Therefore, you will want to provide a mechanism to back out any other changes made to other batch files by the MPS

batch program if an abend occurs.

During dynamic transaction backout processing, CICS/VS issues messages only to the CICS/VS master terminal (CSMT) for MPS batch programs. No messages are written to the CPU console. Therefore, in the event of an MPS batch program abend (for any reason), you will have to verify that backout was successful by examining CSMT.

If you have programs that should not be exposed to the potential of a deadlock abend, then use PROCOPT=E in the programs' PSBs to eliminate deadlock potential. Specifying PROCOPT=E for all sensitive segments gives the program exclusive control over the segments. Since no other tasks sensitive to the same segment types can be scheduled concurrently with this PSB, the program cannot become deadlocked on DL/I resources.

Three informational error messages are issued by program isolation. The one indicating task termination due to invalid data usage conflict indicates a resource contention problem that is very dependent on the current mix of tasks. The task will probably execute successfully if rerun. If not, some evaluation of the mix of other tasks and the resources they reference will be required.

The message indicating insufficient space identifies a problem in getting more space from the CICS/DL/I partition for queuing control blocks. The task being terminated may not be at fault. More likely, some task(s) (probably MPS) is running for an excessive length of time without issuing a checkpoint or a CICS/VS sync point, thereby using up a lot of storage.

The last message indicates that the user is running program isolation without CICS/VS dynamic transaction backout or CICS/VS journaling. DL/I needs these facilities to ensure data base integrity when running with program isolation. DL/I initialization will be terminated. The CICS/VS system will stay active.

Programming Considerations:

- Access method selection:

  Avoid the use of the HISAM access method with program isolation. A change to any segment in a data base record in a HISAM data base results in the entire data base record remaining enqueued for the life of the task (or until a TERMINATE or sync point (end of the CICS/VS logical unit of work) is issued). Consequently other unmodified segments in that record become inaccessible to other tasks until the change is committed. This can increase response time of other tasks that wish to access other segments in the data base record containing a modified segment.

- Application program design:

  Don't design applications with a “control record” (or segment) in the data base that must be updated by every transaction in the application. This type of design will force those transactions to single-thread when they have to access the control information, increasing their response time.

  DL/I does not open a data base in the batch environment until the first request is issued to that data base. Because of this, application programs are often written beginning with a GET NEXT request to every data base named in the program's PSB. In the batch environment this is a good technique because it insures that all data bases can be opened prior to performing any application processing. Under MPS this technique may not be necessary as the data bases were probably opened during DL/I initialization. If programs using this technique are run under MPS with program isolation, the initial GET NEXT request to each data base will cause a record level enqueue of the first record in each data base. Consequently, an MPS batch program using the same initialization technique, starting in another partition, will be suspended until the first program retrieves some other record out of each of the data bases the two programs share. Those batch programs using this technique should be modified before running them under MPS with program isolation.

- CHECKPOINT request:

  MPS batch programs that update data bases should issue frequent CHECKPOINT requests. When a CHECKPOINT request is issued, any segments and records enqueued by program isolation on behalf of that program are released. A batch data base maintenance program run under MPS could conceivably enqueue every segment in the data base. If no CHECKPOINT requests are issued, the enqueued segments and records will remain enqueued until the MPS batch program completes. As online transactions attempt to access those same data base records and segments, they will be suspended until the MPS batch job completes.

  The large number of queue elements created by such an MPS batch program as described above will also require large amounts of queue space. This will increase your working set while the MPS batch job is running. If there is insufficient GETVIS area to supply the necessary queue space, the MPS batch job will be ABENDed.

  Finally, the CICS/VS dynamic log, in which CICS/VS has been recording all the data base changes, will become quite large unless CHECKPOINT requests are issued. A long-running MPS batch update program with no CHECKPOINT requests could cause your CICS/VS system to run out of temporary storage space. If this happens, the batch partition controller task (CSDC) may be put into a wait state indefinitely, or until it is abended by the CICS/VS deadlock-timeout mechanism.

  MPS batch programs which issue CHECKPOINT requests should use the MPS Restart facility to provide a restart mechanism in the event of failure. MPS batch programs not using the MPS Restart facility must contain their own restart mechanism. Refer to *Chapter 12. Data Base Recovery* and the *DL/I Recovery/Restart Guide* for more information on recovery and restart.

  Note: A code is passed back to the high-order byte of the JCB address field in the PCB whenever another task is required to wait on a resource owned by the calling task. This could be used to trigger checkpoints.

- PROCOPT=E:

  As an alternative to the CHECKPOINT request to reduce PI queueing activity, you can specify PROCOPT=E on all sensitive segments in the program's PSB. When an application program schedules a PSB with exclusive control specified for every sensitive segment, DL/I performs no program isolation enqueueing. It is unnecessary, as the exclusive control on every segment guarantees that no other DL/I task will be scheduled that has sensitivity to those segment types. When segments are specified with PROCOPT=E in an MPS batch program, online transactions that attempt to schedule against any of those segments will be returned codes indicating intent conflict with a batch MPS program, rather than being suspended.

  Note: The use of this exclusive control PSB technique does not eliminate any buildup of dynamic log records for a batch program. Only the CHECKPOINT request can do that.

- PROCOPT=GO(P):

  This processing option inhibits all program isolation queuing for requests made under the PCB in which this PROCOPT is specified. This processing option can be specified only on the PCB statement. If “O” has been specified, only “G” or “GP” can be specified as PROCOPTs on the related SENSEG statements. Therefore, the “O” processing option is for read-only purposes.

  This processing option should be used with caution. That is, use this option only if the integrity of the returned data is not important. With this option, DL/I does not check the ownership of the segment returned. This means that the read-only user might get a segment that had just been updated by another user. If the updating user should then abend and be backed out, the read-only user would have a segment containing data that does not exist in the data base. Therefore, the “O” option user should not perform updates in other data bases based on data read with this option.

  Note: 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.

- LOCKED option:

  Any task that relies on replace, delete, or insert intent to protect a segment from later modification by another task must specify the LOCKED option on the read to accomplish the same purpose.

Distributed Data

DL/I support of CICS/VS Intersystem Communication (ISC) makes it possible for DL/I application programs to access data bases that are resident on another CICS/VS system. The application programs don't need to know where the data bases are located. This support lets you:

- Centralize your data bases while distributing the processing of the data bases to other interconnected systems
- Define and maintain data bases where most of the activity for a data base occurs, while providing access to the data from all interconnected systems.

Remote PSB

In the online and MPS batch environments, remote PSB support lets you schedule a PSB on a remote system. This means that an application program can have access to data bases located on DL/I systems different from the one in which the program is running.

To process a data base on a connected or remote system, you specify, during online nucleus generation, the location of the remote PSBs by using the RPSB operand on the CICS/VS DLZACT macro instruction. All data bases referenced by a PSB must reside on the same system unless you are using extended remote PSB support (see “Extended Remote PSB” later in this chapter). In that case, two PSBs are actually used, but the rule still applies to each PSB component.

Intersystem communication support is invoked when the DL/I program request handler detects that a scheduling request is for a PSB on a remote system. CICS/VS ISC invokes the DL/I program request handler in the remote system through a program called the CICS/VS ISC mirror transaction, DFHMIR. This program then issues DL/I requests on behalf of the application program in the connected system. This means that, for your system to process a data base request from another system, you must specify the

DFHMIR program in your DLZACT macro. The other system can then access your data bases.

Because the CICS/VS mirror transaction program issues DL/I requests on behalf of the connected system, the PSBNAME operand must include all PSB names that can be referenced by any other system. Therefore, to help simplify online nucleus generation, an additional operand, REMOTE=YES, can be specified on the TYPE=CONFIG statement instead of, or in conjunction with, specifying the DFHMIR program on the TYPE=PROGRAM statement. Figure 4-6 illustrates a two-system configuration with a centralized data base to show this concept.

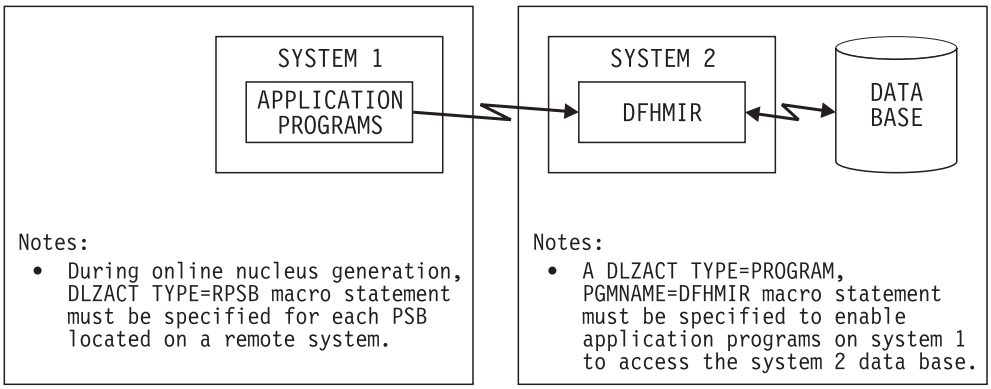


Figure 4-6. Centralized Data Base Configuration

Both systems must have DL/I, but the data base(s) for system 1 is located on system 2, and application programs on system 1 use CICS/VS ISC support to access the data bases(s). Note the use of the RPSB operand on system 1 to identify the PSBs needed on system 2. In turn, system 2 uses the CICS/VS ISC mirror transaction program to allow processing of the data base(s) on system 2 from the application program on system 1. The PSBs on system 2 that are used by the system 1 application program are identified in the DLZACT macro statement that specifies the DFHMIR program. Figure 4-7 illustrates a two-system

configuration where each system maintains its own data bases while providing access to them from application programs running on the connected system. Note that each system now requires the use of both the RPSB and DFHMIR operands in their respective online nucleus generations.

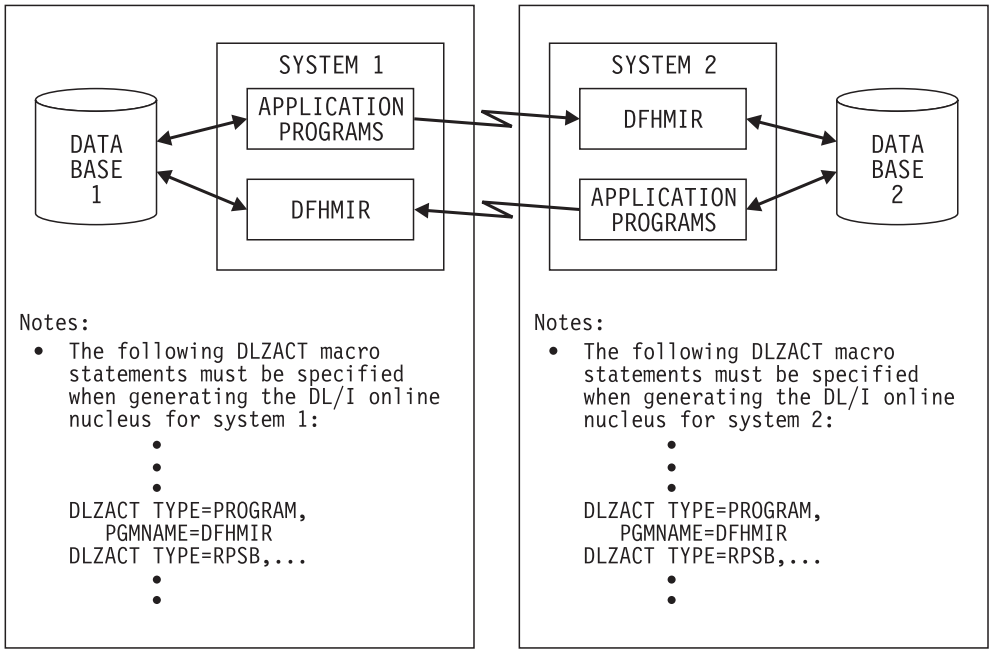


Figure 4-7. Separate Data Base Configuration

REMOTE=YES can be used instead of DFHMIR when all the PSBs on system 1 can be referenced by both resident application programs and application programs located on system 2. Using REMOTE=YES generates the equivalent of a DLZACT TYPE=PROGRAM, PGMNAME=DFHMIR statement for the CICS/VS mirror program, which includes the PSB names of all PSBs that are in the DL/I online nucleus. If some additional PSBs are referenced from the connected system only, then DFHMIR and the additional psbnames must be specified in conjunction with REMOTE=YES.

Programming Notes

While most application programs are not affected by intersystem communication support, there are some factors that you must consider:

- MPS batch application programs that issue their own PSB scheduling requests receive an abnormal status code
- If you have a long running MPS batch job that needs to process a data base on a connected system, be sure that program isolation is used on the system in which the data

  base resides, to prevent possible performance degradation. For example, if program isolation is not used, a long-running MPS-originated task is not recognized as an MPS task in the connected system, and may cause another task in scheduling conflict with the MPS task to wait until the MPS task completes processing.

  See chapter 5 and *DL/I DOS/VS Resource Definition and Utilities* for details on how to specify this support.

Extended Remote PSB

In the online and MPS batch environments, extended remote PSB support lets you schedule a PSB on a remote system and a PSB on the local system simultaneously. This means that an application program can have simultaneous access to data bases located on two DL/I systems.

The conditions under which you might consider use of extended remote PSB support are:

- One or more satellite CICS/VS systems with DL/I DOS/VS are connected to a single central CICS/VS system with DL/I DOS/VS or IMS/VS
- Most of the data required by application programs on the satellite system is resident on that (the local) system. Other needed data is located on the central (the remote) system.

An extended remote PSB is a concatenation of one local PSB and one remote PSB. When scheduled, an extended remote PSB provides an ordered set of Program Control Blocks (PCBs) to the application program. To the program, the PCBs cannot be distinguished from the PCBs of an ordinary PSB. The order of PCBs in an extended remote PSB is determined by its definition in the DLZACT nucleus generation macro and is based on the local and remote PSB names. The PCBs in the PSB whose name comes first in standard EBCDIC collating sequence appears first in the PCB address list.

When an application program schedules an extended remote PSB, an attempt is made to schedule both the local and the remote source PSBs. If scheduling is successful for both, the PCB address lists from the remote and local PSBs are concatenated, and a single list of PCB addresses or indexes is presented to the application program. When the application program issues a data base request, the request is routed to the proper DL/I system. When the program issues a checkpoint request, the request is converted into a PSB checkpoint in both the local and remote DL/I systems. When the program issues an explicit or implicit PSB termination request, the PSB is terminated in both the local and remote DL/I systems.

See chapter 5 and *DL/I DOS/VS Resource Definition and Utilities* for details on how to specify this support.

Logging

DL/I provides a data base change logging mechanism that records every change made to a DL/I data base. DL/I records both “before” and “after” images of all segments changed by application programs. These log records are used by the various DL/I recovery utilities to restore the data base to its original condition after a failure that destroys information in the data base. The use of logging in your recovery procedure is described in detail in chapter 12. At this point you need to be concerned with such things as the choice of a logging medium and the effects that logging can have on performance.

Logging is handled differently in each of the three DL/I operating environments:

- In the batch environment, the DL/I system log should be used.
- In the MPS batch environment, logging is performed in the CICS/VS partition. No log is associated with the MPS batch partition.

- In the online environment, the CICS/VS system journal should be used. It is possible to use the DL/I system log, but you must provide your own dynamic backout capability if you intend to update your data bases online.

Choosing the Log Medium

DL/I log records can be recorded in any one of several different ways. Your choices are:

- On disk as a VSAM data set
- On standard labeled tapes
- On the CICS/VS system journal

In a normal batch environment, you can use either of the first two choices.

In an MPS batch environment, logging is performed in the CICS/VS partition, and no log device is used in the MPS batch partition.

In a CICS/VS environment, the last two choices are available. However, if DL/I program isolation or the CICS/VS recovery facilities (Emergency Restart or Dynamic Transaction Backout) are to be used, the CICS/VS system journal must be used. In addition, the CICS/VS journal usually performs better in a CICS/VS environment. The CICS/VS system journal is a SAM data set and can reside on either disk or tape.

The DL/I change accumulation and forward recovery utilities ignore CICS/VS journal records on the log input. The DL/I backout utility, which runs as a batch job, will not accept a CICS/VS disk journal as input. CICS/VS backout is normally done through dynamic transaction backout.

If the DL/I log is assigned to the CICS/VS system journal, CICS/VS opens and closes the logging device and performs I/O as required. DL/I still maintains write-ahead logging when its log is assigned to the CICS/VS journal.

When using CICS/VS journaling, a block size larger than 1024 bytes can be specified in the CICS/VS journal control table. The maximum block size is 32,767 bytes.

The following table compares the time required to write the default (1K) DL/I log record on a variety of IBM devices. This figure assumes no seek is required on disk and no channel contention is encountered during the write.

IBM Device Type	Data Transfer Time For 1K Bytes (MS)	Average Rotational Delay Time (MS)	Total Log Write Time (MS)
3410/11-1	51.2	15.0	66.2
3410/11-2	25.6	12.0	37.6
3410/11-3	12.8	6.0	18.8
3420/3	8.5	4.0	12.5
3420/5	5.1	2.9	8.0
3420/7	3.2	2.0	5.2
3330/1	1.3	8.4	9.7
3340/44	1.2	10.1	11.3
3350	.9	8.4	9.3
3370	.5	10.1	10.6
3375	.5	10.1	10.6

Multivolume System Log Tapes

All of the utilities that read or write DL/I system log tapes allow the use of multivolume tape logs. The DL/I data base backout utility is able to process multivolume tape logs by treating each volume as a separate file.

Multifile System Log Tapes

Multifile DL/I system log tape support, in the batch environment, allows the use of the same log tape for many batch jobs that are doing minimal updating. This eliminates the need for having a log tape for each batch run, and simplifies keeping track of physical tape reels.

The utilities support multifile logs by means of input control statements.

Performance Considerations

The performance of your application programs can be very dependent on the efficiency of the DL/I logging mechanism. You can affect that efficiency by the design of your data base application. Minimizing the number of physical writes to the log improves logging mechanism efficiency. Physical writes to the log are required whenever:

1. The log buffer fills up
2. A modified data base record is about to be written back to disk and the log records corresponding to the change have not yet been written.

The second condition is known as a “force-write” of the log. Physical writes to the log because of the first condition are performed asynchronously by DL/I. Force-writes are, by their very nature, synchronous. Thus, the execution of your application program will be delayed until the log record is written. If your application program causes many force-writes to the log, you will find that the performance of your application program is dependent on the speed of the log device. (The effect of

using the asynchronous logging option (see the section “Asynchronous Logging Option” below) is to skip the force-writing of the log under the second condition.)

There are several things you can do in designing your data base applications to minimize writing to the log:

- Avoid the use of HISAM. The HISAM access method uses VSAM logical record processing. As a consequence, DL/I has no control over when a physical write to the data base occurs. DL/I assumes that every PUT request to VSAM results in a physical write to the data base. This causes DL/I to force-write the log more often than would normally occur if an HD access method were chosen. With the HD access methods, DL/I has complete knowledge of when physical writes to the data base actually occur. Physical writes tend to be deferred longer, thus allowing a greater opportunity for an asynchronous log write.
- Index data bases; whether HD indexed or HIDAM primary index data bases, or secondary index data bases; are also processed using VSAM logical record processing. Therefore, the same conditions exist when DL/I updates these data bases as exist for HISAM data bases. Be particularly careful with secondary indexes. Every time the source field value changes for a secondary index, DL/I will write at least three log records. If the source field is very volatile, considerable logging activity will occur.
- Consider the relative data base activity when assigning several HD, HDAM, or HIDAM data bases to the same buffer subpool. If two highly active data bases are assigned to the same subpool, activity in one data base is likely to cause DL/I to write back updates to the other data base sooner than would otherwise be necessary. DL/I will do this to reuse space in the subpool. This, in turn, could cause force-writes to the log. Balancing the activity across subpools will reduce the chances of this premature writing.

Whenever a choice exists, the DL/I batch user should log to tape instead of disk. Device access time is generally faster and the CPU time required to write to the logging device is significantly less. Tape logging is especially recommended for short batch jobs since the time to OPEN and CLOSE the log file(s) then becomes a significant portion of the job execution time and the OPEN and CLOSE of the tape file is much faster than the equivalent OPEN and CLOSE of the disk file(s).

Due to the frequency of forced writes to the log device in most environments, the default log buffer size of 1K will usually be sufficiently large so that buffer-full writes are the exception rather than the rule. However, if your installation is frequently waiting for log I/O, and does not often have forced writes to the log, specifying larger buffer sizes could reduce wait time, which would reduce response time. If the CICS journal facility is being used for DL/I logging, the CICS shutdown statistics should be examined. If the journal statistics show that the number of buffer-full conditions are a significant fraction of the number of buffers written, it may be possible to significantly improve DL/I performance by increasing the buffer size of the journal used by DL/I.

Disk Logging Considerations

In Batch: DL/I disk logging in batch is primarily designed to provide a logging facility for the disk-only user. It also provides an alternative for the tape and disk user. The format of tape and disk log records is compatible. DL/I disk logs are supported by the forward recovery utility, the change accumulation utility, the backout utility, and the log print utility. These utilities will process both VSAM disk files and SAM disk files as input.

When DL/I opens the disk log for output logging it specifies RESET, which resets the VSAM catalog high RBA to zero, effectively destroying any log records currently in the data set. Therefore, before you reuse a disk log data set for another program execution, you must first take steps to preserve the existing log records. This could be done by running the DL/I change accumulation utility against the log data set.

Because the volume of log records is difficult to estimate accurately, DL/I provides a variety of options to avoid having to terminate the job if the disk log data set fills up.

1. You can define the VSAM cluster with secondary allocation. Then if the primary allocation fills up during program execution, VSAM will automatically acquire additional space for the data set. This support is inherent in VSAM and is transparent to DL/I or the application program. All DL/I disk log data sets should normally be defined with secondary allocations.
2. You can specify two log data sets. DL/I will switch from one to the other when the log data set currently being used becomes full. When the switch occurs, DL/I will close the current log data set and write a message to the CPU console. When the operator responds to this message, DL/I will then open the other data set and inform the operator. At this point the operator should execute a program in another partition to preserve the old log data set contents in case another log switch becomes necessary.
3. If you specify only one log data set, DL/I will close the data set when it becomes full and issue a message to the CPU console. At this point the operator must execute a job in another partition to preserve the log data set contents. After the log data set contents have been copied or otherwise preserved, the operator can reply to the message and DL/I will reuse the log data set. If you choose to operate with this option, then the asynchronous operator communications feature of the Advanced Functions - DOS/VS Program Product (5746-XE2) could be used to facilitate console operations.

You can specify that the console message and consequent operator interaction is to be bypassed. However, because of the danger of losing a set of log records, especially when operating in the environment described above in point 3, you should be careful about specifying bypassing.

In Online: If you choose to use the CICS/VS journal for logging in the online or MPS environments, you can use either tape or disk logging. The access method used is SAM.

The DL/I data base backout utility supports CICS/VS SAM disk files only when it is invoked under control of the CICS/VS Dynamic Transaction Backout or Emergency Restart Facilities. Support for SAM disk files is necessary if the CICS/VS disk journal is used as the log device.

Asynchronous Logging Option

In the batch environment only, the asynchronous logging option can be used as a performance option. However, the performance improvement is generally small and there is a risk involved in its use that is explained below.

Because of the “write ahead” feature of logging, the speed of the log device may become a critical factor, delaying system execution. Asynchronous logging is designed to prevent this by making writes to the log asynchronously. The potential

risk in its use is that, after some types of failure, the DL/I backout utility cannot be used to correct the data base damage. There is one case where the data base can be physically updated before the corresponding log record is physically written. This can occur when a failure (such as a power failure) prevents DL/I from executing its abnormal termination routine successfully.

In the light of this risk, and the small advantage that its use provides, you should consider carefully before using asynchronous logging.

Limited Data Sharing

DL/I lets you specify that the data in a data base (except HSAM and SHSAM data bases) is to be shared between DL/I subsystems in one host system or between host systems. A DL/I subsystem consists of a single copy of DL/I operating in a VSE partition. It may be supporting either a single batch job or, in conjunction with CICS/VS, multiple MPS batch and online transactions. Of all the subsystems sharing DL/I data at the same time, only one may have update access. The others are restricted to read-only access.

Limited data sharing applies to sharing between subsystems. Sharing between transactions within a single subsystem is done through either intent scheduling or program isolation. (See the section “Processing Contention Control” in this chapter.) For instance, if a DL/I subsystem has update access to data under limited data sharing, then by using program isolation in a CICS/VS environment, multiple transactions within that subsystem could have concurrent update capability. Transactions running within another DL/I subsystem at the same time would be limited to read-only capability.

If you use limited data sharing, you must be aware that there is a situation where data consistency cannot be guaranteed. This occurs when a transaction operating under one subsystem reads data that is being modified under another subsystem. It is possible for the reading transaction to read uncommitted data (data that may be backed out if the writing transaction terminates abnormally), inconsistent data (data that the writing transaction has not completed writing), or data in invalid DL/I structures (data in segments with pointers to segments that the writing transaction has deleted). It is important to note that reading inconsistent or invalid data in these cases may cause the reading transaction to abnormally terminate. Because of the problem situations which may occur with limited data sharing, its use should be restricted to applications where data integrity is not vital.

Buffer Pool Assignment

Data Base Buffering

DL/I allocates and controls its own buffer pool (referred to as the HDBFR pool). The DL/I buffer pool is used mainly for the entry sequenced files (ESDS) of the HD organization (HD, HDAM, and HIDAM). In addition, workareas are provided in certain subpools. For HIDAM, these work areas are used when a root segment is inserted. For HISAM, the work areas are used for the insertion of all root segments, and the insertion of dependent segments when the segment does not fit into the ESDS control interval. They are also used for loading and insertions in simple HISAM and INDEX data bases.

The objective of the buffer pool is to allow blocks of data (VSAM control intervals) to remain in the buffer pool as long as possible. Data in the buffer pool may be

accessed and updated without causing I/O operations as long as there is no need to reuse the buffer space the data occupies. When a retrieve request occurs, the contents of the least recently used buffer is written to the file (if the buffer is not empty and the contents have been altered), and the requested control interval is read into the buffer. A use chain determines the order in which the buffers have been used. Empty buffers are placed at the end of the use chain and are always available for reuse.

VSAM buffer management maintains the index and data buffers for HISAM, simple HISAM, and INDEX data bases. You can specify the number of index and data buffers you want DL/I to use when processing requests using these data bases. If not specified, DL/I assumes three index and two data buffers. DL/I uses these values when defining the application control block for the file(s) to VSAM. In addition, you specify the buffer space in the VSAM access method DEFINE command. These two specifications should be consistent, otherwise VSAM will take the default actions that are described in *Using VSE/VSAM Command and Macros*. The HSAM buffers are dynamically allocated when DL/I is initialized.

HDBFR Pool Control: The HDBFR pool is subdivided into one (the default) or more (up to 255) subpools, as specified by you during DL/I initialization. However, only one subpool may be assigned for each HD, HDAM, or HIDAM data base referenced by the application program; plus one subpool for each HISAM, simple HISAM, or INDEX data base with insert or load sensitivity; plus one additional subpool if any of the data bases used has delete sensitivity.

Default Subpool Allocation: This section describes how DL/I allocates subpools and assigns data bases to them if you don't specify any HDBFR control options during DL/I initialization.

Normally, a subpool consists of 32 fixed-length buffers. The buffer size is generally consistent with the VSAM data base control interval size. It may be 512 or any multiple of 512 bytes. DL/I limits the control interval size to a maximum of 4096 bytes. You have, however, the possibility of specifying that you want to have one or more subpools containing less than 32 buffers. This is especially useful for sequential update processing, where only a small part of the data base information has to be kept in the buffer pool. If not otherwise specified at DL/I initialization, a physical data base is assigned to a specific subpool based on the control interval size and the number of subpools allocated, as follows: The system determines which buffer sizes are required by the data bases that will use the buffer pool. The data bases are then assigned to subpools so that a subpool with the smallest sufficient buffer size is used. If there are several subpools with the same buffer size, the one having the smallest number of data bases already assigned to it is selected. This procedure is repeated until all data bases have been assigned to a subpool. A data base can not be assigned to more than one subpool. However, a number of data bases may be assigned to the same subpool. A data base control interval size never exceeds the subpool buffer size, but may be less than the subpool buffer size. The utilization of the buffer pool is improved when all control intervals in the same subpool are of the same size.

You can override this subpool assignment technique by specifying your own options for assigning data bases to subpools. This, together with the specification of subpool sizes, lets you control the number of buffers allocated to a specific data base.

If you specify more subpools than there are data bases, one additional subpool with 32 512-byte buffers will be allocated for the delete work areas mentioned above; otherwise, these will be taken from the normal subpools when needed.

HISAM and INDEX data bases with insert or load sensitivity are allocated subpools the same as other data bases. After subpools have been assigned for them, one extra subpool (if available) may be used for delete.

Buffer Pool Control Options: DL/I provides you with some options to override buffer subpool sizes and assignments.

These options are specified during initialization, using the HDBFR operand in the DL/I parameter statement (batch) or in the DLZACT TYPE=BUFFER macro statement (online).

One subpool is allocated for each HDBFR operand. For each of these subpools the number of buffers can be specified as a value between 2 and 32. The default value is 32. The performance improvements gained by increasing the buffer pool size are not linear. Depending on the nature of the program, there is usually some point beyond which increasing the size of the buffer pool will not measurably increase the program's performance. For example, one benchmark of a batch program showed that increasing the buffer pool from 6 to 32 buffers only reduced the number of times a request could not be satisfied from the buffer pool (thus requiring an I/O operation) by 1%. Using the default buffer pool allocation (32 buffers) normally results in an increased working set without a corresponding gain in program performance due to reduced I/Os. The size of the buffer pool should be increased as the number of active data bases increases. The type of processing a program does (random or sequential) is much less a factor in determining the size of the buffer pool than the number of active data bases.

With the HDBFR operand, one or more HD data bases can be assigned to each subpool. The subpools assigned to data bases by means of the HDBFR operand are used exclusively for these data bases.

The number of subpools allocated is always greater than or equal to the number of HDBFR operands. If the number of subpools you specify is smaller than the number of HDBFR operands, the number of subpools is increased by the system to the number of HDBFR operands. If there are data bases that are not assigned to a subpool by HDBFR operands and there are no unassigned subpools available, one additional subpool with 32 buffers is allocated and all the remaining data bases are assigned to it. Its buffer size is the largest required by these data bases. If the number of subpools specified is greater than the number of HDBFR operands, any remaining data bases are assigned to the additional subpools, as described earlier in this section under “Default Subpool Allocation.” If there are still subpools available, one is used, if required, for delete work areas.

Explicitly assign the HD data bases to specific DL/I subpools rather than allowing DL/I to make default assignments. There are several factors you should consider when making the subpool assignments:

- Data base CI sizes
- Relative data base activity
- Type of processing (whether read-only or update).

Generally, it is not a good idea to assign data bases with different CI sizes to the same subpool. The buffers in a subpool are all the same size, and are equal in size to the largest CI of the data bases assigned to that subpool. Thus, if two data bases, one with a 1K CI size and the other with a 2K CI size, were assigned to the same subpool, all buffers within the subpool would be 2K. DL/I does not subdivide buffers, so 2K buffers would be used to hold the 1K CIs in this example. This results in an unnecessary increase in the DL/I buffer pool working set. Separating data bases with different CI sizes into different subpools should be your primary consideration when making data base—buffer pool assignments.

When assigning data bases to subpools, attempt to balance buffer pool activity. For example, don't assign two high activity data bases to one subpool and two low activity data bases to another subpool. Instead, put one of the high activity data bases with one of the low activity data bases in one subpool and the other two data bases in another subpool.

Once an MPS batch partition has started, its frequency of reference to its data bases will be much higher than an online task. Therefore, if a data base accessed by an MPS batch program is assigned to the same subpool as one accessed by online tasks, the batch will tend to monopolize the subpool, and slow down online transactions. Data bases primarily accessed by MPS batch programs should be assigned subpools separate from those accessed by online transactions.

Data bases should be assigned to subpools based on the type of processing performed. Try to assign read-only data bases to a different subpool from those frequently updated. This will minimize the number of times DL/I is forced to write its buffers back to the data base.

The sections “Run and Buffer Statistics for Online and MPS” and “Buffer Pool Statistics for Batch” in chapter 10 tell you how to obtain statistics on buffer pool activity. You can use this information to help determine the optimum buffer subpool sizes and assignments.

VSAM Buffer Allocation: VSAM buffer management and VSAM buffers are used by DL/I for SHISAM, HISAM and INDEX data bases. VSAM acquires space for its data set buffers from the partition GETVIS area when the data set is opened. There is no sharing of VSAM buffer space (or control blocks) between VSAM data sets used by DL/I. DL/I does not utilize the VSAM Local Shared Resource facility.

There are three ways in which you can control the amount of buffer space used by VSAM for a DL/I data set:

1. The BUFFERSPACE parameter in the AMS DEFINE command when you establish the data set's catalog entry
2. The HSBFR operand to DL/I when you specify the number of index and data buffers to be used
3. The BUFSP parameter of the DLBL statement for the data set.

The space allocated will be equal to the largest of the three sizes specified. This space will be subdivided by VSAM into index and data buffers. Unless you override DL/I with the HSBFR operand, it will request three index buffers and two data buffers for each data set using VSAM buffer management.

In an online or MPS environment, all VSAM requests made by DL/I will be of the “direct” type. Therefore, only the minimum number of data buffers (2) is necessary.

Requesting additional data buffers will provide no performance improvements but will increase the working set.

You should try to have at least as many index buffers as there are index levels in the KSDS. You can use a LISTCAT of the data set to find this value. Additional index buffers beyond this, up to the number of index records in the data set, may improve performance. Additional buffers will increase the working set of your system, however, so you will have to balance the number of index buffers desired with the amount of real storage available.

Since the VSAM buffer space is page aligned, you may be able to increase the number of index buffers somewhat without increasing real storage requirements. For example, suppose the index and data CI sizes for a data set were each 1K. The DL/I default of three index and two data buffers would require 5K, but three pages of storage. Increasing the number of index buffers to four, using the HSBFR operand to DL/I, will still only require three pages of storage. However, if the index CI size had been 2K, the buffer space would have ended on a page boundary, with the default DL/I index and data buffer specifications. Adding an index buffer in this case would increase the storage requirements. You should determine the index and data CI sizes for each KSDS from a LISTCAT, and specify enough index buffers to completely fill the pages allocated to the VSAM buffers.

If the buffer space specified in either the catalog or the DLBL statement is larger than necessary to hold the index and data buffers requested by DL/I, then VSAM will attempt to fill up any extra buffer space with data buffers. This occurs because the VSAM ACBs generated by DL/I specify both SEQ and DIR processing. Since additional data buffers are unnecessary and undesirable in DL/I, you should let the buffer space default when defining the DL/I data set catalog entries and not use the BUFSP parameter of the DLBL statement for DL/I data bases. Controlling the VSAM buffer allocation with DL/I's HSBFR operand allows you to specify exactly what you want.

Performance Considerations: As a general rule, specification of more than five to eight buffers per data base does not improve DL/I batch performance enough to justify the amount of additional real storage required.

The situation in an online environment is more complex. If the number of CICS/VS tasks that are concurrently accessing a given data base is usually not more than one, the same guideline of five to eight buffers applies. If, however, the average number of CICS/VS tasks that are concurrently accessing that data base is larger, each such task is going to need five to eight buffers for its own use, to maintain good performance. Consequently, the buffer pool for such a data base should be made correspondingly larger.

The transition from intent scheduling to program isolation may cause a serious scheduling bottleneck to be removed. This means that the number of concurrently active DL/I transactions will increase and a greater demand will thereby be placed on other DL/I resources. In particular, the DL/I buffer pool should be reevaluated to see if the size of some or all of its subpools should be increased.

Make sure that no data bases are assigned to a subpool whose buffer size is greater than the CI size defined for that data base. It is best to explicitly assign each data base to a particular subpool to ensure that buffer size and CI size will match up.

It is often a good strategy to assign each highly active data base to its own subpool. A group of low activity data bases having the same CI size might be assigned to a common subpool, as this will tend to reduce the amount of real storage required.

Real Storage Assignment

As with any pageable component, it is important that enough real storage be made available for the proper performance of DL/I. The basic guideline is that enough real storage should exist in the system so that all of the DL/I and VSAM code, control blocks, and buffers that are needed for the “mainline” execution of DL/I requests are normally able to reside in real storage. Appendix B, “DL/I Real Storage Estimates” on page B-1, “Real Storage Estimates,” describes how to estimate the DL/I and VSAM real storage requirements for any given batch or online installation. The *CICS/VS Performance Guide* contains a discussion of CICS/VS real storage requirements.

It is especially important in an online environment that enough real storage be available to the system to keep the system paging rate to a moderate level. This is because a page fault that occurs in the CICS/VS partition causes all active tasks within that partition to wait until the page fault is resolved. By way of contrast, I/O to a data base causes only the CICS/VS task that requested the I/O to wait for its completion. Any other active tasks can proceed without waiting.

If there is any significant concurrent use of DL/I from different partitions (excluding MPS), it is usually best to put all eligible DL/I action modules in the SVA to avoid duplicate copies in real storage. This would also be a good choice for an all-batch environment where real storage is plentiful, since most of the DL/I code will then remain resident in real storage and not have to be loaded for each job step that invokes DL/I.

In a DL/I online environment where there is little non-MPS concurrent use of DL/I by other partitions, it is usually best to have all of the DL/I modules in the CICS/VS partition.

VSAM should be run in the SVA.

If a VSE system containing online DL/I is being run in a virtual machine under VM/370, consider using an NLT and SLC laid out without page boundary alignment so that all DL/I and CICS/VS modules are loaded contiguously. This reduces DL/I and CICS/VS real storage requirements because VM/370 uses 4K pages, while the NLT and SLC do page alignment, if requested, assuming 2K pages. This creates unused gaps in the middle of the 4K pages used by VM/370.

For an application consisting of multiple CSECTs, a good technique to reduce the application working set is to arrange them in order of decreasing frequency of use. This can be done with a series of INCLUDE statements at link-edit time. Remember that one of these CSECTs will be the DL/I language interface module (DLZLI000). This is a small, high usage CSECT; so it should be associated with other high usage CSECTs. It can be AUTOLINKed to the end of the PHASE. The application's working set may increase by a page.

Choosing the Access Method

At this point we have discussed all of the DL/I capabilities that may have an effect on your choice of the access method for your data base. We will now describe the characteristics of the different access methods that DL/I supports. These characteristics will also enter into your choice. Once you understand the advantages and limitations of each method you will be ready to make your decision.

Access Method Characteristics

Earlier in this chapter we discussed the two types of DL/I data base organization: sequential and direct. The sequential organization is supported with four access methods:

- HSAM
- Simple HSAM
- HISAM
- Simple HISAM.

The direct organization is supported with one access method that has two types of access:

- HD randomized access (HDAM)
- HD indexed access (HIDAM)

The access methods for each type of organization are described individually in the following sections.

Sequential Organization Access Methods

HSAM Access Method: HSAM is a hierarchical access method that can only handle sequential processing. You can retrieve data from HSAM data bases, but you can't update the data. Updating can only be done by interleaving modifications as a new data base is created from the old.

HSAM is a useful access method when you are:

- Storing historical data
- Using the data base to collect data or statistics that will not need to be updated
- Always processing the data sequentially.

HSAM stores data base records in the sequence in which you submit them. They and the segments they contain can only be processed sequentially—in the order in which they were loaded. HSAM stores dependent segments in hierarchical sequence.

HSAM data bases are very basic data bases. Since the data is stored in hierarchical sequence, there is no need for pointers or indexes.

Simple HSAM Access Method: Simple HSAM is similar to HSAM except that it contains only root segments. A Simple HSAM data base cannot have a hierarchical structure. This access method is used chiefly as a conversion aid, permitting existing conventional tape and direct access storage device files to be used as data bases.

HISAM Access Method: HISAM is an access method that stores segments in hierarchical sequence with an index to locate root segments. Segments are stored in a logical record until the end of the logical record is reached. If there are still segments remaining in the data base record, they are stored in an overflow data set.

HISAM is well-suited for:

- Sequential access of records
- Sequential access of dependent segments.

Even though your processing has some of the characteristics above, HISAM is not necessarily a good choice if:

- You need to access dependents directly
- There will be a high volume of insertions and deletions
- A lot of the data base records exceed average size and have to use the overflow data set. This is because the segments that go into the overflow data set require additional I/O.

For data base records, HISAM data bases:

- Store records in key sequence
- Can locate a particular record with a key value by using the index.

For dependent segments, HISAM data bases:

- Start each HISAM data base record in a new logical record in the primary data set
- Store any remaining segments from one or more logical records in the overflow data set if the data base records won't fit in the primary data set.

Unlike the direct access methods, which are described below, HISAM doesn't make space that is vacated during deletions available for automatic reuse. The data base must be reorganized to make use of this space. The insertion of new segment occurrences may cause existing segments to be moved in order to keep the hierarchical sequence within the record. The space that is vacated when segments are moved to the overflow data set during this process is reusable by subsequent inserts, but space in the primary data set can only be reclaimed during reorganization.

Simple HISAM Access Method: Simple HISAM is similar to HISAM except that it contains only root segments. A Simple HISAM data base cannot have a hierarchical structure. This access method is used chiefly as a conversion aid, permitting existing key sequenced data sets (KSDS) to be accessed as data bases. Simple HISAM does not use an overflow data set.

Direct Organization Access Methods

Randomized Access: DL/I HD randomized access uses a randomizing routine to locate its root segments, then chains dependent segments together in their hierarchical paths. HD randomized access is efficient for a data base that will primarily use direct access.

The requirements that randomized access satisfies are:

- Direct access of root segments by root keys through a randomizing routine
- Direct access of paths of dependents

- Even distribution of data base records in storage
- New data base records and new segments are added by putting the new data into the nearest available space
- The space created by the deletion of data base records and segments is automatically reusable for other records or segments.

Randomized Access Characteristics: For root segments in a randomized access data base, DL/I:

- Stores them at the location determined by the randomizing routine, rather than in key sequence
- Uses a randomizing routine to locate the root segments
- Returns root segments in physical sequence, not key sequence, when root segments are retrieved sequentially.

With randomized access, dependent segments:

- Are stored anywhere, as close together as possible
- Are all chained together with pointers within a data base record.

An Overview of How Randomized Access Works: When a data base record is stored in an HD randomized data base, one or more direct-address root anchor points (RAPs) are kept at the beginning of each control interval. The RAP points to a chain of root segments. This chain is made up of all root segments that the randomizing routine assigns to this control interval and RAP. They are called *synonyms*. HD randomized also keeps a pointer at the beginning of each control interval that points to any free space in that CI. When you insert a segment, DL/I uses this pointer to locate free space in the control interval. To locate a root segment in a randomized access data base, you provide DL/I with the root key. The randomizing routine uses it to generate the relative block number and the number of the RAP that points to the chain of root segments. The RAP value specifies the location of the first root segment on the synonym chain.

Although HD randomized can place roots and dependents anywhere in the space allocated to the data base, it's good policy to choose HD options that keep roots and dependents close together.

Randomized access performance can be very good. How good depends largely on the randomizing routine you use. Performance also depends on other implementation factors such as:

- The control interval size you use
- The number of RAPs per control interval
- The pattern for chaining together different segments through pointers.

For sequential access of data base records by root key you must use a randomizing routine that stores roots in physical key sequence, or a secondary index.

The efficiency of HD randomized is highest when the most frequently accessed segments tend to be in the same control interval as the root anchor point generated by the randomizing routine. If this is the case, usually only one I/O operation is required to randomly position anywhere in the data base. In order to encourage this condition, take steps to minimize the number of synonyms and to ensure that the specification of the number of bytes of a data base record that can be stored in

the root addressable area is large enough so that the most frequently accessed segments can be together with the anchor point in the same CI.

For HD randomized, you must decide how much of the ESDS is to be used as the root-addressable area (RAA). This involves making a trade off between the number of synonyms and seek time. As the RAA is made progressively larger, the number of synonyms decreases while the average seek time to access a randomly chosen segment in the data base increases. The correct trade off point depends on many factors, such as: the hierarchical structure, average data base record size, and which segments are most frequently accessed.

When designing an HD randomized data base, one strategy is to make the RAA include the entire ESDS so that there is no overflow area. The ESDS should be defined to be significantly larger than the actual amount of space needed by the segments in the data base. If this is done, segments ordinarily put in the overflow area will instead be put as close as possible to the CI containing the root segment (assuming space is found within the scan limits specified in the DBDGEN); otherwise the segment is put at the end of the data base. This can significantly reduce seek time when accessing segments which otherwise would have been in the overflow area. This reduction may be large enough to offset the increased seek time due to the much larger RAA. The main benefit of this strategy is that, since the entire ESDS is root-addressable, the number of synonyms is reduced. This in turn can cause a significant drop in data base I/O operations by increasing the likelihood that the root segment of the desired data base record will be found in the same CI as the anchor point generated by the randomizing routine.

When designing the HD ESDS, consider specifying a CI size large enough to completely hold the average data base record size. This is especially attractive if most of the installation's applications tend to issue groups of DL/I calls on a data base record basis. For this reason, it is generally more efficient for a given amount of DL/I buffer space to have a small number of large CIs rather than a large number of small CIs.

Appendix G, “Randomizing Modules and DL/I User Exit Routine Interfaces” on page G-1 contains details about randomizing modules and gives examples of modules you can use.

Indexed Access: DL/I HD indexed access is the method that is most efficient for an approximately equal amount of direct and sequential processing. It is least efficient in the sequential access of dependents. It uses a separate index data base to locate the root segments of the data base records. Some specific requirements that it satisfies are:

- Direct and sequential access of records by their root keys
- Direct access of paths of dependents
- New data base records and new segments are added by putting the new data into the best available space, as determined by DL/I
- The space created by the deletion of data base records and segments is automatically reusable for other records or segments.

HD randomized access should be the first choice, but if not, indexed access can answer most processing requirements that involve direct processing or an approximately even mixture of direct and sequential processing.

Indexed Access Characteristics: With indexed access, root segments:

- Are initially loaded in key sequence
- Are stored wherever space is available after initial loading
- Are identified through the index, by the root key value that you supply.

With indexed access, dependent segments:

- Are stored anywhere, as close together as possible
- Are all chained together with pointers within a data base record.

An Overview of How Indexed Access Works: Indexed access uses two data bases: one, the primary data base, holds the data. The other is the index data base. The index data base contains entries for all of the root segments in order of their key fields. For each key entry, the index data base contains the address of that root segment in the primary data base.

When you access a root, you supply the key of the root. HD indexed locates the key in the index to find the address of the root, then goes to the primary data base to locate the segment.

HD indexed chains dependent segments together so that when you access a dependent segment, a pointer in each higher level segment locates the next segment below it in the hierarchy.

When you process data base records directly, HD indexed locates the root through the index, then locates the dependent segments of the root by using the pointers.

If you are going to process data base records sequentially, you can specify special pointers in the DBD so that DL/I doesn't have to go to the index each time to locate the next root segment. These pointers chain the roots together.

If you don't chain roots together, HD indexed always goes to the index to locate a root segment. When you process data base records sequentially, HD indexed accesses roots in key sequence by using the index. This only applies to sequential processing; when you access a root segment directly, HD indexed uses the index, and not pointers, to find the root segment you've requested.

Access Method Selection

Selection of the proper DL/I access methods is important to application processing, performance, and DASD space utilization. The choice of DL/I access method is usually independent of application programming and can be changed after production has begun.

Because only HD, HDAM, and HIDAM support logical relationships and secondary indexes, and support pointer options to tune data base accesses; the choice is usually between these DL/I access methods. The major difference between HD randomized or HDAM, and HD indexed or HIDAM is the capability of HD indexed and HIDAM to process root segments sequentially in key field order. From a performance point of view, HD randomized should be the access method that receives first consideration. If an application requires sequenced processing of root segments and no way is found to satisfy this requirement using HD randomized, then use HD indexed.

However, a requirement for sequential accessing of root segments may possibly be satisfied by a secondary index on the root segment of an HD randomized data base rather than by using HD indexed. In this way, most applications would be able to do the more efficient direct accessing of data base records via the randomizing routine.

Don't use HD indexed instead of HD randomized just because of the requirement for a randomizing module. The randomizing algorithms furnished in Appendix G, “Randomizing Modules and DL/I User Exit Routine Interfaces” on page G-1 provide a good basic understanding of how an algorithm works and what its limitations and possibilities are. When randomly retrieving root segments, HD randomized requires fewer I/O operations than HD indexed. In HD randomized, assuming that there are no synonyms, only one I/O operation is required to get to the root segment. In HD indexed, one or more I/O operations may be needed to get to the KSDS index, one to the KSDS data, and another to the ESDS. From this point of view, you can tolerate some degree of synonyms in your HD randomized data base. Assuming 10% synonyms, you will have an average of 1.1 or fewer I/O operations per access to a root segment in HD randomized as opposed to a typical number of 3 or more for indexed.

The decision tree shown in Figure 4-8 is an aid in selecting the access method for your data base after you have taken into consideration all of the factors described in this chapter.

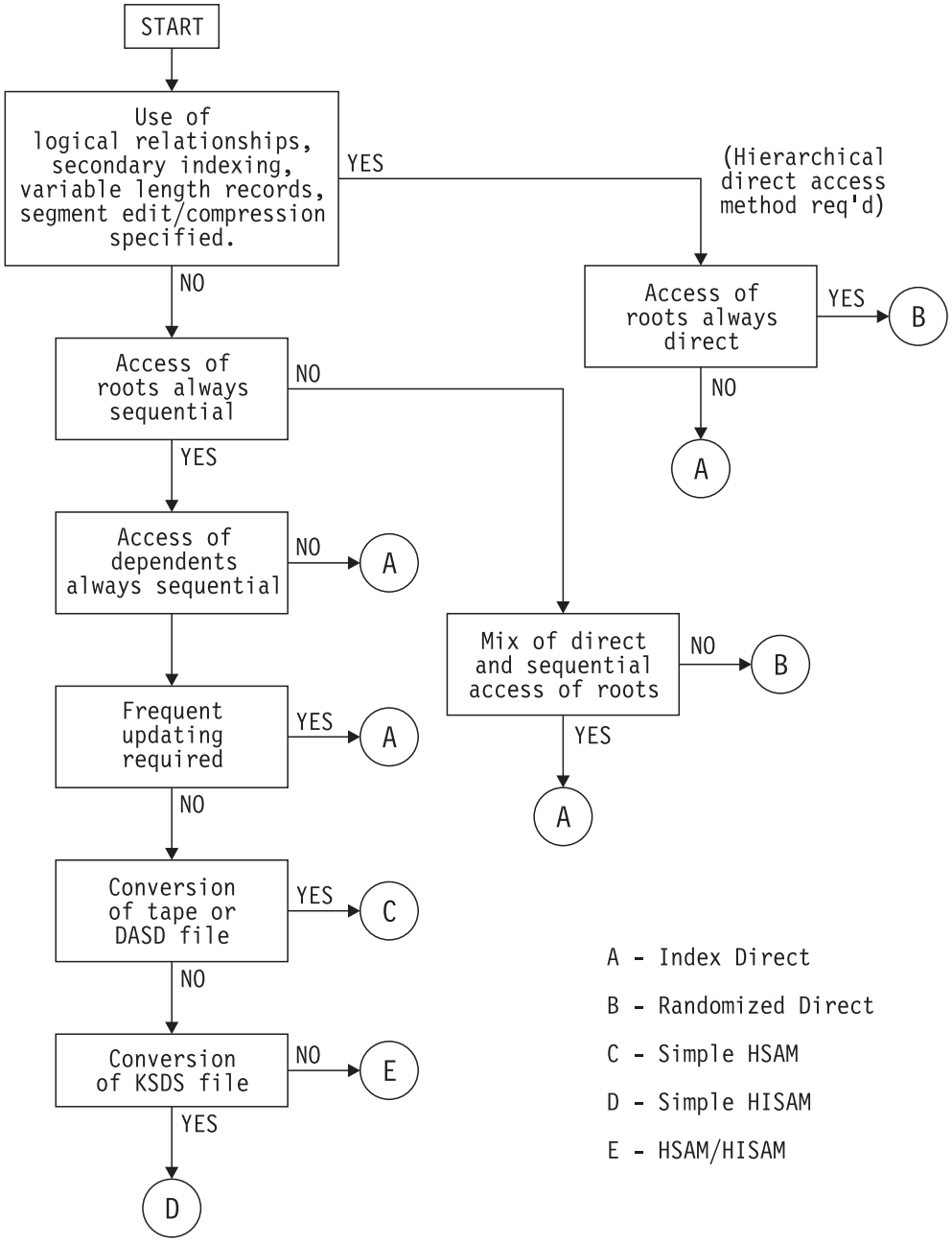


Figure 4-8. Access Method Selection Chart

Defining Data Base Physical Characteristics

Some of the decisions you need to make involve the physical characteristics of the data base as it is recorded in secondary storage. These are described in the following sections. Since these decisions depend on the access method used, you should make that decision first.

Determining VSAM Space Requirements

Before you can initially load and use a data base, you have to estimate how much space the necessary data set(s) will require

and then allocate the space to the data set(s) on the selected storage device.

The space requirement that you need to estimate is the minimum amount required to initially load the data base. If this data base will have data base records inserted after initial loading, you need to estimate and allocate additional space for that requirement.

To estimate the *minimum* space for initial loading, follow a procedure like the following:

- Find the size of an average data base record. To do this, you need the size of each segment type in the data base record and a figure for the average number of occurrences of each segment type in a record. Multiplying these two figures gives the size of an average record.

  The segment size is the size of the data portion plus the size of the prefix section, in bytes. The size of the prefix portion varies with the type of segment and the pointer options that you selected.

  To determine the average number of occurrences of a segment type in a data base record:

  – Find the average number of times the segment occurs under its immediate physical parent. For example, there might be an average of thirty students in a class.

  – Find the figure for the number of occurrences in a data base record by multiplying together the number of occurrences of each segment type in the path from the given segment back to the root segment (which has one occurrence). Following our example, if there was an average of two classes for each subject (the root segment) the total would be 60 occurrences of the student segment (30 students per class x 2 classes per subject x 1 subject).

  To get the average size of the data base record, you have to do the calculation we have just completed for every segment type in the record and add those figures.
- Determine the amount of overhead for VSAM CIs. You can use a figure of 10 bytes for the VSAM CI control information.
- Find the number of CIs or blocks needed.
- Find the number of CIs or blocks needed for free space.
- For HD, HDAM, and HIDAM data bases, include two CIs for DL/I overhead and bit maps.

Selecting a Logical Record Length

If the access method you chose for your data base is HD, HDAM, or HIDAM you can ignore this section. For any other access method, you have to make a decision about the specification of the data management logical record length(s) to be used for each data file. Specifying a logical record length is optional. If you don't specify it, DL/I will calculate a value for you during DBD generation. However, that value may not be the best choice for your data base. The object is

to make the best use of secondary storage space and, by doing that, obtain the best data base performance.

The logical record length requirement depends on the type of data base being defined.

- For HISAM, two logical record lengths are required. The first is the KSDS record length. It must be large enough to contain at least the root segment, the prefix, and DL/I control information. The second is the ESDS record length and must be large enough to contain the largest segment in the data base (other than the root segment), with its prefix and DL/I control information. The second record length must not be less than the first record length. If you omit specifying these record lengths, the value calculated during DBD generation is: the sum of the length of the longest segment type, its prefix, and its DL/I control information; or 4096 bytes, whichever is less. The record length for HISAM data bases must not exceed 4086 bytes.
- For SHISAM, only one logical record length is required. It is the KSDS record length. This record length is the same as the root segment size. It must not exceed 4086 bytes.
- For HSAM and SHSAM, two logical record lengths are required. The first is the input logical record length. The second is the output logical record length. If omitted, a length of 4096 bytes is used for tape files and the track size is used for DASD devices. All HSAM and SHSAM data files are in the fixed-length unblocked format.
- For an INDEX data base, only one logical record length is required. It is the KSDS record length. The value must be large enough to contain the index pointer segment plus the length of its prefix and DL/I control information. The record length must not exceed 4086 bytes.

Figure 4-9 may be helpful in calculating the size in bytes for the maximum segment lengths allowed for the different access methods.

ACCESS METHOD	ALLOCATION IN BYTES						
	SEGMENT PREFIX	DL/I CONTROL INFORMATION		VSAM CI CONTROL INFOR-MATION	MAXIMUM SEGMENT SIZE	MAXIMUM CI SIZE	DEFAULT CI SIZE
		RECORD	BLOCK				
SHISAM	0	0	0	10	4086	4096	2048
HISAM	2	5	0	10	4078	4096	2048
INDEX	6	5	0	10	4074	4096	2048
Primary HD Indexed	2+4T+4CI +8C2+4LP1 +8LP2+4LC +4LP3+PP	N/A	8	10	4068	4096	2048
Primary HD Randomized		N/A	4+ 4RAP	10	4068	4096	2048

Figure 4-9. Maximum Segment Lengths

Note: The maximum CI size for SHISAM, HISAM, and INDEX data bases refers only to the CI size of the data component of the KSDS. The CI size of the index component, however, may be recalculated by VSAM up to a maximum of 8192 bytes.

Notes for Figure 4-9:

```
CI =   VSAM Control Interval
T =    1 if POINTER=TWIN
       1 if POINTER=NOTWIN
       2 if POINTER=TWINBWD
C1 of  physical parent segment = the number of SEGM
       statements that specify
       PARENT=((parent-segment,SNGL)) or default
       to this.
C2 of  physical parent segment = the number of SEGM
       statements that specify
       PARENT=((parent-segment,DBLE)).
LP1 of logical parent segment = the number of
       LCHILD statements defining logical child
       segments of this segment that specify
       POINTER=SNGL or NONE or default to this.
LP2 of logical parent segment = the number of
       LCHILD statements defining logical child
       segments of this segment that specify
       POINTER=DBLE.

LP3 of logical parent segment = 0 if segment
       is a root segment.
       1 if segment is a dependent segment.
LC for logical child segment = 3 if POINTER=LTWIN
       or not specified.  4 if POINTER=LTWINBWD.
PP = 4 for all segments between (and not including)
       the root segment and a logical child, or
       logical parent, or indexed segment in a
       physical path.  If one segment is part of
       more than one such path, PP counts only once.
RAP =  the number of root anchor points as specified
       in the RMNAME operand of the DBD statement
       (minimum is one).
NA =   not applicable.
```

Selecting CI and Block Sizes

If the access method you chose for your data base is HSAM or SHSAM you can ignore this section. For any other access method, you have to make a decision about the specification of the control interval (CI) size to be used for each data file. Specifying CI size is optional. If you don't specify it, DL/I will calculate a value for you during DBD generation, using a control interval size of 2048 bytes wherever possible. However, that value may not be the best choice for your data base. The object is to make the best use of secondary storage space and, by doing that, obtain the best data base performance.

The CI size requirement depends on the type of data base being defined. For SHISAM, HISAM, and INDEX data bases, the number of VSAM records per VSAM control interval is required for each file of the data base. For HD, HDAM, and HIDAM, the control interval size is required.

- For HD, HDAM, and HIDAM, the size of the VSAM ESDS control interval must be specified. It must be a multiple of 512 bytes. The maximum value permitted by DL/I is 4096.
- For HISAM, two blocking factors must be specified. The first is the maximum number of records contained in one control interval for the primary data set.

The second is the maximum number of records contained in one control interval for the overflow data set. If the value you specify is incorrect, the correct value is calculated during DBD generation.

- For INDEX and SHISAM, one blocking factor is required. It is the number of VSAM KSDS records contained in one control interval. If the value you specify is incorrect, the correct value is calculated during DBD generation.

In choosing a CI size, consider the following:

- Try to choose a CI size that allows all often-referenced segments of a data base record to fit into one or more consecutive CIs.
- Large CI sizes favor sequential processing and DASD space utilization. However, if you are doing mostly direct processing, you should determine the segments needed per data base record per transaction.
- The VSAM CI size must be a multiple of 512 bytes. The maximum CI size allowed by DL/I is 4096 bytes. The CI contains 10 bytes of VSAM control information (seven bytes when you specify IMS compatibility).
- The CI size that you select for a SHISAM, HISAM, or INDEX data base refers only to the data component of the KSDS. VSAM may recalculate the CI size of the index component up to a maximum of 8192 bytes.

Figure 4-9 may be helpful in calculating the size in bytes for block and CI sizes.

Specifying Distributed Free Space

If the access method you chose for your data base is HSAM, SHSAM, HISAM, or SHISAM you can ignore this section. For HD, HDAM, or HIDAM you have to make a decision about the amount of free space to be reserved in the data base during a load or reload operation.

Unless you specify otherwise, when an HD, HDAM, or HIDAM data base is loaded, each ESDS CI in the data base is filled with as many segments as it can contain before a new CI is used. This means that a freshly loaded data base using one of these access methods has relatively little space left for subsequent insertions. The only free space in each CI would be that left between the end of the last segment that would fit in the CI and the end of the CI. Segments inserted in a freshly loaded HD, HDAM, or HIDAM data base are usually placed at the end of the data base rather than close to their related segments. Because of this, more seek time is required during both insertion and later retrievals than if the inserted segment had been placed close to its related segments. This will continue until enough segments have been deleted to provide free space within the CIs initially loaded. Even after some deletions have been made, if the resulting free space is not within the range you have instructed DL/I to search for free space for insertions, new segments will still be put at the end of the data base.

You can see this condition by comparing the high-RBA values in a LISTCAT of the ESDS portion of the data base over a period of time. If the high-RBA values constantly increase as time goes on, despite any delete activity, the free space is not being created where it is needed for new insertions.

By leaving free space in the ESDS as the data base is being loaded, segments inserted later can be placed close to their related segments rather than at the end of the data base. This reduces seek time for both the insert and for later retrievals, improving performance.

However, reserving lots of free space during load will spread your data base over more cylinders. This may cause slightly poorer performance on retrievals after an initial load than if you had not reserved any free space. Retrieval performance should improve as more segments are inserted, and eventually become better than if no free space had been reserved.

The amount of free space to be reserved depends on record size and the number of inserted segments anticipated. There are no firm rules for determining the necessary free space. You will have to try various values experimentally to find the optimum for each data base.

Reserving free space will be of most benefit when you expect a high number of insertions without offsetting deletions in the same area. If the insert activity is concentrated at the end of the data base (always adding new records with higher keys to an HD indexed data base, for instance), then reserving free space will provide little, if any, benefit.

Although distributed free space is helpful for both indexed and randomized access, there is more improvement with indexed. This is because, without distributed free space, the indexed ESDS is loaded contiguously. This forces all inserts to be made at the end of the data base. In randomized access, only the overflow area is loaded contiguously when distributed free space is not specified. There will generally be usable space scattered throughout the root addressable area (RAA) because of the random distribution of segments produced by the randomizing routine.

Free space can be specified in either or both of two ways: as a percentage of space to be reserved in each CI as it is initially loaded, or by leaving every "nth" CI empty during loading. The free space percentage can be any value from zero (the default) to 99%. Depending on segment size, the actual space may be larger. The frequency of free CIs can be any value from zero (the default) to 100, excluding one. Of the two methods of reserving free space, reserving a percentage in each CI should give better performance. By leaving some free space in every CI, DL/I will be able to insert segments in the CI where their related segments are located. However, within the limits to the values specified above, you can use either or both types to get any combination of free and/or partially free control intervals. For instance, you could specify that every fifth CI (5, 10, 15, ...) would be left free and at least 40 percent of all other CIs would also be left as free space. This free space would be used at insert time to place inserted segments as close to related segments as possible. Be sure not to reserve so much free space that there is insufficient room in a CI to hold the largest segment being inserted when the data base is loaded.

By leaving every "nth" CI empty, you force DL/I to perform an additional I/O operation at load time to read and write the CI with the free space. This will, however, still take less time than seeking to the end of the data set later to insert the segment, which would be required if no free space were reserved. This method is not recommended for randomized access data bases. This is because every "nth" CI in the ESDS, including the root addressable area, would be left free at load time. This means that a root segment that randomizes to such a "free" CI will be placed in the nearest available CI instead. A later retrieve of that root segment requires two I/O operations: one to get the root anchor point in the free CI and one to access the CI pointed to by that root anchor point.

You specify free space in the DBD generation. To obtain free space for existing data bases you will have to:

1. Unload the data base
2. Change the DBD specifications
3. Reload the data base.

Specifying distributed free space will cause an increase in the time required to load the data base. This increase is small when every "nth" CI is left free since it means only that a CI is periodically written with no data. The increase is more significant for the free space percentage value because fewer segments will fit into each CI, requiring more CIs to be written.

Selecting the Physical Storage Device Type

You have to make a decision about the type of physical storage device on which all data files for this data base will be stored. If the access method you chose for the data base is HSAM or SHSAM, the device can be tape; otherwise, it must be one of the types of direct access devices that are supported by DL/I.

Selecting the Scanning for Available Storage Space

If the access method you chose for your data base is HSAM, SHSAM, HISAM, or SHISAM you can ignore this section. For HD, HDAM, or HIDAM you have to make a decision about the number of cylinders (for direct access devices) or blocks (for FBA devices) to be scanned in both directions from the current position when searching for available storage space during segment insertions.

The number of cylinders can be any integer value between 0 and 255. The number of blocks can be any integer value between 0 and 32767. If you omit this specification, the default value is three cylinders or the equivalent for fixed block devices. If space cannot be found within the defined bounds, space at the current end of the data base is used. If you specify zero, only the current cylinder or fixed block is scanned.

Selecting VSAM Options

If the access method you chose for your data base is HSAM or SHSAM, you can ignore this section. For any other access method, you have to make a decision about the specification of VSAM options.

Use share option 1 or 2 for all VSAM data sets. Share option 1 is the only option that ensures full read and write integrity. Because DL/I-MPS programs residing in different partitions are really accessing the data bases through the common CICS/VS partition, VSAM share option 1 is sufficient if the data bases are not shareable.

DL/I always opens its data base data sets for updates, regardless of the intent of the PSB (except for the case of read-only data sharing (PROCOPT=GO)). Because of this, share option 2 provides no functional benefit for VSAM's data sets when the data bases are not shareable.

VSAM share option 2 is used to define data sets for shareable data bases. Specifying share option 2 alone allows the data base to be shared between subsystems on the same host system. If the data base is to be shared across host systems, both the data set and its VSAM catalog must be on devices shared

between the host systems. VSAM share option 2 ensures write integrity only. Therefore, data consistency for read-only subsystems is not guaranteed.

Share option 3 should never be used as it allows for concurrent scheduling of update jobs in different partitions with no warnings or data set integrity protection. It is especially important that share option 3 not be used with DL/I data base data sets as this could allow the high RBA value in the catalog to be updated incorrectly when the data sets are closed. Because DL/I depends on the high RBA value in the catalog for its operation, use of share option 3 can cause loss of information and internal data base pointer damage. You should only access DL/I data bases from multiple partitions using DL/I MPS.

Share option 4 should not be used with DL/I's VSAM data sets either, as this can also result in improper DL/I operation.

Chapter 5. Implementing the Design

This chapter describes the actual steps that are involved in implementing the physical data base, using the DL/I data definition language. There are ten sections:

1. DL/I Data Definition Language
2. DBDGEN
3. PSBGEN
4. ACBGEN
5. Specifying DL/I Definitional Capabilities
6. Specifying DL/I Operational Capabilities
7. DL/I Virtual Storage Estimates
8. DL/I Real Storage Estimates
9. Specifying the Data Base Access Method
10. Specifying Data Base Physical Characteristics

The first section introduces the two-level DL/I data definition language. The second section describes the Data Base Description generation and how it is used to implement the system view of the data base design. The third section describes the Program Specification Block generation and how it is used to define an application view of the data base. The fourth describes the purpose of the Application Control Blocks generation and how it is accomplished. The next tells how to specify the DL/I definitional capabilities that were chosen in the last chapter. The sixth tells how to specify the DL/I operational capabilities that were chosen in the last chapter. The next two sections describe the tasks of making DL/I real and virtual storage estimates. The actual procedures are described in Appendix A, “DL/I Virtual Storage Estimates” on page A-1 and Appendix B, “DL/I Real Storage Estimates” on page B-1. The ninth section tells how to specify the data base access method. The last section tells how to specify data base physical characteristics.

Once the design of a data base has been completed, an access method has been chosen, and decisions have been made as to the details of physical implementation, it is time to implement the design as a physical data base. This involves a number of tasks. Several of these tasks involve the generation of DL/I control blocks. Three of them (DBDGEN, PSBGEN, and ACTGEN) can be accomplished in either of two ways. You can code DL/I control statements and execute them as a normal application program job or, if you have a 3270-type terminal available, you can use the Interactive Macro Facility (IMF) to do these generations.

IMF gives you easy-to-use interactive procedures for creating and modifying the DL/I control blocks associated with the DBD, PSB, and ACT generations. Using a series of formatted display panels, IMF interactively prompts you for required information. Once the information for one of the functions has been entered, IMF allows you to review, modify, or delete it. When you are ready, you can request IMF to generate the control blocks. For complete information on IMF and how to use it, see *DL/I DOS/VS Interactive Resource Definition and Utilities*.

If you choose to not use IMF, the *DL/I DOS/VS Resource Definition and Utilities* gives you complete information on how to code and execute the application programs to perform the DBD, PSB, and ACT generations. In any case, you should read this chapter in preparation for those tasks.

DL/I Data Definition Language

This chapter introduces the two-level DL/I data base definition language. You use this language to define to DL/I the physical and logical characteristics of data bases, and the views of those data bases that apply to each application program.

The first level, called the data base description (DBD), applies to the system view of the data. It describes to DL/I the contents of the data base, names the segments and defines their hierarchical relationships, and specifies the physical organization and characteristics of the data base data set. The second level, the program specification block (PSB), applies to the application view of the data. It defines the application data structure (view of the data base) required by a particular application program.

Before the data base descriptions and program specification blocks can be used by DL/I, they must be merged and expanded into an internal format. DL/I provides a utility that creates a data management control block (DMB) for each related DBD CSECT and an expanded PSB for each related PSB CSECT. When DL/I is initialized, the DMBs and PSBs for the applications are loaded into storage and control is passed to the application program.

DBDGEN (Data Base Description Generation)

After you finish the design of your data bases, you must specify each of them to DL/I. This process is called data base description generation (DBDGEN). You use the first level of the DL/I data base definition language for data base description generation.

Figure 5-1 illustrates the execution of a DBD generation. To generate a DBD, you must first code a set of DL/I control statements that specify the data base characteristics you want. These control statements are presented to the DBD generation procedure as a normal application program job. They are processed as assembler language macro instructions. The result of a DBD generation is the creation of a DL/I DBD CSECT, which is cataloged and link-edited into a core image library. The DBD CSECT is used for subsequent processing of the data base.

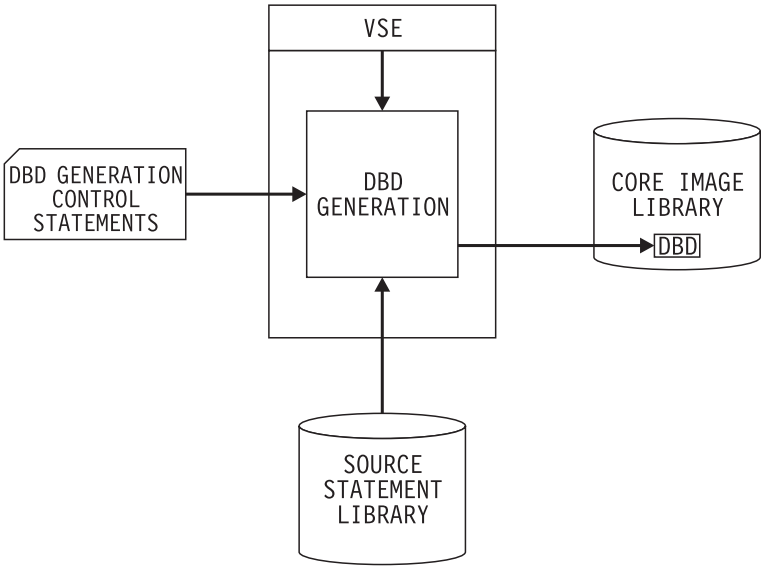


Figure 5-1. Data Base Description Generation (DBDGEN)

Figure 5-2 shows the sequence of the control statements in the DBD input stream. We will not discuss the control statements themselves in this book. Instead, we will describe how you use them to implement some of the decisions you made in the last chapter. The details of the formats of the control statements, and how to code them, are given in *DL/I DOS/VS Resource Definition and Utilities*.

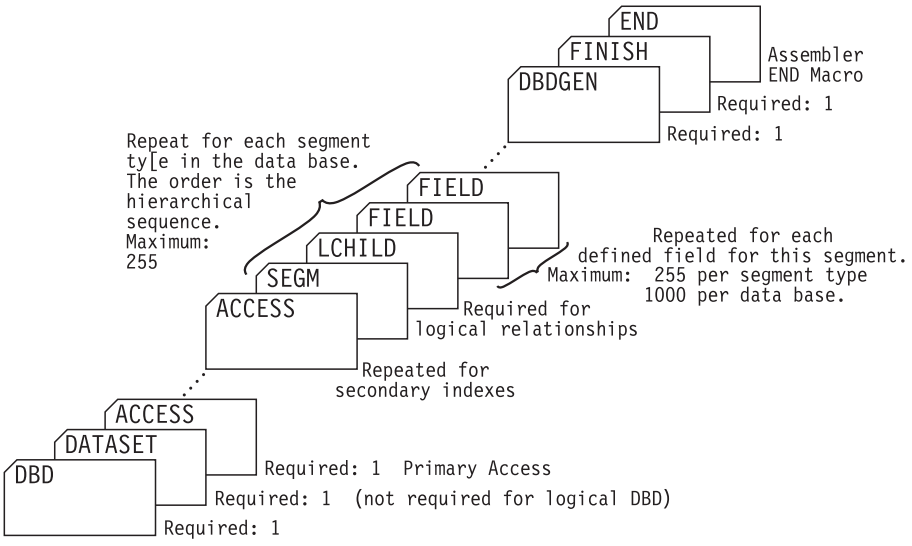


Figure 5-2. DBDGEN Control Statement Sequence

PSBGEN (Program Specification Block Generation)

Before a particular application program can be executed under DL/I, you must describe it to DL/I and specify the application views (logical data structures) it will use. Application views are the particular views of data bases that apply to your program. Each application view is represented by a program communication block (PCB). You must generate one or more PCBs for each data base your program uses. All of the PCBs associated with your program are generated together as a program specification block (PSB). The process that accomplishes this is called a program specification block generation (PSBGEN). You use the second level of the DL/I data base definition language for program specification block generation.

Figure 5-3 illustrates the execution of a PSB generation. To generate a PSB, you must first code a set of DL/I control statements that describe the PCBs your program uses. These control statements are presented to the PSB generation procedure as a normal application program job. They are processed as assembler language macro instructions. The result of a PSB generation is the creation of a DL/I PSB CSECT, which is cataloged and link-edited into a core image library. The PSB CSECT is used as input to the application control blocks creation and maintenance utility, which builds other DL/I control blocks for use at application program execution time.

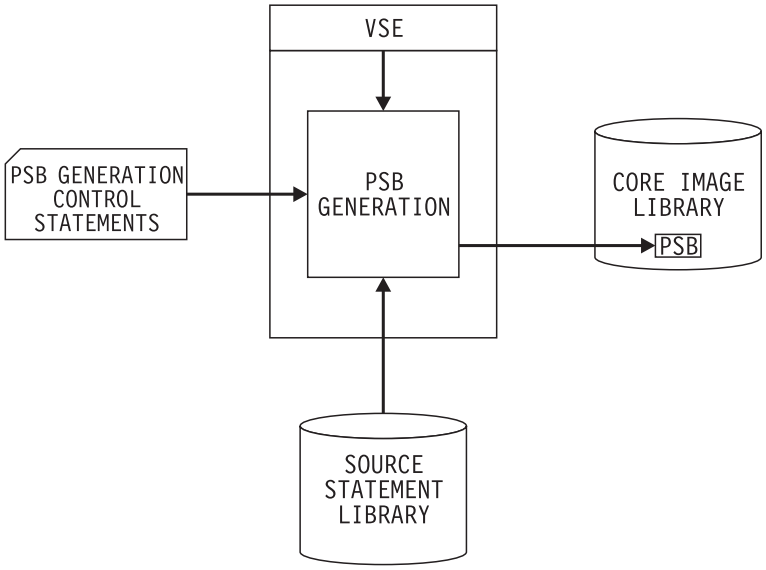


Figure 5-3. Program Specification Block Generation (PSBGEN)

Figure 5-4 shows the sequence of the control statements in the PSB input stream. We will not discuss the control statements themselves in this book. Instead, we will describe how you use them to implement some of the decisions you made in the last chapter. The details of the formats of the control statements, and how to code them, are given in *DL/I DOS/VS Resource Definition and Utilities*.

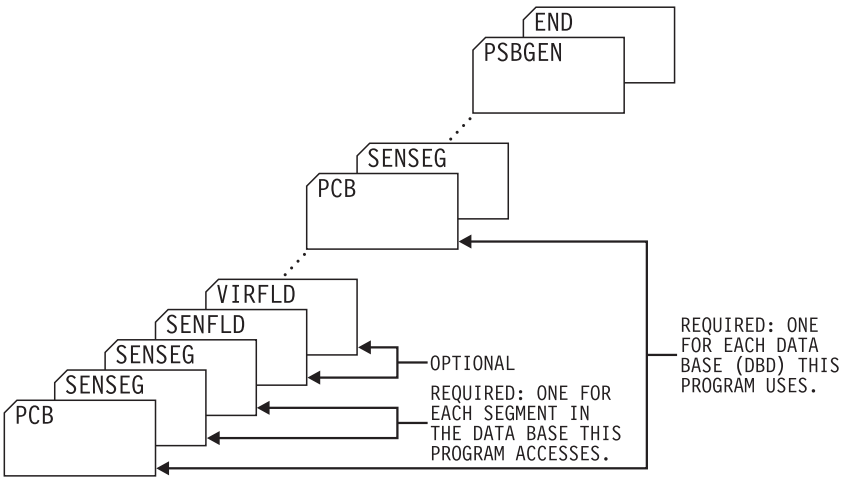


Figure 5-4. PSBGEN Control Statement Sequence

ACBGEN (Application Control Blocks Generation)

The previously defined system (DBD) and application (PSB) views must now be tied together so that DL/I can provide the correct data base management services for your application program. This involves the creation of internal control blocks (DL/I application control blocks, or ACBs) that are necessary for program execution. To do this, you run the application control blocks creation and maintenance utility.

The application control blocks creation and maintenance utility is executed as a normal application program. Input control statements modify its execution as required. You can specify the names of the PSBs for which blocks are to be built; the output destination on which the blocks will be stored; and whether all DMBs, none, or only those not already existing are to be built.

You can also request that the DL/I Documentation Aid facility create and store the DL/I data base definitions. This feature is available to DL/I users who have the Structured Query Language/Data System (SQL/DS) and Interactive SQL Facility (ISQL) program products installed on their VSE systems.

Through this facility, information about a DL/I data base is automatically collected and stored in SQL/DS tables whenever the data base definition is created or modified. This data can be accessed interactively through ISQL for display or for generation of printed reports.

At the start of execution, the PSB for the application program and its related DBD(s) are loaded from a core image library. An expanded PSB is then built from the PSB CSECT. A data management block (DMB) is created for each related DBD CSECT.

The output of the application control blocks creation and maintenance utility must be link-edited and cataloged into a core image library (see Figure 5-5). The core image library then contains one DMB and one utility PSB for each DBD, and one expanded PSB for each original PSB. When the DL/I system is initialized, these DL/I control blocks for the application program are loaded into storage.

The control statement requirements for this program are described in detail in *DL/I DOS/VS Resource Definition and Utilities*.

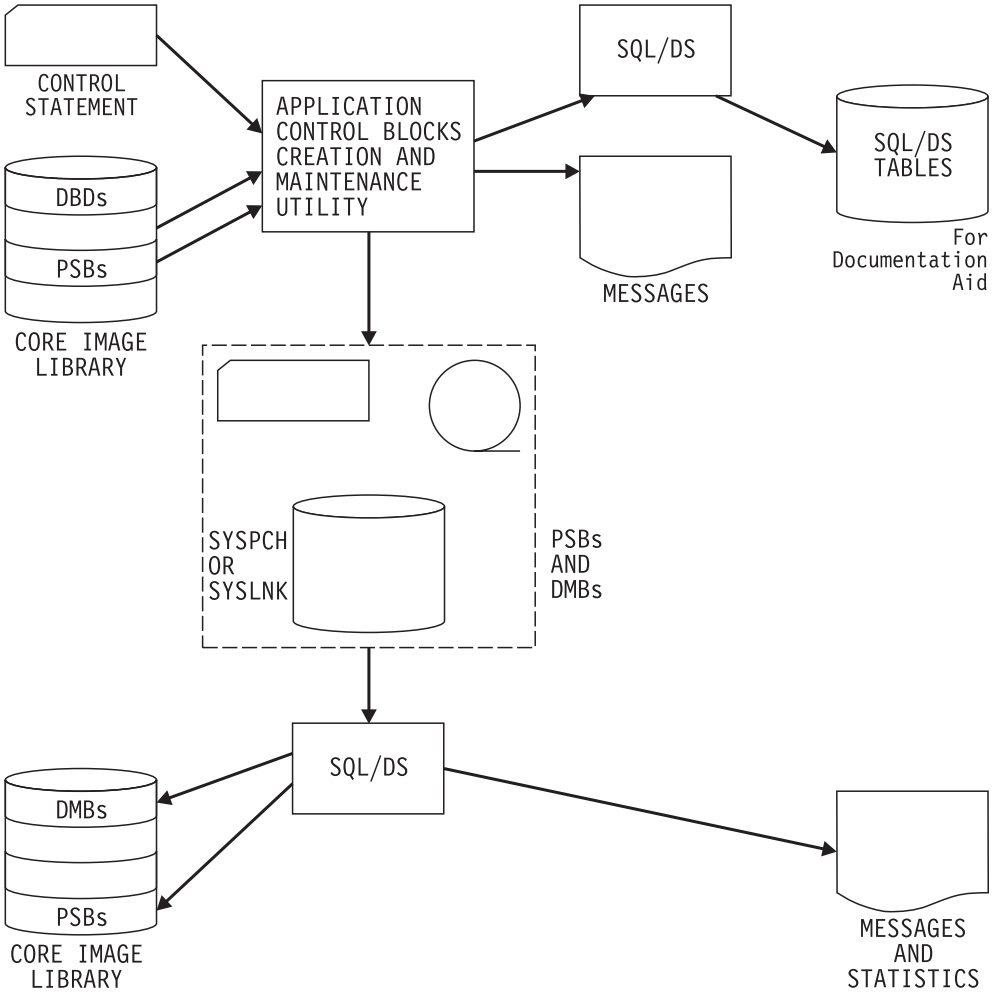


Figure 5-5. Application Control Blocks Creation and Maintenance Utility

Specifying DL/I Definitional Capabilities

Defining Fields

You use the FIELD statement in the DBD generation for your data base to define fields to DL/I. Each FIELD statement defines one field. The following characteristics of the field must be identified to DL/I:

- The name by which the field is referenced
- The field data type
- The length of the field
- The segment in which the field is located
- The location of the field within the segment.

Defining the Field Name

Each field must have a name. The name is specified by using the NAME operand. It is suggested that all field names start with an alphabetic character and contain only alphameric characters. Field names may be a maximum of eight characters in length.

Example:

```
NAME=NAMEFLD
```

Defining Field Data Type

The type of data to be stored in the field is specified by using the TYPE operand of the FIELD statement. The possible types are:

```
'X' hexadecimal
'H' halfword binary (implied length of two bytes)
'F' fullword binary (implied length of four bytes)
'P' packed decimal
'Z' zoned decimal
'C' character
'E' floating point (short, length of four bytes)
'D' floating point (long, length of eight bytes)
'L' floating point (extended, length of sixteen bytes)
```

If the data type is not specified, it will default to 'C'.

Example:

```
TYPE=F
```

Defining Field Length

The length of the field may be either implicitly defined, based on the field type, or explicitly defined by using the BYTES operand of the FIELD statement. The length must be specified for field types X, P, C, and Z. It may be specified to override the implied lengths for field types H and F. It may not be specified for field types E, D, and L.

Example:

```
BYTES=3
```

Specifying the Segment

The segment in which a field is located is identified by placing the FIELD statement defining this field after the SEGM statement for the segment in which it belongs, and before any other SEGM statements.

Defining Field Location Within a Segment

Unless otherwise specified, the first field specified for a segment is assumed to be located at the start of the segment, and each field defined after the first is assumed to start at the end of the previous one. The START operand of the FIELD statement can be used to override this default and specify a specific location for a field.

The starting location for a field can be specified either as a number, indicating the location within the segment at which the field is to start (the first position is location 1), or the name of a field already defined for this segment, indicating that this field is to start at the same location as the specified field.

Examples:

```
START=20
START=NAMEFLD
```

Defining Segments

You use the SEGM statement in the DBD generation for your data base to define segments to DL/I. Each SEGM statement defines one segment. These characteristics of each segment must be specified:

- The length of the segment
- The name by which it will be referenced.

Defining Segment Length

If all of the fields that belong to a segment are defined, DL/I will calculate the length of the segment such that it will contain all of the fields. If all the fields are not specified, or if the segment length has to be specified for some other reason, the BYTES operand of the SEGM statement can be used to define a specific length for a segment. Either one or two parameters can be specified for the BYTES operand. The first is the maximum segment length. The second, if specified, identifies the segment as having variable length (valid for HD access methods only) and specifies the minimum length for the segment.

Examples:

```
BYTES=30
BYTES=(30,12)
```

Defining the Segment Name

The name of the segment is specified by using the NAME operand of the SEGM statement. The segment name cannot exceed eight characters in length, and it is suggested that it start with an alphabetic character and contain only alphameric characters.

Example:

```
NAME=SEGNAME
```

Defining Segment Contents

The fields that belong to a segment are defined by placing their FIELD statements after the SEGM statement for the segment, and before the next SEGM or DBDGEN statement.

Examples:

```
SEGM NAME=A,...
   FIELD NAME=A1,..
   FIELD NAME=A2,..
   FIELD NAME=A3,..
SEGM NAME=B,...
```

Defining Physical Structures

To define the structure of a physical data base to DL/I, you must specify:

- The parent/child relationships
- The left-to-right ordering of the children of each parent
- The connection technique to be used
- Additional direct pointer options
- Any extended structure connections.

Defining Parent/Child Relationships

You use the PARENT operand of the SEGM statement of the child segment to define the parent/child relationship. You specify the name of the parent segment in the PARENT operand.

Example:

```
PARENT=SEGA
```

Defining the Left-to-Right Ordering of Children

The left-to-right order of occurrence of children segment types corresponds to the order in which their SEGM statements occur in the DBDGEN, with the first SEGM statement defining the leftmost, and the last the rightmost.

Defining the Connection Technique

The technique used to connect the parents and children is implicit in the access method used. The sequential method is specified by using HSAM or HISAM. Direct pointer connections are supported by the HD access methods. If the data base contains only one segment type, SHSAM or SHISAM should be used instead of HSAM or HISAM respectively (except that HISAM must be used instead of SHISAM when you use program isolation for the data base).

The access method chosen is specified by using the ACCESS operand of the DBD statement.

Example:

```
ACCESS=HD
```

Additional Direct Pointer Options

If one of the HD access methods has been selected, additional options controlling pointers can be specified.

Physical Child Last Pointer: The physical child last pointer is specified by adding a second parameter to the PARENT operand of the SEGM statement of the physical child for which you want the extra pointer. The parameter is coded as DBLE, indicating that both physical child first and physical child last pointers are to be included in the parent.

Example:

```
PARENT=((PARSEG,DBLE))
```

Physical Twin Backward Pointer: The physical twin backward pointer is specified by coding TWINBWD in the POINTER operand of the SEGM statement for the segment in which you want the pointer.

Example:

```
POINTER=TWINBWD
```

Suppression of Physical Twin Pointers: The generation of physical twin pointers can be suppressed by coding NOTWIN in the POINTER operand of the SEGM statement for the segment in which you want to suppress the pointers.

Example:

```
POINTER=NOTWIN
```

WARNING: This should be done *only* if only one occurrence of the segment type can exist for each occurrence of its parent.

Defining a Bidirectional Logical Relationship

In order to define a bidirectional logical relationship, the following items must be specified:

- The logical child segment
- The connections of the logical child to the logical and physical parents
- The logical child segment format
- Insert, replace, and delete rules
- Any additional pointer options.

Defining the Logical Child Segment

You use the SEGM statement in the DBD generation for your data base to define logical child segments to DL/I. For bidirectional relationships, two SEGM statements must be coded. One defines the logical child as a dependent of the physical parent. The other defines it as a dependent of the logical parent. The second definition is known as a *virtual logical child*.

Defining the Connections

The physical and logical parents in the relationship are identified by using the PARENT operands in the SEGM statements for the logical child and the virtual logical child. In addition, an LCHILD statement must be supplied for the logical parent segment to identify the logical child and the virtual logical child. Also, a SOURCE operand is coded in the SEGM statement of the virtual logical child to identify the logical child.

PARENT Operand for Logical Child: The parent operand for the logical child segment is coded as follows:

```
PARENT=((pparent,DBLE),(lparent,V,lpdbd))
```

where

pparent

is the segment name of the physical parent.

DBLE

serves the same purpose as for a normal segment, causing generation of a physical child last pointer in the physical parent, pointing to the logical child. This parameter is optional.

lparent

is the segment name of the logical parent.

lpdbd

is the name of the physical data base in which the logical parent resides. This operand may be omitted if the logical parent is in the same data base as the logical child. (The V is required for source compatibility with IMS.)

PARENT Operand for Virtual Logical Child: The parent operand for the virtual logical child segment is coded as follows:

```
PARENT=lparent
```

where

lparent

is the name of the logical parent segment.

The DBLE parameter is not valid for a virtual logical child definition. The equivalent function (specifying a logical child last pointer in the logical parent) is accomplished with the POINTER operand of the LCHILD statement (see "Defining Additional Pointer Options" below).

LCHILD Statement for the Logical Parent: An LCHILD statement must be supplied for each logical child, following the SEGM statement for the logical parent. For a bidirectional relationship, both the logical child and the virtual logical child must be identified.

LCHILD Specification of Logical Child: The logical child is identified by using the NAME operand, coded in the following manner:

```
NAME=lcname
```

or

```
NAME=(lcname,lcdbd)
```

where

lcname

is the segment name of the logical child

lcdbd

is the name of the data base the segment is located in if different from that of the logical parent.

LCHILD Specification of Virtual Logical Child: The virtual logical child is identified by using the PAIR operand of the LCHILD statement, as follows:

```
PAIR=vlcname
```

where

vlcname

is the segment name of the virtual logical child.

Virtual Logical Child Specification of Logical Child: The logical child is identified in the SEGM statement for the virtual logical child by using the SOURCE operand, coded in the following manner:

```
SOURCE=lcname
```

or

```
SOURCE=(lcname,D,lcdbd)
```

where

lcname

is the segment name of the logical child

lcdbd

is the name of the data base the segment is located in if different from that of the logical parent. (The D is required for source compatibility with IMS.)

Defining the Logical Child Segment Format

As stated previously, two SEGM statements are coded for the logical child.

Defining a Normal Logical Child: Logical child segments are defined in the same manner as normal segments. The two characteristics that must be defined are the segment name and the segment length.

Defining Segment Length: A logical child segment differs from a normal segment in that certain fields are required to be present in the segment. Each logical child segment must be defined as containing the concatenated key of its logical parent as its first set of fields.

If all of the fields that belong to the segment are defined, DL/I will calculate the length of the segment such that it will contain the fields. If all the fields are not specified, or if the length must be specified for some other reason, the BYTES operand of the SEGM statement can be used to define a specific length for a segment.

Example:

```
BYTES=30
```

Defining the Segment Name: The name of the segment is specified by using the NAME operand in the SEGM statement. The segment name cannot exceed eight characters in length, and it is suggested that it start with an alphabetic character and contain only alphameric characters.

Example:

```
NAME=LCNAME
```

Defining a Virtual Logical Child: Virtual logical child definitions are based on the logical child definition.

Defining Segment Length: The segment length will be calculated by DL/I. It will be the length of the physical parent's concatenated key plus the length of any intersection data.

Defining the Segment Name: The name of the segment is specified by using the NAME operand in the SEGM statement. The segment name cannot exceed eight characters in length, and it is suggested that it start with an alphabetic character and contain only alphameric characters.

Example:

```
NAME=VLCNAME
```

Defining the Rules

The insert, delete, and replace rules for the physical parent, logical parent, and logical child are specified by using the RULES operand in their respective SEGM statements. Valid rules specifications are P, L, or V, with the exception of the replace rule for the logical child, where V is the only valid value.

The format of the rules operand is:

```
RULES=idr
```

where

idr

represents the insert, delete, and replace rules.

Example:

```
RULES=PLV
```

Defining Additional Pointer Options

Physical Twin Pointer Options: The POINTER operand in the SEGM statement can be used for a logical child segment the same as for a normal segment, to control the generation of physical twin pointers; but if used for a logical child in a bidirectional relationship, a second parameter (LTWIN) must be added so that the logical twin pointer will be generated.

Example:

```
POINTER=(TWINBWD,LTWIN)
```

Logical Child Last Pointer: The logical child last pointer is specified by using the POINTER operand in the LCHILD statement for the logical parent.

Example:

```
POINTER=DBLE
```

Logical Twin Backward Pointer: The logical twin backward pointer is specified by using the POINTER operand in the SEGM statement for the logical child. If the logical twin pointer option is specified, the physical twin pointer option must be explicitly defined as either TWIN (twin forward only), TWINBWD (twin forward and backward), or NOTWIN (no physical twin pointers).

Example:

```
POINTER=(TWIN,LTWINBWD)
```

Suppression of Logical Twin Pointers

The generation of logical twin pointers can be suppressed by explicitly specifying the physical twin pointer option in the POINTER operand of the SEGM statement for the segment in which the pointer is to be suppressed, as in any of the following:

Examples:

POINTER=NOTWIN
POINTER=TWIN
POINTER=TWINBWD

WARNING: This should be done *only* if only one occurrence of the segment type can exist for each occurrence of its logical parent.

Example of Bidirectional Logical Relationships

The following is an example of specifying a bidirectional logical relationship when the logical parent is in another data base:

```
In Logical Child Data Base

Physical -- SEGM NAME=PPSEG
Parent
Logical  -- SEGM NAME=LCSEG,PARENT=((PPSEG),(LPSEG,V,LPDBD))
Child

In Logical Parent Data Base

Logical -- SEGM NAME=LPSEG
Parent        LCHILD NAME=(LCSEG,LCDBD),PAIR=VLCSEG
Virtual -- SEGM NAME=VLCSEG,SOURCE=(LCSEG,D,LCDBD),POINTER=PAIRED
Logical
Child
```

Defining a Unidirectional Logical Relationship

In order to define a unidirectional logical relationship, the following items must be specified:

- The logical child segment
- The connection of the logical child to the logical and physical parents
- The logical child segment format
- Insert, replace, and delete rules.

Defining the Logical Child Segment

logical child segment, defining the

You use the SEGM statement in the DBD generation for your data base to define logical child segments to DL/I. The SEGM statement defines the logical child as a dependent of the physical parent.

Defining the Connections

The physical and logical parents in the relationship are identified by using the PARENT operand of the SEGM statement for the logical child. In addition, an LCHILD statement must be supplied for the logical parent segment, identifying the logical child.

PARENT Operand for Logical Child: The PARENT operand for the logical child segment is coded as follows:

```
PARENT=((pparent,DBLE),(lparent,V,lpdbd))
```

where

pparent

is the segment name of the physical parent.

DBLE

serves the same purpose as it does for a normal segment, causing generation of a physical child last pointer in the physical parent, pointing to the logical child. This parameter is optional.

lparent

is the segment name of the logical parent.

lpdbd

is the name of the physical data base in which the logical parent resides. This operand may be omitted if the logical parent is in the same data base as the logical child. (The V is required for source compatibility with IMS.)

LCHILD Statement for the Logical Parent: An LCHILD statement must be supplied after the SEGM statement for the logical parent for each logical child. It is used to identify the logical child and to define the relationship as unidirectional.

LCHILD Specification of Logical Child: The logical child is identified by using the NAME operand, coded in the following manner:

```
NAME=lcname
```

or

```
NAME=(lcname,lcdbd)
```

where

lcname

is the segment name of the logical child.

lcdbd

is the name of the data base the segment is located in if different from that of the logical parent.

LCHILD Specification of Unidirectional Relationship: The logical relationship is specified as being unidirectional by using the POINTER operand of the LCHILD statement as follows:

```
POINTER=NONE
```

Defining the Logical Child Segment Format

Logical child segments are defined in the same way as normal segments. The two characteristics that must be defined are the segment name and the segment length.

Defining Segment Length: A logical child segment differs from a normal segment in that certain fields are required to be present in the segment. Each logical child segment must be defined as containing the concatenated key of its logical parent as its first set of fields.

If all of the fields that belong to the segment are defined, DL/I will calculate the length of the segment such that it will contain the fields. If all the fields are not specified, or if the length must be specified for some other reason, the BYTES operand in the SEGM statement can be used to define a specific length for a segment.

Example:

```
BYTES=30
```

Defining the Segment Name: The name of the segment is specified by using the NAME operand in the SEGM statement. The segment name cannot exceed eight characters in length, and it is suggested that it start with an alphabetic character and contain only alphameric characters.

Example:

```
NAME=LCNAME
```

Physical Twin Pointer Options: The POINTER operand in the SEGM statement can be used for a logical child segment the same as for a normal segment, to control the generation of physical twin pointers.

Example:

```
POINTER=TWINBWD
```

Defining the Rules for Unidirectional Relationships

The insert, delete, and replace rules for the logical parent are specified by using the RULES operand in its SEGM statement. Valid rules specifications are P, L, or V.

The format of the rules parameter is:

```
RULES=idr
```

where

idr

represents the insert, delete, and replace rules.

Example:

```
RULES=PLV
```

Example of Unidirectional Logical Relationships

The following is an example of specifying a unidirectional logical relationship when the logical parent is in another data base:

```
In Logical Child Data Base

Physical -- SEGM NAME=PPSEG
Parent
Logical  -- SEGM NAME=LCSEG,PARENT=((PPSEG),(LPSEG,V,LPDBD))
Child

In Logical Parent Data Base

Logical -- SEGM NAME=LPSEG
Parent        LCHILD NAME=(LCSEG,LCDBD)
```

Defining a Logical Data Base

A logical data base is a hierarchical data base derived from one or more physical data bases that contain logical relationships.

You identify the data base definition for a logical data base by specifying ACCESS=LOGICAL in the DBD statement. In addition, you *don't* code a DATASET statement. It is not needed because the logical data base is derived from data bases that already have their own data sets.

Defining Fields for a Logical Data Base

Fields must not be defined in a logical data base definition. Fields to be referenced in a logical data base must have been defined as fields in the physical data base. A field in a logical data base must be the same type, same length, be located in the same position, and have the same name as the field on which it is based.

Defining Segments for a Logical Data Base

You use the SEGM statement in the DBD generation to define segments to DL/I. Each SEGM statement defines one segment. These characteristics of each segment must be specified: the name by which the segment will be referenced, the segment or segments on which that segment is based, and the data base in which those segments are found. (The format of a segment in a logical data base must be the same as the format in the physical data base of the segment (or segments) on which it is based.)

Defining the Segment Name: The name of the segment is specified by using the NAME operand in the SEGM statement. The segment name cannot exceed eight characters in length, and it is suggested that it start with an alphabetic character and contain only alphameric characters.

Example:

```
NAME=SEGNAME
```

Defining Base Segments: You use the SOURCE operand in the SEGM statement to define the segments, and the data bases in which they are found, on which a segment in a logical data base is based. A simple segment is based on a single segment from a physical data base, while a concatenated segment is based on a logical child segment and its logical or physical parent.

Defining the Base Segment for a Simple Segment: The base segment for a simple segment in a logical data base is specified by using the SOURCE operand in its SEGM statement as follows:

```
SOURCE=(segname,dbname)
```

where

segname

is the name of the segment in the physical data base

dbname

is the name of the physical data base.

Defining the Base Segment for a Concatenated Segment: The base segment for a concatenated segment in a logical data base is specified by using the SOURCE operand in its SEGM statement as follows:

```
SOURCE=((lcseg,lcdbn),(lpseg,lpdbn))
```

where

lcseg

is the name of the logical child segment in its physical data base

lcdbn

is the name of its physical data base

lpseg

is the name of the logical parent in its physical data base

lpdbn

is the name of its physical data base.

or

```
SOURCE=((vlcseg,vlcdbn),(ppseg,ppdbn))
```

where

vlcseg

is the name of the virtual logical child segment in its physical data base

vlcdbn

is the name of its physical data base

ppseg

is the name of the physical parent in its physical data base

ppdbn

is the name of its physical data base.

Defining Structures for a Logical Data Base

In defining the structure of the logical data base to DL/I, the following must be specified:

- The parent/child relationships
- The left-to-right ordering of children for a parent.

Defining Parent/Child Relationships: The parent/child relationship is defined by specifying the name of the parent segment in the PARENT operand in the SEGM statement for the child segment.

Example:

```
PARENT=SEGA
```

Defining the Left-to-Right Ordering of Children: The left-to-right order of occurrence of children segment types corresponds to the order in which their SEGM statements occur in the DBDGEN, with the first SEGM defining the leftmost, and the last the rightmost. This need not correspond to the ordering in the underlying physical data base.

Defining Sequencing for Data Base Records

Sequencing by Order of Creation

Sequencing by order of creation is specified by using the second parameter of the RULES operand of the SEGM statement for the root segment. The parameter LAST defines chronological sequencing, and FIRST specifies reverse chronological sequencing. The operand is coded as follows:

```
RULES=(,FIRST)
```

or

```
RULES=(idr,LAST)
```

where

idr

represents the insert, delete, and replace rules.

for segments involved in a logical relationship.

Sequencing Under Application Control

Sequencing of data base records under control of the creating application program is specified by using the second parameter of the RULES operand of the SEGM statement for the root segment, as follows:

```
RULES=(,HERE)
```

or

```
RULES=(idr,HERE)
```

where

idr

represents the insert, delete, and replace rules.

for segments involved in a logical relationship.

Sequencing by Field Value

Sequencing by Field Value for HD: The primary access for an HD data base is specified by the first ACCESS statement supplied for the root segment. The field to be used to control sequencing is identified by using the SEQFLD parameter on that ACCESS statement.

Example:

```
SEQFLD=FLDNAME
```

For HD randomized access, a sequence field that can have non-unique values must be identified by specifying SEQVAL=DUPLICATE in the ACCESS statement.

For HD indexed access, the name of the VSAM data set and the generated data base for the index must be specified by using the REF operand.

Example:

```
REF=IDBNAME
```

The name cannot exceed seven characters in length, must start with an alphabetic character, contain only alphameric characters, and must not use the @ character. It must be unique for your system, as it is used as the name of a module in your core image library.

For HD randomized, the RMRTN operand is used to specify the name of your randomizing routine.

Sequencing by Field Value for Non-HD (Including HIDAM and HDAM): The field to be used to control sequencing of non-HD data base records is identified by using the NAME operand in the FIELD statement for the field.

Example:

```
NAME=(FLDNAME,SEQ)
```

For HSAM, SHSAM, or HDAM, a sequence field that can have non-unique values must be identified by specifying a third parameter for the NAME operand.

Example:

```
NAME=(FLDNAME,SEQ,M)
```

If a sequence field has non-unique values, the sequence of records having identical sequence fields can be controlled by the application program creating the entries or be specified as chronological (default) or reverse chronological order (see above).

Defining Indexes

HISAM and SHISAM Indexes

HISAM and SHISAM are supported by VSAM key sequenced data sets (KSDS), for which VSAM builds and maintains the indexes. There are no DL/I definitional requirements for these indexes.

HD Primary Index

The primary index for an HD indexed data base is defined in the first ACCESS statement for the root segment. The information required is:

- The name of the VSAM data set and generated data base for the index
- The name of the root segment
- The name of the field in the root segment to be used as the sequence field.

The VSAM data set and index data base name are specified in the REF operand.

Example:

```
REF=IDBNAME
```

The root segment name is specified in the SEGM operand.

Example:

```
SEGM=RSEGM
```

The key field is identified in the SEQFLD operand.

Example:

```
SEQFLD=FLDNAME
```

HD Secondary Index

Secondary indexes for HD data bases are defined by using the ACCESS statement, one for each index to be defined. The following information is needed:

- The name of the segment to be accessed through the index (the target segment)
- The name of the field or fields to be used as the key
- The name of the segment containing the key field(s) if other than the target segment (the sequence segment)
- Whether or not non-unique values may exist for the key
- The name to be used for the key field in references using the index, and the VSAM data set and generated data base for the index
- An index suppression value, if required
- The name of an index suppression routine, if required.

The target segment is identified by using the SEGM operand:

Example:

```
SEGM=TSEGM
```

The sequence field or fields (up to five) are specified with the SEQFLD operand.

Examples:

```
SEQFLD=SFLD
SEQFLD=(SFLD1,SFLD2,...)
```

The sequence segment, if other than the target segment, is designated in the SEQSEG operand.

Example:

```
SEQSEG=SSEGM
```

If non-unique values can exist for the key field, this fact is specified by using the SEQVAL operand.

Example:

```
SEQVAL=DUPLICATE
```

The key field, VSAM data set, and index data base name are specified with the REF operand.

Example:

```
REF=KDDNAME
```

The index suppression value is supplied by using the SUPVAL operand.

Example:

```
SUPVAL=0
```

The name of the index suppression routine is specified in the SUPRTN operand.

Example:

```
SUPRTN=RTNAME
```

Defining Indexes for HIDAM and HDAM

Index definition for HDAM and HIDAM requires the explicit definition of the index data base as well as the identification of the index in the primary data base.

Definition of the Index Data Base

The index data base is defined in the same manner as any other data base, using the same statements. The following information must be supplied:

- Specification of index data base type
- The VSAM data set name
- The physical device type on which the data set will reside
- Optionally, the VSAM logical record size and blocking factor
- The name and length of the root segment (only one segment is allowed)
- The name, length, and type (optional) of the key (sequence) field in the index root segment
- Any additional fields in the root segment
- The name of the target segment
- The name of the target segment's data base
- The name of the key field in the primary data base.

The data base is defined as being an index data base by specifying ACCESS=INDEX in the DBD statement.

The VSAM data set name is identified by using the DD1 operand in the DATASET statement.

Example:

```
DD1=VSDSNAME
```

The device type is specified by using the DEVICE operand in the DATASET statement.

Example:

```
DEVICE=3350
```

The VSAM logical record size and blocking factor are specified by using the RECORD and BLOCK operands in the DATASET statement.

Example:

```
RECORD=28,BLOCK=12
```

Either or both of these operands may be omitted.

The root segment is defined by using the SEGM statement. The name is supplied in the NAME operand. The length can be specified in the BYTES operand, or it can be omitted and DL/I will calculate the length such that it will contain all of the defined fields.

Example:

```
NAME=IRSNAME,BYTES=24
```

The key (sequence) field is defined as the first field in the segment by using the FIELD statement. The name is defined by using the NAME operand.

Example:

```
NAME=(KFNAME,SEQ)
```

The length of the field is specified by using the BYTES operand. For a primary index, this length must be equal to the length of the sequence field for the root segment in the primary data base. For a secondary index, the length equals the sum of the lengths of the fields identified as either SRCH or SUBSEQ fields. (A /SX field is four bytes long.)

Example:

```
BYTES=12
```

The field type can be specified by using the TYPE operand. If omitted, DL/I will assume that the type is character (C).

Example:

```
TYPE=X
```

Additional fields representing duplicate data (DDATA) or application maintained fields in the index root segment can be defined by using FIELD statements following the one for the key field.

The name of the target segment and its data base are both specified by using the NAME operand in an LCHILD statement supplied after the SEGM statement, as follows:

```
NAME=(rsgname,pdbname)
```

where

rsgname

is the name of the root segment in the primary data base

pdbname

is the name of the primary data base.

The name of the field in the primary data base to be used as the key field is specified with the INDEX operand in the same LCHILD statement. For a primary index, this is the name of the sequence field in the root segment. For a secondary index, it corresponds to the name supplied in the NAME operand in the XDFLD statement in the primary data base.

Example:

```
INDEX=PKFIELD
```

Primary Index Specification in the HIDAM Data Base

The primary index for a HIDAM data base is identified by using the first LCHILD statement for the root segment. The information required is:

- The name of the index data base to be defined
- The name of the root segment in the index data base
- The field in the root segment in the primary data base to be used as the key.

The index data base name and index root segment names are supplied by using the NAME operand in the LCHILD statement, as follows:

```
NAME=(rsegname,idbname)
```

where

rsegname

is the name of the root segment in the index data base

idbname

is the name of the index data base.

The field in the root segment to be used as the key for the primary data base is identified by using the NAME operand in its FIELD statement:

```
NAME=(fldname,SEQ)
```

HIDAM/HDAM Secondary Index

Secondary indexes for HIDAM or HDAM data bases are defined by using the LCHILD and XDFLD statements, one pair for each index to be defined. The following information must be provided:

- The segment to be accessed through the index (the target segment)
- The name of the field or fields to be used as the key
- The name of the segment containing the key field(s) (the source segment), if other than the target segment.
- The names of up to five fields to be appended to the key to ensure uniqueness
- The names of up to five fields from the primary data base to be duplicated in the index data base root segment
- The name to be used for the key field in references using the index
- The name of the index data base
- The name of the root segment in the index data base
- An index suppression value, if required
- The name of an index suppression routine, if required.

The target segment is specified by placing the LCHILD/XDFLD pair after its SEGM statement and before the next SEGM statement.

The sequence field or fields (up to five) are specified with the SRCH operand in the XDFLD statement.

Examples:

```
SRCH=SFLD
SRCH=(SFLD1,SFLD2,...)
```

The source segment, if other than the target segment, is designated with the SEGMENT operand in the XDFLD statement.

Example:

```
SEGMENT=SSEGM
```

If non-unique values can exist for the key field, up to five fields from the source segment, its concatenated key, or its RBA can be appended to the key to ensure uniqueness. These fields are selected with the SUBSEQ operand in the XDFLD statement.

Examples:

```
SUBSEQ=SSFLD
SUBSEQ=(SSFLD1,/CK01,/SXAB,...)
```

(If the RBA is to be used, a special /SX field must be defined for the source segment and, if a portion of the concatenated key is required, a /CK field must be defined. See below for a discussion of how this is done.)

Up to five fields from the source segment or its concatenated key can be copied into the index root segment. These fields are selected with the DDATA operand in the XDFLD statement.

Examples:

```
DDATA=SSFLD
DDATA=(SSFLD1,/CK01,....)
```

(If a portion of the concatenated key is required, a /CK field must be defined. See below for a discussion of how this is done.)

The name to be used for the key field in references using the index is specified with the NAME operand in the XDFLD statement.

Example:

```
NAME=REFNAME
```

The name of the index data base and the name of the root segment in the index data base are specified by using the NAME operand in the LCHILD statement, as follows:

```
NAME=(irsname,idbname)
```

where

irsname

is the name of the root segment in the index data base

idbname

is the name of the index data base.

The index suppression value is supplied by using the NULLVAL operand in the XDFLD statement.

Example:

```
NULLVAL=0
```

The name of the index suppression routine is specified with the EXTRTN operand in the XDFLD statement.

Example:

```
EXTRTN=RTNAME
```

One additional operand must be coded for the LCHILD statement to distinguish it from an LCHILD for a logical child. The POINTER operand must be specified as follows:

```
POINTER=INDX
```

Defining /SX Field Types: In order to use the RBA of the source segment as a qualifier for the key, a special /SX field must be defined for the source segment. This field type is identified by the characters /SX as the first three characters of its name.

Example:

```
NAME=/SXAB
```

No other operands are allowed on the FIELD statement for a /SX field.

Defining /CK Field Types: In order to use a part of the concatenated key of the source segment as a qualifier for the key, or to have it duplicated in the index data base root segment, a special /CK field must be defined for the source segment.

The following information must be provided:

- The name by which it may be referenced in the SUBSEQ or DDATA operands in the XDFLD statement
- The part of the concatenated key referred to.

The name is specified by using the NAME operand in the FIELD statement.

Example:

```
NAME=/CK01
```

The first three characters of the name must be /CK.

The START and BYTES operands are used to specify the part of the concatenated key referred to. The START operand identifies the starting location within the concatenated key.

Example:

```
START=12
```

(The first position is location 1.)

The BYTES operand specifies the length of the field.

Example:

```
BYTES=6
```

Defining Sequencing for Dependent Segments

Sequencing by Order of Creation

Sequencing by order of creation is specified by using the second parameter of the RULES operand in the SEGM statement for the dependent segment. The LAST parameter defines chronological sequencing, and FIRST specifies reverse chronological sequencing. The operand is coded as follows:

```
RULES=(,FIRST)
```

or

```
RULES=(idr,LAST)
```

where

idr

represents the insert, delete, and replace rules.

for segments involved in a logical relationship.

Sequencing Under Application Control

Sequencing of dependent segments under control of the creating application program is specified by using the second parameter of the RULES operand of the SEGM statement for the dependent segment, as follows:

```
RULES=(,HERE)
```

or

```
RULES=(idr,HERE)
```

where

idr

represents the insert, delete, and replace rules.

for segments involved in a logical relationship.

Sequencing by Field Value

The field to be used to control sequencing of a dependent segment is identified by using the NAME operand in the FIELD statement for the field.

Example:

```
NAME=(FLDNAME,SEQ)
```

A sequence field that can have non-unique values must be identified by specifying a third parameter for the NAME operand.

Example:

```
NAME=(FLDNAME,SEQ,M)
```

If a sequence field has non-unique values, the sequence of dependents having identical sequence fields can be controlled by the application program creating the entries, or be specified as chronological (default) or reverse chronological order (see above).

Defining Sequencing for Logical Relationships

Defining sequencing for logical relationships involves defining the sequence of the logical parent as seen as a dependent of the physical parent and, for bidirectional relationships, the sequence of the physical parent as seen as a dependent of the logical parent.

Defining the Sequence of Logical Parents

Sequencing by Order of Creation: Sequencing by order of creation is specified by using the second parameter of the RULES operand in the SEGM statement for the logical child segment. The LAST parameter defines chronological sequencing and FIRST specifies reverse chronological sequencing. The operand is coded as follows:

```
RULES=(,FIRST)
```

or

```
RULES=(idr,LAST)
```

where

idr

represents the insert, delete, and replace rules.

for segments involved in a bidirectional logical relationship.

Sequencing Under Application Control: Sequencing of logical parent segments under control of the creating application program is specified by using the second parameter of the RULES operand of the SEGM statement for the logical child, as follows:

```
RULES=(,HERE)
```

or

```
RULES=(idr,HERE)
```

where

idr

represents the insert, delete, and replace rules.

for segments involved in a bidirectional logical relationship.

Sequencing by Field Value: The field to be used to control sequencing of logical parents is identified by using the NAME operand in the FIELD statement for the field.

Example:

```
NAME=(FLDNAME,SEQ)
```

This field must be located in the intersection data part of the logical child (not in the concatenated key area).

A sequence field that can have non-unique values must be identified by specifying a third parameter for the NAME operand.

Example:

```
NAME=(FLDNAME,SEQ,M)
```

If a sequence field has non-unique values, the sequence of logical parents having identical sequence fields can be controlled by the application program creating the entries, or be specified as chronological (default) or reverse chronological order (see above).

Defining the Sequence of Physical Parents (Bidirectional Only)

Sequencing by Order of Creation: Sequencing by order of creation is specified by using the RULES operand of the LCHILD statement for the logical child segment. This statement follows the SEGM statement for the logical parent. The LAST parameter defines chronological sequencing and FIRST specifies reverse chronological sequencing.

Example:

```
RULES=FIRST
```

Sequencing Under Application Control: Sequencing of physical parent segments under control of the creating application program is specified by using the RULES operand in the LCHILD statement for the logical child.

Example:

```
RULES=HERE
```

This statement follows the SEGM statement for the logical parent.

Sequencing by Field Value: The fields to be used to control sequencing of physical parents are identified by using the NAME operand in the FIELD statements for the fields.

Example:

```
NAME=(FLDNAME,SEQ)
```

These fields must be defined in the virtual logical child and may be part of the concatenated key or the intersection data. Up to five fields in the virtual logical child may be identified as sequence fields. The fields will be concatenated to form the sequence field, with the first defined being the leftmost.

A sequence field that can have non-unique values must be identified by specifying a third parameter in the NAME operand.

Example:

```
NAME=(FLDNAME,SEQ,M)
```

If a sequence field has non-unique values, the sequence of logical parents having identical sequence fields can be controlled by the application program creating the entries, or be specified as chronological (default) or reverse chronological order (see above).

Defining the Application View

An application view is specified by generating a Program Specification Block (PSB). A PSB consists of one or more Program Communication Blocks (PCBs), which define views of individual data bases.

In defining an application's view of a data base, the following information is needed:

- The name of the data base on which the view is based
- The longest concatenated key length in the view
- An indication of whether or not multiple positioning is to be used
- If secondary indexing is to be used, the reference name (index data base name) for the index
- The segments to appear in the view and, optionally, their contents
- The organization of the segments in the application view
- The type of processing to be done for each segment.

Specifying Base Data Base

The physical or logical data base on which the view is based is defined with the DBDNAME operand in the PCB statement.

Example:

```
DBDNAME=BASNAME
```

Specifying the Longest Concatenated Key Length

The KEYLEN operand in the PCB statement is used to define the longest concatenated key length of any segment in the view. When using secondary indexes, if the search field for the index is longer than the sequence field of the target segment, the search field length should be used in place of the sequence field length in the concatenated key length calculation.

Example:

```
KEYLEN=100
```

Specifying Multiple Positioning

If multiple positioning is required, it is defined for a view by specifying POS=MULTIPLE in the PCB statement.

Specifying Secondary Index Usage

If the application view of the data base is using access through a secondary index, the PROCSEQ operand must be specified in the PCB statement to identify the index:

Example:

```
PROCSEQ=INDXNM
```

For HD, this is the REF name for the index. For HDAM or HIDAM, it is the index data base name.

Specifying Segments

The segments that appear in an application view are specified by using SENSEG statements. The NAME operand is used to specify the segment name.

Example:

```
NAME=BSEGNAME
```

Specifying Segment Contents

The default application view of a segment is identical to the view in the base data base. The SENFLD and VIRFLD statements can be used to re-define the view of the segment. SENFLD statements are used to present alternate views of fields that exist in the base data base, while VIRFLD statements define fields that do not exist there.

Note that if any fields are defined for a segment by using SENFLDs or VIRFLDs, a SENFLD must be supplied for *every* field in the base segment that the application wishes to see.

Redefining Existing Fields: Each field is identified to DL/I with a SENFLD statement. Four characteristics of the field are required:

- The name of the field in the base data base
- The field type
- The length of the field
- The location of the field in this view of the segment.

In addition, you can specify the name of a routine to be entered whenever the field is retrieved or stored.

Defining the Field Name: The name of the field in the base data base is specified by using the NAME operand in the SENFLD statement.

Example:

```
NAME=NAMEFLD
```

(Fields in an application view may not be based on /CK or /SX special field types from HIDAM or HDAM data bases.)

Defining Field Type: The type of data to be stored in the field is specified by using the TYPE operand in the SENFLD statement. The possible types are:

```
'X' hexadecimal
'H' halfword binary (implied length of two bytes)
'F' fullword binary (implied length of four bytes)
'P' packed decimal
'Z' zoned decimal
'C' character
'E' floating point (short, length of four bytes)
'D' floating point (long, length of eight bytes)
'L' floating point (extended, length of sixteen bytes)
```

If the data type is not specified, it will default to the type specified in the base data base.

Example:

```
TYPE=F
```

Defining Field Length: The length of the field may be either implicitly defined, based on the field type, or explicitly defined by using the BYTES operand in the SENFLD statement. The length can be specified to override the implied lengths for field types H or F. It can not be specified for field types E, D, or L. If omitted for types X, P, C, or Z, the length of the field in the base data base will be used.

Example:

```
BYTES=3
```

Defining Field Location Within a Segment: Unless otherwise specified, the first field specified for a segment is assumed to be located at the start of the segment, and each field defined subsequent to the first is assumed to start at the end of the previous one. The START operand in the SENFLD statement can be used to override this default and specify a specific location for a field.

The starting location for a field can be specified either as a number, indicating the location within the segment at which the field is to start (the first position is location 1); or the name of a field already defined for this segment, indicating that this field is to start at the same location as the specified field.

Examples:

```
START=20
START=NAMEFLD
```

Defining a Field Exit Routine: The name of a module, located in your core image library, to be entered whenever the field is retrieved or stored can be specified by using the RTNAME operand in the SENFLD statement.

Example:

```
RTNAME=FERNAME
```

This operand is optional.

Defining Virtual Fields: Each field is identified to DL/I with a VIRFLD statement. These characteristics of the field need to be identified to DL/I:

- The name of the field in the base data base
- The field type
- The length of the field
- The location of the field in this view of the segment
- An initial value for the field.

In addition, you can specify the name of a routine to be entered whenever the field is retrieved or stored.

Defining the Field Name: The name of the field is specified by using the NAME operand in the VIRFLD statement.

Example:

```
NAME=NAMEFLD
```

Defining Field Type: The type of data to be stored in the field is specified by using the TYPE operand in the VIRFLD statement. The possible types are:

```
'X' hexadecimal
'H' halfword binary (implied length of two bytes)
'F' fullword binary (implied length of four bytes)
'P' packed decimal
'Z' zoned decimal
'C' character
'E' floating point (short, length of four bytes)
'D' floating point (long, length of eight bytes)
'L' floating point (extended, length of sixteen bytes)
```

If the data type is not specified, it will default to type 'C'.

Example:

```
TYPE=F
```

Defining Field Length: The length of the field may be either implicitly defined, based on the field type, or explicitly defined by using the BYTES operand in the VIRFLD statement. The length must be specified for field types X, P, C, or Z. It may be specified to override the implied lengths for field types H or F. It can not be specified for field types E, D, or L.

Example:

```
BYTES=3
```

Defining Field Location Within a Segment: Unless otherwise specified, the first field specified for a segment is assumed to be located at the start of the segment, and each field defined subsequent to the first is assumed to start at the end of the previous one. The START operand in the VIRFLD statement can be used to override this default and specify a specific location for a field.

The starting location for a field may be specified either as a number, indicating the location within the segment at which the field is to start (the first position is location 1); or the name of a field already defined for this segment, indicating that this field is to start at the same location as the specified field.

Examples:

```
START=20
START=NAMEFLD
```

Defining an Initial Value: A value to be stored in a virtual field whenever its segment is retrieved can be specified by using the VALUE operand in the VIRFLD statement.

Example:

```
VALUE=C'INITIAL'
```

This operand is optional.

Defining a Field Exit Routine: The name of a module, located in your core image library, to be entered whenever the field is retrieved or stored can be specified by using the RTNAME operand in the SENFLD statement.

Example:

```
RTNAME=FERNAME
```

This operand is optional.

Defining Segment Organization

In defining the application view of the structure of the data base to DL/I, you need to specify:

- The parent/child relationships
- The left-to-right ordering of children for a parent.

Defining Parent/Child Relationships: The parent/child relationship is defined by specifying the name of the parent segment in the PARENT operand in the SENSEG statement for the child segment.

Example:

```
PARENT=SEGA
```

Defining the Left-to-Right Ordering of Children: The left-to-right order of occurrence of children segment types will correspond to the order in which their SENSEG statements occur in the PSBGEN, with the first SENSEG statement defining the leftmost, and the last SENSEG statement the rightmost.

Note that the left-to-right order of segment types in an application view of an HSAM or HISAM data base must be the same as their order in the base data base.

Defining Segment Processing Options

The type of processing operations allowed against a segment or field type can be specified for an application view.

Defining Process Controls for Fields: The REPLACE operand on the SENFLD statement can be used to prohibit an application program from changing a field in a segment, while allowing it to change other fields in the same segment.

Example:

```
REPLACE=NO
```

Note that if any SENFLDs are supplied, one must be supplied for every field in the base segment that the application expects to see.

Defining Process Controls for Segments: The PROCOPT operand in the PCB and SENSEG statements is used to control the types of operations an application program is allowed to perform on a segment basis.

The PROCOPT operand in the PCB statement is used to specify the default option to be used for all segments whose SENSEGs don't explicitly state a processing option. If the PROCOPT operand is not specified on the PCB statement, the default will be R, or read-only.

The PROCOPT operand in the SENSEG statement restricts the type of operations the application program can perform against a segment.

The format of the operand in either case is:

```
PROCOPT=opqr
```

where

opqr

represents the fact that up to four options can be specified.

Additional Specifications

In addition to defining the views of each data base, two additional pieces of information must be provided for the PSB:

- The name by which it can be referenced
- The programming language in which the application program is written.

The PSB name is specified by using the NAME operand in the PSBGEN statement.

Example:

```
NAME=PSBNAME
```

The programming language of the application program is specified in the LANG operand in the PSBGEN statement.

Example:

```
LANG=PL/I
```

Data Formats

Variable Length Segments

If the access method you chose for your data base is HSAM, SHSAM, HISAM, or SHISAM you can ignore this section.

You use the SEGM statement in the DBD generation for your data base to specify that a particular segment type has variable length. You do this by coding the BYTES operand. There are two parameters in the BYTES operand:

max-var-length

is the maximum length of the data portion of the segment, including the two-byte length field

min-var-length

is the minimum length of the data portion of the segment, including the two-byte length field.

Segment Edit/Compression

If the access method you chose for your data base is HSAM, SHSAM, HISAM, or SHISAM you can ignore this section.

You use the SEGM statement in the DBD generation for your data base to specify that a particular segment type is to be processed by a segment edit/compression routine. You do this by coding the COMPRTN operand in the statement. A parameter in the operand names the compression routine. Another, optional, COMPRTN parameter (INIT) indicates that the routine requires initialization and termination processing control.

Segment Sensitivity

You use the SENSEG statement in the PSB generation for your application program to specify that the program is sensitive to a particular segment. There must be one SENSEG statement for each segment the program is sensitive to. The SENSEG statement has three possible operands:

NAME

The name of this segment.

PARENT

The name of the parent segment of this segment. If this segment is a root segment, this operand must be omitted or specified as PARENT=0.

PROCOPT

The processing options (optional) for this segment. See the section "Processing Options" below.

Field Level Sensitivity

Sensitive Fields: You use the SENFLD statement in the PSB generation for your application program to specify that the program is sensitive to a particular field in a particular segment. If you don't specify any SENFLD statements for a particular segment, the program is sensitive to all fields in that segment. If you specify one or more SENFLD statements for fields in the segment, the program is sensitive to only those fields. The SENFLD statement has six possible operands:

NAME

The name of this field.

BYTES

The length (in bytes) of this field (optional).

START

The starting position of this field in the segment (optional).

TYPE

The type of data to be contained in this field (optional).

RTNAME

The name of a user-written field exit routine (optional).

REPLACE

Specifies whether or not the application program is allowed to modify this field (optional). If omitted, YES is assumed.

Virtual Fields: You use the VIRFLD statement in the PSB generation for your application program to specify a virtual field for a particular segment. The VIRFLD statement has six possible operands:

NAME

The name of the virtual field.

BYTES

The length (in bytes) of the field (optional).

START

The starting position of the field in the segment (optional).

TYPE

The type of data to be contained in the field (optional).

VALUE

An initial value for the field (optional).

RTNAME

The name of a user-written field exit routine (optional).

Processing Controls

Processing Options

You specify processing options in the PSB generation for your application program in either, or both, of two ways:

1. In the PCB statement, by using the PROCOPT operand
2. In the SENSEG statement, by using the PROCOPT operand.

PROCOPTs specified in the PCB statement apply to all the segments that your program is sensitive to in this data base. PROCOPTs specified in the SENSEG statement apply to that segment alone. The PROCOPTs in the SENSEG statement over-ride PROCOPTs in the PCB statement.

Specifying DL/I Operational Capabilities

Online Operation

The following sections describe the tasks that need to be done to make DL/I online operation possible. Before going on, you should be familiar with CICS/VS concepts and facilities. The appropriate publications are listed in the preface of this manual.

Examples of coding the various CICS/VS-DL/I macros for the tables described in the following sections are given in *DL/I DOS/VS Resource Definition and Utilities*

CICS/VS System Generation Requirements and Options

Before the DL/I system can be executed in the CICS/VS environment, you must redefine certain CICS/VS components to allow the addition of DL/I. Those components are discussed individually in the following sections.

CICS/VS System Generation (DFHSG): The DFHSG macro tailors the CICS/VS system components to support the CICS/VS services you want. To include the DL/I-related services in the CICS/VS system, you have to code

```
DL1=YES
```

in the DFHSG TYPE=INITIAL macro statement.

System Initialization Table (SIT): The SIT supplies the CICS/VS system initialization program with the information necessary to initialize the CICS/VS system to the unique environment you request. You can generate several SITs and select the appropriate one at initialization time. You can dynamically change some of the parameters during initialization, if required.

The SIT is created by assembling and link-editing the DFHSIT macro. By coding the DL1 operand, you can cause one of three actions:

- Module DLZNUC is loaded during CICS/VS system initialization
- Module DLZNUC with a two-character suffix supplied by you is loaded during CICS/VS system initialization
- Loading of DL/I is not done at CICS/VS initialization.

Journal Control Table (JCT): The JCT describes your journal data sets to CICS/VS (optional). There must be one entry in the table to define the CICS/VS system journal. If you use program isolation, the DL/I log must be assigned to the CICS/VS system journal.

File Control Table (FCT): The file control table describes to CICS/VS any of your files that are processed by CICS/VS file management. To generate entries in the table and to request services related to the file, you use the DFHFCT TYPE=DATASET macro statement and its DATASET, ACCMETH, and OPEN operands.

A DFHFCT entry must be made for every physical DBD that can be accessed in the CICS/VS-DL/I online environment.

This means that all of the following must be entered in the FCT, if used:

HD randomized data bases

HDAM data bases

HD indexed data bases

HIDAM data bases

HIDAM primary index data bases

HIDAM secondary index data bases

HDAM secondary index data bases

HISAM data bases

SHISAM data bases.

When a logical data base is made from an HDAM data base and a HIDAM data base, and each of these physical data bases has a secondary index, the FCT would include the

HDAM data base

HDAM secondary index data base

HIDAM data base

HIDAM primary index data base

HIDAM secondary index data base.

There would not be an entry in the FCT for the logical data base.

The DL/I entries in the FCT define your data bases and control when these data bases are opened. You can specify that they be opened during initialization or that the opening be deferred. If opening is deferred, each data base must be opened by executing the DL/I system call STRT before the data base is used.

The FCT must be assembled and link-edited to include your DL/I entries. Because these entries are only referenced during DL/I initialization, they should be grouped with other low-activity entries in your FCT.

Processing Program Table (PPT): The DL/I system termination program, DLZSTP00, must always be defined in your PPT. If you want a DL/I formatted dump to be produced on a CICS/VS system ABEND, you must also make an entry for DLZFSDP0 (this is good practice because it can help in problem determination when a system ABEND occurs). Because both DLZSTP00 and DLZFSDP0 are only used once per CICS/VS session, they should be specified as RES=PGOUT for best CICS/VS performance. If program isolation is used, include DFHDBP (CICS/VS dynamic transaction backout program) in the PPT. If the Execution Diagnostic Facility (EDF) is used, the PPT must also contain an entry for module DLZHLPI to process HLPI commands.

Program List Table (PLT): Make an entry in part 2 of the PLT for module DLZSTP00, after the DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM statement.

Program Control Table (PCT): The PCT defines all CICS/VS transactions that can be processed by the system. Each transaction is described in a table entry that includes:

- Transaction identifier
- Transaction priority value
- Security key
- Name of program that (initially) processes the transaction.

You can create several PCTs, each identified by unique suffix characters.

If you use program isolation, all DL/I transactions that modify data bases must be marked for dynamic transaction backout (DTB=YES). If program isolation is not used, then all transactions that modify DL/I data bases should also use dynamic transaction backout. Failure to do this can cause loss of data base integrity.

Storage Layout Control (SLC) Option: You create a storage layout control table by using a series of DLZSLC macro statements, which are executed as a standard application program job. The resulting object module is link-edited into a private core image library. You can generate several SLC tables. The particular table to be used with the execution of a particular application program is specified in the DLZACT macro.

To generate an SLC, you code three types of DLZSLC macro statements:

- TYPE=INITIAL

  In this statement you specify one operand:

  NAME

  The phase name of the SLC in the core image library.

- TYPE=ENTRY

  You code a separate ENTRY type statement for every module to be loaded, and one for the buffer. The sequence of the ENTRY statements is the sequence in which the modules are loaded and the buffer allocated during DL/I initialization.

  In this statement you specify three operands:

MODULE

The name of the action module.

LOAD

Whether the module or buffer is to be loaded into low (the default) or high storage.

ALIGN

Whether or not the module loading is to be aligned on a page boundary. The default is not aligned. The buffer is always aligned.

If all action modules are loaded with the same options and in the default sequence provided in the initializer, you can code only one ENTRY statement with the name ALL.

Essentially, the SLC consists of two lists:

1. High activity modules, arranged in descending sequence of activity (most active first) loaded into low storage.
2. Low activity or unreferenced modules, arranged in ascending sequence of activity (least active first) loaded into high storage.

The high usage modules in your DL/I system are the call analyzer (DLZDLA00), the buffer handler (DLZDBH00), the retrieve module (DLZDLR00), and the HD buffer pool (BUFFER). Make these the first entries in your SLC and specify alignment and loading in low storage. DLZDLA00, DLZDLR00, and DLZDBH00 can be placed in the SVA instead of the SLC.

The lowest usage module in your system is probably the open/close module (DLZDLOC0). Specify alignment and loading in high storage.

Usage of the other modules varies, depending on the functions of your application programs. The following comments will help you sequence them in descending frequency of activity (high activity first):

- Data base logger (DLZRDBL0). If your system is read-only (processing option G specified in all PCB and SENSEG statements) specify alignment and loading in high storage. If there is high insert, delete, or replace activity specify alignment and loading in low storage.
- Delete/replace (DLZDLD00). If there is high delete or replace activity, specify alignment and loading in low storage; otherwise, specify alignment and loading in high storage. This module can be placed in the SVA instead of the SLC.
- Load/insert (DLZDDLE0). If there is high insert activity, specify alignment and loading in low storage; otherwise, specify alignment and loading in high storage. This module can be placed in the SVA instead of the SLC.
- Space management (DLZDHDS0). If there is high HD insert or delete activity, specify alignment and loading in low storage; otherwise, specify alignment and loading in high storage. This module can be placed in the SVA instead of the SLC.
- Index maintenance (DLZXMT0). If there is high index insert or delete activity, specify alignment and loading in low storage. This would be caused by HD indexed or HIDAM root segment insert or delete activity or secondary index changes caused by source segment insertion, replacement, or deletion. Otherwise, specify alignment and loading in high storage. This module can be placed in the SVA instead of the SLC.

- Program isolation enqueue/dequeue module (DLZQUEF0). If PI is used (specified or defaulted in the ACT), specify alignment and loading in low storage; otherwise, specify alignment and loading in high storage.
- Program isolation enqueue/dequeue work area (DLZQUEFW). If PI is used (specified or defaulted in the ACT), specify alignment and loading in low storage; otherwise, specify alignment and loading in high storage. This module should be adjacent to DLZQUEF0.
- Field level sensitivity copy (DLZCPY10). If field level sensitivity is used (SENFLD or VIRFLD statements used in PSB and the sensitive fields are referenced), specify alignment and loading in low storage; otherwise, specify alignment and loading in high storage.

If you don't specify all of the modules in the SLC, the remaining modules are loaded according to the DL/I default sequence.

- TYPE=FINAL

  This statement defines the end of the SLC statements.

Online Nucleus Generation Requirements

Before you can do any processing of DL/I applications in an online environment, you have to generate a DL/I online nucleus tailored to your particular needs. This section describes how that is done. The result is a DL/I online nucleus CSECT that is link-edited into a core image library. The CSECT consists of a system contents directory (SCD), a table of partition specification table (PST) prefixes, a PSB directory for each PSB specified, and an application control table (ACT). Also included are scheduling and termination routines, the online program request handler, and error routines.

> *Creating an ACT Interactively*. The ACT creation and generation procedures described in this section can also be done interactively on a 3270-type terminal by using the Interactive Macro Facility (IMF). See the introduction to this chapter and *DL/I DOS/VS Interactive Resource Definition and Utilities* for more information.

Application Control Table (ACT) Generation: The application control table associates online application programs with one or more DL/I data bases. The ACT is generated by creating DLZACT macro statements and executing them as a standard assembler job. There are several types of DLZACT statement. Each one that you are concerned with is discussed separately below.

TYPE=INITIAL: The INITIAL type of DLZACT statement must be the first one in the ACT assembly. It establishes the CSECT into which the ACT is assembled. There is one optional operand:

SUFFIX

A two-character alphameric suffix that is appended to the standard module name (DLZNUC). This becomes the name of the module in the linkage editor output library. By using different suffixes, you can create several different nucleus modules, for use under different circumstances.

TYPE=CONFIG: The CONFIG type of DLZACT statement defines your CICS/VS-DL/I DOS/VS online environment. The information in this statement is used to build the PST prefix table. There can be only one of this type of DLZACT statement in a DL/I ACT generation. There are several optional operands:

MAXTASK

The maximum number of DL/I tasks that can be processed concurrently. This can be a number from 1 to 255. If not specified, a value of 10 is assumed.

Note: The DL/I maximum task parameter can't be changed dynamically during CICS/VS execution as can the CICS/VS MXT and AMXT parameters. It can only be changed by assembling a new ACT. Since only dispatchable tasks can issue PCB calls, and the number of dispatchable tasks is limited by AMXT, there is no advantage in setting the DL/I maximum task parameter higher than the CICS/VS AMXT parameter.

CMAXTSK

The current maximum number of concurrent DL/I DOS/VS tasks allowed. This can be a number from 1 to 255. It can't be greater than the maximum number of tasks specified. If omitted, the value specified for the maximum number of tasks is used.

This operand lets you change dynamically the maximum number of DL/I tasks within the system, by using a special DL/I system call (CMXT).

Normally, the current maximum number of tasks should be the same as the value specified for the maximum number of tasks. When there is heavy demand on your CICS/VS system by non-DL/I tasks, you can lower the value for this parameter, restricting the number of current DL/I tasks. This lets a larger proportion of non-DL/I tasks execute concurrently.

BFRPOOL

The number of buffer subpools to be acquired and formatted during DL/I online system initialization. The value can be from 0 to 255. If you specify zero, or don't specify any value, a request for the entry of a value will be made through the system log during DL/I online system initialization.

If no buffer pool control options are specified, a subpool consists of 32 fixed-length buffers. The buffer size is generally consistent with the VSAM data base control interval size, and can be 512 or any multiple of 512 bytes to a maximum of 4K bytes. The buffer size value is determined at DL/I system initialization. It is based on the value specified here, the number of data bases, and the size of the VSAM control intervals.

PASS

The password (one to eight alphameric characters) associated with the functions of system control calls. If you don't specify a password, “DLZPASS1” will be used as the default.

SLC

The phase name of the storage layout control table you want to use.

PI

Whether or not you want to use program isolation. You must use program isolation if this is the local system when using extended remote PSB support, and it is recommended that you use it if this is the remote system.

REMOTE

Whether or not this DL/I online nucleus is for a system that will process requests from remote systems. If you specify YES, all PSBs defined in this (local) system's DL/I nucleus can be accessed from other (remote) systems through the CICS/VS mirror program DFHMIR. Specifying this parameter causes the generation of a DLZACT TYPE=PROGRAM, PGMNAME=DFHMIR statement for the CICS/VS mirror program, which includes the PSB names of all PSBs defined in this DL/I nucleus.

If there are PSBs that are to be accessed only from remote systems, a DLZACT TYPE=PROGRAM,PGMNAME=DFHMIR statement must be used to define those PSBs. If you also specify YES for this parameter, all PSBs that can be accessed by the local system are *added* to the list of PSBs specified in that statement.

TYPE=PROGRAM: The DLZACT TYPE=PROGRAM statement defines which PSBs may be scheduled by a CICS/VS application program. The name of any program issuing a scheduling call must be listed in the ACT. You can include as many as 4095 TYPE=PROGRAM statements in an ACT generation. There is also a limit of 4095 unique program names and unique PSB names permitted in these statements.

There are two operands:

PGMNAME

An application program name of from one to eight alphameric characters.

PSBNAME

The names of the PSBs associated with this program. PSB names are from one to seven alphameric characters long. The first PSB named in this list is the default PSB for the application program named in this statement.

When generating a DL/I online nucleus for a system that processes DL/I requests from another system, you must include all PSB names that can be referenced by the other system, in PROGRAM type statements for this system. (See the discussion of remote system specification in the section on the DLZACT TYPE=CONFIG statement.)

TYPE=RPSB: The RPSB type of DLZACT statement defines the PSBs that are involved in remote operation. All RPSB type statements must immediately follow the last PROGRAM type DLZACT statement.

Remote PSB

There are several operands:

PSB

A unique name identifying this PSB as a remote PSB.

RNAME

An optional operand specifying the name of the PSB in the remote system. If omitted, the name specified in the PSB operand is used.

SYSID

The four-character identifier of the CICS/VS system to which this remote PSB and associated data bases are assigned.

LANG

An optional operand used when your application runs in the MPS batch environment, to specify that the application programs using this PSB are written in PL/I.

Extended Remote PSB

There are several operands:

PSB

A unique name identifying this PSB as a remote PSB.

LNAME

The name of the local PSB component.

RNAME

The name of the PSB in the remote system.

SYSID

The four-character identifier of the CICS/VS system to which this remote PSB and associated data bases are assigned.

LANG

An optional operand used when your application runs in the MPS batch environment, to specify that the application programs using this PSB are written in PL/I.

TYPE=BUFFER: Use of the BUFFER type of DLZACT statement is optional. It can be used to specify information about DL/I buffer subpools and VSAM buffer usage. There are two optional operands, each with several parameters:

HDBFR

The parameters in this operand give information about a DL/I subpool:

bufno

The number of buffers to be allocated to this subpool. The value can be from 2 to 32. If you don't specify a value, 32 is assumed.

dbdname1,dbdname2,...

The names of DBDs that are allocated to this subpool.

HSBFR

The parameters in this operand give information about VSAM buffer allocation for HISAM and INDEX data bases:

indno

The number of index buffers for a KSDS

ksdsbuf

The number of data buffers for a KSDS

esdsbuf

The number of data buffers for the ESDS (HISAM only)

dbdname

The name of the HISAM or INDEX DBD referenced by the application program.

TYPE=FINAL: The FINAL type of DLZACT statement defines the end of the ACT generation to the assembler.

Reassembling DL/I for New/Updated CICS/VS

A set of DL/I macros and modules (A.books) must be reassembled, relinked, and recataloged if a new (later) release of CICS/VS is installed or if significant CICS/VS 1.6 service is applied.

When upgrading to a new level of CICS/VS or applying significant CICS/VS service updates, all DL/I ACTs must be reassembled and relinked (for the online nucleus), using the DL/I DLZACT macro.

Listed below are all of the sample job streams for reassembling, relinking, and recataloging the DL/I modules, macros, and source books which are CICS/VS dependent. For additional information you should refer to the following documents:

- The CICS/VS "Memo To User" and CICS/VS "Program Directory".
- "CICS/DOS/VS Installation and Operations Guide".

  These job streams are only examples. You should refer to the documents referenced above and create the applicable job streams, using the samples provided below. You must provide the job control information as it pertains to your individual system configurations and requirements.

- ALL USERS:
  - You should also have your VSE system libraries online to insure the proper library search when reassembling and relinking these DL/I members.
  - Fill in the file id's, volume serial numbers, and tape address for your system configuration.
  - You must have a tape mounted which is used as output for the text decks from the assembler and for input to the VSE MAINT program.
  - Verify for each macro or module that the assembly, link edit, and MAINT librarian program ran correctly.
  - After executing the following jobs, reassemble and relink all ACTs using the DL/I DLZACT macro.
- SIPO/E USERS:
  - The service (generation) libraries for CICS/VS and DL/I should both be restored and referenced with LIBDEFs.
  - The assembly of DLZMPI00 also requires the service (generation) library for VSE.
  - Check the production relocatable and source libraries for DL/I. If the production library contains a module or source book which you have assembled and cataloged, copy that module or source book from the service (generation) library into the production library. Service (generation) and production libraries must be kept at the same level.

```
// JOB ASSEM1   ASSEMBLE AND CATALOG NEW DLZPRECC MACRO
* ************************************************************* *
* *                    RUN THIS JOB FIRST!                    * *
* *                                                           * *
* * THIS JOB STREAM ASSEMBLES THE DLZPRECC MACRO (A.BOOK) AND   * *
* * CATALOGS THE OUTPUT AS AN E.BOOK (E.DLZPRECC) INTO YOUR     * *
* * DL/I SOURCE STATEMENT LIBRARY FOR USE IN ASSEMBLIES FOUND   * *
* * LATER IN THIS SERIES OF JOB STREAMS                         * *
* *                                                           * *
* * PROVIDE THE INFORMATION REQUIRED AS NOTED IN THE COMMENTS   * *
* *                                                           * *
* ************************************************************* *
// OPTION EDECK,NOXREF,NODECK
// DLBL CICSSL,'cics/vs ssl name' * PROVIDE FILE ID FOR THIS LIB. *
// EXTENT ,volser                          * PROVIDE VOLSER INFO. *
// DLBL DLISL,'dl/i ssl name'     * PROVIDE FILE ID FOR THIS LIB. *
// EXTENT ,volser                          * PROVIDE VOLSER INFO. *
// LIBDEF SL,SEARCH=(DLISL,CICSSL)
// MTC REW,X'adr'                * PROVIDE ADDRESS OF OUTPUT TAPE *
// ASSGN SYSPCH,X'adr'           * PROVIDE ADDRESS OF OUTPUT TAPE *
// EXEC ASSEMBLY
        PRINT  NOGEN
        COPY DLZPRECC
        END
/*
// RESET SYSPCH
// MTC WTM,X'adr',2
// MTC REW,X'adr'                ** PROVIDE ADDRESS OF TAPE USED
// ASSGN SYSIPT,X'adr'                          IN PRIOR JOB STEP **
// DLBL DLISL,'dl/i ssl name'        ** PROVIDE FILE ID FOR LIB. **
// EXTENT ,volser                        ** PROVIDE VOLSER INFO. **
// LIBDEF SL,TO=DLISL
// EXEC MAINT
/*
/&
```

```
// JOB ASSEM2     ASSEMBLE,CATALOG,LINKEDIT DL/I-CICS/VS PROGRAMS
* ************************************************************* *
* *                                                           * *
* *  REASSEMBLE THE FOLLOWING DL/I MODULES (A.BOOKS), CATALOG   * *
* *  THEM INTO THE DL/I RELOCATABLE LIBRARY (AS MODULES), THEN  * *
* *  LINKEDIT THEM INTO THE DL/I CORE IMAGE LIBRARY.            * *
* *                                                           * *
* *  ALL OF THE FOLLOWING NINE DL/I MODULES (A.BOOKS) MUST BE   * *
* *  REASSEMBLED AND RELINKEDITED AND RECATALOGED.              * *
* *  EACH OF THE NINE DL/I MODULES REQUIRES ITS OWN SEPARATE    * *
* *  JOB STREAM (THAT IS ONE COMPLETE JOB STREAM FOR EACH DL/I  * *
* *  MODULE).  SUBSTITUTE THE MODULE NAME BELOW IN THE          * *
* *  ASSEMBLY EXEC JOB STEP ON THE STATEMENT WHICH READS:       * *
* *                  ' COPY DLZxxxxx '                          * *
* *                                                           * *
* *  DLZBPC00    DLZMPC00    DLZMPUR0    DLZMSTP0    DLZMSTR0   * *
* *  DLZOLI00    DLZRDBL1    DLZSTP00    DLZSTTL                * *
* *                                                           * *
* * NOTE:   SIPO/E USERS ALSO NEED LIBDEF STATEMENTS FOR THE    * *
* *         SERVICE (GENERATION) LIBRARIES FOR THE FOLLOWING    * *
* *         MACROS: LOCK, UNLOCK, DTL, BTWAIT, DIMOD.          * *
* *                                                           * *
* ************************************************************* *
// OPTION DECK,CATAL,NOXREF
// DLBL CICSSL,'cics/vs ssl name' * PROVIDE FILE ID FOR THIS LIB. *
// EXTENT ,volser                          * PROVIDE VOLSER INFO. *
// DLBL DLISL,'dl/i ssl name'     * PROVIDE FILE ID FOR THIS LIB. *
// EXTENT ,volser                          * PROVIDE VOLSER INFO. *
// LIBDEF SL,SEARCH=(DLISL,CICSSL)
// MTC REW,X'adr'                * PROVIDE ADDRESS OF OUTPUT TAPE *
// ASSGN SYSPCH,X'adr'             * ASSIGN SYSPCH TO OUTPUT TAPE *
// EXEC ASSEMBLY
        PRINT  NOGEN
        COPY DLZxxxxx
        END
/*
// DLBL DLIRL,'dl/i rl name'      * PROVIDE FILE ID FOR THIS LIB. *
// EXTENT ,volser                          * PROVIDE VOLSER INFO. *
// LIBDEF RL,SEARCH=DLIRL
```

```
* ************************************************************* *
* *  THIS PART OF THE JOB STREAM WILL LINKEDIT THE REASSEMBLED  * *
* *  DL/I MODULES FROM THE DL/I RELOCATABLE LIBRARY AS PHASES   * *
* *  TO THE DL/I CORE IMAGE LIBRARY.                            * *
* *                                                             * *
* *  SIPO/E USERS: USE THE PRODUCTION CORE IMAGE LIBRARY WHICH  * *
* *                CONTAINS THE DL/I PHASES.                    * *
* ************************************************************* *
// DLBL DLICL,'dl/i cil name'     * PROVIDE FILE ID FOR THIS LIB. *
// EXTENT ,volser                          * PROVIDE VOLSER INFO. *
// LIBDEF CL,TO=DLICL
// EXEC LNKEDT
// RESET SYSPCH
// MTC WTM,X,'adr',2      * PROVIDE ADR OF TAPE USED IN PRIOR JOB *
// MTC REW,X'adr'                              * REWIND THE TAPE *
// ASSGN SYSIPT,X'adr'    * ASSIGN SYSIPT TO ADDRESS OF TAPE USED *
* ************************************************************* *
* *  THIS NEXT JOB STEP WILL CATALOG THE TEXT DECKS CREATED BY  * *
* *  THE REASSEMBLIES FOR THE DL/I MODULES INTO THE DL/I        * *
* *  RELOCATABLE LIBRARY WHICH CONTAINS ALL DL/I MODULES (TEXT) * *
* *                                                             * *
* *  SIPO/E USERS: THE LIBDEF TO BE USED IS FOR THE SERVICE     * *
* *                (GENERATION) RELOCATABLE LIBRARY.  IF THIS   * *
* *                MODULE IS ALSO FROM THE DL/I PRODUCTION      * *
* *                LIBRARY, COPY IT FROM THE DL/I SERVICE       * *
* *                (GENERATION) RELOCATABLE LIBRARY TO THE      * *
* *                PRODUCTION RELOCATABLE LIBRARY, TO KEEP      * *
* *                YOUR PRODUCTION AND GENERATION LIBRARIES     * *
* *                AT THE SAME LEVEL.                           * *
* ************************************************************* *
// DLBL DLIRL,'dl/i rl name'      * PROVIDE FILE ID FOR THIS LIB. *
// EXTENT ,volser                          * PROVIDE VOLSER INFO. *
// LIBDEF RL,TO=DLIRL
// EXEC MAINT
/*
/&
```

```
// JOB ASSEM3      DL/I-CICS PROGRAMS AND CATALOG INTO RELO. LIB.
* ************************************************************* *
* *   ASSEMBLE THE FOLLOWING DL/I MODULES (A.BOOKS) AND         * *
* *   RECATALOG THEM INTO THE DL/I PRIVATE RELOCATABLE LIBRARY. * *
* *                                                             * *
* *   EACH DL/I MODULE LISTED BELOW REQUIRES ITS OWN SEPARATE   * *
* *   JOB STREAM (THAT IS ONE COMPLETE JOB STREAM FOR EACH DL/I * *
* *   MODULE).  SUBSTITUTE THE MODULE NAME BELOW IN THE         * *
* *   ASSEMBLY EXEC JOB STEP ON THE STATEMENT WHICH READS:      * *
* *                  ' COPY DLZxxxxx '                          * *
* *                                                             * *
* *   DLZEIPO0  DLZFTDP0  DLZISC00  DLZLOC00  DLZODP  DLZSTRO0  * *
* *                                                             * *
* ************************************************************* *
// OPTION DECK,NOXREF
// DLBL CICSSL,'cics/vs ssl name' * PROVIDE FILE ID FOR THIS LIB. *
// EXTENT ,volser                          * PROVIDE VOLSER INFO. *
// DLBL DLISL,'dl/i ssl name'     * PROVIDE FILE ID FOR THIS LIB. *
// EXTENT ,volser                          * PROVIDE VOLSER INFO. *
// LIBDEF SL,SEARCH=(DLISL,CICSSL)
// MTC REW,X'adr'           * PROVIDE ADDRESS OF TAPE USED IN JOB *
// ASSGN SYSPCH,X'adr'    * ASSIGN SYSPCH TO ADDRESS OF TAPE USED *
// EXEC ASSEMBLY
        PRINT  NOGEN
        COPY DLZxxxxx
        END
/*
// RESET SYSPCH
// MTC WTM,X'adr',2      * PROVIDE ADR OF TAPE USED IN PRIOR STEP *
// MTC REW,X'adr'                              * REWIND THE TAPE *
// ASSGN SYSIPT,X'adr'    * ASSIGN SYSIPT TO ADDRESS OF TAPE USED *
* ************************************************************* *
* *                                                           * *
* *  THIS NEXT STEP CATALOGS THE TEXT DECK INTO THE DL/I      * *
* *                   RELOCATABLE LIBRARY.                    * *
* *                                                           * *
* *  SIPO/E USERS: THE LIBDEF TO BE USED IS THE SERVICE       * *
* *                (GENERATION) LIBRARY. IF THIS MODULE IS ALSO * *
* *                IN THE PRODUCTION LIBRARY, COPY IT FROM THE  * *
* *                SERVICE (GENERATION) LIBRARY TO THE        * *
* *                PRODUCTION LIBRARY.                        * *
* *                                                           * *
* ************************************************************* *
// DLBL DLIRL,'dl/i rl name'      * PROVIDE FILE ID FOR THIS LIB. *
// EXTENT ,volser                          * PROVIDE VOLSER INFO. *
// LIBDEF RL,TO=DLIRL
// EXEC MAINT
/*
/&
```

```
// JOB ASSEM4     ASSEMBLE AND CATALOG NEW E.DLZTRACE MACRO
* ************************************************************ *
* *                ONLINE USERS ONLY MUST                      * *
* * RUN THIS JOB IF YOU HAVE TRACE PROGRAMS WITH OUTPUT=CICS.   * *
* *                                                            * *
* * THIS JOB ASSEMBLES THE DLZTRACE MACRO (FROM A.BOOK) AND     * *
* * CATALOGS THE ASSEMBLED MACRO AS AN E.BOOK INTO THE DL/I     * *
* * PRIMARY SOURCE LIBRARY FOR USE IN LATER ASSEMBLIES. AFTER   * *
* * EXECUTING THIS JOB, YOU MUST REASSEMBLE AND RELINK ALL OF   * *
* * YOUR TRACE PROGRAMS THAT HAVE THE OPTION 'OUTPUT=CICS'.     * *
* *                                                            * *
* ************************************************************ *
// OPTION EDECK,NOXREF,NODECK
// DLBL CICSSL,'cics/vs ssl name' * PROVIDE FILE ID FOR THIS LIB. *
// EXTENT ,volser                          * PROVIDE VOLSER INFO. *
// DLBL DLISL,'dl/i sl name'      * PROVIDE FILE ID FOR THIS LIB. *
// EXTENT ,volser                          * PROVIDE VOLSER INFO. *
// LIBDEF SL,SEARCH=(DLISL,CICSSL)
// MTC REW,X'adr'                  * PROVIDE ADDRESS OF TAPE USED *
// ASSGN SYSPCH,X'adr'         * ASSIGN SYSPCH TO ADDRESS OF TAPE *
// EXEC ASSEMBLY
        PRINT  NOGEN
        COPY DLZTRACE
        END
/*
// RESET SYSPCH
// MTC WTM,X'adr',2      * PROVIDE ADR OF TAPE USED IN PRIOR STEP *
// MTC REW,X'adr'                               * REWIND THE TAPE *
// ASSGN SYSIPT,X'adr'         * ASSIGN SYSIPT TO ADDRESS OF TAPE *
// DLBL DLISL,'dl/i ssl name'     * PROVIDE FILE ID FOR THIS LIB. *
// EXTENT ,volser                          * PROVIDE VOLSER INFO. *
// LIBDEF SL,TO=DLISL
// EXEC MAINT
/*
/&
```

MPS Operation

To use MPS, you must have CICS/VS installed on your system in addition to DL/I. You also have to make entries in various CICS/VS and DL/I control tables. These same control tables are also required for CICS/VS online operation and are discussed in the section called “Online Operation” earlier in this chapter. Only the additional requirements needed for MPS are discussed in the following sections.

Examples of coding the various CICS/VS-DL/I macros for the tables described in the following sections are given in *DL/I DOS/VS Resource Definition and Utilities* and *DL/I DOS/VS Interactive Resource Definition and Utilities*.

CICS/VS File Control Table (FCT)

You must make an entry in the file control table (FCT) for every DL/I data base that is referenced by an MPS batch program. You should specify only:

- The name of the data base (the name specified in the DBD statement in the DBDGEN)
- The DL/I access method
- Whether the data base is to be opened automatically or whether it will be opened at a later time by using the DL/I system call STRT.

These entries should be put at the end of the FCT.

CICS/VS Journal Control Table (JCT)

The JCT describes your journal data sets to CICS/VS (optional). There must be one entry in the table to define the CICS/VS system journal. If you use program isolation, the DL/I log must be assigned to the CICS/VS system journal.

CICS/VS Program Control Table (PCT)

The PCT defines all transactions that can be processed by the system. Each transaction is described in a table entry that includes:

- Transaction identifier
- Transaction priority value
- Security key
- Name of program that (initially) processes the transaction.

You have to make entries in the PCT to define six MPS transactions. They are:

- CSDA (start MPS operation)
- CSDB (master partition controller)
- CSDC (batch partition controller)
- CSDD (stop MPS operation)
- CSDE (run and buffer statistics)
- CSDP (purge temporary storage).

These entries should be put toward the end of the PCT. Their class should be specified as LONG.

Only transactions CSDA, CSDD, CSDE, and CSDP would be entered from a terminal. Since they are special transactions, you may want to put transaction security on them so that only the master terminal operator and other authorized persons are allowed to execute them.

If you use program isolation, all DL/I transactions that modify data bases or other recoverable resources must be marked for dynamic transaction backout. For MPS, this includes the CSDB, CSDC, and CSDP transactions. Failure to do this can cause loss of data base integrity.

CICS/VS Program Processing Table (PPT)

The programs referenced by the MPS transactions, described above in the section on the PCT, must be defined in the PPT. They are:

- DLZMSTR0
- DLZMPC00
- DLZBPC00
- DLZMSTP0
- DLZMPUR0

These entries should be placed near the end of the PPT. Because of their low usage, DLZMSTR0 and DLZMSTP0 must be specified as RES=PGOUT. DLZMPC00 and DLZBPC00 must be specified as RES=YES. DLZMPC00 should also be specified as RELOAD=YES. If you want a DL/I formatted dump to be printed in case of a CICS/VS abend, you must make an entry for the DL/I formatted system dump program (DLZFSDP0), with RES=PGOUT specified. These specifications could also be made with the DFHALT macro.

CICS/VS System Initialization Table (SIT)

You must specify in the SIT that the DL/I nucleus module DLZNUC or an alternate nucleus module is to be loaded during CICS/VS initialization. In addition, DL/I requires entries in a shutdown program list table (PLT) and, optionally, in an initialization PLT. Specify the suffixes of the appropriate initialization and shutdown PLTs in the PLTPI and PLTSD parameters, respectively. In order to support emergency restarts and warm starts of the CICS/VS system, START=AUTO should be specified.

CICS/VS Program List Table (PLT)

MPS batch programs cannot be executed until the master partition controller is initiated in the CICS/VS partition. This can be done:

1. By a terminal operator entering the CSDA transaction
2. By having CICS/VS execute the MPS start program (DLZMSTR0) as part of its system initialization processing.

For automatic initialization, the start program must be made an entry in a startup PLT. The suffix of this PLT must be specified in the PLTPI parameter of the SIT.

If you want MPS to stop automatically when CICS/VS starts its shutdown processing, make an entry in the shutdown PLT for the MPS stop program DLZMSTP0. This entry must be before the DFHDELIM entry. The suffix of this PLT must be specified in the PLTSD parameter of the SIT.

Code the PLT names in the PPT.

DL/I Application Control Table (ACT)

When planning your DL/I application control table (ACT) generation when you are using MPS, consider the following items in addition to the requirements discussed in the section called “Online Nucleus Generation Requirements” earlier in this chapter:

- Review your ACT environmental parameters (maximum number of tasks, current maximum number of tasks, etc.) and include any changes needed for MPS. Each active MPS batch program requires a batch partition controller transaction in the online system. To handle these, you may need to increase the values for the maximum number of tasks and the current maximum number of tasks. If you are adding new data bases into the online system (for instance, those that were previously used only in the batch environment, but are now to be accessed by MPS batch programs) you should include these in estimating the size of your buffer pool and DMB assignments.
- Review your definition of valid application program and PSB combinations for the online system. The name of any program issuing a scheduling call must be listed in the ACT. All PSBs that the program is authorized to use must be listed with it in the ACT.

  Because all MPS batch programs actually communicate with DL/I through the batch partition controller running in the online partition, you must make an ACT entry for the BPC. List with it the names of all PSBs that will be used by MPS batch programs.

CICS/VS Temporary Storage Table (TST)

The temporary storage queue name "DLZTSQ00 must be specified in the TST as TYPE=RECOVERY. This is necessary for the function of the MPS Restart Facility. This queue name is reserved and must not be used for other application purposes.

Temporary storage is also required by dynamic transaction backout (DTB) for each active batch partition controller (BPC) in the online partition. If sufficient temporary storage space is not defined, BPCs and other tasks may go into a wait state. Care should be taken to avoid this situation by defining enough space.

Processing Contention Control

If the access method you chose for your data base is HSAM, SHSAM, or SHISAM, or if your application is to run in the batch environment, you can ignore this section.

You use a DLZACT TYPE=CONFIG statement in an application control table (ACT) generation to specify your choice of whether you want to use program isolation or intent scheduling. You specify your choice by using the PI operand. If you specify NO in the operand, intent scheduling is in effect. If you specify YES, or omit the operand, program isolation is in effect.

Distributed Data

The following sections apply to online and MPS batch environments.

Remote PSB

Use a DLZACT TYPE=RPSB statement in the ACT generation to specify that a PSB name is to represent a remote system. You provide additional information in the following operands:

PSB

A unique name identifying this PSB as a remote PSB

RNAME

An optional operand specifying the name of the PSB in the remote system. If omitted, the name specified in the PSB operand is used.

SYSID

The four-character identifier of the CICS/VS system to which this remote PSB and associated data bases are assigned

LANG

An optional operand used when your application runs in the MPS batch environment, to specify that the application programs using this PSB are written in PL/I.

Extended Remote PSB

To synchronize updates in a remote system with updates in the local system, you must define an extended remote PSB. To define an extended remote PSB, use the following operands in a DLZACT TYPE=RPSB statement in the ACT generation:

PSB

A unique name identifying this PSB as a remote PSB

LNAME

The name of the local PSB component

RNAME

The name of the PSB in the remote system

SYSID

The four-character identifier of the CICS/VS system to which this remote PSB and associated data bases are assigned

LANG

An optional operand used when your application runs in the MPS batch environment, to specify that the application programs using this PSB are written in PL/I.

Logging

DL/I System Log

It is recommended that the CICS/VS system journal be used instead of the DL/I system log in the online and MPS batch environments.

In the batch environment, use the DL/I parameter statement at program execution time to specify the use of the DL/I system log. You do this by coding the LOG operand in the statement. The LOG operand has parameters that let you specify whether the log is to be written on tape or disk. If you choose disk, you can specify that there are to be either one or two disk extents for the log, and whether the operator is to be notified when an extent is full.

The DL/I disk log is a VSAM ESDS. Therefore, before using disk logging you must define the disk log data set(s) in the VSAM catalog by using the DEFINE command. The data set must be defined with the reusable and recovery attributes. The DEFINE must have the following characteristics:

1. The control interval size must be 1024 bytes.
2. The record size must be 1017 bytes.
3. The log file(s) must be reusable.

Note: The cluster names can be any names desired. The data name must be DSKLOG1. If two log files are defined, the second must be DSKLOG2.

This is an example of disk log file definition:

```
// JOB DISKLOG DEFINE
// EXEC IDCAMS,SIZE=200K
   DEFINE CLUSTER(NAME(DLI.LOG1)NONINDEXED) -
   DATA (NAME(DSKLOG1) VOLUME(xxxxxx) TRK(nn) -
   RECORDSIZE(1017 1017) -
   RECOVERY REUSE CONTROLINTERVALSIZE(1024))
/*
/&
```

You also have to provide ASSGN statements and a set of DLBL/EXTENT statements (one set for each log extent) in the job control statements for your job. The DLBL filename must be DSKLOG1 for the first file and DSKLOG2 for the second file.

In order for logging to be active, you must also include a job control statement in the program input stream that sets bit 6 of the UPSI byte to 0.

You control multivolume and multifile log tapes in normal operating system job control statements.

CICS/VS System Journal

If your application runs in the batch environment you can ignore this section.

You specify the use of the CICS/VS system journal in the CICS/VS journal control table (JCT). You can specify that the journal is to be recorded on either tape or disk. If on disk, you must make assignments for the SAM disk journal file.

Asynchronous Logging

In the batch environment, you use the DL/I parameter statement at program execution time to specify the use of asynchronous logging. You do this by including the ASLOG=YES operand in the statement. Omitting the operand disables asynchronous logging.

Limited Data Sharing

If the access method you chose for your data base is HSAM or SHSAM you can ignore this section.

You define limited data sharing for a data base by:

- Specifying in the PSB generation that the data base may be opened for read-only access. You do this by specifying processing option GO or GOP for all PCBs that reference the data base.
- Specifying VSAM share option 2 for the data set(s) of the data base. You do this by using the SHAREOPTIONS parameter in the VSAM DEFINE CLUSTER command when you define your data base data set(s).

Note: Specifying processing option GO or GOP alone does not define a data base as shareable. The VSAM share option must also be specified. Specifying share option 2 alone allows sharing among subsystems in the same host. Locating the data sets and their VSAM catalogs on shared devices (which could be the same one) automatically extends the sharing across host machines. Sharing of data bases located on 2314 devices can only be between subsystems on a single host machine.

Buffer Pool Assignment

HD Access Methods

Online Environment: You control buffer pool assignment for HD, HDAM, and HIDAM data bases in the online environment by using the DLZACT TYPE=BUFFER statement in the ACT generation. You do this specification in the HDBFR operand, which describes one DL/I subpool:

bufno

The number of buffers to be allocated in this subpool.

dbdname1,dbdname2,...

The names of the DBDs allocated to this subpool.

Batch Environment: You control buffer pool assignment for HD, HDAM, and HIDAM data bases in the batch environment by using the DL/I parameter statement at program execution time. You do this specification in the buff and HDBFR operands:

buff

The number of data base subpools required for this execution. If omitted, one subpool is assumed.

HDBFR

The parameters in this operand describe one DL/I subpool. The operand can be repeated to describe as many subpools as needed. The parameters are:

bufno

The number of buffers to be allocated in this subpool.

dbdname1,dbdname2,...

The names of the DBDs allocated to this subpool.

HS Access Methods

Online Environment: You can control the amount of buffer space used by VSAM for HISAM, SHISAM, or INDEX data base data sets in three ways:

1. The BUFFERSPACE parameter in the VSAM DEFINE CLUSTER command when you create the data set's catalog entry.
2. The BUFSP parameter of the DLBL statement for the data set in the job control statements.
3. The HSBFR operand in the DLZACT TYPE=BUFFER statement in the ACT generation.

The space allocated is equal to the largest of the three amounts specified.

You control buffer assignment within this space by using the HSBFR operand in the DLZACT TYPE=BUFFER statement in the ACT generation. If you don't use HSBFR, VSAM allocates three index buffers and two data buffers for each data set.

The parameters in the HSBFR operand are:

indno

The number of index buffers for a KSDS. If omitted, three are assumed.

ksdsbuf

The number of data buffers for a KSDS. If omitted, two are assumed.

esdsbuf

The number of data buffers for the ESDS (HISAM only). If omitted, two are assumed.

dbdname

The name of the associated DBD.

Batch Environment: You can control the amount of buffer space used by VSAM for HISAM, SHISAM, or INDEX data base data sets in three ways:

1. The BUFFERSPACE parameter in the VSAM DEFINE CLUSTER command when you create the data set's catalog entry.

2. The BUFSP parameter of the DLBL statement for the data set in the job control statements.
3. The HSBFR operand in the DL/I parameter statement at program execution time.

The space allocated is equal to the largest of the three amounts specified.

You control buffer assignment within this space by using the HSBFR operand in the DL/I parameter statement at program execution time. If you don't use HSBFR, VSAM allocates three index buffers and two data buffers for each data set.

The parameters in the HSBFR operand are:

indno

The number of index buffers for a KSDS. If omitted, three are assumed.

ksdsbuf

The number of data buffers for a KSDS. If omitted, two are assumed.

esdsbuf

The number of data buffers for the ESDS (HISAM only). If omitted, two are assumed.

dbdname3

The name of the associated DBD.

DL/I Virtual Storage Estimates

In order for DL/I to execute in a virtual partition, storage space must be available for several major items. The particular items that must be included vary depending on the DL/I environment (online, MPS batch, or batch). You must perform a computation involving the sizes of these components to determine the size required for the virtual storage partition. Appendix A, “DL/I Virtual Storage Estimates” on page A-1 gives the details on how to perform this calculation.

DL/I Real Storage Estimates

In order for DL/I to execute with good performance, sufficient real storage must be allocated keep the paging activity at a reasonable level. For this reason, it is necessary to make a real storage estimate computation. Appendix B, “DL/I Real Storage Estimates” on page B-1 shows you how to do this.

Specifying the Data Base Access Method

You use the DBD statement in the DBD generation to specify the data base access method for your data base. The access method is specified by using the ACCESS operand.

If your access method is HD randomized or HDAM, you also have to specify other operands or parameters in the DBD statement.

For *HD randomized*, the operands are:

CIANPT

The number of root anchor points for each VSAM control interval

PRIMCI

The maximum number of control intervals to be located in the prime (root-addressable) area

RILIM

The maximum number of bytes that can be inserted into the primary area with one set of contiguous inserts for a single data base record.

For *HDAM* the operand is RMNAME. It has the following parameters:

mod-name

The name of your randomizing module

anch

The number of root anchor points for each VSAM control interval

rbn

The maximum number of control intervals to be located in the prime (root-addressable) area

bytes

The maximum number of bytes that can be inserted into the primary area with one set of contiguous inserts for a single data base record.

Specifying Data Base Physical Characteristics

Logical Record Length

If the access method you chose for your data base is HD, HDAM, or HIDAM you can ignore this section.

You use the DATASET statement in the DBD generation for your data base to define the logical record length of your data base. The logical record length is specified by using the RECORD operand. You code one or two parameters for the RECORD operand, depending on the access method of the data base.

- HISAM and SHISAM

  Two logical record lengths are required. The first is the KSDS record length. The second is the ESDS record length.

- HSAM and SHSAM

  Two logical record lengths are required. The first is the input record length. The second is the output record length.

- INDEX

  One logical record length is required. It is the KSDS record length.

CI and Block Size

If the access method you chose for your data base is HSAM or SHSAM you can ignore this section.

You use the DATASET statement in the DBD generation for your data base to define the CI or block size of your data base. The CI or block size is specified by using the BLOCK operand. You code one or two parameters for the BLOCK operand, depending on the access method of the data base.

- HD, HDAM, or HIDAM

  One parameter is required. It is the size of the VSAM control interval.

- HISAM

  Two parameters are required. The first is the maximum number of VSAM KSDS records contained in one control interval. The second is the maximum number of ESDS records contained in one control interval.

- INDEX and SHISAM

  One parameter is required. It is the maximum number of VSAM KSDS records contained in one control interval.

Distributed Free Space

If the access method you chose for your data base is HSAM, SHSAM, HISAM, or SHISAM you can ignore this section.

For initial loading of your data base, you use the DATASET statement in the DBD generation for the data base to define the number of free CIs or the percentage of free space in each CI. The free space is specified by using the FRSPC operand. You code one or two parameters for the FRSPC operand, depending on whether you want to specify free CIs or a percentage in each CI, or both.

fbff

The first parameter, fbff, specifies that every nth CI is to be left free.

fspf

The second parameter, fspf, specifies the percentage of each CI that is to be left free.

Device Type

You use the DATASET statement in the DBD generation for your data base to specify the physical device type, on which your data base will be stored. The device type is specified by using the DEVICE operand. The available devices are:

`TAPE, FBA, 3375, 3350, 3340, 3330, and 2314.`

Scanning Range

If the access method you chose for your data base is HSAM, SHSAM, HISAM, or SHISAM you can ignore this section.

You use the DATASET statement in the DBD generation for your data base to specify the range of cylinders or FBA blocks to be scanned for available storage space. The scanning range is specified by using the SCAN operand.

Allocating Data Sets

If your data base uses the HSAM or simple HSAM access method, you allocate the data set by using normal operating system job control statements.

All other DL/I access methods use VSAM, so data set allocation is done by using the VSE access method services utility functions. You have to define a VSAM catalog, VSAM data space, and VSAM data sets. These are discussed briefly here. For the details on how to use the VSAM commands see *Using VSE/VSAM Commands and Macros*.

VSAM Catalog

You have to define a master catalog first; then, optionally, any number of user catalogs. Use the access method services DEFINE MASTERCATALOG and DEFINE USERCATALOG commands.

VSAM Data Spaces

This is the direct access storage space assigned to VSAM. VSAM allocates space to VSAM data sets from this space. Use the DEFINE SPACE command.

VSAM Data Sets

Your data base is made up of one or more data sets, depending on the DL/I access method. Each data set has to be defined to VSAM. Use the DEFINE CLUSTER command. Take the file attribute and key information needed in the command from the output listing of the DBD generation for the data base. This helps ensure that you have the right information.

VSAM Options

If the access method you chose for your data base is HSAM or SHSAM, you can ignore this section.

You specify the VSAM share option for your data base in the VSAM DEFINE CLUSTER command when you define your data base data set(s). The share option is specified by using the SHAREOPTIONS parameter.

Part 4. Using Data Bases

Chapter 6. DL/I Program Execution

This chapter gives information about application program execution in the DL/I environments. There are three sections:

1. Preparing for Program Execution
2. Program Execution in the DL/I Operating Environments
3. Testing Programs with a Data Base

The first section reviews the steps that must be completed before a data base and application programs are ready for use together. The second section describes the three DL/I operating environments as they apply to program execution, and describes system control flow for each. The last section describes the kinds of testing to be done on data base application programs, and the generation of test data bases to be used in that testing.

Preparing For Program Execution

DL/I operates in the Disk Operating System/Virtual Storage Extended (DOS/VSE) environment under VSE/Advanced Functions (VSE). The DL/I facilities make it possible for user-written online, multiple partition support (MPS) batch, or batch programs to access data bases for processing or maintenance; and for batch programs to do the initial loading of data bases. Online programs operate under the control of CICS/DOS/VS, while using DL/I for data base access. Figure 6-1 shows the DL/I batch system environment: a single operating system partition containing the user's application program and the DL/I system.

The major parts of the DL/I system are:

- The system control facility
- The facility providing the functions for creating, accessing, updating, adding, and deleting data base segments
- The controls for keeping a record of these operations in a log.

The user's application program is given control by the system control facility. Input and output are usually performed within the application program, using standard operating system data management facilities. All data base operations are initiated by the application program through the interface with DL/I. This interface controls the execution of DL/I High Level Programming Interface commands or DL/I CALL statements from the application program. Parameters in the commands or statements provide the information necessary to perform specific data base operations on specific data segments in specific data bases. An application program can interface with more than one DL/I data base.

After receiving an HLPI command or CALL statement, the DL/I facility interprets the parameters to identify the data base and data segment to be operated on, then issues an input/output request to the appropriate operating system access method. If the desired data segment already exists in the data base buffers, no input/output operation is performed. When the requested operation is insertion, replacement, or

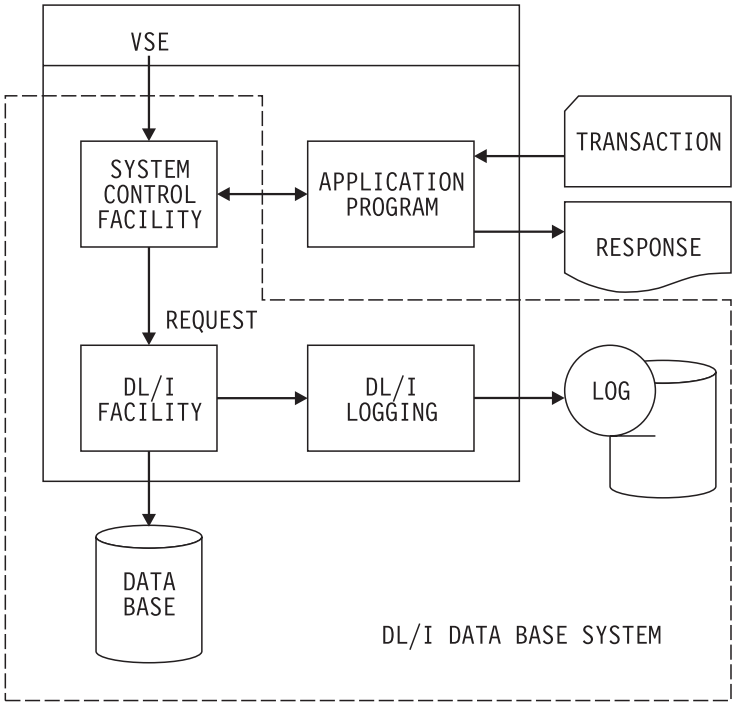


Figure 6-1. DL/I Batch System Environment

deletion, a record of the data base modification is written on the DL/I log, if logging has been requested.

Figure 6-2 shows DL/I operation under CICS/VS, where the data base requests are issued by the online application program. DL/I resolves any conflict that might be created when different transactions attempt to update the same data base simultaneously. Execution of the request is identical to that in the batch environment.

Figure 6-3 shows DL/I multiple partition support (MPS) operation. All DL/I requests from an MPS batch partition are passed to an online task associated with it, called the batch partition controller (BPC). The BPC then issues a call to the DL/I facility, like other online programs. After completion of the requested function, the results are made available to the MPS batch application program.

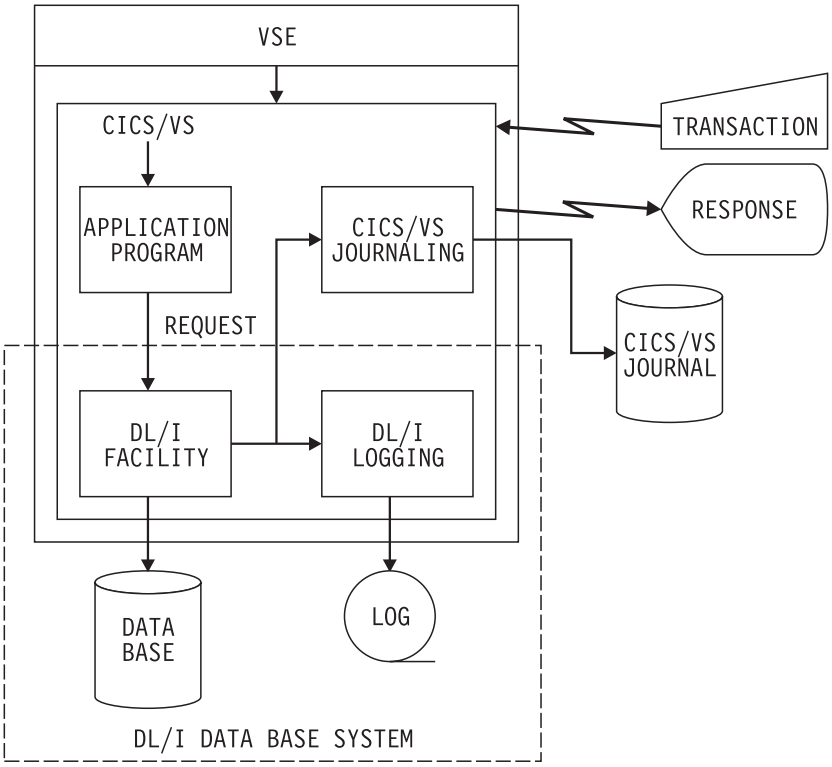


Figure 6-2. DL/I Online System Environment

DL/I System Requirements

DL/I consists of a set of macros and a set of preassembled modules. The modules contain the executable code necessary to operate DL/I. They are in object form and must be link-edited and cataloged into a core image library during installation. They are loaded from the core image library during DL/I system initialization.

The supplied macros must be cataloged into a source statement library before performing data base description and program specification block generations.

The DL/I system uses the data management access method modules of the Virtual Storage Access Method (VSAM) and Sequential Access Method (SAM) for the physical storage of segments. Each of the DL/I data base access methods described in chapter 4 uses one of these VSAM or SAM access methods. Simple HSAM and HSAM use SAM. All the others use VSAM.

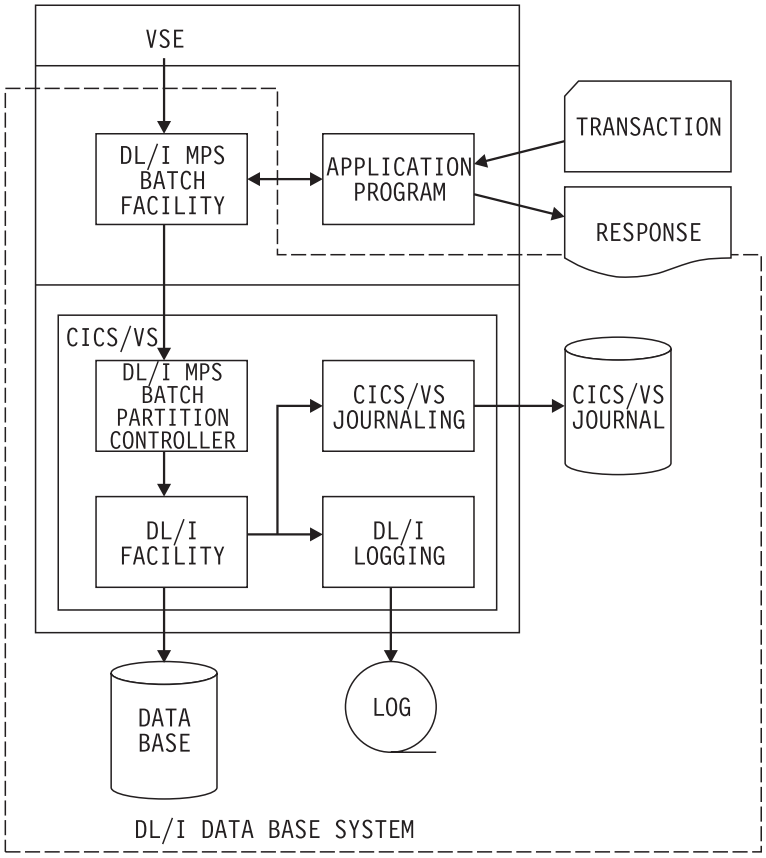


Figure 6-3. DL/I MPS System Environment

Requirements for Execution

Before application programs can use the DL/I facilities to handle information in DL/I data bases, there are a number of things that must be done. These are:

- DBD generation
- PSB generation
- ACB generation
- Data base space allocation
- Compilation and link-editing of application programs.

Figure 6-4 shows the program elements necessary for batch and online data base processing.

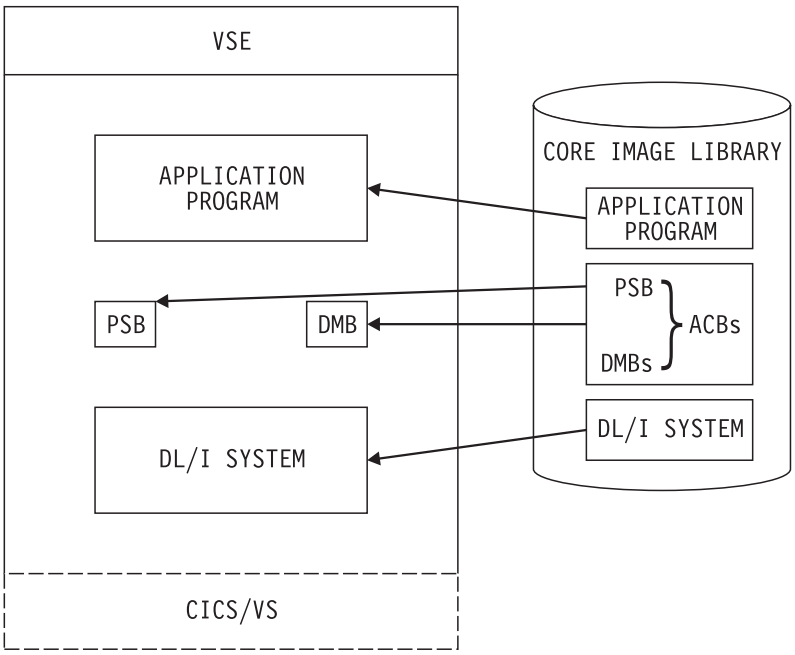


Figure 6-4. Essential DL/I Program Elements for Batch and Online Execution

Figure 6-5 shows the different configuration that is necessary for running multiple partition support in addition to online execution. All control blocks, CICS/VS, and most of the DL/I system are in the online partition with the online application programs. The MPS batch program runs in a separate partition containing the application program and some DL/I code needed as an interface with the DL/I system in the online partition.

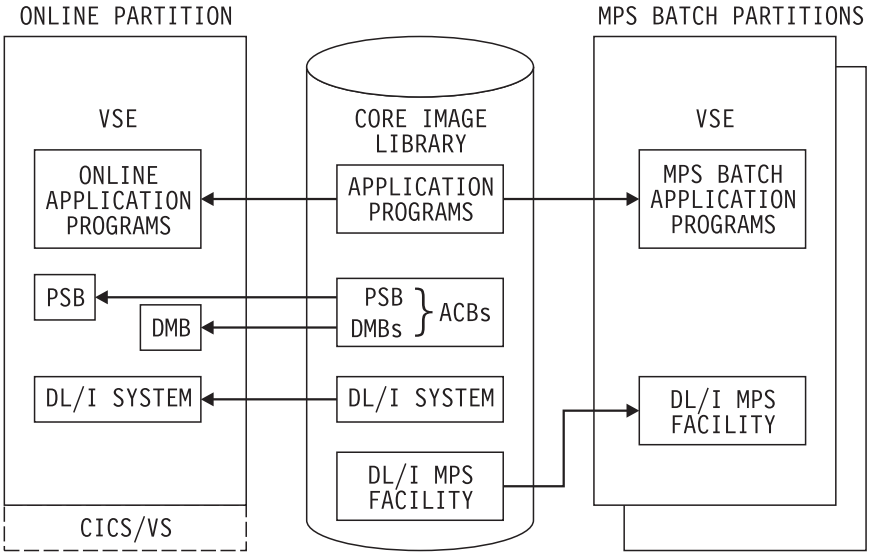


Figure 6-5. Essential DL/I Program Elements for MPS and Concurrent Online Execution

Data Base Description (DBD) Generation

Before a DL/I data base can be created and used, the data base must be described to DL/I. This is accomplished by the data base description (DBD) generation procedure. Figure 6-6 shows the execution of a DBD generation.

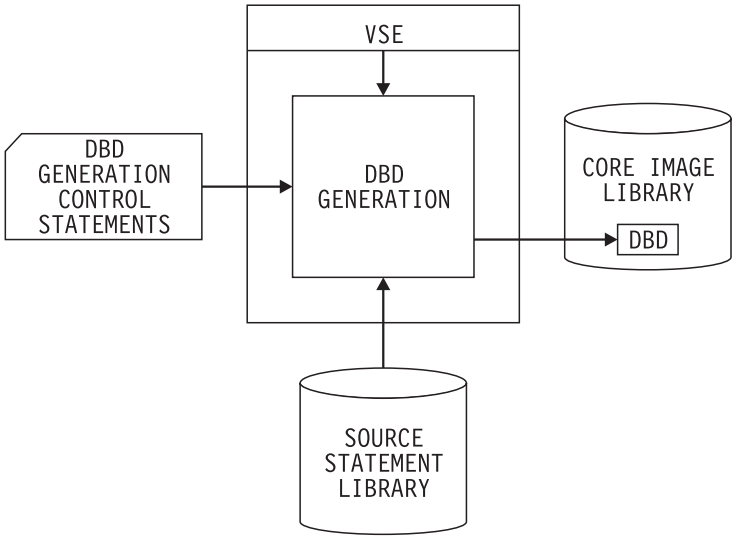


Figure 6-6. DBD Generation Required For Every Data Base

DBD generation is the execution of DL/I macro instructions to create a description of the characteristics of a data base. You can execute the DBD generation procedure as a normal application program job, or you can use the Interactive Macro Facility (IMF). More about DBD generation is found in chapter 5. *DL/I DOS/VS Resource Definition and Utilities* gives the details of how to code DBD macros and perform a DBD generation.

Program Specification Block (PSB) Generation

Before an application program can make use of data bases, a program specification block (PSB) generation must be performed for that program. The PSB defines the logical structure of the data bases as seen by the program. Figure 6-7 shows the execution of a PSB generation.

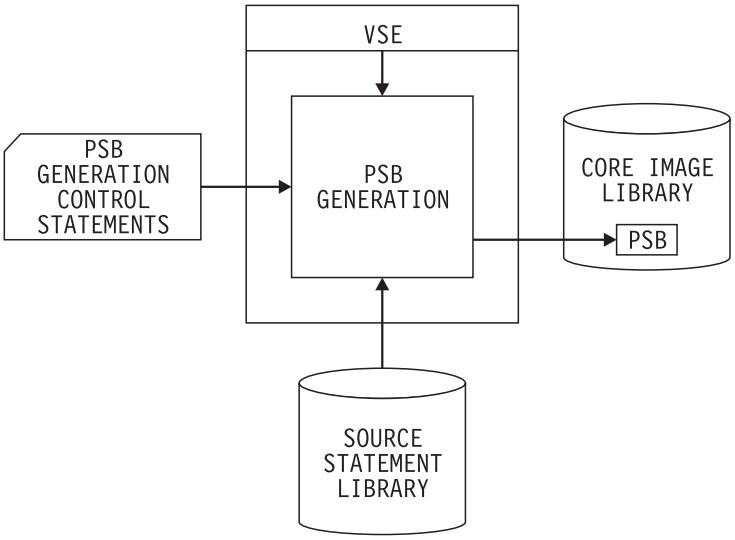


Figure 6-7. PSB Generation Required for Every Application Program

PSB generation is the execution of DL/I-supplied macro instructions to define an application program's use of one or more data bases. You can execute the PSB generation procedure as a normal application program job, or you can use the Interactive Macro Facility (IMF). There is more about PSB generation in chapter 5. *DL/I DOS/VS Resource Definition and Utilities* describes the details of how to code PSB macros and perform a PSB generation.

Application Control Blocks Creation and Maintenance

The previously defined physical (DBD) and logical (PSB) data structures must now be tied together so that DL/I can provide the correct data base management services for the application program. This is done, prior to program execution, through the creation of internal control blocks called application control blocks (ACBs). The ACBs are created by the application control blocks creation and maintenance utility. Figure 6-8 shows, in diagram form, the execution of the utility as a normal application program. Figure 6-9 shows the execution of the ACBGEN utility when the DL/I Documentation Aid option is specified. The DL/I Documentation Aid facility is described in Chapter 9, "Data Administration and Security". More about ACB creation and maintenance is found in chapter 5. *DL/I DOS/VS Resource Definition and Utilities* describes the details of how to use the ACB creation and maintenance utility.

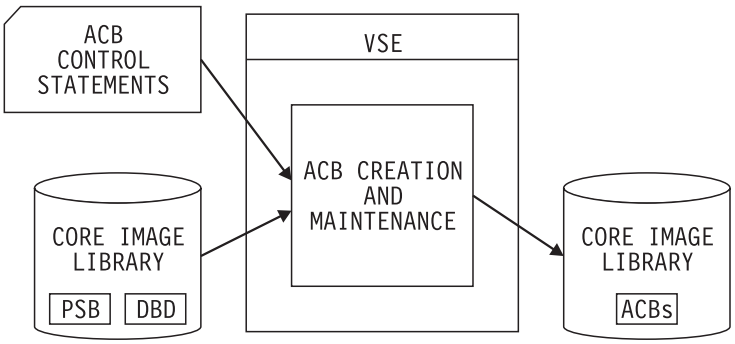


Figure 6-8. DL/I ACB Creation and Maintenance Required for Each PSB

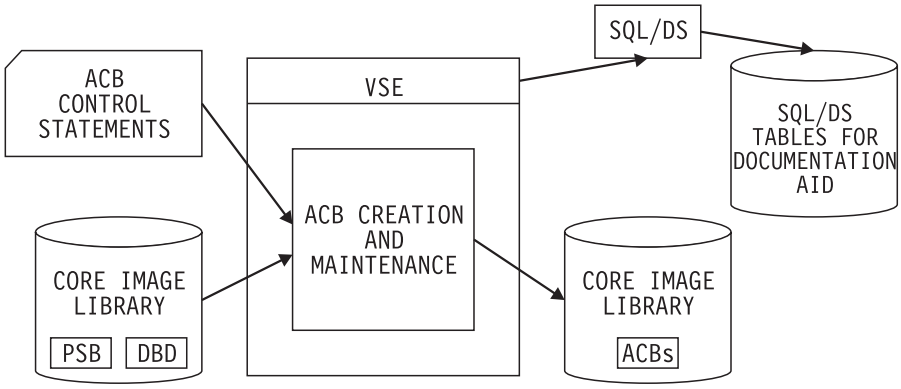


Figure 6-9. DL/I ACB Creation and Maintenance Required for Each PSB Using the Documentation Aid Option

Data Base Space Allocation

In addition to the functions described in the preceding sections, the files for every simple HISAM, HISAM, HD randomized, and HD indexed data base must be defined to VSAM by using the DOS/VS VSAM Access Method Services utility. The DEFINE command establishes the original definition of VSAM files, the VSAM master catalog, and VSAM data areas. Refer to *Using VSE/VSAM Commands and Macros* for further information concerning the DEFINE command.

Application Programs

At this point, the DL/I system and the DL/I data bases are ready for use. The final requirement is for the application programs that create and maintain the data bases that you have defined. The details of how to write application programs using DL/I are found in *DL/I DOS/VS Application Programming: High Level Programming Interface* or in *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces*.

Program Execution in the DL/I Operating Environments

Online

System Execution

Initialization of CICS/VS includes loading the DL/I facility and opening the DL/I log, if used. It does not necessarily include opening DL/I data bases for online access, since DL/I allows data bases to be added or removed during operation of the online system.

System Control Flow

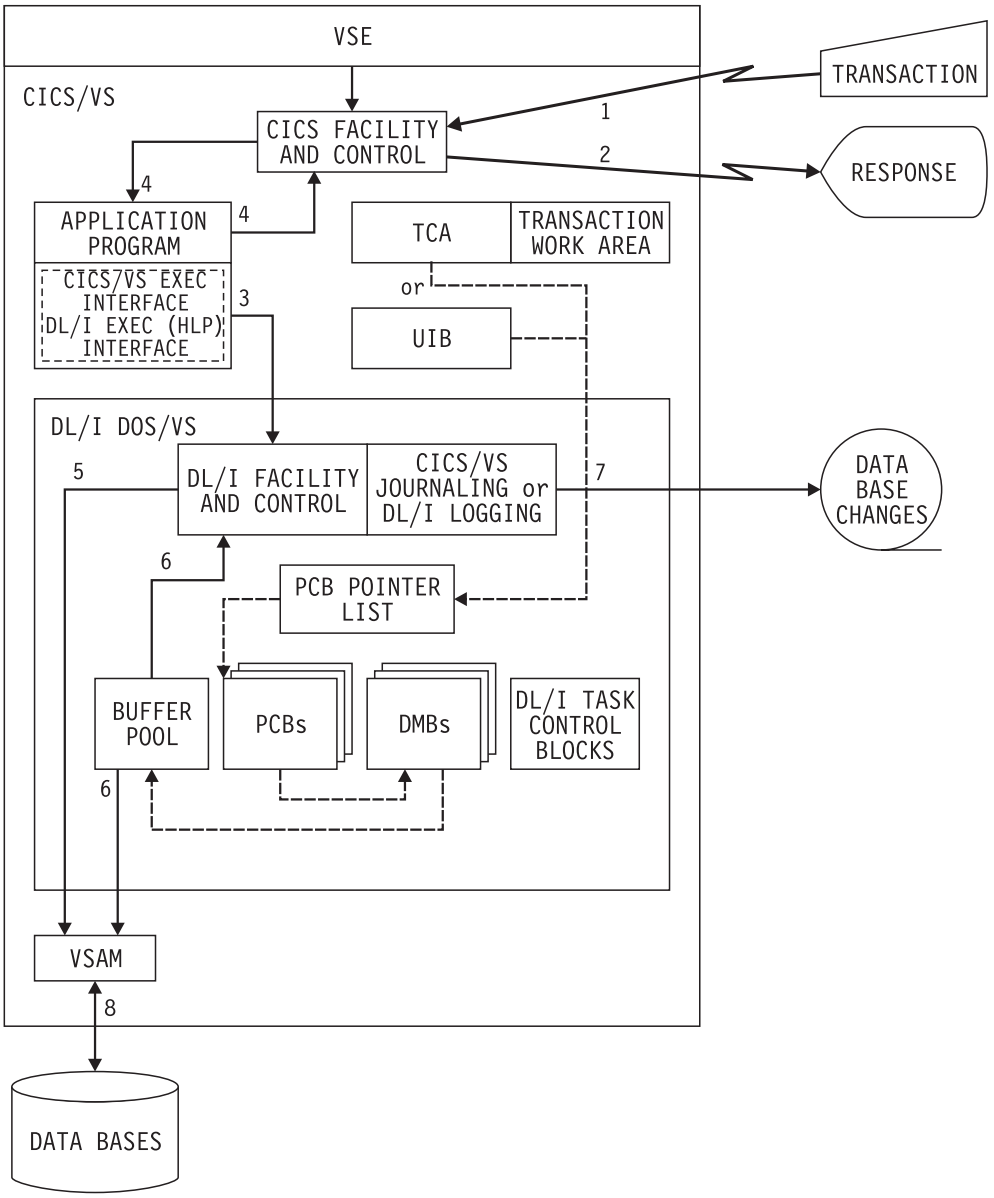


Figure 6-10. Online Data Base System Flow

Figure 6-10 shows the online data base system control flow under CICS/VS after CICS/VS initialization is complete.

Arrows 1 and 2 in the figure indicate CICS/VS transaction input and output between the terminal and the application program. CICS/VS creates a task based on either an input message from a terminal or an activity initiated by CICS/VS. If an application program is needed that is not already loaded, CICS/VS loads the program. All application programs that use DL/I must have a language interface program or CICS/VS EXEC stub link-edited with the program. The language interface accepts data base requests from the application program and passes control to the DL/I data base modules (arrow 3).

After the online program begins execution, the following functions are performed:

- The program issues a DL/I scheduling request to acquire a PSB prior to issuing requests involving data. This is necessary because, unlike the batch environment, many CICS/VS transactions accessing the same data base can be active at one time. Concurrent updating and deletion of data bases, as well as the possibility of deadlock conditions, makes it necessary for DL/I to schedule resources selectively. In scheduling a PSB for a program, DL/I considers the processing options specified, the access method type, and segment sensitivity.
- The application program makes CICS/VS requests. These can include transient data input or output, storage requests, and non-DL/I file activity (arrows 4). Control is passed back and forth between CICS/VS and the application program (or DL/I) during processing of these CICS/VS requests.
- After control is passed to the DL/I modules for execution of a data base request, the following functions are performed:
  - The parameters in the request are checked for valid content.
  - If the data base request involves segment retrieval, the information contained in control blocks and data base buffers is used to attempt to satisfy the request. If the request can be satisfied, the desired data segment is placed in the input/output work area named by the application program in the data base request.
  - If the retrieval request cannot be satisfied with information already contained in the data base buffer pool, the VSAM data management access method modules are invoked (arrow 5).
  - The VSAM modules perform the necessary input requests to bring the needed data into the data base buffer pool (arrows 6 and 8).
  - If the data base request involves updating or deletion, the segment must first be retrieved. Then the segment is deleted or updated in the buffer pool (arrow 6).
  - If the data base request involves segment insertion, the segment is placed in the data base buffer pool (arrow 6) and subsequently written onto direct access storage (arrows 5 and 8).
  - DL/I data base modifications are made to the information in the buffer pool. The changes are written onto disk either when the buffer pool needs more buffers, or when the application program terminates or takes a checkpoint or CICS/VS sync point (arrows 6 and 8).
  - When the data base request involves segment insertion, updating, or deletion, a record of the data base modification is placed on the DL/I system log or CICS/VS journaling tape (arrow 7). This logical information can be used later for data base recovery or reconstruction with the DL/I utility programs.
- When the program no longer needs the services of DL/I a TERM request is issued. This request is the opposite of a DL/I scheduling request. It causes the program to relinquish all DL/I resources, including the PSB. The program is no longer able to access DL/I data bases until the next scheduling request is issued. The same thing happens when a CICS/VS “RETURN” or a CICS/VS “SYNCPOINT” is issued.

MPS Batch

System Execution

Multiple partition support (MPS) uses the DL/I resources and multitasking facilities of DL/I-CICS/VS. MPS is started in the CICS/VS partition in either of two ways:

- A terminal operator enters transaction CSDA
- Automatically by CICS/VS during its system initialization processing. To cause the automatic start, an entry for the DL/I MPS start program (DLZMSTR0) must have been made previously in a startup CICS/VS program list table (PLT).

Once MPS has been initialized, the batch partition can be started in an operating system or ICCF partition by executing DLZMPI00 (// EXEC DLZMPI00....), with a parameter statement that specifies DLI or DLR, the program name, and the PSB name. The statement is submitted to DL/I through either the system input (SYSIPT) or system log (SYSLOG) device.

When a DL/I application program is initiated in an MPS batch environment, the first step is the loading of the application program. DL/I then signals a DL/I routine in the CICS/VS online MPS partition, which initiates a mirror task in the online partition. The mirror task schedules the PSB for the MPS application. Control is then returned to DL/I, which returns it to the application program.

When the application makes a DL/I request, DL/I signals the mirror task and provides a pointer to the request. The request is executed by the mirror task. At completion of the request, MPS notifies DL/I that the request is complete. It passes back the request status and the requested data (if any). DL/I passes the information on to the application program through the DIB (or PCB mask, depending on whether HLPI or the CALL interface was used to make the request) and the user's I/O area.

Logging is performed in the online partition, using the CICS/VS journaling services.

MPS can be terminated by entering the CICS/VS stop transaction CSDD, or automatically if an entry has been made previously in a shutdown PLT for the program DLZMSTP0.

System Control Flow

Figure 6-11 shows the control flow for an MPS batch program after it has been given control. A parameter address list is provided by DL/I when the application program is entered. This list provides data base PCB addresses that are used by the application program when issuing data base requests (see arrow 1).

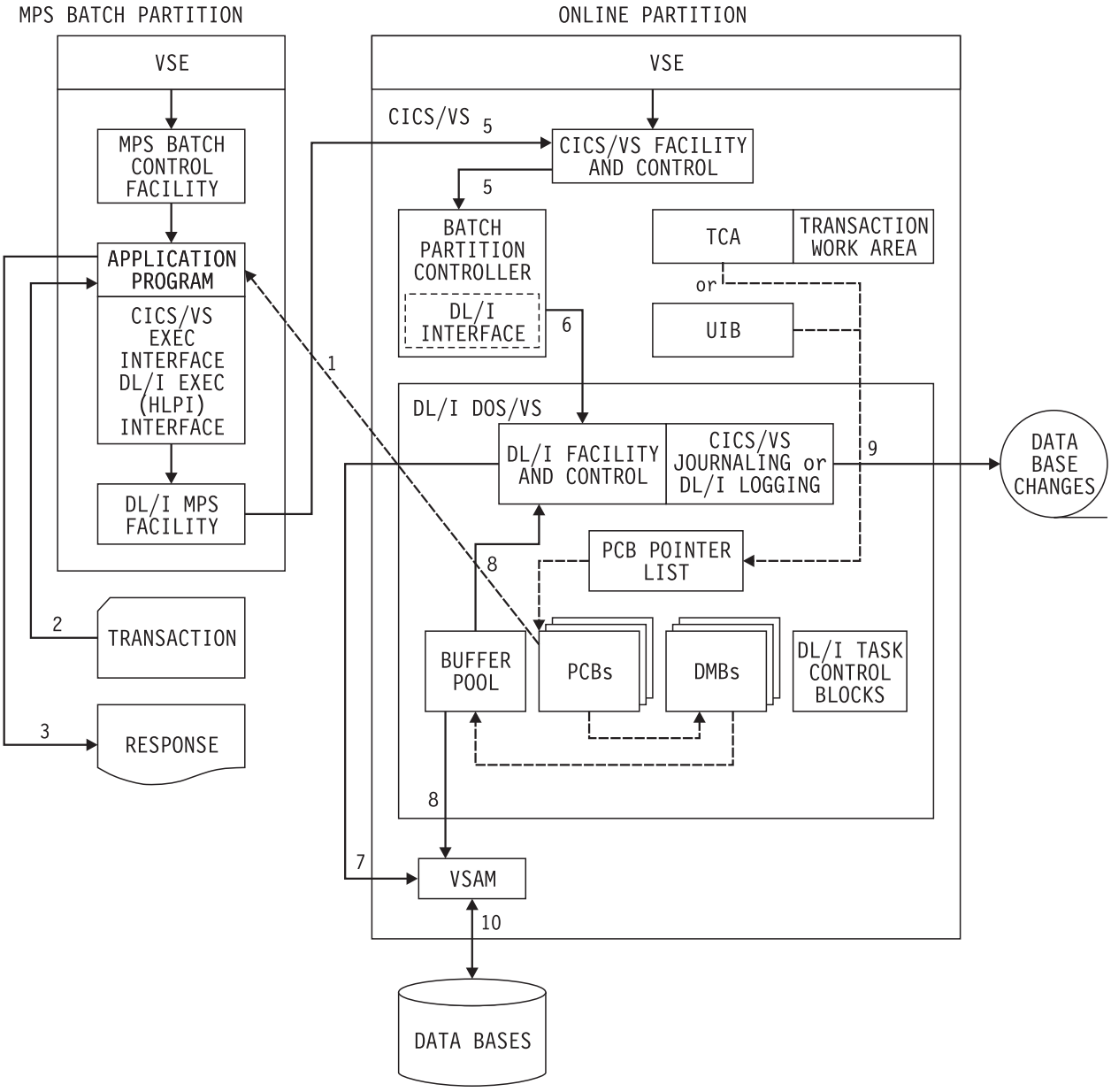


Figure 6-11. MPS Batch Data Base System Flow

Arrows 2 and 3 in the figure indicate the input and output functions of the application program. These functions are the same as in non-DL/I application programs.

All application programs that operate under DL/I have a “language interface” automatically link-edited (AUTOLINK linkage editor function) with the application program. The purpose of the language interface is to accept a data base request from the application program and pass control to the DL/I MPS facility (arrow 4). It does this by:

- Saving the application program's registers in a user save area
- Locating the next DL/I routine entry point address and branching to it.

The MPS facility communicates with the online mirror task created by DL/I for this MPS batch job. This task is the batch partition controller (BPC) (arrows 5). The

BPC issues the DL/I request (arrow 6). The call parameter list is validated by DL/I as for an ordinary online task; except for parameter addresses, which were verified by the MPS facility in the batch partition.

The same operations occur then as in regular online processing (arrows 7 - 10):

- The parameters in the request are checked for valid content.
- If the data base request involves segment retrieval, the information contained in control blocks and data base buffers is used to attempt to satisfy the request. If the request can be satisfied, the desired data segment is placed in the input/output work area named by the application program in the data base request.
- If the retrieval request cannot be satisfied with information already contained in the data base buffer pool, the VSAM data management access method modules are invoked (arrow 7).
- The VSAM modules perform the necessary input requests to bring the requested data into the data base buffer pool (arrows 8 and 10).
- If the data base request involves updating or deletion, the segment must first be retrieved. Then the segment is deleted or updated in the buffer pool (arrow 8).
- If the data base request involves segment insertion, the segment is placed in the data base buffer pool (arrow 8) and subsequently written onto direct access storage (arrows 7 and 10).
- DL/I data base modifications are made to the information in the buffer pool. The changes are written onto disk either when the buffer pool needs more buffers, or when the application program terminates (arrows 8 and 10).
- When the data base request involves segment insertion, updating, or deletion, a record of the data base modification is placed on the CICS/VS journaling tape or DL/I system log (arrow 9). This logical information can be used later for data base recovery or reconstruction with the DL/I utility programs.

Programming Note

In some cases you may find a marked increase in elapsed time for DL/I application programs run under MPS, as compared to the elapsed time when the same programs run as non-MPS jobs. See “MPS Performance Considerations” in chapter 4 for more information.

Batch

System Execution

When DL/I is initiated, the control blocks associated with the application program are automatically obtained and initialized. This control block initialization is part of the execution of the batch processing job step. It precedes the actual loading of the application program.

The DL/I system environment is controlled by the system control facility. The primary functions of the system control facility are to:

- Initiate the data base system block loading process by passing control to the DL/I control block load and relocate module (Figure 6-12).
- Load the DL/I facility and data base logger modules, then pass control to the application program (Figure 6-13).
- Terminate data base system batch execution by returning control to the operating system.

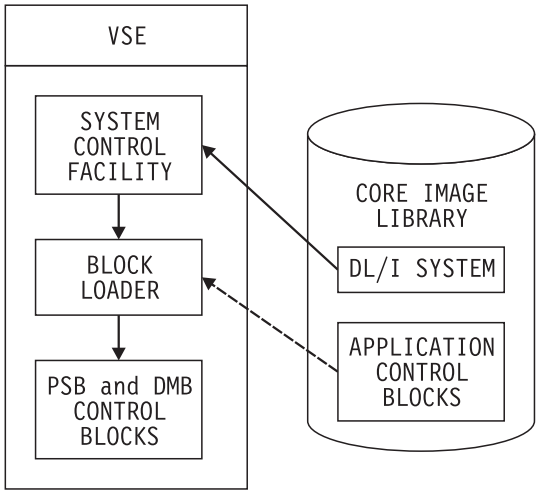


Figure 6-12. Initializing the Data Base System

During DL/I system initialization, parameter information is submitted to DL/I through either the system input (SYSIPT) or system log (SYSLOG) device. This parameter information includes the names of the application program to be executed, and the associated PSB (as specified in the PSB generation).

The ACB PSB is loaded from a core image library. The PSB contains at least one PCB. Each PCB contains the name of a DMB. The required DMBs are also loaded from a core image library and initialized.

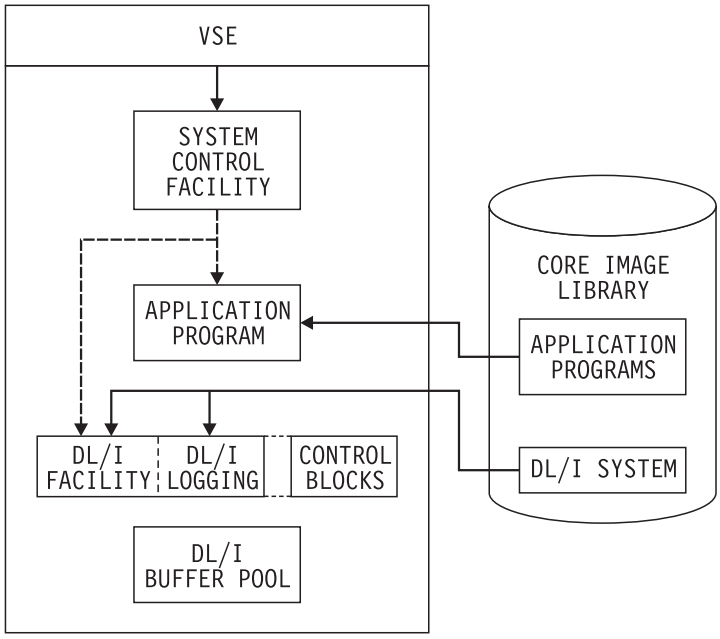


Figure 6-13. Initializing the Batch System

The number of buffer subpools required for job execution can be specified in the application program parameter statement. (By default, each subpool contains 32 fixed length buffers). The buffers are used for the VSAM ESDS and for work areas required by the application. During DL/I initialization, each data base is assigned to a buffer subpool. The assignment is based on the size of the data base control interval, the number of subpools allocated, and any user-specified allocations.

After the buffer pools are assigned, the remaining system functions are initialized (Figure 6-13). This includes loading the DL/I modules and the data base logger module. The DL/I action modules and VSAM are reentrant and may optionally reside in the operating system shared virtual area (SVA). When the modules are placed in the SVA, a single copy of the code can be used by multiple partitions within the system.

Finally, the application program to be executed is loaded from a core image library, and control is given to it.

System Control Flow

Figure 6-14 shows the data base system control flow after the application program has been given control. A parameter address list is provided by the DL/I system control facility when the application program is entered. This list provides data base PCB addresses that are used by the application program when issuing data base requests (see arrow 1).

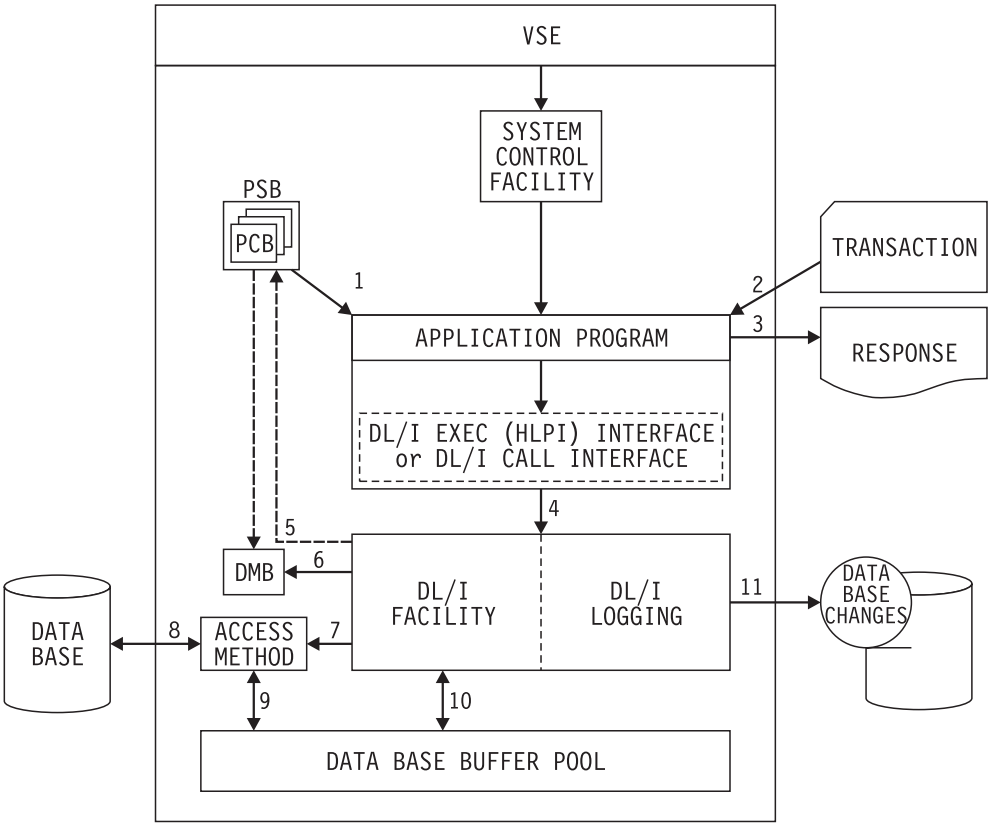


Figure 6-14. Batch Data Base System Flow

Arrows 2 and 3 in the figure indicate the input and output functions of the application program. These functions are the same as in non-DL/I application programs.

All application programs that operate under DL/I have a “language interface” automatically link-edited (AUTOLINK linkage editor function) with the application program. The purpose of the language interface is to accept a data base request from the application program and pass control to the DL/I facility (arrow 4). It does this by:

- Saving the application program's registers in a user save area
- Locating the next DL/I routine entry point address and branching to it.

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

- The parameters in the request are checked for valid content (see arrows 5 and 6).
- If the data base request involves segment retrieval, the information contained in control blocks and data base buffers is used to attempt to satisfy the request. If the request can be satisfied, the desired data segment is placed in the input/output work area named by the application program in the data base request.
- If the retrieval request cannot be satisfied with information already contained in the data base buffer pool, the appropriate data management access method modules are invoked (arrow 7).
- The data management access method modules perform the necessary input requests to place necessary data in the data base buffer pool (arrows 8 and 9).
- If the data base request involves updating or deletion, the segment must first be retrieved. Then the segment is deleted or updated in the buffer pool (arrow 10).
- If the data base request involves segment insertion, the segment is placed in the data base buffer pool (arrow 10) and subsequently written onto direct access storage (arrows 8 and 9).
- DL/I data base modifications are made to the information in the buffer pool. The changes are written onto disk either when the buffer pool needs more buffers, or when the application program terminates (arrows 8 and 9).
- When the data base request involves segment insertion, updating, or deletion, a record of the data base modification is placed on the DL/I system log tape or disk (arrow 11). This log information can be used later for data base recovery or reconstruction with the DL/I utility programs.

DL/I in the CMS/DOS Environment

Batch DL/I programs can be written and tested in the CMS/DOS environment. This includes all programs written in COBOL, PL/I, and Assembler language. RPG II is also included, but requires BSE, Release 2.

Data base description generation and program specification block generation can also be executed. However, the application control block generation must be submitted to a VSE virtual machine for execution. DL/I utilities are not supported and must also be executed in a VSE virtual machine.

This support lets you:

- Interactively code DL/I control blocks and application programs that contain imbedded DL/I requests.
- Store and maintain macros used to generate DL/I control blocks, and programs created under CMS, in the CMS library. Production libraries can be isolated from the test environment.
- Modify and compile programs using the CMS/DOS text manipulation and EXEC facilities. CMS EXECs are similar to VSE procedures.
- Link-edit and execute batch DL/I programs either interactively or in CMSBATCH. Online DL/I application programs requiring access to CICS/VS must be submitted to a VSE virtual machine for link-editing, cataloging, and execution.

The following restrictions apply:

- All the existing guidelines and restrictions that apply to VSAM data set creation, maintenance, and application program use apply to DL/I data sets.
- The CMS/DOS restriction on writing to sequential files applies to SHSAM and HSAM.
- To assemble a DBD or PSB under CMS/DOS, you must first copy the following DBDGEN and PSBGEN macros from the operating system source statement library to a CMS MACLIB.

DBDGEN Macros

DATASET	DLZCKDDN	LCHILD
DLZCAP	DLZHIERS	DLZALPHA
DLZDEVSI	DLZSOURS	DLZDBGLB
DLZSETFL	FINISH	DLZSEGPT
FIELD	DBDGEN	DLZXTDBD
XDFLD	DLZLRECL	SEGM
DBD	DLZXPARM	

PSBGEN Macros

DLZCKOPT	SENSEG	VIRFLD
PSBGEN	PCB	DLZALPHA
DLZPCBPD	SENFLD	

For more information about operating under CMS/DOS, see the *VM/CMS Users Guide*, GC20-1819, or the *VM/SP User's Guide*, SC19-6210, and the *VM/370 System Programmer's Guide*, GC20-1807, or the, *VM/SP System Programmer's Guide*SC19-6203. For information on compiling COBOL programs, see the *VM/370 CMS Users Guide for COBOL*, SC28-6469.

Testing Programs with a Test Data Base

Application programs that access data bases must be tested before they are transferred to production status, the same as any other program. In the case of data base application programs, a test data base is needed to avoid the risk of damaging a production data base. This section describes requirements that are important for data base program testing, and suggests some ways of generating test data bases.

Program Testing Requirements

Test Requirements

Depending on your system configuration, user requirements, and the design characteristics of your data base system, you may need to test:

- That DL/I request sequences will execute and that the results are as they should be.
  - This kind of test may require only a few records. In this case, you can use the DL/I test program, DLZDLTxx, to produce the records.
  - You may want to extract records from an existing data base. In this case, the DL/I DOS/VS Space Management Utilities IUP (Program Number 5796-PKF) may meet your requirements.
- That requests will execute through all possible application program decision paths.

  - In this case, you may need to approximate your production data base. The Space Management Utilities can help you in this case too.
- How performance compares with that of a model. For example, for system test or regression tests.
  - For this kind of test, you may need a copy of a subset of the production data base. Again, the Space Management Utilities can be of help.

Data Base Requirements

To perform the tests outlined above accurately and completely, you need test data bases that approximate, as closely as possible, the production data bases. Often, a test data base can be a copy of a subset of a production data base. You can copy and modify the DBD of the production data base. If the production data base is defined in a data dictionary, that definition will give you much of the information you need.

If you need access to your online production data base to test your online or MPS batch application programs, you should consider using CICS/VS intersystem communication (ISC). In an ISC environment, you would set up a production CICS/VS system and a test CICS/VS system. Your production system would control access to your online production data bases. However, you could grant limited remote access to these data bases to application programs running in your test system. In this environment, your test system would have access to your production data. At the same time, using the CICS/VS dynamic backout facility, your production system would be protected against failures in your test system.

By selecting the PSBs, which the ISC mirror could schedule on behalf of a remote system, you could prevent update access to your production data bases from being granted. You could also restrict the view of the data base, the segment types, and the fields within these segments that could be accessed by a remote system.

Sample Data Requirements

It is important for the sample data to approximate the real data so that the two will have the same characteristics; such as the range of field values. The kind of sample data needed depends on whether you're testing requests or program logic.

- To test requests, you need values in only those fields that are sequence fields or that are specifically referenced in the requests.
- To test program logic, you need data in all fields accessed in the program logic (for adds, compares, etc.)

Again, you might use a copy of a subset of the real data base. However, if you do, find out if any fields contain sensitive data and load fictitious data in those fields in the test data base.

Application Program Requirements

In order to design a test data base that effectively tests the operational application programs being developed, you must be familiar with those programs. Much of the information you need is in the application program design documentation, in the PSBs, and in the data dictionary if you use one.

Generating a Test Data Base

You can develop a test data base just as you'd develop a production data base. To do that, perform the tasks described in chapters 4 and 5 of this manual, keeping in mind the special requirements for test data bases. If your installation has testing standards and procedures, you should follow them.

Try to use the same development aids to develop the test data base that you use for production data bases. That will help to avoid problems caused by differences that might occur between the two types of data base.

DL/I Test Program, DLZDLTxx

If you need a test data base with relatively few data base records (to test DL/I request sequences, for example) you can use this program. If you don't have a machine readable data base to begin with, you can define a PCB, then use DLZDLTxx to insert segments. This eliminates the need for a load program to generate your test data base. Information about this test program is found in "Module and Program Aids," in *DL/I DOS/VS Diagnostic Guide*.

DL/I DOS/VS Space Management Utilities IUP (SMU)

The Space Management Utilities IUP are a group of IBM programs that are used as data base administration productivity aids. You can use one of them, the segment restructure utility, to create a test data base by structuring segments and hierarchies based on a subset of the actual data base. *Space Management Utilities Product Description/Operations Manual*, SH20-2107, gives details on how to do this.

DB/DC Data Dictionary

The *DOS/VS DB/DC Data Dictionary* (Program Number 5746-XXC) is a development and administrative tool used to manage information about an installation's data processing resources. If you have your production data base defined in the data dictionary, you can use it to help define your test data base. You can search the definitions in the dictionary to determine which elements you want to include in your test data base. Then, you can copy selected parts of the data dictionary definitions and assign the copy a test status. This becomes the definition of your test data base.

You can also use the DBD and PSB generation facility of the data dictionary to provide DBDs and PSBs for the test data base.

The DB/DC Data Dictionary is described in *DOS/VS DB/DC Data Dictionary General Information Manual*, GH20-9193.

Chapter 7. Loading Data Bases

This chapter tells you how to write and use application programs to initially load data bases. There are two sections:

1. Load Program
2. Sample Load Program

The first section tells you how to write a basic data base load program, then goes on to describe the additional prefix resolution requirements of HD access method loading programs. The second section uses the sample load program shipped with DL/I as an example.

Load Program

After a DBD has been generated and space has been allocated, a data base is ready for loading. Loading is the process of storing data in the data base for the first time. This section describes what is required to accomplish the loading.

Basic Loading

Initial data base loading is done by an application program written by someone in your installation. This program must run in the *batch* environment. Load programs cannot run in the online or MPS batch environments. Input to the loading program is a sequential file containing the data that is to become the data base records. This data may be newly created, or it may already exist in one or more files. In the latter case, the files may have to be sorted and merged to get the records into the correct sequence for the loading program. In some cases, a program will have to be written to clean up or correct data in the original files. For instance, redundant data should be removed.

If the access method specified for this data base is simple HSAM, HSAM, simple HISAM, HISAM, HD indexed, or HIDAM; the data base records must be presorted in ascending order of the value of the key field of the root segments. The load program must load them in this order. If the access method is HD randomized or HDAM, presorting of root segments is optional. If the access method is HSAM, HISAM, HD, HDAM, or HIDAM, and the data base record contains more than the root segment, all of the segments in the data base record must be presorted in order of their hierarchical relationship and the value of their key fields, if any. The load program then loads them in hierarchical order.

The load program reads the input file and builds the segments in an I/O area, with the length and field placement of each as described in the physical DBD for that data base. The program inserts the segments in hierarchical sequence in the space reserved for the data base data set.

The load program uses only one type of DL/I request. This is either the DL/I High Level Programming Interface LOAD command, or the DL/I CALL interface ISRT statement. No other DL/I requests are permitted in this program. Details about the use of these requests in loading programs are given in *DL/I DOS/VS Application*

Programming: High Level Programming Interface or *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces*.

The processing option (PROCOPT) specified in the PCB(s) for this program must be L or LS. If LS is specified, the data base must be loaded sequentially. PCBs associated with load programs must refer to physical DBDs, not logical DBDs.

Figure 7-1 is a diagram of the basic loading process.

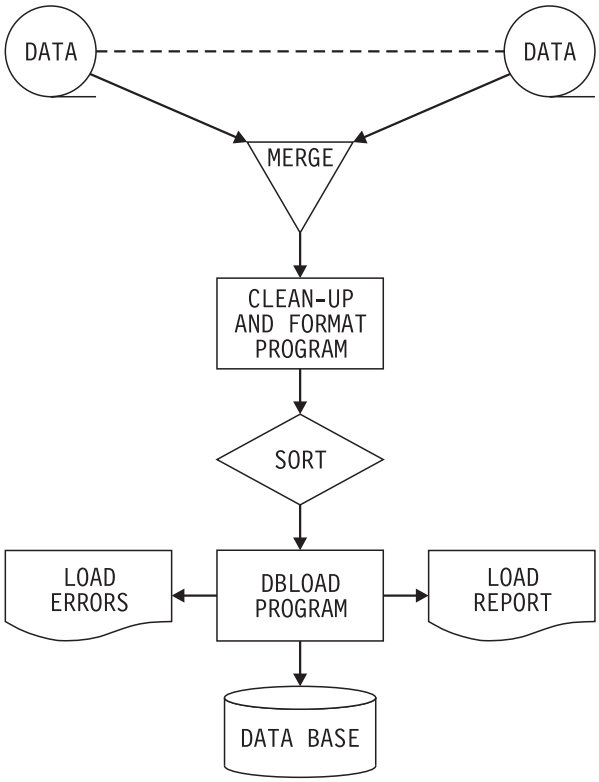


Figure 7-1. Basic Data Base Loading Process

Note: You must load at least one data base record in every data base that will be used online, before trying to use the online DL/I system.

Load Program Status Code Testing

As with any DL/I application program, DL/I will return a status code after the execution of each DL/I request in your load program. The codes that are returned, and how they should be handled, are somewhat different for the two DL/I interfaces. See *DL/I DOS/VS Application Programming: High Level Programming Interface* for details if your load program uses HLPI, or *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces* if it uses the CALL interface.

Loading HD Indexed or HIDAM Data Bases

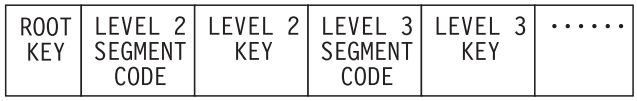
For initial loading of an HD indexed or HIDAM data base, you must specify PROCOPT=LS in the PCB. Data base records must be inserted in ascending root key sequence. Dependents of the root segment must be inserted after it, in hierarchical sequence.

Note: If you need to sort dependent segments into hierarchical sequence by segment type, you will have to write a program to prepare your input data

for the sort operation by adding a sort control field to each segment. The sort control field defines the hierarchical path to the particular segment and the sequencing within the path. The sort control field is made up of:

- The root segment key field
- The key fields of the segments in the hierarchical path between the root segment and this segment (one segment at each hierarchical level)
- The one-byte segment codes of those same segments.

The sort control field is laid out like this:



Notes:

1. The segment code is one byte containing a binary number from one to 255 that is used by DL/I, instead of a segment name, to identify a segment type. The values are assigned to the segment types in ascending sequence, starting with the root segment type and continuing through all dependent segment types, following the hierarchical sequence you defined in the DBD.
2. For every level, the key field length should be equal to the length of the longest segment key field on that level. Keys shorter than that in length should be left-adjusted and padded on the right with characters having a low value in the sort sequence.
3. If no sequence field has been defined in the segment, you should provide one. Its value could be a simple dependent-segment counter provided by a user-written “clean-up and format” program. Segments on the lowest level need not have a key field if no sequence field was defined; however, their sequence below their parent might be different after the sort.

When loading an HD indexed or HIDAM data base, DL/I also loads the primary index data base automatically. As each root segment is stored, DL/I generates the index segment for it and stores the index segment in the INDEX data base.

The INDEX data base is a KSDS. Index segments created both during and after initial loading are placed in this data set.

The data stored in an HD indexed or HIDAM data base is stored in an ESDS. The HD space search algorithm is used by DL/I to locate the most suitable space available for inserting each segment.

Loading HD Randomized or HDAM Data Bases

For initial loading of an HD randomized or HDAM data base, you must specify PROCOPT=L in the PCB. The data base records do not need to be sorted into root key order, but the segments must be inserted in hierarchical order. For performance reasons it is advantageous if the data base records are sorted into physical sequence. Physical sequence is the ascending sequence of the block and root anchor point values as generated by the randomizing algorithms. You can do this by passing each root key through the randomizing module for address conversion, then sorting on the generated address plus the root key value.

When loading a given data base record, your randomizing module generates a relative block (control interval) and anchor point number for the root segment of the record. These values are passed to DL/I. DL/I attempts to store the root segment in the control interval specified. If space is available in that control interval, the root segment is stored there. A four byte direct address pointer to the root segment is placed in the specified anchor point position in the anchor point area of that control interval. When enough space is not available in the control interval specified by the randomizing module, DL/I uses the HD space search algorithm to find the available space closest to that control interval. The root segment is stored in that space. A pointer to the root segment is placed in the original anchor point position and relative block number specified by the randomizing module.

When your randomizing module generates the same relative block and anchor point number for more than one root segment, the specified anchor point points to the root segment with the lowest key. The rest are chained to it in ascending key sequence through physical twin pointers. All root segments chained from a given anchor point are called synonyms since they all have the same relative block and anchor point numbers. Long synonym chains can affect performance. To keep them short, increase the number of root anchor points specified for each control interval in the primary (root addressable) area. (Specified in the ACCESS parameter in the DBD statement of DBD generation). It may also be necessary to modify your randomizing module. The DL/I DOS/VS Space Management Utilities IUP (Program Number 5796-PKF) can be used to analyze synonym chain lengths. See “Space Management Utilities IUP” in chapter 10 for details.

When a root segment is loaded into the primary area, the remaining segments in the data base record are stored following it until the value specified in the BYTES parameter of the DBD statement has been reached. Any remaining segments in the data base record are stored in the overflow area.

Differences Between HD Randomized and HD Indexed

When a new root segment is presented to HD indexed or HIDAM, it is placed in the next available block in the space allocated to the data base. An indexing segment is generated in the INDEX data base. The indexing segment contains, in its data portion, the sequence field of the root segment being inserted; and, in its prefix, the relative block address of the root segment.

When the same root segment is presented to HD randomized or HDAM, an address is generated for it in your randomizing module. The address consists of a relative block number and an anchor point number. If any root segment having a lower key field value already exists in that block and anchor point, the new root segment is chained to the existing root. Figure 7-2 shows these differences in diagram form.

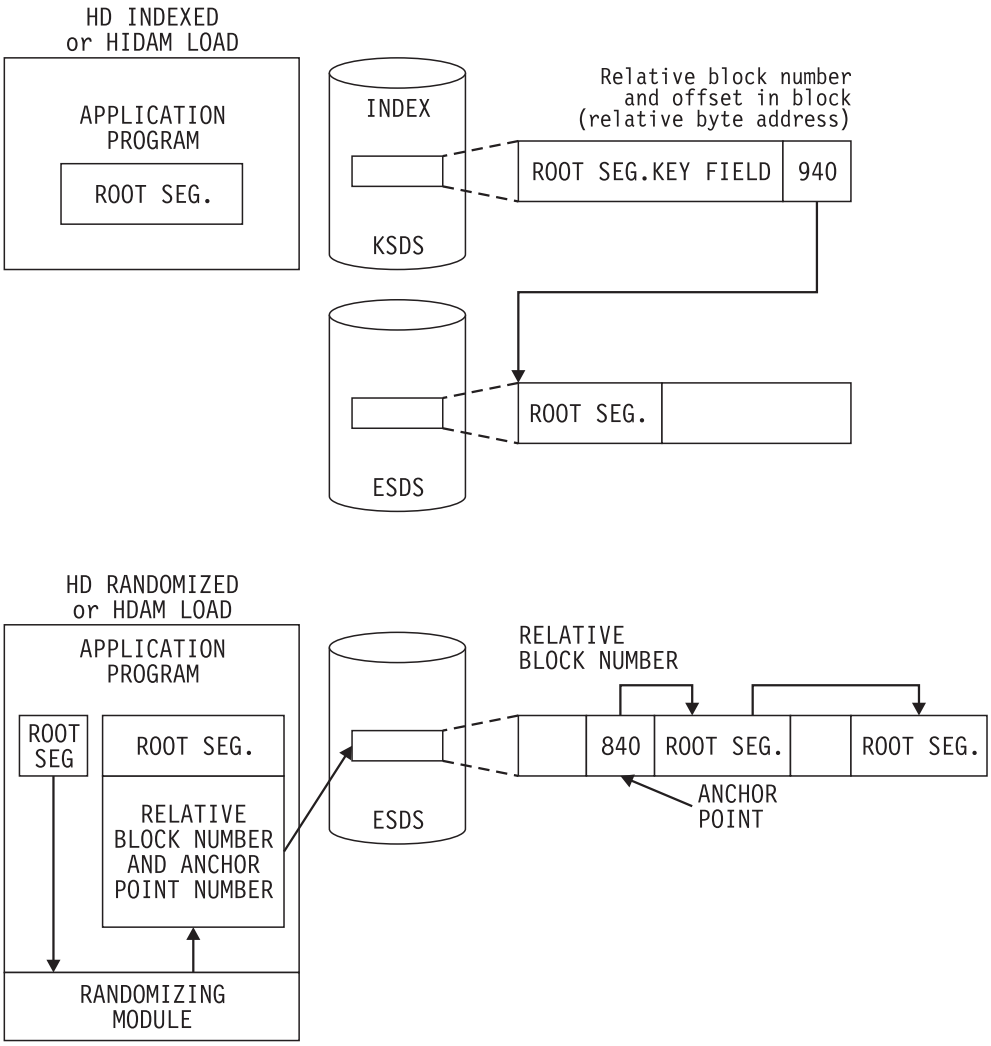


Figure 7-2. Comparison of HD Access Methods in Data Base Loading

Space Management Search Algorithm

The DL/I Space Manager acquires and frees data base space for HDAM, HIDAM, HD randomized and HD indexed access methods. The desired location for the segment is passed to it by the DL/I action module that is inserting a segment. When data base space is requested, the space manager attempts to get that space at a point that will give the best performance when retrieving the segment in subsequent applications. If the requested space does not exist at the optimum location, the space manager then scans the data base for space as near the optimum location as possible. The scan delta provided at data base generation time by the DATASET SCAN= parameter is used when scanning for space on cylinders (or FBA blocks) adjacent to the most desirable cylinder (or FBA block).

For an HDAM root segment, the best location is in the VSAM Control Interval (CI) that contains the root anchor point or root segment that will be pointing to it. An HDAM dependent segment is best located in the CI containing the segment that points to it. This may be either the parent or twin of the segment being inserted.

For a HIDAM root segment, the best location is the CI with the root containing the next highest key. This is true because when DL/I positions itself, it often searches

HIDAM data bases until the next highest key is found. The dependent segment placement is similar to the HDAM explanation above.

The HD access method results in either an HDAM or HIDAM structure, and the same search rules explained above also apply.

The sequence of searches used by the space manager to locate the space for the segment to be inserted are described below:

- The optimum point as previously described. (Note: This control interval will already be in the buffer pool).
- In any control interval that is already in the buffer pool, and is on the same track.
- In any control interval that is already in the buffer pool, and is on the same cylinder.
- In any control interval that is on the same track, and has sufficient space according to the bit map. (Note: The bit map contains a '1' in the bit position that corresponds to each control interval that has enough space for the largest segment).
- In any control interval that is on the same cylinder, and has sufficient space according to the bit map.
- In any control interval plus or minus n control intervals, and has sufficient space according to the bit map. (Note: The value for 'n' is determined by the SCAN= parameter previously described. The bit map search starts at the cylinders (or FBA blocks) adjacent to the most desirable cylinder (or FBA block) and continues outward in both directions until the value 'n' is reached. If 'n' is zero, this search is skipped).
- In any control interval that is at the end of the dataset, and is already in the buffer pool.
- In any control interval that is at the end of the dataset, and has sufficient space according to the bit map.

Loading Requiring Prefix Resolution

The data base loading program discussed above under “Basic Loading” is required in every initial loading situation. However, there are situations that can arise that the loading program cannot handle. These involve the resolving of pointers in the prefixes of logically related segments. During the initial loading of data bases having logical relationships or secondary indexes, the sequence in which logical parent segments is loaded is normally not the same as the sequence in which the logical child segments are loaded. This makes it impossible to generate all of the pointers just mentioned. To handle this situation, DL/I provides two things:

- A work file that is automatically created whenever you load a data base that contains logical child and/or logical parent segments. This work file contains the information needed to update the pointers.
- A set of utility programs that use the work file information to perform the pointer updating.

The three following sections describe the loading of data bases with logical relationships, the loading of data bases with secondary indexes, and the utility programs that perform the pointer resolution.

With Logical Relationships

If a segment is a logical child, both the logical parent's fully concatenated key and the logical child's intersection data, if any, are assembled in the loading program's I/O area for loading. The data for the logical parent is loaded by a separate LOAD command or ISRT statement.

When you load a data base containing segments involved in logical relationships, one or more of the logical relationship resolution utilities may have to be executed. The section “Utility Programs” describes these programs.

When you load a data base containing logical relationships you must provide, with the execution of your load program, job control statements that define an input and an output file:

- The input file was created by the data base Prereorganization utility. It contains control information. The filename must be specified as CONTROL. The logical unit assignment must be SYS012. The file can only be DASD.
- The output file is created automatically during the loading of your data base. It contains logical relationship information. The filename must be WORKFIL. The logical unit assignment must be SYS013. The file can be either tape or DASD.

If you load only logical parent segments (no logical child segments), you can bypass the need for executing the logical relationship resolution utilities by:

- Specifying // ASSGN SYS013,IGN in the job control of your loading program, so that no work file is generated. A message is printed as a warning, but processing continues.
- Subsequently loading the logical child segments in an update type job (one with a PCB specifying a PROCOPT of A or I).

All work files produced during the loading of data bases participating in logical relationships must be supplied as input to an execution of the Prefix Resolution utility. Be sure that the logical parent of each logical child loaded during initial data base load is also loaded before the prefix resolution utility is executed.

With Secondary Indexes

There are two ways of creating secondary indexes. One way is to create them automatically during initial loading or reloading. This is accomplished if all DLBL statements for the secondary index data bases are included in the job stream for the initial load or reload. However, because the index records are not normally in ascending key sequence, this usually leads to a significant performance degradation.

The other way of creating secondary indexes is to delay their creation until later using logical relationship utilities. One way of delaying their creation is to omit the DLBL statement(s) for the secondary index data base(s). This prevents the indexes from being created automatically. When using the HD Reload utility, there is another way to delay their creation. Leave the DLBL statement(s) as is, and include a BLDINDEX=NO control statement following the DLI parameter statement. If the BLDINDEX=NO statement is not provided, your labels will be used as submitted. In both cases, the logical relationship resolution utilities must be used to build the secondary index data base(s) and ASSGN and TLBL (or DLBL and EXTENT) statements must be provided for the work file (WORKFIL).

Note: When using the method of omitting the DLBL statements, message DLZ020I with a VSAM return code of X'80' occurs for every secondary index data

base (because the index maintenance routine tries to open them), but loading does continue.

When loading a data base that has a secondary index, but no index source segments in the initial load, supply DLBL statements for the secondary index data bases so the Prefix Update utility will work properly.

Utility Programs

DL/I provides a set of four utility programs to help you with the initial loading of data bases containing logical relationships. They are called logical relationship resolution utilities. These utilities are not concerned with the actual loading of data into data bases. That is the function of the loading program written by someone in your installation. The major function of the utilities is the resolution of pointers in the prefixes of logically related segments.

The four programs are:

- The data base Prereorganization utility (DLZURPR0)

  This program generates a control data set that controls the execution of the load program and the other logical relationship resolution utilities. It must be run before them.

- The data base Scan utility (DLZURGS0)

  This program searches designated data bases for segments that are involved in logical relationships. It generates one or more output records for each such segment. These output records are input to the prefix resolution utility.

- The data base Prefix Resolution utility (DLZURG10)

  The main function of this program is to combine and sort (in physical data base sequence) all work files that are defined as input to it. These input files are generated by the pre-reorganization utility, the scan utility, or your load program. The prefix resolution utility also checks for missing logical parents.

- The data base Prefix Update utility (DLZURGP0)

  This program applies the necessary changes to the prefixes of segments involved in logical relationships that could not be resolved previously. Its input is the file generated by the Prefix Resolution utility. Following the execution of this program, the data base(s) are ready for use.

You can code the DL/I logical relationship utility job streams or, if you have a 3270-type terminal available, you can use the Interactive Utility Generation (IUG) facility to generate job streams for the utilities.

For complete information on IUG and how to use it, see *DL/I DOS/VS Interactive Resource Definition and Utilities*.

If you choose not to use IUG, *DL/I DOS/VS Resource Definition and Utilities* gives you complete information on how to code and execute the utilities necessary to perform the various load functions.

Sample Load Program

Shipped with DL/I are several programs that make up a sample DL/I application. This application includes two data bases, and makes use of logical relationships and secondary indexing. The application and the programs are described in *DL/I DOS/VS Guide For New Users*.

One of the programs in this set is a load program that initially loads the two data bases. It can serve as an example of how to write and use a load program. One of the data bases contains inventory data. The other contains information about customers.

There are different versions of the load program, depending on which request interface you wish to use (DL/I HLPI or DL/I CALL) and for each programming language supported by that interface.

The data to be loaded into the two data bases is in the form of 80-byte card image input data records. The data for the inventory data base is expected first, and is separated from the customer data by a single record having “CUSTOMER” in the first eight bytes, like this:

```
....INVENTORY DATA....
....INVENTORY DATA....
      •
      •
CUSTOMER
....CUSTOMER DATA....
....CUSTOMER DATA....
      •
      •
```

The program reads the card images from SYSRDR, constructs the data base segments from the card images, and issues DL/I requests to load the segments.

Parts of the source code for the version of the load program using COBOL and the HLPI commands are shown here as an example of load program logic and coding. Note the use of the variable (SEG-NAME) in the LOAD commands. The actual name of each segment is the first field of the input record. Code not shown moves the actual name into SEG-NAME for each segment. There is a variable length segment (STSCHIS) in the customer data base. When an input record for this segment is located a special LOAD command is executed.

```
CBL XOPTS(DLI)...
IDENTIFICATION DIVISION.
PROGRAM ID. DLZCBL10.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
          •
          •
77 SW-CUSTOMER          PIC 9 VALUE 0.
77 INV-REC              PIC 9(5) VALUE 0 COMP-3.
77 CUST-REC             PIC 9(5) VALUE 0 COMP-3.
77 SEG-NAME             PIC X(9).
01 IO.
   02 IO-LEFT           PIC X(64).
   02 IO-RIGHT          PIC X(80).
01 IO-VAR.
   02 SEG-LGT           PIC 9(4) COMP.
   02 IO1               PIC X(142).
01 MSGE-LINE.
   02 MSGE1             PIC X(6).
   02 MSGE2             PIC X(60).
          •
          •
PROCEDURE DIVISION.
ENTRY 'DLITCBL'.
          •
          •
     IF SW-CUSTOMER = 1
          GO TO LOAD-CUSTOMER-DB
     ELSE NEXT SENTENCE.
*
* LOAD INVENTORY DATA BASE
*
 LOAD-INVENTORY-DB.
    EXEC DLI
        LOAD USING PCB(1)
        SEGMENT((SEG-NAME)) FROM(IO) SEGLENGTH(144)
    END-EXEC.
    IF DIBSTAT = '  ' ADD 1 TO INV-REC
        GO TO READ-CARD
    ELSE MOVE DIBSTAT TO MSGE1
        MOVE 'UNEXPECTED DLI STATUS CODE-INV' TO MSGE2
        PERFORM PRINT-MESSAGE
        GO TO END-NOK.
```

```
*
* LOAD CUSTOMER DATA BASE
*
 LOAD-CUSTOMER-DB.
   IF SEG-NAME = 'STSCHIS'
      MOVE IO TO IO1
      EXEC DLI
         LOAD USING PCB(2)
         SEGMENT((SEG-NAME)) FROM(IO-VAR) SEGLENGTH(144)
      END-EXEC
   ELSE
      EXEC DLI
         LOAD USING PCB(2)
         VARIABLE SEGMENT(STSCHIS) FROM(IO) SEGLENGTH(144)
      END-EXEC.
   IF DIBSTAT = '  ' ADD 1 TO CUST-REC
      GO TO READ-CARD
   ELSE MOVE DIBSTAT TO MSGE1
       MOVE 'UNEXPECTED DLI STATUS CODE-CUST' TO MSGE2
       PERFORM PRINT-MESSAGE
       GO TO END-NOK.
        •
        •
 END-NOK.
* DISPLAY FAILURE MESSAGE
        •
        •
 GOBACK.
```

Chapter 8. Modifying Data Base Structure, Organization, and Environment

This chapter tells you how to modify the structure, organization, or environment of an existing data base. There are eight sections:

1. Adding Segment Types
2. Deleting Segment Types
3. Moving Segment Types
4. Changing Segment Size
5. Adding a Secondary Index
6. Adding a Logical Relationship
7. Adding or Converting to Variable Length Segments
8. Converting to Use of Segment Edit/Compression

The first four sections tell you how to make changes involving the segments or fields of the data base. The last four sections tell you how to make use of various DL/I capabilities with existing data bases.

Circumstances will arise that make it necessary or desirable to modify your data base (that is, change its structure, organization, or environment). Over time, user requirements may change, forcing changes in the data base design. You may wish to use new or different options or features. Perhaps you've simply found a more efficient way to structure the data base.

This chapter describes various types of changes you can make to your data base. It gives procedures to guide you through making each individual type of change. If you're making more than one change at a time, you should proceed carefully, after planning exactly what you need to do. The flowchart in Figure 8-1, when used with the individual procedures in this chapter, will guide you in making some types of multiple changes to the data base.

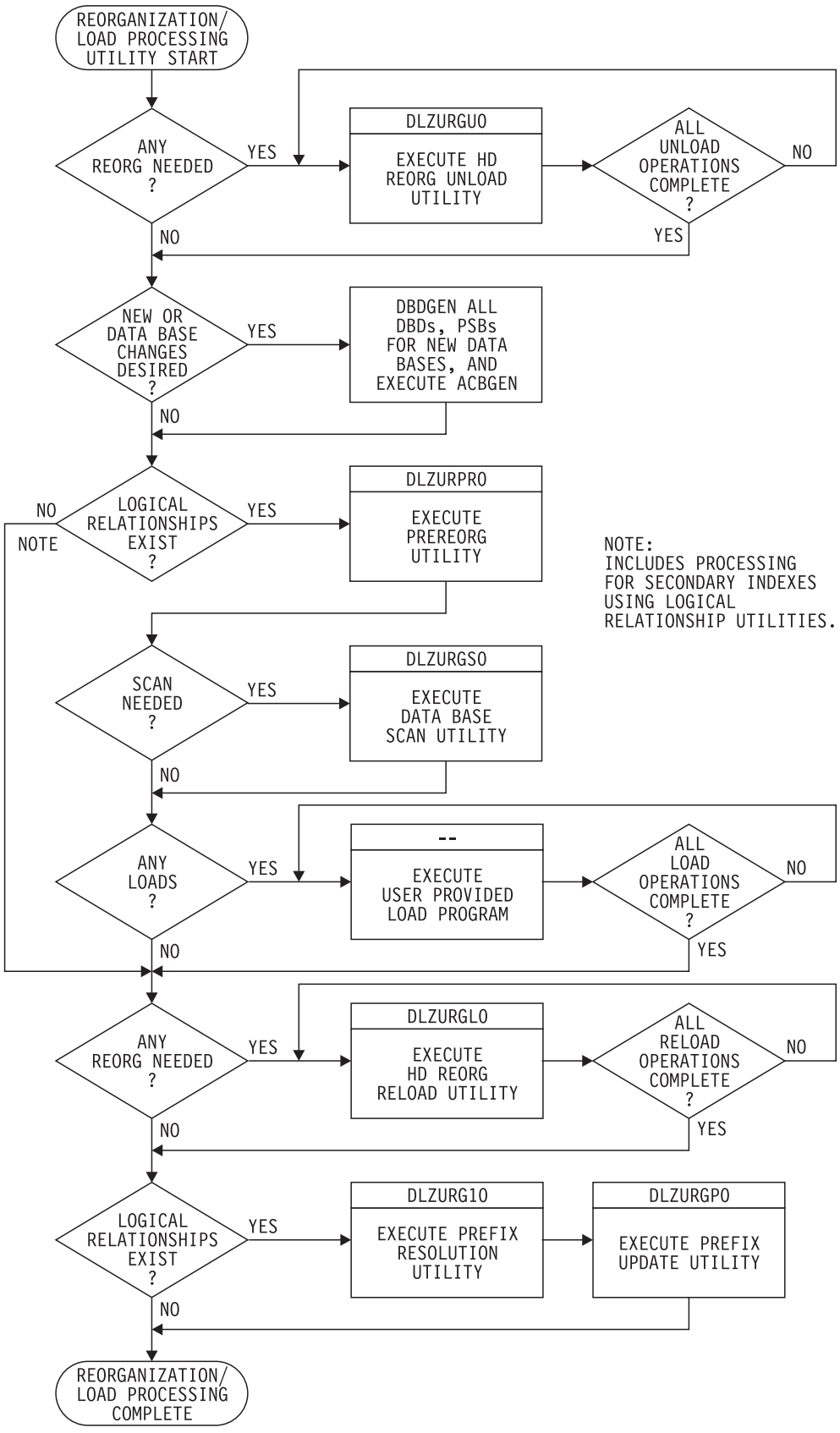


Figure 8-1. Utility Program Usage Sequence

Adding Segment Types

You can add a segment type to a data base:

- By unloading and reloading with the reorganization utilities
- Without unloading or reloading under certain conditions
- By using your own unload and reload program.

Which method you use depends on the conditions explained in the following sections.

Using the Reorganization Utilities

You can add a segment type to a data base record using the reorganization utilities if:

- The segment type to be added is at the lowest level of a path in the hierarchy. Figure 8-2 shows an existing data base record (indicated by solid lines) and the places where a new segment type can be added (indicated by dotted lines).
- The existing relative hierarchical order of segments in the data base record doesn't change. In other words, the existing parent-child relationships don't change.
- The existing segment names don't change.

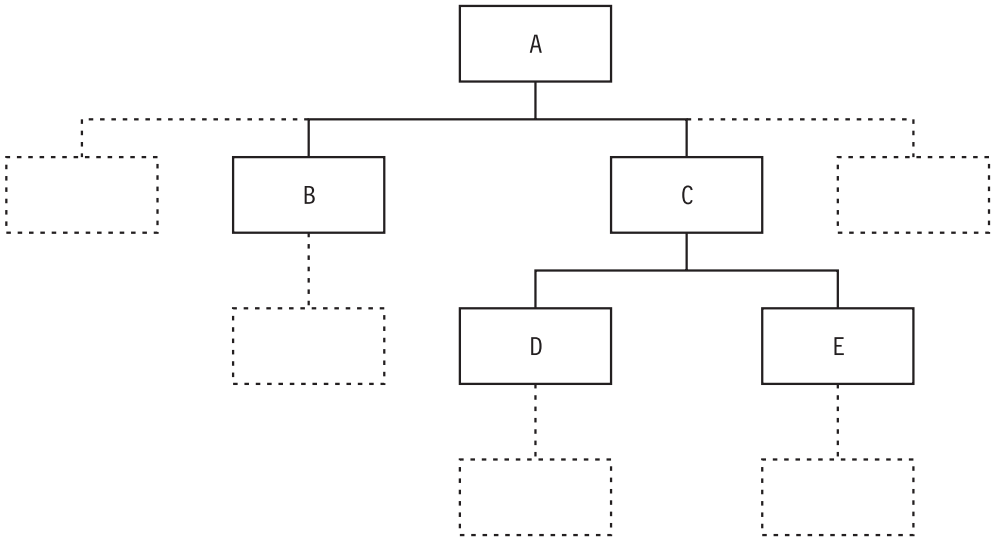


Figure 8-2. Where Segment Types Can Be Added in a Data Base Record

To use the reorganization utilities to add a segment type to a data base:

1. Unload your data base using the existing DBD. No data base updates are allowed between unload and reload.
2. Modify the DBD by adding a SEGM statement for the new segment type. Perform a DBDGEN.
3. This change will not affect any existing application programs unless they are modified to access the new segment.
4. If any existing application programs are modified to access the new segment type, make any necessary changes to the PSBs for those programs and perform a PSBGEN. If you use the DB/DC Data Dictionary, it can show you which application programs and PCBs are affected by the DBD change you've made.

5. Perform an ACBGEN.
6. The change you are making may result in different data base space requirements. Recalculate data base space if necessary. Delete the VSAM space allocated for the old cluster(s) and define space for the new cluster(s). See the section “Determining VSAM Space Requirements” in chapter 4 for a description of how to calculate data base space.
7. Reload your data base using the modified DBD. Remember to make an image copy of your data base as soon as it is reloaded.
8. If your data base uses logical relationships or secondary indexes, you will have to run some of the reorganization utilities before and after reloading to resolve prefix information. The flowchart in Figure 8-1 shows you which utilities and the order in which they must be run.
9. Code and execute an application program to insert the initial data for the new segment type into the data base.

Without Unloading or Reloading

You can add a segment type to an HSAM or HISAM data base record without first unloading the data base if:

- The segment type to be added becomes the last segment type in the hierarchy.
- The segment type to be added fits in the existing logical record.

To add a segment type to a data base without unloading and reloading:

1. Modify the DBD by adding a SEGM statement for the new segment type. Perform a DBDGEN.
2. This change will not affect any existing application programs unless they are modified to access the new segment.
3. If any existing application programs are modified to access the new segment type, make any necessary changes to the PSBs for those programs and perform a PSBGEN. If you use the DB/DC Data Dictionary, it can show you which application programs and PCBs are affected by the DBD change you've made.
4. Perform an ACBGEN.
5. Code and execute an application program to insert the initial data for the new segment type into the data base.

Using Your Own Program

If your conditions for adding a segment type don't meet the qualifications described in the two preceding sections you will have to write and execute your own unload and reload program.

Deleting Segment Types

You can delete a segment type from a data base by using either the reorganization utilities or your own unload and reload program.

Note: Remember that when you delete a segment type you must also delete all segment types (if any) that are dependent on it.

To use the reorganization utilities to delete a segment type from a data base:

1. Code and generate a PSB naming the physical DBD containing the segment type being deleted. Include SENSEG (and SENFLD, if desired) statements for only the the segments (and fields) that are to be kept. Do not include SENSEG

statements for the segment type to be deleted, or for its dependent segments, if any.
2. Unload your data base using the selective unload facility of the HD unload utility, and the specially generated PSB. On the batch initialization parameter statement, you must code PLU instead of ULU, and the name of the PSB instead of a DBD name.
3. Modify the existing DBD by removing the SEGM statements for:
   - The segment type being deleted
   - All dependent child segments of the deleted segment.
4. Determine whether the change you are making will affect the code in any application programs. If so, make sure the code is changed.
5. If any existing application programs are modified because of the change, make any necessary changes to the PSBs for those programs and perform a PSBGEN. If you use the DB/DC Data Dictionary, it can show you which application programs and PCBs are affected by the DBD change you've made.
6. The change you are making may result in different data base space requirements. Recalculate data base space if necessary. Delete the space allocated for the old cluster(s) and define space for the new cluster(s). See the section “Determining VSAM Space Requirements” in chapter 4 for a description of how to calculate data base space.
7. Perform an ACBGEN.
8. Reload your data base using the modified DBD. Remember to make an image copy of your data base as soon as it is reloaded.
9. If your data base uses secondary indexes, you can run some of the reorganization utilities before and after reloading to resolve prefix information. The flowchart in Figure 8-1 shows you which utilities and the order in which they must be run. Alternatively, you may have the secondary indexes recreated when the data base is reloaded.

Changing Segment Size

You can increase or decrease segment size by using the selective unload feature of the HD unload utility with field level sensitivity. Selective unload cannot be used if your data base contains logical relationships. To increase or decrease segment size:

1. Determine whether the change you are making will affect the code in any application programs. If so, make sure the code gets changed.
2. Code a PSB, using field level sensitivity, that represents the new segment sizes.
3. Perform an ACBGEN.
4. Unload the data base using the HD unload utility. In the utility's parameter statement: code PLU instead of ULU, and the name of the PSB instead of a DBD name.
5. Code a new DBD for the data base that represents the new segment sizes as described in the new PSB.
6. Perform an ACBGEN.
7. Recalculate data base space. You need to do this because the change you are making will result in different requirements for data base space. See the section “Determining VSAM Space Requirements” in chapter 4 for a description of how to calculate data base space.
8. Delete the space allocated for the old cluster(s) and define space for the new cluster(s).

9. Reload the data base, using the HD reload utility and the new DBD. Remember to make an image copy of your data base as soon as it's reloaded.
10. If your data base uses secondary indexes, you can run some of the reorganization utilities before and after reloading to resolve prefix information. The flowchart in Figure 8-1 tells you which utilities and the order in which they must be run. Alternatively, you may have the secondary indexes recreated when the data base is reloaded.

Changing the Position of Data in a Segment

You can change the position of data in a segment by using the selective unload feature of the HD unload utility with field level sensitivity. Selective unload cannot be used if your data base contains logical relationships. To change the position of data in a segment:

1. Determine whether the change you are making will affect the code in any application programs. If so, make sure the code gets changed.
2. Code a PSB that represents the new arrangement of fields in the segment type, using field level sensitivity.
3. Perform an ACBGEN.
4. Unload the data base using the HD unload utility. In the utility's parameter statement: code PLU instead of ULU, and the name of the PSB instead of a DBD name.
5. Code a new DBD for the data base that represents the new placement of fields as described in the new PSB.
6. Perform an ACBGEN.
7. Reload the data base, using the HD reload utility and the new DBD. Remember to make an image copy of your data base as soon as it is reloaded.
8. If your data base uses secondary indexes, you can run some of the reorganization utilities before and after reloading to resolve prefix information. The flowchart in Figure 8-1 tells you which utilities and the order in which they must be run. Alternatively, you may have the secondary indexes recreated when the data base is reloaded.

Adding a Logical Relationship

Logical relationships are explained in detail in chapter 4. This section contains a series of examples and procedures for adding a logically-related data base to an existing data base (or bases). The examples are not intended to cover every possible situation in which you might want to add a logical relationship. However, if the examples don't fit your specific requirements, you should be able to gather enough information from them to decide:

- Whether adding a logical relationship to your existing data base is possible
- If it is possible, what you must do to add it.

Figure 8-3 summarizes the steps you must take in reorganizing a data base to add a logical relationship.

The examples always show the logical parent as a root segment. This is not a requirement. The examples are still valid when the logical parent is at a lower level in the hierarchy.

When adding logical relationships to existing data bases, you should always make the change on a test data base. Only after the change is thoroughly tested should it be implemented using production data bases.

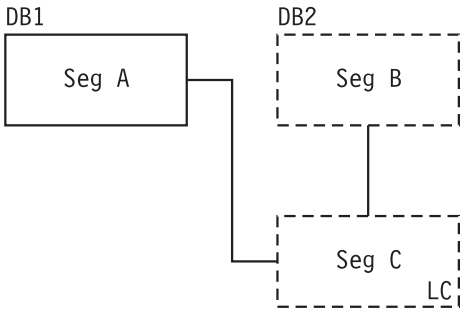
In the following examples, these conventions are used:

- Existing data bases are shown using *solid* lines
- The data base being added is shown using *dashed* lines
- The logical parent–logical child relationships are labeled LP and LC, respectively.
- Unidirectional logical relationships are indicated with a single line. Bidirectional relationships are indicated with a double line.
- The terms DB1, DB2, and DB3 refer to data base 1, data base 2, and data base 3.

ACTION REQUIRED TO ACCOMPLISH REORGANIZATION			
FROM:	TO:	UNIDIRECTIONAL DIRECT POINTERS	BIDIRECTIONAL DIRECT POINTERS
UNIDIRECTIONAL DIRECT POINTERS	REORGANIZE LP DATA BASE ONLY	1. SCAN LC DATA BASE 2. RESOLUTION/UPDATE	1. SCAN LC DATA BASE 2. RESOLUTION/UPDATE NOTE: LC SEGMENT WON'T CONTAIN LT ↑ UNLESS REORGANIZED
	REORGANIZE LC DATA BASE ONLY	1. FINISHED	NOT VALID LCF/LCL ↑ BEING CREATED ON LP DATA BASE
	REORGANIZE BOTH DATA BASES	1. RESOLUTION/UPDATE	1. RESOLUTION/UPDATE
BIDIRECTIONAL DIRECT POINTERS	REORGANIZE LP DATA BASE ONLY	NOT VALID * THE COUNTER WON'T BE RESOLVED	1. SCAN LC DATA BASE 2. RESOLUTION/UPDATE LCF/LCL ISN'T UNLOADED AND RELOADED
	REORGANIZE LC DATA BASE ONLY	NOT VALID * LCF/LCL EXISTS NOW IN LP DATA BASE	1. SCAN LP DATA BASE 2. RESOLUTION/UPDATE
	REORGANIZE BOTH DATA BASES	NOT VALID * THE COUNTER WON'T BE RESOLVED	1. RESOLUTION/UPDATE

Figure 8-3. Steps in Reorganizing a Data Base to Add a Logical Relationship

Example 1. DB1 EXISTS, DB2 IS TO BE ADDED (Unidirectional Relationship)

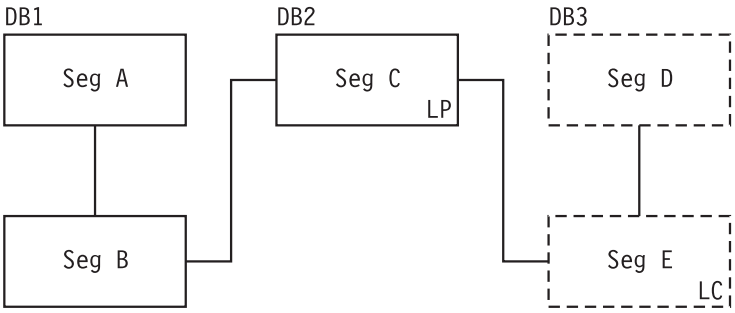


DB1 must be reorganized to add the logical child counter field to segment A's prefix.

Procedure

1. Unload DB1 using the existing DBD and the HD unload utility.
2. Modify the DBD for DB1 by adding an LCHILD statement for segment C. Code a new DBD for DB2. Perform DBDGENs for both data bases. See chapter 4 for a complete discussion of logical relationships. See *DL/I DOS/VS Resource Definition and Utilities* for details on how to implement DBDs for logical relationships.
3. Perform an ACBGEN.
4. Recalculate data base space for DB1, if necessary. Calculate space for DB2. Delete the space allocated for the old cluster(s) of DB1 and define space for the new cluster(s). See the section “Determining VSAM Space Requirements” in chapter 4 for a description of how to calculate data base space.
5. Run the prereorganization utility, specifying DBR in the control statement for DB1 and DBIL in the control statement for DB2.
6. Reload DB1 using the modified DBD and the HD reload utility.
7. Load DB2 using an initial load program. See chapter 7 for a description of how to write an initial load program.
8. Run the prefix resolution utility using as input the CONTROL file and the work files that are output from the three previous steps.
9. Run the prefix update utility using as input the work file that is output from the previous step.
10. Remember to make an image copy of both data bases as soon as they are loaded.

Example 2. DB1 AND DB2 EXIST, DB3 IS TO BE ADDED (Unidirectional Relationships)



In this example, the logical child counter exists in segment C's prefix. This gives you two options on how to add DB3. You can scan DB2 or reorganize it.

Procedure Using Scan

1. Modify the DBD for DB2 by adding an LCHILD statement for segment E. Code a new DBD for DB3. Perform DBDGENs for both data bases. See chapter 4 for a complete discussion of logical relationships. See *DL/I DOS/VS Resource Definition and Utilities* for details on how to implement DBDs for logical relationships.
2. Code PSB(s) to access new paths in data base(s).
3. Perform an ACBGEN.
4. Calculate data base space for DB3. Define space for the new VSAM cluster(s). See the section “Determining VSAM Space Requirements” in chapter 4 for a description of how to calculate data base space.
5. Run the prereorganization utility, specifying DBIL in the control statement for DB3. (The output from the prereorganization utility will say that a scan of DB2 is required.)
6. Run the scan utility against DB2.
7. Load DB3 using an initial load program. See chapter 7 for a description of how to write an initial load program.
8. Run the prefix resolution utility using as input the CONTROL file and the work files that are output from the three previous steps.
9. Run the prefix update utility using as input the work file that is output from the previous step.
10. Remember to make an image copy of both data bases as soon as they are loaded.

Procedure When Reorganizing DB2

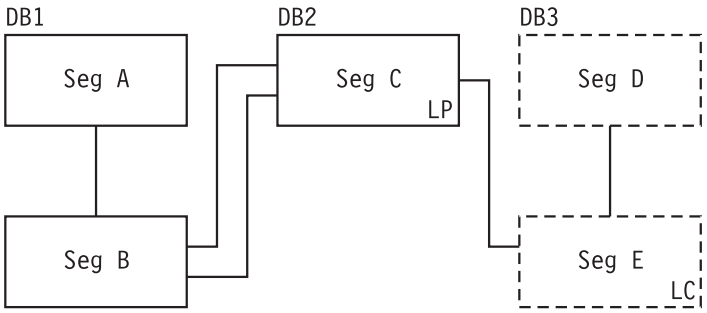
1. Unload DB2 using the existing DBD and the HD unload utility.
2. Modify the DBD for DB2 by adding an LCHILD statement for segment E. Code a new DBD for DB3. Perform DBDGENs for both data bases. See chapter 4 for a complete discussion of logical relationships. See *DL/I DOS/VS Resource Definition and Utilities* for details on how to implement DBDs for logical relationships.
3. Perform an ACBGEN.
4. Calculate data base space for DB3. Define space for the new VSAM cluster(s). See the section “Determining VSAM Space Requirements” in chapter 4 for a description of how to calculate data base space.

5. Run the prereorganization utility specifying DBR in the control statement for DB2, and DBIL in the control statement for DB3. (The output from the pre-reorganization utility will say that a scan of DB1 is required.)
6. Run the scan utility against DB1.
7. Reload DB2 using the modified DBD and the HD reload utility.
8. Load DB3 using an initial load program. See chapter 7 for a description of how to write an initial load program.
9. Run the prefix resolution utility using as input the CONTROL file and the work files that are output from the four previous steps.
10. Run the prefix update utility using as input the work file that is output from the previous step.
11. Remember to make an image copy of both data bases as soon as they are loaded.

Procedure When Reorganizing DB1 and DB2

1. Unload DB1 and DB2 using existing DBDs and the HD unload utility.
2. Modify the DBD for DB2 by adding an LCHILD statement for segment E. Code a new DBD for DB3. Perform DBDGENs for both data bases. See chapter 4 for a complete discussion of logical relationships. See *DL/I DOS/VS Resource Definition and Utilities* for details on how to implement DBDs for logical relationships.
3. Perform an ACBGEN.
4. Calculate data base space for DB3. Define space for the new VSAM cluster(s). See the section “Determining VSAM Space Requirements” in chapter 4 for a description of how to calculate data base space.
5. Run the prereorganization utility specifying DBR in the control statements for DB1 and DB2, and DBIL in the control statement for DB3.
6. Reload DB1 and DB2 with the HD reload utility, using the existing DBD for DB1 and the modified DBD for DB2.
7. Load DB3 using an initial load program. See chapter 7 for a description of how to write an initial load program.
8. Run the prefix resolution utility using as input the CONTROL file and the work files that are output from the three previous steps.
9. Run the prefix update utility using as input the work file that is output from the previous step.
10. Remember to make an image copy of both data bases as soon as they are loaded.

Example 3. DB1 AND DB2 EXIST, DB3 IS TO BE ADDED (Unidirectional Relationships)



DB2 must be reorganized to add the logical child counter field to segment C's prefix. DB1 must be reorganized or scanned to reestablish pointers to segment C.

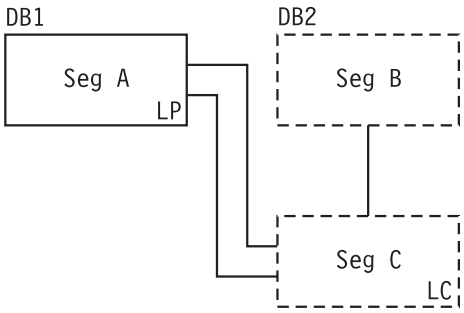
Procedure Using Scan

1. Unload DB2 using the existing DBD and the HD unload utility.
2. Modify the DBD for DB2 by adding an LCHILD statement for segment E. Code a new DBD for DB3. Perform DBDGENs for both data bases. See chapter 4 for a complete discussion of logical relationships. See *DL/I DOS/VS Resource Definition and Utilities* for details on how to implement DBDs for logical relationships.
3. Perform an ACBGEN on all the PSBs that reference the data bases.
4. Calculate data base space for DB3. Define space for the new VSAM cluster(s). See the section “Determining VSAM Space Requirements” in chapter 4 for a description of how to calculate data base space.
5. Run the prereorganization utility specifying DBR in the control statement for DB2, and DBIL in the control statement for DB3. (The output from the pre-reorganization utility will say that a scan of DB1 is required.)
6. Run the scan utility against DB1.
7. Reload DB2 using the modified DBD and the HD reload utility.
8. Load DB3 using an initial load program. See chapter 7 for a description of how to write an initial load program.
9. Run the prefix resolution utility using as input the CONTROL file and the work files that are output from the four previous steps.
10. Run the prefix update utility using as input the work file that is output from the previous step.
11. Remember to make an image copy of both data bases as soon as they are loaded.

Procedure When Reorganizing DB1 and DB2

1. Unload DB1 and DB2 using existing DBDs and the HD unload utility.
2. Modify the DBD for DB2 by adding an LCHILD statement for segment E. Code a new DBD for DB3. Perform DBDGENs for both data bases. See chapter 4 for a complete discussion of logical relationships. See *DL/I DOS/VS Resource Definition and Utilities* for details on how to implement DBDs for logical relationships.
3. Perform an ACBGEN.
4. Calculate data base space for DB3. Define space for the new VSAM cluster(s). See the section “Determining VSAM Space Requirements” in chapter 4 for a description of how to calculate data base space.
5. Run the prereorganization utility specifying DBR in the control statements for DB1 and DB2, and DBIL in the control statement for DB3.
6. Reload DB1 and DB2 with the HD reload utility, using the existing DBD for DB1 and the modified DBD for DB2.
7. Load DB3 using an initial load program. See chapter 7 for a description of how to write an initial load program.
8. Run the prefix resolution utility using as input the CONTROL file and the work files that are output from the three previous steps.
9. Run the prefix update utility using as input the work file that is output from the previous step.
10. Remember to make an image copy of both data bases as soon as they are loaded.

Example 4. DB1 EXISTS, DB2 IS TO BE ADDED (Bidirectional Relationship)

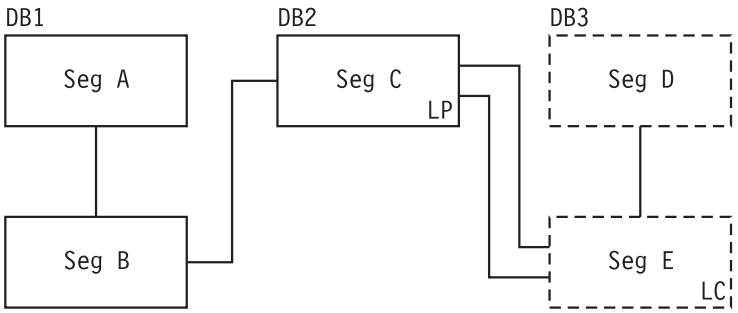


DB1 must be reorganized to add the logical child pointers in segment A's prefix.

Procedure

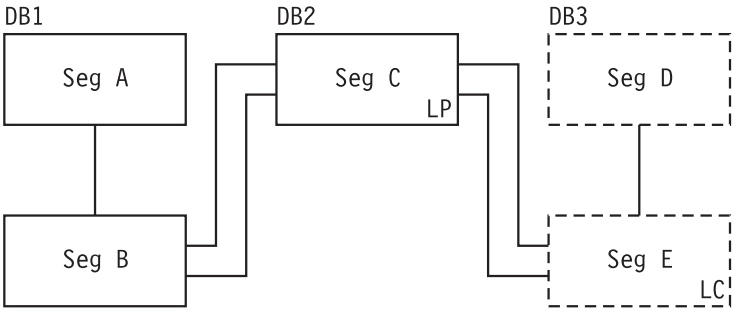
1. Unload DB1 using the existing DBD and the HD unload utility.
2. Modify the DBD for DB1 by adding an LCHILD statement for segment C. Code a new DBD for DB2. Perform DBDGENs for both data bases. See chapter 4 for a complete discussion of logical relationships. See *DL/I DOS/VS Resource Definition and Utilities* for details on how to implement DBDs for logical relationships.
3. Perform an ACBGEN.
4. Recalculate data base space for DB1, and calculate space for DB2. Delete the space allocated for the old VSAM cluster(s) and define space for the new cluster(s). See the section “Determining VSAM Space Requirements” in chapter 4 for a description of how to calculate data base space.
5. Run the prereorganization utility specifying DBR in the control statement for DB1, and DBIL in the control statement for DB2.
6. Reload DB1 using the modified DBD and the HD reload utility.
7. Load DB2 using an initial load program. See chapter 7 for a description of how to write an initial load program.
8. Run the prefix resolution utility using as input the CONTROL file and the work files that are output from the three previous steps.
9. Run the prefix update utility using as input the work file that is output from the previous step.
10. Remember to make an image copy of both data bases as soon as they are loaded.

Example 5. DB1 AND DB2 EXIST, DB3 IS TO BE ADDED



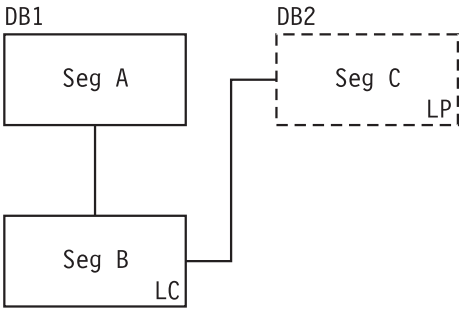
DB2 must be reorganized to add the logical child pointers to segment C's prefix. The procedure for this example (and all conditions and considerations) is exactly the same as the second, third, and fourth procedures for example 2.

Example 6. DB1 AND DB2 EXIST, DB3 IS TO BE ADDED



DB2 must be reorganized to add the logical child pointers to segment C's prefix. Logical child pointers from segment C to segment B are not unloaded; therefore, DB1 must be reorganized or scanned. The procedure for this example (and all conditions and considerations) is exactly the same as the two procedures in example 3.

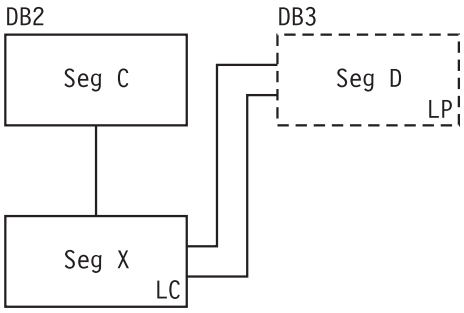
Example 7. DB1 EXISTS, DB2 IS TO BE ADDED (Unidirectional Relationship)



DB1 and DB2 must be loaded using an initial load program. Although it appears that a reorganization of DB1 and initial load of DB2 would work (no error messages are issued), the counter field in DB2 will be zero. This results in an abnormal termination when you later attempt to delete a segment B. Only when logical children are loaded using an initial load program is the logical parent counter incremented.

The procedure for this example (and all conditions and considerations) is exactly the same as for example 2.

Example 8. DB2 EXISTS, DB3 IS TO BE ADDED (Bidirectional Relationship)



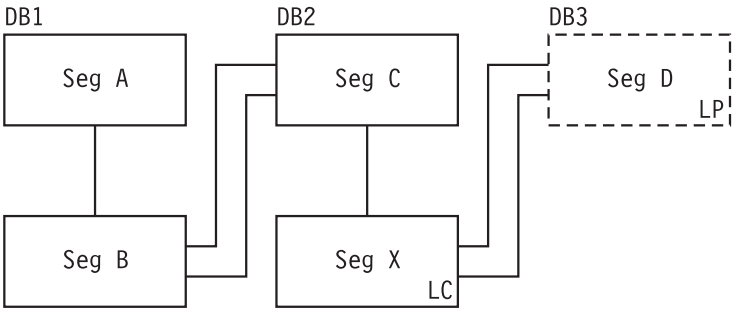
Segment X might be considered a logical child if the key of segment D is at the right location in segment X. DB2 must be reorganized, because an initial load (DBIL) of the logical parent (segment D) assumes on initial load (DBIL) of the logical child.

Do *not* under any circumstances specify DBR in the control statement for DB2. If you did, the reload utility wouldn't generate work records for segment D. This means the logical child pointer in segment D would never be resolved.

Procedure

1. Unload DB2 using the existing DBD and the HD unload utility.
2. Code a new DBD for DB2 and DB3. See chapter 5 for an explanation of how the DBD is implemented for logical relationships.
3. Perform an ACBGEN.
4. Recalculate data base space for DB2, and calculate space for DB1. See the section “Determining VSAM Space Requirements” in chapter 4 for a description of how to calculate data base space.
5. Delete the space allocated for the old VSAM cluster(s) and define space for the new cluster(s).
6. Run the prereorganization utility specifying DBIL in the control statement for DB2 and DB3.
7. Reload DB2 using the new DBD and the HD reload utility.
8. Load DB3 using an initial load program. See chapter 7 for a description of how to write an initial load program.
9. Run the prefix resolution utility using as input the work files that are output from steps 9 and 10.
10. Run the prefix update utility using as input the work file that is output from step 11.
11. Remember to make an image copy of both data bases as soon as they are loaded.

Example 9. DB1 and DB2 EXIST, DB3 IS TO BE ADDED (Bidirectional Relationships)



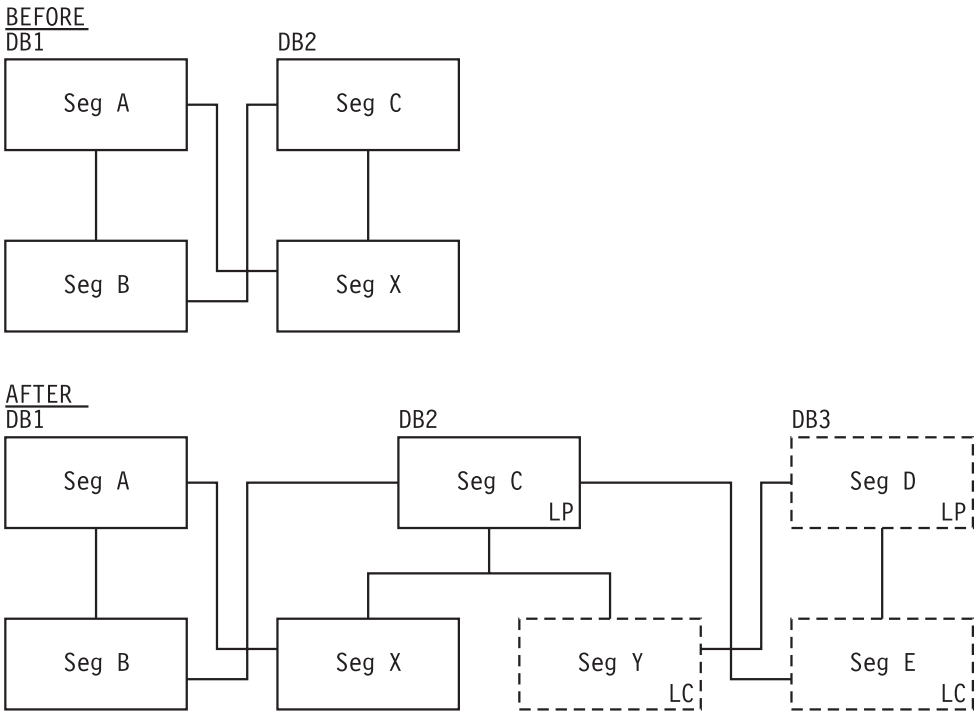
DB1 and DB2 must be reorganized. DB3 must be loaded using an initial load program. Because you must specify DBIL in the control statement for DB3 (a logical parent data base), you must also specify DBIL for DB2 (a logical child data base). DB2 is also a logical parent data base. Therefore, you must specify DBIL in the control statement for DB1 (a logical child data base).

Do not specify DBR in the control statement for DB2 or DB1.

Procedure

1. Unload DB1 and DB2 using the existing DBD and the HD unload utility.
2. Code a new DBD for DB2 and DB3. See chapter 5 for an explanation of how the DBD is implemented for logical relationships.
3. Perform an ACBGEN.
4. Recalculate data base space for DB1 and DB2, and calculate space for DB3. See the section “Determining VSAM Space Requirements” in chapter 4 for a description of how to calculate data base space.
5. Delete the space allocated for the old VSAM cluster(s) and define space for the new cluster(s).
6. Run the prereorganization utility specifying DBIL in the control statement for DB1, DB2, and DB3.
7. Reload DB1 and DB2 using the new DBDs and the HD reload utility.
8. Load DB3 using an initial load program. See chapter 7 for a description of how to write an initial load program.
9. Run the prefix resolution utility using as input the work files that are output from steps 9 and 10.
10. Run the prefix update utility using as input the work file that is output from step 11.
11. Remember to make an image copy of both data bases as soon as they are loaded.

Example 10. DB1 and DB2 EXIST, SEGMENT Y AND DB3 ARE TO BE ADDED



In this example, all three data bases must be loaded using an initial load program. DB2 must be initially loaded to add occurrences of segment Y. During initial load, the counts of segments X and Y are added to the counters in segments A and D.

If DB1 was scanned or reorganized, the counter in segment A, after the prefix update utility was run, would have twice the count of X segments as were initially loaded. The counter in segment C would have only the count of E segments because:

- The counter would have been built with zero, and
- Only the E logical children would have been loaded using an initial load program (the B segments would have been scanned or reloaded).

Steps in Reorganizing a Data Base to Add a Logical Relationship

Figure 8-3 shows you:

- When a logically related data base must be scanned
- When both sides of a logical relationship must be reorganized
- When the prefix resolution and prefix update utilities must be run

Note that the figure applies to reorganizations only. When initially loading data bases, you have to run the prefix resolution and update utilities whenever work data sets are generated.

Figure 8-3 covers all reorganization situations, whether or not data base pointers are being changed.

Summary on Use of Utilities When Adding Logical Relationships

- Counters are incremented only by counting logical children loaded using an initial load program.
- Counter problems can't be corrected by reorganizing data bases. The data bases must be loaded using an initial load program.
- LCF and LCL pointers are not unloaded and reloaded. They must be recreated by the prefix resolution and update utilities.
- Never specify DBIL in the control statement for a logical parent data base unless DBIL is specified for all its logical child data bases.

Adding a Secondary Index

Secondary indexes are explained in detail in chapter 4. If you want to add a secondary index to your data base:

1. Determine whether the change you are making will affect the code in any application programs. If so, make sure the code gets changed.
2. Unload your data base using the existing DBD and the HD unload utility.
3. Code new DBDs. See chapter 5 for an explanation of how the DBD is implemented for secondary indexes. Note that you need two new DBDs, one for the existing data base and one for the new secondary index data base.
4. If the change you are making affected the code in application programs, make any necessary changes to the PSBs for those application programs. Remember that if you have the DB/DC Data Dictionary, it can help you determine which application programs and PCBs may be affected by the DBD changes you've made.
5. Perform an ACBGEN.
6. Delete the space allocated for the cluster and define space for the new cluster. In addition, define space for the secondary index.
7. Reload the data base using the new DBD and the HD reload utility.
8. Run the prefix resolution utility using as input the work file that is output from step 7.
9. When you add a secondary index, remember to change your job control statements. You need a DS statement for the secondary index data set even when you're not using the secondary index to process the main data base. You also need to change your reorganization procedures when adding a secondary index. Whenever you reorganize the data set the secondary index points to, you have to execute the reorganization utilities to rebuild the secondary index.

Adding or Converting to Variable Length Segments

Variable-length segments are explained in detail in chapter 4. If you want to change selected segments in your data base from fixed to variable length—or convert the entire data base to variable-length segments—, the object in conversion is to put a size field in the segment you want to make variable length and then get the segment defined as variable length in the DBD. You can do that by using field level sensitivity to define a virtual field for the length field.

To convert selected segments or the entire data base:

1. Determine whether the change you are making will affect the code in any application programs. If so, make sure the code gets changed.

2. Code a PSB, using field level sensitivity to define the two-byte size fields as virtual fields.
3. Unload the data base using the HD unload utility. In the utility's parameter statement: code PLU instead of ULU, and the name of the PSB instead of a DBD name.
4. Code a new DBD for the data base that represents the changed segments as variable length, with their sizes as described in the new PSB.
5. Perform an ACBGEN.
6. Recalculate data base space. You need to do this because the change you are making will result in different requirements for data base space. See the section “Determining VSAM Space Requirements” in chapter 4 for a description of how to calculate data base space.
7. Delete the space allocated for the old cluster(s) and define space for the new cluster(s).
8. Reload the data base, using the HD reload utility and the new DBD. Remember to make an image copy of your data base as soon as it is reloaded.
9. If your data base uses logical relationships or secondary indexes, you will have to run some of the reorganization utilities before and after reloading to resolve prefix information. The flowchart in Figure 8-1 tells you which utilities and the order in which they must be run.

Converting to Use of Segment Edit/Compression

The segment edit/compression facility is explained in detail in chapter 4. If you want to convert an existing data base so it can use the facility:

1. Determine whether the change you are making will affect the code in any application programs. If so, make sure the code gets changed.
2. Unload your data base using the existing DBD.
3. Code a new DBD. The new DBD must specify the name of your edit routine for the segment types you want edited.
4. Perform an ACBGEN.
5. Recalculate data base space. You need to do this because the change you are making will result in different requirements for data base space. See the section “Determining VSAM Space Requirements” in chapter 4 for a description of how to calculate data base space.
6. Delete the old data base space and define new data base space. If you are using VSAM, use the Access Method Services DEFINE CLUSTER command to define VSAM data sets.
7. Reload the data base using the new DBD. Remember to make an image copy of your data base as soon as it is reloaded.
8. If your data base uses logical relationships or secondary indexes, you will have to run some of the reorganization utilities before and after reloading to resolve prefix information. The flowchart Figure 8-1 tells you which utilities and the order in which they must be run.

Part 5. Controlling Data Base Operation

Chapter 9. Data Administration and Security

This chapter describes methods of providing for the integrity and security of the information in a data base. There are four sections:

1. Maintaining Data Integrity
2. Controlling Data Access
3. Controlling Processing Authority
4. Controlling Access by Non-DL/I Programs

The first section identifies facilities that can be used to help maintain and monitor the integrity of the data base. The second shows how to provide data base security by limiting data access through segment and field level sensitivity definition. The third section shows how to provide security by limiting processing authority through processing option specification. The last section shows how to increase security by preventing access by non-DL/I programs, through use of data encryption.

Maintaining Data Integrity

An important consideration in data base administration is the ability to keep track of the data and how it is used. By monitoring the data base and its use, you can identify areas where a potential problem may exist. This early detection provides you an opportunity to take corrective measures.

DL/I users can use the following facilities to assist in maintaining data integrity:

- data dictionary
- the DL/I Documentation Aid facility.

Use of the Data Dictionary

A data dictionary is a tool for monitoring and maintaining data integrity. Whether you create your own dictionary or use a program such as the *DOS/VS DB/DC Data Dictionary* (Program Number 5746-XXC), you can:

- Keep track of relationships among entities in your computing environment. For example, your dictionary will tell you which programs use which data elements.
- Define authorization matrixes
- Define terminals, programs, users, data, and their relationships with each other (through the extensibility facility of the DB/DC Data Dictionary)
- List, for each user, these kinds of information:
  - Programs that may be used
  - Types of transactions that may be entered
  - Data sets that may be read
  - Data sets that may be modified
  - Categories of data within a data set that may be read
  - Categories of data that may be modified.

Using this information, you can produce reports that show potential or actual security danger areas.

DL/I Documentation Aid

The DL/I Documentation Aid is intended to provide an ease-of-use facility in documenting DL/I definitions that can be accessed directly by ISQL. This permits you to monitor the structure of the data base and its definitions.

The DL/I Documentation Aid facility is available to DL/I users who have SQL/DS and ISQL installed on their VSE systems.

When the Documentation Aid facility is invoked during ACBGEN, data base description (DBD) and program specification block (PSB) data information is automatically created for the DL/I data base. This information is inserted into a special group of SQL/DS tables.

SQL/DS Documentation Aid Tables

The following SQL/DS tables contain information on the data base description (DBD) and describe the physical characteristics of the DL/I data base:

DBDBASICDATA Names the data base, its organization, and defines each data file that makes up the data base. Only one DBD and one DATASET statement per DBD generation.

DBDACCESSDATA Defines either the primary randomized access, primary indexed access, or secondary indexed access for HD organization DL/I data bases.

DBDSEGMENTDATA Defines all occurrences of a segment type that are placed in the data base being defined. It defines a segment type, the segment's position in a data base hierarchy, the physical characteristics of the segment, and how the segment is related to other segments.

DBDLCHILDDATA Defines a logical relationship between two segment types.

DBDFIELDDATA Defines the fields within a segment.

The following SQL/DS tables contain information on the program specification block (PSB) and describe the program and its use of logical data structures.

PSBBASICDATA Names the Program Specification Block and the language of the application program using this PDB.

PSBPCBDATA Defines the Program Communication Block within the PSB.

PSBSEGMENTDATA Identifies the segments within the data base which the PSB is sensitive.

PSBFIELDDATA Identifies the fields in a physical segment which are included in this logical view of the segment. Also, defines virtual fields that do not exist in the physical segment but are used in the PB's view of the segment.

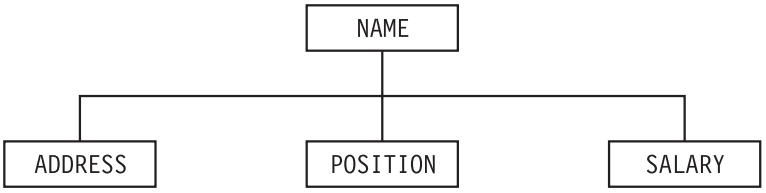
The information in these tables can be accessed through ISQL, the interactive facility of SQL/DS. An SQL/DS user can formulate a query routine or use one those supplied by IBM to display the information contained in the SQL/DS Documentation Aid tables.

For additional information on using the DL/I Documentation Aid facility, see *DL/I DOS/VS Resource Definition and Utilities*.

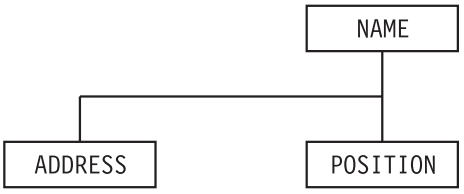
An example of the storage requirements for DL/I Documentation Aid is provided in Appendix C, “Example of Storage Requirements for DL/I Documentation Aid” on page C-1.

Controlling Data Access

Segment Sensitivity




```
DBD   NAME=PAYROLL,...
DATASET ...
SEGM  NAME=NAME,PARENT=0...
FIELD NAME=...
SEGM  NAME=ADDRESS,PARENT=NAME,...
FIELD NAME=...
SEGM  NAME=POSITION,PARENT=NAME,...
FIELD NAME=...
SEGM  NAME=SALARY,PARENT=NAME,...
FIELD NAME=...
```




```
PCB TYPE=DB,DBDNAME=PAYROLL,...
SENSEG NAME=NAME,PARENT=0,...
SENSEG NAME=ADDRESS,PARENT=NAME,...
SENSEG NAME=POSITION,PARENT=NAME,...
```

Figure 9-1. Using the PCB to Mask Segments

A PCB defines a program's view of a data base. Thus, it can be thought of as a “mask” placed over the total data base structure as defined by the DBD, hiding certain parts of it. By limiting the scope of the PCB, it is possible to limit the program's access to only certain segments of the data base. The program (and the programmer) cannot access any other segments in the data base.

Figure 9-1 shows an example. The data structure in the upper part of the figure represents a PAYROLL data base as perceived by you, and as defined by the DBD. The significant parts of the macro statements used in the DBD generation are also shown. For certain applications it may not be necessary (or desirable) to allow access to the SALARY segment. By not including a SENSEG statement for the SALARY segment in the PCB generation for these applications, you *can make it seem that this segment does not exist*. You have, in effect, changed the data structure so that it looks like the lower part of the figure.

Note that for this method to be successful in maintaining security, the segment being “masked off” must not be in the search path to a segment the application *is* allowed to access. In that case, the application would have access to the segment that is supposed to be hidden.

Field Level Sensitivity

You can achieve the same “masking” effect at the *field* level by using field level sensitivity. Field level sensitivity lets you specify only those fields in the physical definition of a given segment that an application program is to be allowed to access. If SALARY and NAME were fields in the same segment, you could still restrict access to SALARY without denying access to other fields in the segment. This is done in the PSB generation by using SENFLD statements for the segment containing the field in question. Include SENFLD statements for all fields the application is allowed to access (NAME, for instance). Omit a SENFLD statement for the field to be restricted (there would be no SENFLD statement for SALARY).

Controlling Processing Authority

Even though you allow an application program to have access to certain data, it may be necessary to control the type of processing the application can perform on that data. For instance, a certain application may need to *read* the SALARY segment, but it should be prevented from making any change to that data as it is stored in the data base. Other programs would be given the authority to *insert*, *update*, and *delete* the segment. You do this with the PROCOPT operand of either the PCB statement or the SENSEG statement in the PSB generation. You can also prevent a program from replacing the data in a particular field within a segment. You do this in the PSB generation with the REPLACE operand in a SENFLD statement that defines the field. See chapter 4 of this book for a discussion of the processing options. See *DL/I DOS/VS Resource Definition and Utilities* for the details of how to do a PSB generation.

Controlling Access by Non-DL/I Programs

A potential security exposure is the attempted access of DL/I data bases through non-DL/I programs. Data base encryption is a method of guarding against this exposure. By encrypting the data in your data bases you make the data unintelligible even though read by non-DL/I programs. You can do this by writing your own encryption routine, to be entered at either the segment edit/compression routine exit or the user field routine exit. The segment edit/compression routine can be used only with variable length segments. The user field routine can be used to encrypt individual fields.

To use the segment edit/compression routine: Name your routine in the COMPRTN operand of a SEGM statement in the DBD generation for the data base. Then, before the segment is written to the data base, DL/I will pass control to your routine, which will encrypt it. Each time the segment is retrieved, it will be decrypted by your routine before being presented to the application program.

To use the user field routine exit: Name your routine in the RTNAME operand of a SENFLD statement in the PSB generation for the data base. Then, before the field is written to the data base, DL/I will pass control to your routine, which will encrypt

the field. Each time the field is retrieved, it will be decrypted by your routine before being presented to the application program.

Remember that you must not change the key or the location of the key field.

See Chapter 4 and Appendix G, “Randomizing Modules and DL/I User Exit Routine Interfaces” on page G-1 for more details about the exits.

Chapter 10. Monitoring the Data Base

This chapter describes tools available for monitoring data base performance, and how to use them. There is one section:

Monitoring Tools and How to Use Them

The chapter describes various tools available for monitoring the performance of data bases; and how to use them in the areas of performance, need for reorganization, and access contention.

Monitoring Tools and How to Use Them

Run and Buffer Statistics for Online and MPS

You can use the run and buffer statistics to make adjustments in DL/I system parameters or reorder work loads.

The run and buffer statistics function operates as a CICS/VS transaction. It captures DL/I system statistics in the online and MPS batch environments and writes them to the CICS/VS destination “CSSL.” CSSL is defined as an extra-partition transient data set by using the DFHDCT macro. The data written to CSSL is automatically printed during CICS/VS shut-down, and includes performance information from the entire CICS/VS partition.

The function inspects DL/I control blocks to retrieve and format the following information:

- Run statistics
  - Number of PSB scheduling calls
  - Number of times at program isolation deadlock
  - Number of times at current maximum task
  - Number of duplicate PSBs created.
- Buffer statistics
  - Number of buffer handler requests
  - Number of requests satisfied from buffer pool
  - Number of buffer read requests issued
  - Number of buffer alter requests
  - Number of buffer write requests issued
  - Number of buffer blocks written
  - Number of new blocks created in pool (control interval is created because it does not exist in the data base)
  - Number of chained write requests issued
  - Number of blocks written on write chain
  - Number of retrievals by keyed calls
  - Number of retrievals by sequential logical reads
  - Number of permanent write errors encountered.

  (The run and buffer statistics above are cumulative for the current invocation of CICS/VS.)

- Buffer pool configuration
  - Number of buffer subpools
  - Number of buffers in each subpool
  - Buffer size in each subpool
  - Number of DMBs assigned in each subpool.

The output format is the same as that for CICS/VS statistics: an explanatory title for the statistic, followed by the value. A row of asterisks separates sections of the output.

Examples of the use of the statistics are:

- If the “number of times at program isolation deadlock” value is excessive (indicating a bottleneck for data base access), some of the transactions using the data base could be deferred to a later time. The same action may be required when the “number of times at current maximum task” value indicates a bottleneck for other DL/I resources.
- Excessive I/O can be detected through some of the buffer statistics. For example, an excessive number of updates would show up in the “number of buffer blocks written” value. I/O could be reduced by increasing the number of buffers in a subpool.

The transaction can be invoked from the CICS/VS master terminal, or at system shutdown as part of termination processing. In either case, the data is sent to destination CSSL.

To invoke the function from the terminal, enter the transaction identifier “CSDE” without any other parameters. Entries must have been made during CICS/VS system generation for DLZSTTL in the program control table (DFHPCT) and program processing table (DFHPPT). An entry can be made in the CICS/VS application load table (DFHALT) if the program is to be loaded in a certain order. The function will be invoked automatically during CICS/VS shutdown if the program name DLZSTTL is included in the program list table (DFHPLT) during CICS/VS system generation. The entry must be placed after the DFHELIM parameters to indicate that execution should occur during the second quiesce phase.

See the *CICS/VS Performance Guide* for further details.

Buffer Pool Statistics for Batch

The DL/I system maintains statistics in the buffer pool prefix on the activity of the entire buffer pool. Information is available about the buffer size of each subpool and the data bases that access each subpool. The buffer pool contains one pool prefix that manages all of the subpools. Each subpool contains identifying information (buffer prefix) for the buffers in the subpool.

You can use this information in determining the optimum number of subpools for a given application program. Just prior to the termination of the program, access the contents of the data base pool work area. Analysis of this information will help you determine the size of the buffer pool for subsequent executions. Figure 10-1, with the following description, shows how to get the data.

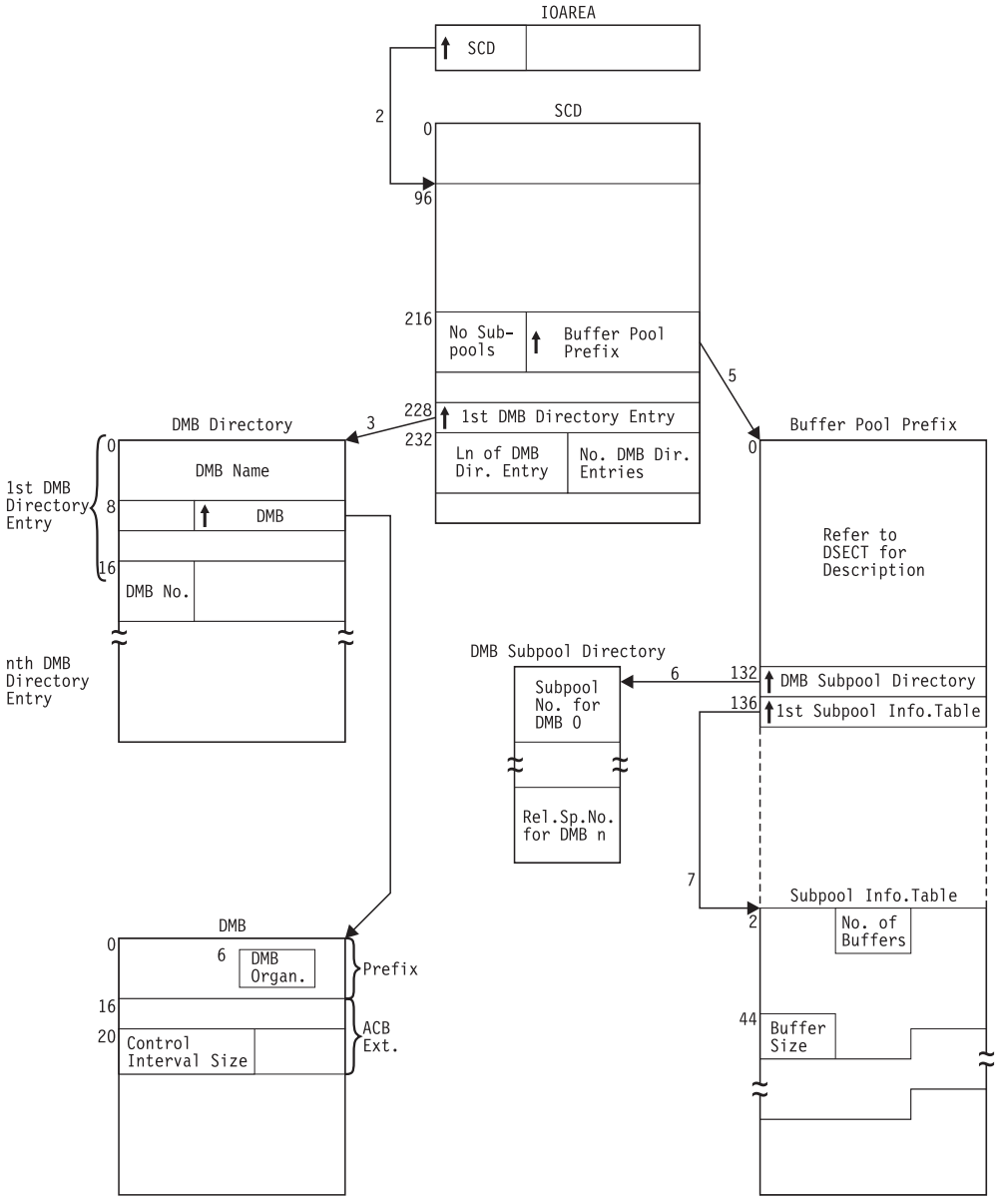


Figure 10-1. DL/I Blocks Used for Buffer Pool Statistics

1. Get the address of the DL/I system contents directory (SCD) control block. It is the major control block within the DL/I system. The SCD contains pointers to the major control block and entry point addresses for the primary DL/I modules. Because of this, it is also useful for diagnosing problems.

   You can obtain the address of the SCD by issuing a GSCD (get SCD) call. The format of the GSCD call in assembler language is:

```
        LA      1,GSCDPARM
        CALL    ASMTDLI
```

   where:

```
GSCDPARM   DC   A(GSCD)      FUNCTION CODE ADDRESS
PCBADDR    DC   A(pcb-addr)  PCB ADDRESS - NOT USED BUT
                             MUST BE VALID
           DC   X'80'
           DC   AL3(IOAREA)  INPUT/OUTPUT AREA ADDRESS
GSCD       DC   CL4'GSCD'    FUNCTION CODE
IOAREA     DC   CL8''        INPUT/OUTPUT AREA
```

   Upon return, the first four bytes in IOAREA contain the SCD address plus 96 (X'60'). The second four bytes contain the address of the partition specification table (PST).

   The GSCD call may be issued in a PL/I program by using the standard DL/I call format with three parameters (excluding the count):

   - 'GSCD' ad the function code,
   - a valid PCB, and
   - an I/O area of at least eight bytes.

   Although the GSCD call may be issued in COBOL, the addresses returned in the I/O area would be unusable in a COBOL program. In this case, it is recommended that an assembler language subroutine be used to issue the GSCD call and process the returned addresses.

   Note: The GSCD call is not supported by the High Level Programming interface (HLPI).

   Normally, no status codes are returned with the GSCD call. The only exception is when the call is issued by an online task through intersystem communication support while the task is scheduled to a PSB located on a remote system. In this case, a status code of 'AD' is returned in the PCB, indicating an invalid function code was specified.

2. The following fields in the SCD should be of use to you:

   SCDBFPL

   one byte containing the number of subpools allocated to the application, followed by a three-byte address pointer to the DL/I buffer pool control block prefix.

   SCDDLIDM

   four bytes containing the address of the first DMB directory entry.

   SCDDLIDL

   two bytes containing the length of the DMB directory entries.

   SCDDLIDN

   two bytes containing the number of DMB directory entries.

3. A DMB directory entry exists for each DMB allocated to the application. Some fields that should be of help to you are:

   DDIRSYM

   eight characters containing the symbolic name of the DMB.

   DDIRADDR

   four bytes containing an address pointer to the referenced DMB.

   DDIRNUMB

   two bytes containing the sequence number, relative to one, of this DMB. This number minus one may be used as an offset value in the DMB subpool directory to determine the subpool to which this DMB is assigned. Numbers are assigned sequentially when DMBs are loaded during DL/I initialization.

4. These fields in the DMB should be inspected by the application:

   DMBORG

   one byte containing the data base organization for this DMB. This field is in the 16-byte DMB prefix and should be tested for HDAM (X'06') or HIDAM (X'07') data bases.

   DMBCINV

   two bytes containing the VSAM control interval size for the DMB. This field is in the DL/I application control block (ACB) extension immediately following the DMB prefix. The control interval size should be matched against the assigned subpool size to determine if buffer space is being used efficiently.

5. The buffer pool control block prefix contains the statistics for the entire buffer pool. These statistics are not maintained for each subpool. The following DSECT, and description of some of the key fields it contains, should be useful for analysis of statistics:

```
BFPL     DSECT
BFPLID   DS   OF
         DC   CL4' ' POOL ID
         DC   3F'0' RESERVED
BFPLRQCT DC   F'0' NUMBER OF BLOCK REQ  RECEIVED
BFPLINPL DC   F'0' NUMBER OF REQ  SATISFIED FROM POOL
BFPLRDCT DC   F'0' NUMBER OF READ REQ  ISSUED
BFPLALTR DC   F'0' NUMBER OF BUFFER ALT  RECEIVED
BFPLOSWT DC   F'0' NUMBER OF WRITES ISSUED
BFPLBKWT DC   F'0' NUMBER OF BLOCKS WRITTEN
BFPLNWBK DC   F'0' NEW BLOCKS CREATED IN POOL
BFPLCHWT DC   F'0' NUMBER OF CHAINED WRITES ISSUED
BFPLCHBK DC   F'0' NUMBER OF BLOCKS WRITTEN ON WC
BFPLISTL DC   F'0' NUMBER OF RETRIEVES BY KEY CALLS
BFPLIGET DC   F'0' NUMBER OF GN CALLS RECEIVED
BFPLWERR DC   X'0' NUMBER OF PERMANENT WRITE ERROR
*                       BUFFERS IN POOL
BFPLWERT DC   X'0' LARGEST NUMBER OF WRITE ERROR
*                       BUFFERS EVER IN POOL
BFPLCOUT DC   X'0' NR OF ROWS/COLS  IN MATRIX CURRENTLY
*                     IN USE
BFPLROCO DC   X'0' MASK SHOW AVAILABLE ROWS/COLS  IN
*                       MATRIX
BFPLNQW1 DC   F'0' ENQ/DEQ WORKAREA 1
BFPLEXCI EQU  X'00'
BFPLPECI EQU  X'04'
BFPLSUPO EQU  X'08'
BFPLNQW2 DC   F'0' ENQ/DEQ WORKAREA 2
BFPLSW00 EQU  X'00' SWITCH
BFPLSW80 EQU  X'80' SWITCH
BFPLINMA DC   4F'0' INTERLOCK DETECTION MATRIX
BFPLINW1 DS   0CL16 INTERLOCK DET  WORKAREA 1
BFPLINRO DC   2F'0'
BFPLINCO DC   2F'0'
BFPLINW2 DC   4F'0' INTERLOCK DET  WORKAREA 2
BFPLPSI1 DC   F'0' FIELD 1 FOR PSEUDO INTERLOCK
BFPLPSIF DC   X'0' PST PREF NUMBER OF FIRST WAITING
*                       FOR MATRIX
BFPLPSIL DC   X'0' PST PREF  NUMBER  OF LAST WAITING
*                       FOR MATRIX
         DS   H  RESERVED
BFPLPRAD DC   F'0' ADDR OF BEG  OF BUFFER PREF  AREA
BFPLSUBD DC   F'0' ADDR OF BEG  OF DMB-SUBPOOL-DIR
BFPLSUIN DS   OF BEGINNING OF SPECIFIC TABLE FOR EACH
*                       SUBPOOL
```

BFPLRQCT

contains the number of requests made to the data base buffer handler by higher-level DL/I modules. It includes the count of requests for logical records, segments, or physical blocks; as well as the count of "service" requests that may not be related to I/O, such as a request to get buffer space.

BFPLINPL

contains the number of requests defined under BFPLRQCT that are satisfied from data already in the DL/I managed buffer pool without a physical I/O operation. This number should be a fraction of BFPLRQCT. The value can be increased by increasing the size of the data base buffer pool.

Note that this count is only for DL/I managed buffers that are for HD randomized, HDAM, and the data portion (ESDS) of HD indexed and HIDAM data bases. Not included are requests related to VSAM managed buffers, which includes all indexes and the ESDS for HISAM. This number is cumulative for all data bases using DL/I managed buffers.

BFPLRDCT

contains the number of physical I/O read requests performed by the buffer handler. This number should be a fraction of BFPLRQCT. The value can normally be decreased by increasing the size of the data base buffer pool.

Note that this count is only for DL/I managed buffers that are for HD randomized, HDAM, and the data portion (ESDS) of HD indexed and HIDAM data bases. Not included are requests related to VSAM managed buffers, which includes all indexes and the ESDS for HISAM. This number is cumulative for all data bases using DL/I managed buffers.

BFPLALTR

contains the number of DL/I managed data base buffers altered because of delete operations, replace operations, or insert operations to data bases.

Note that this count is only for DL/I managed buffers that are for HD randomized, HDAM, and the data portion (ESDS) of HD indexed and HIDAM data bases. Not included are requests related to VSAM managed buffers, which includes all indexes and the ESDS for HISAM. This number is cumulative for all data bases using DL/I managed buffers.

BFPLOSWT and BFPLBKWT

both contain the number of physical I/O write operations performed on DL/I managed buffers. This number should be a fraction of BFPLRQCT.

Note that this count is only for DL/I managed buffers that are for HD randomized, HDAM, and the data portion (ESDS) of HD indexed and HIDAM data bases. Not included are requests related to VSAM managed buffers, which includes all indexes and the ESDS for HISAM. This number is cumulative for all data bases using DL/I managed buffers.

BFPLNWBK

contains the number of new data base blocks, created in the DL/I managed buffer pool, that are subsequently written to data base storage. This value is the sum of new physical blocks in all data bases used.

BFPLCHWT

contains the number of write operations performed to write new data base blocks. New data base blocks are written in groups, if possible. This count applies only to DL/I managed buffers.

BFPLCHBK

contains the number of blocks written in the mode described by the parameter BFPLCHWT. This count applies only to DL/I managed buffers.

BFPLISTL

contains the number of retrievals by key field value made by DL/I module calls to the buffer handler. Requests for retrieval by key apply only to VSAM managed buffers (primary and secondary indexes).

BFPLIGET

contains the number of GET NEXT requests received. Requests for retrieval by key apply only to VSAM managed buffers (primary and secondary indexes).

BFPLWERR

contains the number of permanent write errors that occurred on blocks currently in the DL/I managed buffer pool.

BFPLWERT

contains the largest number of permanent write errors ever encountered in the DL/I buffer pool during execution.

BFPLSUBD

contains an address pointer to the DMB subpool directory.

BFPLSUIN

contains an address pointer to the first subpool information table.

6. The DMB subpool directory contains a one-byte subpool number, relative to zero, for each DMB allocated. The DMB sequence number is used as an offset value in the DMB subpool directory. It allows a DMB to be identified with a specific subpool.
7. The subpool information table contains, in the second byte, the number of buffers in the subpool. The subpool information table also contains, in byte 44, the subpool buffer size in field SUBBFSIZ. This one-byte field is expressed as:

```
X'01'   512-byte buffer
X'02'   1024-byte buffer
X'03'   1536-byte buffer
X'04'   2048-byte buffer
X'05'   2560-byte buffer
X'06'   3072-byte buffer
X'07'   3584-byte buffer
X'08'   4096-byte buffer.
```

Buffer Statistics Output

Whenever you are using preprinted forms in a batch environment, you should not use SYSLST as your designated output device since DL/I will automatically use SYSLST to print statistical information. If you do not want this statistical information written to your preprinted forms, you should designate a device other than SYSLST (e.g., SYS003) for your printed output.

Trace Facility

DL/I provides a trace facility that, while its primary purpose is for problem determination, can be used to assist in monitoring. Nine different types of traces are available. Any combination of them can be selected through an option parameter. The trace types are:

- User calls
- DL/I module flow
- Retrieve job control block (JCB) values
- Current position in data base
- Exit/return to VSAM
- Exit/return to buffer handler
- Requests to index maintenance
- Requests to online buffer handler
- Program isolation queueing calls.

These can be further limited by specifying data base, key range, call number, function, PSB, segment type, and other conditions. The trace facility can be used in the batch, MPS batch, or online DL/I environments.

A Trace Print Utility is provided to let you print trace entries from tape or disk input files. A control statement lets you specify that only specific information be selected from the input file for printing.

The DL/I trace facility and the trace print utility are described in *DL/I DOS/VS Diagnostic Guide*.

Data Base Utilities

Some of the DL/I data base utilities provide output that is useful in monitoring your data base. The HISAM Reorganization Unload and HISAM Reorganization Reload utilities, and the HD Reorganization Unload and HD Reorganization Reload utilities all produce output statistics about records and segments.

For detailed information on using DL/I utilities, see *DL/I DOS/VS Resource Definition and Utilities*. Job streams for these utilities can be generated interactively as described in *DL/I DOS/VS Interactive Resource Definition and Utilities*.

CICS/VS Monitoring Facilities (CMF) Hooks

The CICS/VS Monitoring Facilities (CMF) let you, as a CICS/VS DL/I DOS/VS user, collect performance data during online processing, for later offline analysis. Although three monitoring classes are provided by CICS/VS (accounting, performance, and exception), DL/I uses only the performance class. For detailed information about CICS/VS Monitoring Facilities, see the *CICS/VS Customization Guide*.

Monitoring data recorded in the CICS/VS journal data sets can be formatted using the CICS/VS Performance Analysis Reporting System (CICS/PARS) (FDP), Program Number 5798-DAB. This field developed program, used in conjunction with the DOS/VS System for Generalized Performance Analysis Reporting (DOS/GPAR) (FDP), Program Number 5798-DAA, prints analysis reports and summary reports of DL/I performance data as user clocks and counters.

Note: These utilities are not shipped as part of DL/I. If you want to use them, you have to order them separately.

Monitored DL/I Events

CMF performance data for DL/I is collected only for the online environment (including MPS batch mirror program, DLZBPC00). Performance data is collected for seven DL/I events:

- Task scheduling delay
- PSB scheduling delay
- Program isolation (PI) queue search delay
- Program isolation (PI) task wait delay
- VSAM I/O wait time
- Time spent in DL/I to process DL/I requests
- Time spent waiting for a response to a remote request.

Performance data is also collected to record the number of DL/I requests performed by DL/I functions. At task termination, twenty-two counters are updated with the following information:

- Count of DL/I requests by function (21 counters)
- Current number of program isolation resources (1 counter).

The event types are defined as follows:

- Task scheduling delay

  The clock runs from the time a task is suspended due to the CMAX task limitation until it is resumed by termination of another task. (CMAX is the maximum number of DL/I tasks currently allowed to be simultaneously scheduled in a given DL/I system.) The counter associated with this event will be incremented each time a task is suspended due to the CMAX limitation.

  A delay may also be caused during task scheduling if sufficient CICS/VS storage is not available to build a DL/I PST (partition specification table) and related control blocks. This delay is not measured by this (or any other) DL/I event type.

- PSB scheduling delay

  The clock runs from the time a task is suspended due to a scheduling conflict until the scheduling conflict is resolved. All tasks suspended for intent conflict are resumed by termination of another task. Therefore, some of these tasks may be suspended and resumed several times before the scheduling conflict is resolved. When this occurs, the counter associated with this event will not be incremented each time the task is suspended.

- Program isolation (PI) queue search delay

  The clock runs from the time a search is initiated for a particular RDB (Resource Descriptor Block) element on the queue, until the RDB is found. The elapsed time represents the time spent for queue search, and varies with the number of queues and enqueued resources. The associated counter is incremented when the search is completed. PI queue search time on a remote system is not monitored as a separate DL/I event on the local system.

- Program isolation (PI) task wait delay

  The clock runs from the time a CICS/VS WAIT is issued during program isolation task enqueueing until the task is resumed. This elapsed time measures the data base contention among tasks with update intent for the same segment *occurrence*. The counter associated with this event is incremented when the task is resumed. PI suspend time on a remote system is not measured as a separate DL/I event on the local system.

- VSAM I/O wait time

  The clock measures I/O wait time for local VSAM I/O requests issued by DL/I. The clock runs from the time the I/O wait begins until the task is given control after I/O completion. The counter associated with this event counts the total number of VSAM I/O waits for this task. Since DL/I buffer management determines when to schedule I/O, these counts may actually be on behalf of a previous task's request. I/O wait time in a remote system is not monitored as a separate DL/I event on the local system.

- Time spent in DL/I to process DL/I requests

  The clock runs from the time the DL/I Program Request Handler is entered to process a DL/I request until control is returned by DL/I to the application program. This event measures the time this task spent performing DL/I requests. Both local and remote data base access time is included in this event. The counter associated with this event is incremented each time control is returned to the application program. DL/I system calls (such as CMXT, STRT, etc.) and the GSCD call are included in this event, but are not included in the “Count of DL/I requests by function” event.

- Time spent waiting for a response to a remote request

  The clock runs from the time a data base request is given to the CICS/VS Intercommunication Routines for processing on a remote system until control is returned to DL/I. The counter associated with this event represents the total number of requests issued by the task that involve a remote data base.

- Count of DL/I requests by function

  A counter is incremented for each of the following DL/I requests processed on the local DL/I system:

  Note: DL/I HLPI GET commands are counted as GET HOLD requests.

  'DLET' - Delete
  'GHN ' - Get Hold Next
  'GHNP' - Get Hold Next within Parent
  'GHU ' - Get Hold Unique
  'GN ' - Get Next
  'GNP ' - Get Next within Parent
  'GU ' - Get Unique
  'ISRT' - Insert
  'REPL' - Replace
  'CHKP' - Checkpoint

Clocks and Counters

Clocks: You define the CMF clocks and counters for the DL/I events you wish to monitor by the entries you include in the CICS/VS Monitoring Control Table (MCT). Seven of these events require a clock to record the duration of the event. The following table shows the clocks associated with each DL/I event.

CLOCK NUMBER	EVENT
1	Task scheduling delay
2	PSB scheduling delay
3	PI queue search delay
4	PI task wait delay
5	VSAM I/O wait time
6	Total time in DL/I (local and remote)
7	Remote time

A CMF counter is allocated by CICS/VS for each of the seven DL/I event clocks. These counters are used to record the number of occurrences of each DL/I event.

Note that it is possible for a clock to be started and not stopped during the execution of a task, e.g. the task is cancelled while suspended due to the maximum task scheduling limitation.

For DL/I system calls (CMXT, STRT, etc.) and the GSCD call, the only event that is monitored is “Time spent in DL/I to process DL/I requests”.

Counters top The “Count of DL/I requests by function” event does not require a CMF clock. This event requires twenty-two counters, in addition to the seven counters used with the clocks, to record the number of DL/I requests by function.***:***

The following table shows the information recorded by each counter when the "Count of DL/I requests by function" event is defined in the CICS/VS Monitoring Control Table:

COUNTER NUMBER		USE
1		'PCB' Schedule (successful only)
2		Number of resources owned (PI only)
COUNTER NUMBER (local)	COUNTER NUMBER (remote)	DL/I FUNCTION
3	13	'GU' Get Unique
4	14	'GN' Get Next
5	15	'GNP' Get Next Within Parent
6	16	'GHU' Get Hold Unique
7	17	'GHN' Get Hold Next
8	18	'GHNP' Get Hold Next Within Parent
9	19	'ISRT' Insert
10	20	'DLET' Delete
11	21	'REPL' Replace
12	22	'CHKP' Checkpoint

Activating the CICS/VS Monitoring Facilities

You activate the monitoring facilities for CMF Hooks by defining entries in the CICS/VS Monitoring Control Table, Journal Control Table, and the System Initialization Table. While CICS/VS is active, you can activate/deactivate an event monitoring class using the CSTT transaction.

Monitoring Control Table: Use the DFHMCT TYPE=MCT macro to define clocks and counters to record the DL/I monitored events.

Use the DFHMCT TYPE=RECORD macro to define the CICS/VS user journal to which the data is to be sent for each class of data being collected (in this case, the performance class).

This is an example of how to code your Monitoring Control Table to activate all of the DL/I clocks and counters:

```
// JOB MCTPP ASSEMBLE
// OPTION CATAL,NODECK,NOXREF
// EXEC ASSEMBLY
  DFHMCT TYPE=INITIAL,SUFFIX=PP
*
* CLOCK/COUNTER FOR TASK SCHEDULING DELAY
*
  DFHMCT TYPE=EMP,ID=(PP,1),CLASS=PER,PER=SCLOCK(1)
  DFHMCT TYPE=EMP,ID=(PP,2),CLASS=PER,PER=PCLOCK(1)
*
* CLOCK/COUNTER FOR PSB SCHEDULING DELAY
*
  DFHMCT TYPE=EMP,ID=(PP,3),CLASS=PER,PER=SCLOCK(2)
  DFHMCT TYPE=EMP,ID=(PP,4),CLASS=PER,PER=PCLOCK(2)
*
* CLOCK/COUNTER FOR PROG ISOLATION QUEUE SEARCH DELAY
*
  DFHMCT TYPE=EMP,ID=(PP,5),CLASS=PER,PER=SCLOCK(3)
  DFHMCT TYPE=EMP,ID=(PP,6),CLASS=PER,PER=PCLOCK(3)
*
* CLOCK/COUNTER FOR PROG ISOLATION TASK SUSPENSION DELAY
*
  DFHMCT TYPE=EMP,ID=(PP,7),CLASS=PER,PER=SCLOCK(4)
  DFHMCT TYPE=EMP,ID=(PP,8),CLASS=PER,PER=PCLOCK(4)
*
* CLOCK/COUNTER FOR VSAM I/O WAIT TIME
*
  DFHMCT TYPE=EMP,ID=(PP,9),CLASS=PER,PER=SCLOCK(5)
  DFHMCT TYPE=EMP,ID=(PP,10),CLASS=PER,PER=PCLOCK(5)
*
* CLOCK/COUNTER FOR TOTAL TIME TO PROCESS DL/I REQUESTS
*
  DFHMCT TYPE=EMP,ID=(PP,11),CLASS=PER,PER=SCLOCK(6)
  DFHMCT TYPE=EMP,ID=(PP,12),CLASS=PER,PER=PCLOCK(6)
*
* CLOCK/COUNTER FOR TOTAL TIME TO PROCESS REMOTE DL/I REQUESTS
*
  DFHMCT TYPE=EMP,ID=(PP,13),CLASS=PER,PER=SCLOCK(7)
  DFHMCT TYPE=EMP,ID=(PP,14),CLASS=PER,PER=PCLOCK(7)
*
* DL/I FUNCTION COUNTERS
*
  DFHMCT TYPE=EMP,ID=(PP,15),CLASS=PER,PER=(MLTCNT(1,22))
*
* CICS/VS USER JOURNAL
*
  DFHMCT TYPE=RECORD,CLASS=PER,DATASET=2,MAXBUF=2040,FREQ=100
  DFHMCT TYPE=FINAL
         END
/*
// EXEC LNKEDT
/&
```

Journal Control Table: Journals used to record the monitoring facilities output data must be defined in the Journal Control Table (JCT) as user journals by specifying a JFILEID value between 02 and 99 in the DFHJCT TYPE=ENTRY macro. FORMAT=SMF must also be specified in this macro so that the SMF block format is used instead of the CICS/VS block format.

This is an example of how to code the JCT:

```
// JOB CMF HOOKS JCT DEFINITION
// OPTION CATAL,NODECK,NOXREF
// EXEC ASSEMBLY
         PRINT NOGEN
         DFHJCT TYPE=INITIAL,SUFFIX=PP
         DFHJCT TYPE=ENTRY,JFILEID=SYSTEM,BUFSIZE=1024,BUFSUV=1024,    *
               DEVADDR=SYS011,JOUROPT=(CRUCIAL,INPUT),JTYPE=DISK1,     *
               JDEVICE=3330
         SPACE 2
         DFHJCT TYPE=ENTRY,JFILEID=2,BUFSIZE=4096,BUFSUV=4096,         *
               JTYPE=TAPE1,OPEN=INITIAL,FORMAT=SMF,                    *
               DEVADDR=SYS010
         DFHJCT TYPE=FINAL
         END   DFHJCTBA
/*
// EXEC LNKEDT
/&
```

System Initialization Table: To activate the CMF PERFORMANCE monitoring class at system initialization time, use the MONITOR operand of the DFHSIT macro (MONITOR=PER). You can also specify the MONITOR operand as an override parameter during CICS/VS initialization.

PPT and PCT Entries: The entries required for event monitoring are included in CICS/VS PPT and PCT by specifying FN=STANDARD on the DFHPPT TYPE=GROUP and the DFHPCT TYPE=GROUP macro instructions.

CSTT Transaction: You can also use the CSTT transaction to activate and deactivate any of the monitoring classes. See the *CICS/VS Operator's Guide* for details.

Space Management Utilities IUP

The DL/I DOS/VS Space Management Utilities (Program Number 5796-PKF) are a collection of utilities that can be of help in monitoring your data base.

The HD pointer checker utility monitors space utilization and detects and reports direct pointer problems. If pointer problems are detected, all bad pointer records are identified and the actual disk addresses are displayed for correction. Disk space used is analyzed, and free space and pointer statistics are reported. The statistics reported are:

- Summary of pointer types at both data base and segment levels
- Segment occurrences, overflow occurrences, deleted segment occurrences, and counter field values
- Full blocks, empty blocks, and free space
- Bit map
- Root key values for use by the HD tuning aid utility
- Average data base record size
- Number of dependents in same block with root
- Actual versus randomized root address.

The HD tuning aid utility uses the root key file produced by the HD pointer checker utility. An “actual roots per block” map is created and a root anchor point (RAP) chain analysis is performed. This utility shows you how the data is actually stored in each block throughout the data base. You can use this utility to conduct iterative RAP chain analyses to determine the effects of changing data base parameters or randomizing routines.

The HDAM physical block reload utility takes the output from the DL/I HD unload utility, passes the keys through your randomizing

routine, and sorts on randomized address. The sorted file is used to produce a RAP chain analysis and as input to the DL/I HD reload utility. Reload is then done in physical block sequence. You can use this utility to conduct iterative RAP chain analysis to determine the effects of changing data base parameters or randomizing routines.

See *DL/I DOS/VS Space Management Utilities Program Description/Operations Manual*, SH20-2107 for details about the programs and how to use them.

Chapter 11. Improving Data Base Performance

This chapter describes the availability and use of tools for improving data base performance. There is one section:

Tools For Improving Performance and How To Use Them

This chapter describes the tools that are available for use in improving the performance of operating data bases, how to decide which of the tools to use, and how to use them.

After your data base system is operating and applications are running, you will be concerned with how efficiently the system is doing its job. Even though the end users are getting useful work from the system, it may not be making the best use of the resources available to it. You want to make sure that the total performance is the optimum that can be attained. The first step toward this goal is to monitor the operation of the system. Chapter 10 tells you how to do that. When you analyze the data that comes from your monitoring operations, certain areas where performance could be improved may show up. This chapter shows you how to use the tools available to you to make changes that will improve your system's performance in many of those areas.

Many of the changes you can make are not simple ones, and you may want to make more than one change at the same time. For instance, you might want to reorganize a data base and change its DL/I access method at the same time. This chapter gives you many procedures for performing different types of changes. If you combine procedures to do multiple changes, use care and plan carefully before you begin.

Some of the changes you can make may affect the logic in application programs. You can often use your data dictionary to analyze the effect before making changes. In addition, some changes require that you code new DBDs and PSBs. If you make the necessary changes in the dictionary first, you can then use the dictionary to help create the new DBDs and PSBs.

Tools For Improving Performance and How To Use Them

Reorganization Utilities

When speaking of reorganizing a data base we can be thinking in terms of either its structure or the storage space it uses.

You might want to change the *structure* of a data base when changing user requirements necessitate changes in the data base design, when you wish to use new or different options or features, or perhaps simply when you've found a more efficient way to structure the data base. Structural changes to a data base can often be made using the reorganization utilities discussed in this chapter. However, structural changes are discussed in chapter 8, and the use of the DL/I utilities in making such changes is part of the information given there. In this chapter we will

be principally concerned with using the utilities in reorganizing data bases to change their storage requirements or layout.

You might want to change the *storage* of a data base, that is, the way in which data base records and segments are stored, when use of space in storage has become inefficient. This happens as a data base grows and parts of a data base record get widely scattered or available space for adding new records is used up.

When Should You Reorganize?

You should reorganize your data base when performance is decreasing to an unacceptable level because segments in a data base record are stored across too many CIs or blocks. And you should reorganize when you're running out of free space in your data base.

The various aids that exist for monitoring a data base help you to decide when it's time to reorganize. These tools are discussed in chapter 10.

Steps in Reorganizing

There are three basic steps in reorganizing a data base (when you're not making structural changes to it):

1. Unload the existing data base
2. Delete the old data base space and define new data base space. Note that for VSAM, data base space refers to the clusters defined to VSAM for data base data sets.
3. Reload the data base.

Protecting Your Data Base

Because you delete your data base in the process of reorganizing it, you should protect it in case of system failure or in case the reorganization fails. You can do this by renaming the space it occupies, then defining new data base space. You should make a copy of your data base as soon as it's reloaded and before any application programs are run against it. You need a backup copy in case of system failure. You can copy your data base using the data base image copy utility, which is described in detail in *DL/I DOS/VS Resource Definition and Utilities*.

Choosing the Right Utilities

Reorganization of a data base is done by using a combination of utilities. Which utilities you need to use, and how many, depends on the type of data base and whether it uses logical relationships or secondary indexes.

If your data base doesn't use logical relationships or secondary indexes, you simply run the appropriate unload and reload utilities, which are:

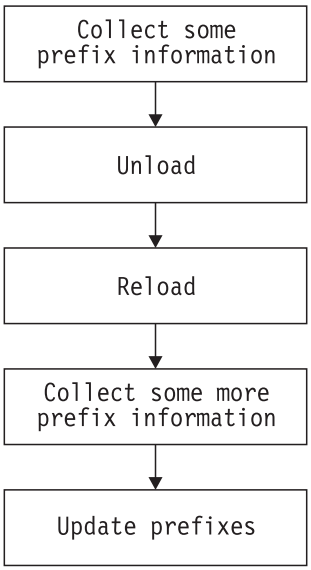
- For HD, HDAM, and HIDAM data bases: the HD reorganization unload utility and the HD reorganization reload utility.
- For HISAM and SHISAM data bases: the HISAM reorganization unload utility and the HISAM reorganization reload utility
- For the primary index data bases of HD indexed or HIDAM data bases, the secondary index of any HD data base (if reorganized separately from the main data bases): the HISAM reorganization unload utility and the HISAM reorganization reload utility

If your data base does use logical relationships or secondary indexes, you have to run the HD reorganization unload and reload utilities. In addition, you have to run

other utilities to collect, sort, and restore pointer information from the segment prefixes. Remember: when a data base is reorganized, the location of segments changes. If logical relationships or secondary indexes are being used, prefixes have to be updated to reflect new segment locations. The various utilities involved in updating segment prefixes are:

- Data base prereorganization utility
- Data base scan utility
- Data base prefix resolution utility
- Data base prefix update utility

If logical relationships or secondary indexes exist (requiring use of the latter four utilities), the sequence in which operations occur is something like:



Because of this, for instance, the HD reorganization reload utility doesn't just reload the data base if a secondary index or logical relationship exists. It reloads the data base using as one input a data set containing some of the prefix information that's been collected. And it produces as output from the reload, a data set containing more prefix information. As the various utilities do their processing they use data sets produced by previously-executed utilities and produce data sets for use by subsequently-executed utilities.

As we have said, DL/I provides a number of utility programs that you can use in reorganizing data bases to improve performance. These utilities are divided into two types: those dealing directly with data base reorganization, called physical reorganization utilities; and those dealing with resolution of logical relationships, called logical relationship resolution utilities. Each utility is described briefly in a separate section below.

You can code the DL/I reorganization utility job streams or, if you have a 3270-type terminal available, you can use the Interactive Utility Generation (IUG) facility to generate job streams for the utilities.

For information on IUG and how to use it, see *DL/I DOS/VS Interactive Resource Definition and Utilities*.

If you choose not to use IUG, *DL/I DOS/VS Resource Definition and Utilities* gives you information on how to code and execute the utilities necessary to perform the various reorganization functions.

You do not have to use DL/I reorganization utilities to reorganize your data base. You can write your own programs to unload and reload data. You'd probably want to do this only if you're making structural changes to your data base that can't be done by using the utilities. Information about when these utilities can be used to make structural changes to a data base is contained in chapter 8.

Data Base Prereorganization

This is one of the logical relationship resolution utilities. It is not used when the data bases involved are known to contain no logical relationships or for which no secondary indexes exist. When you are reorganizing data bases that do involve logical relationships, or that may involve them, you must run this utility before any of the other logical relationship utilities. It is also used when reorganizing data bases for which secondary indexes exist.

The input to this utility consists of control statements that name the data base or bases that are being reorganized.

Its output consists of a message listing and a control file that is used as input to the data base scan utility, the data base prefix resolution utility, and the data base work file generator module.

Data Base Scan

This is one of the logical relationship resolution utilities. Its function is to search one or more data bases for all segments that are involved in logical relationships. For each segment found, the utility writes one or more output records (depending on the relationships in which the segment is involved) to an output work file.

Input to the data base scan utility consists of control statements, the control file created by the data base prereorganization utility, and the data bases to be scanned. The data bases can be HD, HDAM, or HIDAM (with its primary index). If the data base being scanned contains logical relationships or secondary indexes, all related data bases must also be online.

Output consists of an output work file to be used as input to the data base prefix resolution utility, and a message listing.

Data Base Prefix Resolution

This is one of the logical relationship resolution utilities. It is used to accumulate information from work files generated during the reorganization of the data base or bases. It combines and sorts all work files specified as input to it. The input work files contain records that describe each segment of a data base in each of its logical or secondary index relationships. For example, there may be one record describing the use of a segment as a logical parent of logical child type A, another record describing its use as a logical parent of logical child type B, and so on. Based on this input data, the prefix resolution utility determines the actual values that must be placed in the prefix fields of each segment to complete all logical and secondary index relationships defined for the data base or bases being reorganized. The output work files from this utility contain the data that is to be placed in each segment prefix.

The input to this utility consists of the control file created by the data base prereorganization utility, all work files generated by the data base scan utility or reload utilities, and a control statement.

Output consists of a work file containing all sorted input work file records for logical relationships, a similar file for secondary index relationships, and a message listing.

Data Base Prefix Update

This is one of the logical relationship resolution utilities. It is used to update logical parent, logical twin, and logical child pointer fields in the prefixes of segments involved in logical relationships or secondary indexing. Using the update records generated by the prefix resolution utility, the prefix update utility reads input records from the input work file, reads the corresponding segments, and applies the changes indicated in input records to the segment prefixes.

Input to this utility consists of one or two work files created by the data base prefix resolution utility, one or more data bases (HD, HDAM, or HIDAM (with its primary index)), and a control statement.

Output consists of the updated data base and a message listing.

HD Reorganization Unload

This is one of the physical reorganization utilities. It is designed for use in reorganizing HD, HDAM, or HIDAM data bases; and for making structural changes to HISAM and simple HISAM data bases. (The HISAM utilities should be used for reorganizing HISAM and simple HISAM data bases when there are no structural changes.)

This utility can also be used to reformat data, add new fields for an application program, and move a subset of the data base to another location for faster processing. Segment and field level sensitivity functions are used to accomplish these changes. The data base is unloaded using a special PSB that defines the changed segments and fields, then reloaded using a physical DBD that matches the PSB. The only change you have to make in using the unload utility is to name, in the control statement, the PSB you want to use for unloading.

The HD reorganization unload utility operates by issuing GET NEXT requests to access the segments of the specified data base. It then writes a sequential output file containing each active segment and DL/I utility prefix information, arranged in hierarchical sequence. This file is used by the HD reorganization reload utility.

The input to the program consists of a DMB and a unique PSB loaded from a core image library, the data base to be unloaded, and any primary or secondary index data bases related to the data base to be unloaded. (These data bases are not unloaded.)

The output consists of one or two copies of the unloaded data base, as specified, and a message listing.

HD Reorganization Reload

This is one of the physical reorganization utilities. It is used to reload a data base from the sequential file created by the HD reorganization unload utility.

You have to execute the VSAM Access Method Services utility command DELETE to remove the name of the file from the VSAM catalog to release the space allocated to it. Then you have to define the new file to VSAM to cause its definition to be recorded on the VSAM catalog.

One of the inputs to the HD reorganization reload utility is the DMB of the data base being reorganized. By replacing the original DMB with a new one, certain structural changes can be made to the data base during reorganization. There are certain rules and restrictions you must follow when you do this:

- The original DMB must have been used when the data base was unloaded with the HD reorganization unload utility
- New segment types can be added in the DBD if they don't change the hierarchical relationship of existing segment types
- Names of existing segment types must not be changed
- Except for the sequence field, the definition of any field can be changed, added, or deleted
- The DL/I access method can be changed to any other with the exception of HD randomized (HDAM) to HD indexed (HIDAM).
- Segment pointer options for HD, HDAM, or HIDAM can be changed
- An existing segment type can be deleted from the DBD if all occurrences of this segment type had been deleted from the data base before the HD reorganization unload utility was executed.
- Before executing the reload utility, the new DBD must be assembled, link-edited in a core image library, and the existing DMB deleted from the core image library. The application control blocks creation and maintenance utility must be executed for all PSBs that reference the changed DBD, and its output must be link-edited and cataloged in a core image library. (This paragraph does not apply when no structural changes are made. The original DMB of the data base is used in that case.)

If any data bases are logically related to the data base being reorganized, the data base scan utility must be executed as indicated by the data base prereorganization utility.

The HD reorganization reload utility operates by reading records from the file created by the unload utility and issuing INSERT requests to insert the segments. The result is a data base organized in an efficient accessing format, with all fragmented free space consolidated.

Input to the HD reorganization reload utility consists of a DMB and a unique PSB loaded from a core image library, a copy of the previously unloaded data base, and a control file created by the prereorganization utility, if logical relationships exist.

Output consists of the reorganized data base; a work file to be used by the prefix resolution utility, if logical relationships exist; and a message listing.

HISAM Reorganization Unload

This is one of the physical reorganization utilities. The HISAM reorganization unload utility is used in conjunction with the HISAM reorganization reload utility to reorganize a HISAM data base to remove unused space that came about as the result of segment deletions or other reasons. This results in faster, more efficient operation of the data base and better use of storage space. You can change the data base logical record length and blocking factors to create a more efficient physical data base structure. This is done by generating a new DBD and creating a new DMB from it, with the application control blocks creation and maintenance utility, *before unloading* the data base.

If you want to reorganize a primary or secondary index data base, without reorganizing the associated data base, you can do it by using this and the HISAM reorganization reload utility. All that you have to do is specify the index data base name in the unload utility control statement.

The HISAM reorganization unload utility operates by reading the data base KSDS records, picking up any overflow dependent segments from the ESDS, dropping deleted root and dependent segments, and writing a sequential file arranged in a physically related sequence.

Input to this utility consists of a control statement, the DBD of the data base to be unloaded, and the KSDS and ESDS files of the data base.

The output consists of one or two copies of the unloaded data base and a message listing.

HISAM Reorganization Reload

This is one of the physical reorganization utilities. This utility is used to reload a data base that was unloaded and reorganized with the HISAM reorganization unload utility. It is used only with HISAM, simple HISAM, and index data bases.

Before executing this utility, you have to execute the VSAM Access Method Services utility command DELETE to remove the name of the file from the VSAM catalog to release the space allocated to it. Then you have to define the new file to VSAM to cause its definition to be recorded on the VSAM catalog.

The HISAM reorganization reload utility operates by reading the file created by the HISAM reorganization unload utility and writing the records to the KSDS or ESDS, as required.

Input to the utility consists of a control statement and the input file containing the unloaded data base.

Output consists of the reloaded KSDS and ESDS (for HISAM) files and a message listing.

Partial Data Base Reorganization

This is one of the physical reorganization utilities. The partial data base reorganization utility lets you reorganize a selected range of data base records into a designated target area in the data base. It can be used with HD, HDAM, or HIDAM data bases. For an HD indexed or HIDAM data base, the range is defined by a set of low-high key values in the primary index data set. For an HD randomized or HDAM data base, the range is defined by a set of low-high relative

block numbers in the primary area. Data base structural changes can't be made with this utility.

The DL/I DOS/VS Space Management Utilities IUP (program number 5796-PKF), can be used to produce a report that will help you identify the ranges that need reorganization, and a target area.

The partial reorganization takes place in two steps:

- The first step does prereorganization functions to check for user errors, without requiring use of the data base. The range of the data base to be reorganized is determined. Any logically related data bases needing pointer resolution are identified. Control tables are then built for the processing in the second step. A special PSB is required for the second step. PSB source statements can be produced for creating it. This PSBGEN and the associated ACBGEN must be run before the second step is executed. Once these generations have been done, they don't have to be repeated for subsequent reorganizations of the same data base. A report is produced containing information on the data base and data set(s) being reorganized, the range(s) selected for reorganization, and the names of any other data bases that may also have to be scanned.

  Input to the first step of the partial data base reorganization utility consists of control statements and the DBDs of the data bases involved (located in a core image library).

  Output consists of control tables, PSB source statements, the report mentioned above, and a message listing.

- The second step does the actual reorganization of the data base. It is executed as a normal batch job, and requires the use of the data base.

  This job uses control statements and the control tables created in the first step. The root and dependent segments specified are unloaded in hierarchical order into work files. The space occupied by the unloaded segments is freed. The unloaded segments are then inserted into the designated target area of the data base. The old and new segment addresses are recorded for subsequent pointer resolution. All of the segments of any logically related data bases indicated in the data base table created in the first step are scanned. Work records are produced for segments that require pointer resolution. The segments in each data base affected by the reorganization are accessed in a physical sequential order, and the old pointer values are replaced with the resolved values.

  Input to this step consists of the control tables created in the first step, control statements, the data bases involved, and their DBDs and the control blocks created in the special PSB and ACB generations.

  Output consists of the partially reorganized data base.

Procedures

Reorganizing a HISAM or SHISAM Data Base: To reorganize a HISAM data base when there are no structural changes:

1. Unload the data base using the HISAM reorganization unload utility.
2. Delete the space allocated for the old VSAM data set cluster(s) and define space for the new cluster(s).
3. Reload the index data base using the HISAM reorganization reload utility. Remember to make an image copy of your data base as soon as it is reloaded.

Reorganizing an HD Data Base (No Logical Relationships or Secondary Indexes): To reorganize an HD data base when it doesn't use logical relationships or secondary indexes:

1. Unload the data base using the HD reorganization unload utility.
2. Any time you unload a data set, you should delete and redefine space before reloading.
3. Reload the data base using the HD reorganization reload utility.

Reorganizing a Primary or Secondary Index: HD indexed and HIDAM has a primary index. HD, HDAM, and HIDAM have separate secondary index data bases when secondary indexing is being used. Both types of index are reorganized in the same way. To reorganize a primary or secondary index:

1. Unload the index data base using the HISAM reorganization unload utility.
2. Any time you unload a data set, you should delete and redefine space before reloading.
3. Reload the index data base using the HISAM reorganization reload utility. Remember to make an image copy of your data base as soon as it is reloaded.

Reorganizing an HD, HDAM, or HIDAM Data Base (With Logical Relationships or Secondary Indexes): Figure 11-1 shows you the steps you must go through to reorganize an HD, HDAM, or HIDAM data base that uses logical relationships or secondary indexes. Those steps are:

1. Run the HD unload utility to unload the data base.
2. Run the prereorganization utility to get information needed later to resolve secondary index or logical relationships.
3. If data bases *not* being reorganized contain segments involved in a logical relationship with the data base being reorganized, run the scan utility. The scan utility is run against the data base that is *not* being reorganized. The scan utility gets some more of the information needed later to resolve logical relationships.
4. Any time you unload a data set, you should delete and redefine space before reloading.
5. Run the HD reload utility to reload the data base.
6. Run the prefix resolution utility to accumulate and sort the prefix information collected in steps 3 and 5.
7. If the reloaded data base uses logical relationships, run the prefix update utility against the reloaded data base. It updates the prefixes of all segments involved in logical relationships.
8. Remember to make an image copy of your data base.

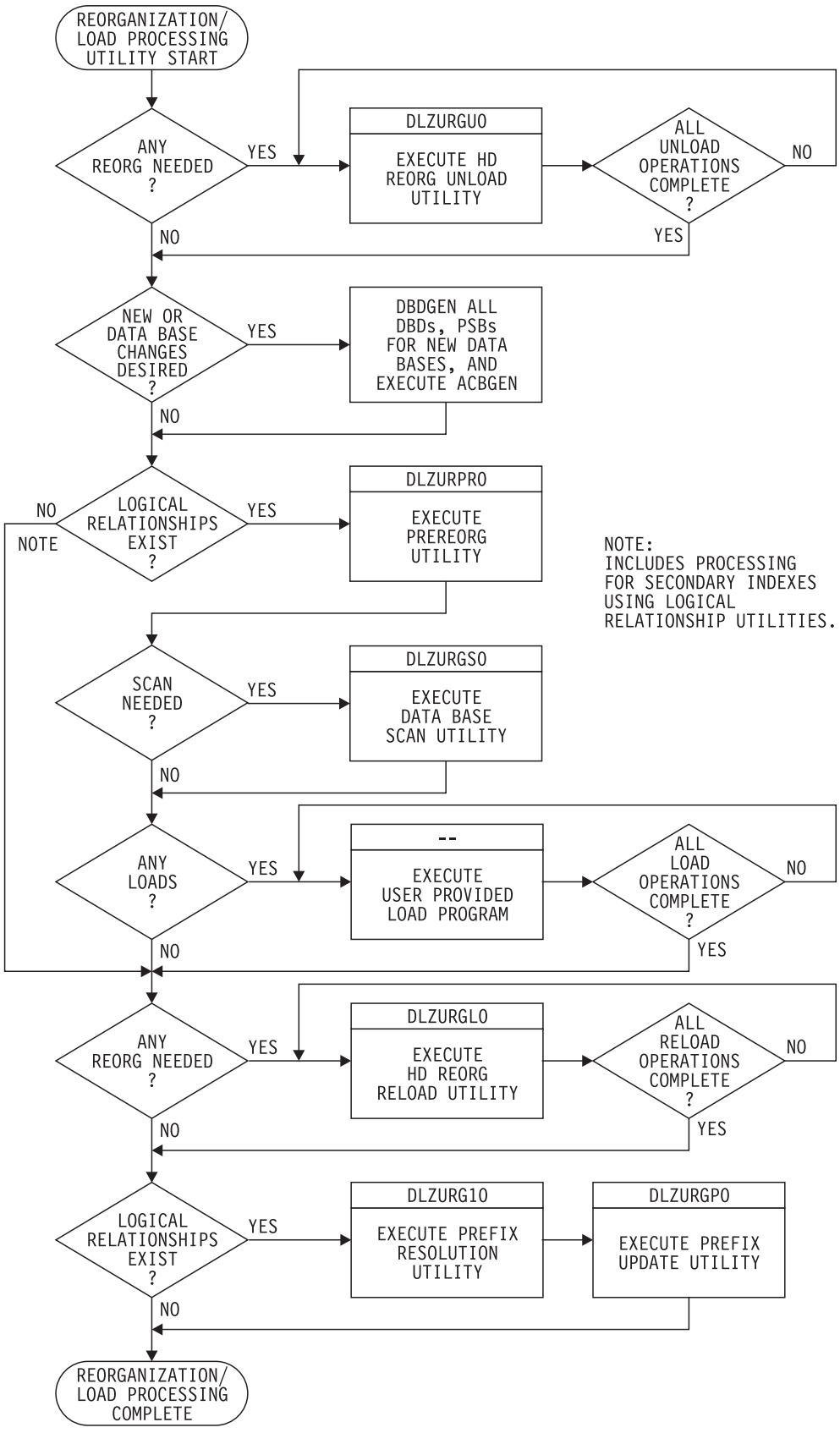


Figure 11-1. Sequence of Execution of Utilities When Making Data Base Changes During Reorganization

HD Parameters

The HD options you can choose are described in chapter 4. If you want to improve performance, you may want to reread that chapter and reassess the original choices you made.

You can adjust HD options by using the reorganization utilities. The procedure is:

1. Determine whether the change you are making will affect the code in any application programs. It should only do so if you are changing to a sequential randomizing module.
2. Unload your data base using the existing DBD and the appropriate unload utility.
3. Code a new DBD. Remember that if you changed your CI or block size, you need to allocate buffers for the new size. See the section in chapter 4 called "Buffer Pool Assignment" for a discussion of what to consider in choosing buffer number and size, and how they are specified.
4. If the change you are making affected the code in application programs, make any necessary changes to the PSBs for those application programs. Remember that if you have the DB/DC Data Dictionary, it can help you determine which application programs and PCBs may be affected by the DBD changes you've made.
5. Perform an ACBGEN.
6. Determine whether you need to recalculate data base space. See the section in chapter 4 called "Determining VSAM Space Requirements" for a description of how to calculate space.
7. Delete the space allocated for the old data base data set cluster(s) and define space for the new cluster(s).
8. Reload your data base using the new DBD and the appropriate reload utility. Remember to make an image copy of your data base as soon as it is reloaded.
9. If your data base uses logical relationships, you will have to run some of the reorganization utilities before and after reloading to resolve prefix information. You may also want to do this for secondary indexes, as it may improve performance. The flowchart in Figure 11-1 tells you which utilities and the order in which they must be run.

VSAM Buffers and Options

The size and number of buffers you can choose is described in chapter 4 in the section called "Buffer Pool Assignment." The performance implications of choosing a buffer size and number are discussed there. So, if you want to improve performance, you may want to reread that section and reassess the original choices you made.

Buffers

If you are using VSAM, you can monitor buffers using the run and buffer statistics as described in chapter 10. By analyzing those statistics, you can see whether you need to make changes. You can experiment by changing the number and/or size of buffers. After monitoring again for a while, you can tell whether or not an improvement has been made.

Another way to improve performance, this time for a specific application, is to reserve subpools for use by certain data sets. For example, if you have an index data set with a CI size of 512 bytes, you might want to reserve a subpool for it that contains 512-byte buffers. You can do this by *not* defining 512-byte CI sizes for

any other data sets in the data base. (Remember, CI sizes are specified *by data set* in the BLOCK= operand in the DATASET statement in the DBD.) If you then allocate enough 512-byte buffers to hold all of the CIs in your index, all CIs read into the buffer pool will remain in the buffer pool.

Options

The VSAM options you can choose are described in chapter 4 in the section called "Selecting VSAM Options." If you want to improve performance, you may want to reread that section and reassess the original choices you made.

Space Allocation

You would want to change the amount of space allocated for your data base in two situations. The first is when you are running out of primary space. You do not ever want to use your secondary space allocation since this can greatly decrease performance. You would also want to change the amount of space allocated for your data base when the number of I/O operations required to process a DL/I request is so many that performance is unacceptable. This can happen if data in the data base is spread across too much DASD space.

If you decide to change the amount of space allocated for your data base, the reorganization utilities must be run to put the data base in its new space. The procedure for doing this is:

1. Unload your data base using the existing DBD and the appropriate unload utility.
2. Recalculate data base space. See the section in chapter 4 called "Determining VSAM Space Requirements" for a description of how to calculate data base space.
3. For non-VSAM data sets, delete the old data base space and define new data base space. For VSAM data sets, delete the space allocated for the old cluster(s) and define space for the new cluster(s).
4. Remember that if you are changing the space in the root addressable area of an HD randomized or HDAM data base, you may have to adjust other HD parameters. In this case you will have to code a new DBD before reloading.
5. If you changed the DBD, perform an ACBGEN.
6. Reload your data base using either the existing DBD (if no changes were made to the DBD) or the new DBD. Use the appropriate reload utility.
7. If your data base uses logical relationships or secondary indexes, you will have to run some of the reorganization utilities before and after reloading to resolve prefix information. The flowchart in Figure 11-1 tells you which utilities and the order in which they must be run.

DASD Types

Several situations might prompt you to try to improve your data base performance by changing direct-access storage devices (DASD). One is when application requirements change, dictating that a faster or slower device be used. Another reason for change might be to take advantage of new devices offering better performance. Finally, you might want to change devices to get data base data sets on two different devices so contention for use of a device is minimized. For example, in an HD indexed or HIDAM data base, there is one data set containing data and another containing an index. When both are on the same device, extra time is required for seek operations as the arm moves between the index and data

base data sets. By putting one of the data sets on a different device, the amount of arm movement is decreased, thereby improving performance.

You can change your data base (or part of it) from one device to another by using the reorganization utilities. To change direct-access storage devices, you need to:

1. Unload your data base using the existing DBD and the appropriate unload utility.
2. Recalculate CI or block size to maximize use of track space on the new device. Information on calculating CI or block size is contained in the section called "Selecting CI and Block Sizes" in chapter 4.
3. Code a new DBD.
4. Perform an ACBGEN.
5. Delete the space allocated for the old data base data set cluster(s) and define space for the new cluster(s).
6. Reload your data base using the new DBD and the appropriate reload utility. Remember to make an image copy of your data base as soon as it is reloaded.
7. If your data base uses logical relationships, you will have to run some of the reorganization utilities before and after reloading to resolve prefix information. You may want to do this also for secondary indexes, since it may improve performance. The flowchart in Figure 11-1 tells you which utilities and the order in which they must be run.

DL/I Access Methods

When you originally chose a DL/I access method for your data base, you chose it based on such things as:

- The type of processing you needed to do (sequential, direct, or both)
- The volatility of your data.

If the characteristics of your applications have changed over a period of time, performance may be improved by changing to another DL/I access method. Chapter 4 describes which type of DL/I access method to choose, given your application's characteristics. This section assumes you've decided to change access methods and tells you:

- Given your existing DL/I access method, what things you may want or need to change to convert to a different DL/I access method
- How to do the conversion.

The reorganization utilities described earlier in the chapter can be used to change DL/I access methods among the HISAM, HD, HDAM, and HIDAM access methods. There is one exception to this: HD randomized or HDAM cannot be changed to HISAM, HD indexed, or HIDAM unless its data base physical records are in root key sequence. This is because HISAM, HD indexed, and HIDAM data bases must be loaded with data base records in root key sequence. When the HD unload utility unloads HD randomized or HDAM data base, it unloads it using GET NEXT requests. GET NEXT requests against an HD randomized or HDAM data base unload the data base records in the physical sequence in which they were stored by the randomizing module. This will not be root key sequence unless you used a sequential randomizing module (one that put the data base records into the data base in physical root key sequence).

From HISAM to HD Indexed

Here are the things you need to consider before changing your DL/I access method from HISAM to HD indexed or HIDAM:

- Determine whether you are going to set aside free space in the new data base.

  Unlike HISAM, in an HD indexed or HIDAM data base you can set aside periodic blocks or CIs of free space, or a percentage of free space in each block or CI. This free space can then be used for inserting data base records or segments into the data base after initial load. See the section in chapter 4 called “Specifying Distributed Free Space” for a description of free space and how it is specified.

- Determine what type of pointers you are going to use in the data base. Unlike HISAM, HD indexed and HIDAM use direct-address pointers to point from one segment in the data base to the next. See chapter 4 for a description of types of pointers and how to specify them.

- Reassess your choice of logical record length and block size and convert to CI size. See the section in chapter 4 called “Selecting CI and Block Sizes” for a discussion of what to consider in choosing a CI size and how it is specified.

- Reassess your choice of data base buffer sizes and the number of buffers you have allocated. Remember that if you have changed your CI or block size, you need to allocate buffers for the new size. See the section in chapter 4 called “Buffer Pool Assignment” for a discussion of what to consider in choosing buffer number and size and how they are specified.

- Recalculate data base space. You need to do this because the changes you are making will result in different space requirements. See the section in chapter 4 called “Determining VSAM Space Requirements” for a description of how to calculate data base size.

Once you have determined what changes you want or need to make, you're ready to change your DL/I access method. To do this:

1. Unload your data base using the existing DBD and the HD unload utility.
2. Code a new DBD that reflects the changes you want or need to make.
3. If there are changes you want or need to make that are not specified in the DBD (such as changing data base buffer sizes or the amount of space allocated for the data base), make those changes.
4. Perform an ACBGEN.
5. Delete the space allocated for the old VSAM data set cluster(s) and define space for the new cluster(s).
6. Reload the data base using the new DBD and the HD reload utility. Remember to make an image copy of your data base as soon as its reloaded.

From HISAM to HD Randomized

Here are the things you need to consider before changing your DL/I access method from HISAM to HD randomized or HDAM:

- Determine whether you are going to set aside free space in the new data base.

  Unlike HISAM, in an HD randomized or HDAM data base you can set aside periodic blocks or CIs of free space, or a percentage of free space in each block or CI. This free space can then be used for inserting data base records or segments into the data base after initial load. See the section in chapter 4 called “Specifying Distributed Free Space” for a description of free space and how it's specified.

- Determine what type of pointers you are going to use in the data base. Unlike HISAM, HD randomized and HDAM use direct-address pointers to point from one segment in the data base to the next. See chapter 4 for a description of types of pointers and how to specify them.
- Determine which randomizing module you are going to use. Unlike HISAM, HD randomized and HDAM use a randomizing module. The randomizing module generates information that determines where a data base record is to be stored. See the section in hdref refid=ranmod. called “Randomizing Modules for HD Randomized Data Bases” for information on choosing a randomizing module and how it is specified.
- Determine which HD options you are going to use. Unlike HISAM, an HD randomized or HDAM data base is divided into two parts within the same VSAM data set: a primary (root addressable) area and an overflow area. The root addressable area contains all root segments and is the primary storage area for dependent segments in a data base record. The overflow area is for storage of segments that don't fit in the primary area. The HD options we are talking about here are the ones that pertain to choices you make about the root addressable area. These are:
  - The maximum number of bytes of a data base record to be put in the primary area when segments in the data base record are inserted consecutively (without intervening processing operations)
  - The number of blocks or CIs in the primary area
  - The number of root anchor points (RAPs) in a block or CI in the primary area. (A RAP is a field that points to a root segment.)

  See chapter 4 for information on choosing the options and how they are specified.
- Reassess your choice of logical record length and block size and convert to CI size. See the section in chapter 4 called “Selecting CI and Block Sizes” for a discussion of what to consider in choosing a CI size and how it is specified.
- Reassess your choice of data base buffer sizes and the number of buffers you have allocated. Remember that if you have changed your CI or block size, you need to allocate buffers for the new size. See the section in chapter 4 called “Buffer Pool Assignment” for a discussion of what to consider in choosing buffer number and size and how they are specified.
- Recalculate data base space. You need to do this because the changes you are making will result in different space requirements. See the section in chapter 4 called “Determining VSAM Space Requirements” for a description of how to calculate data base size.

After you have determined what changes you want or need to make, you are ready to change your DL/I access method. To do this:

1. Unload your data base using the existing DBD and the HD unload utility.
2. Code a new DBD that reflects the changes you want or need to make.
3. If there are changes you want or need to make that are not specified in the DBD (such as changing data base buffer sizes or the amount of space allocated for the data base), make those changes. Remember that HD randomized and HDAM require only one data set, while HISAM requires two.
4. Perform an ACBGEN.
5. Delete the space allocated for the old VSAM data set cluster(s) and define space for the new cluster(s).

6. Reload the data base using the new DBD and the HD reload utility. Remember to make an image copy of your data base as soon as it's reloaded.

From HD Indexed to HISAM

Here are the things you need to consider before changing your DL/I access method from HD indexed or HIDAM to HISAM:

- Reassess your choice of CI size and convert to logical record length and block size. See the section in chapter 4 called “Selecting a Logical Record Length” for a discussion of what to consider in choosing a logical record length and how it is specified. See the section in chapter 4 called “Selecting CI and Block Sizes” for a discussion of what to consider in choosing a block size and how it is specified.
- Reassess your choice of data base buffer sizes and the number of buffers you have allocated. Remember that if you have changed your CI or block size, you need to allocate buffers for the new size. See the section in chapter 4 called “Buffer Pool Assignment” for a discussion of what to consider in choosing buffer number and size and how they are specified.
- Recalculate data base space. You need to do this because the changes you are making will result in different space requirements. See the section in chapter 4 called “Determining VSAM Space Requirements” for a description of how to calculate data base size.

After you have determined what changes you want or need to make, you are ready to change your DL/I access method. To do this:

1. Unload your data base using the existing DBD and the HD unload utility.
2. Code a new DBD that reflects the changes you want or need to make. Remember that you will not be specifying direct-address pointers or free space in the DBD, since HISAM, unlike HD indexed and HIDAM, does not allow use of these.
3. If there are changes you want or need to make that are not specified in the DBD (such as changing data base buffer sizes or the amount of space allocated for the data base), make those changes.
4. Perform an ACBGEN.
5. Delete the space allocated for the old VSAM data set cluster(s) and define space for the new cluster(s).
6. Reload the data base using the new DBD and the HD reload utility. Remember to make an image copy of your data base as soon as it is reloaded.

From HD Indexed to HD Randomized

Here are the things you need to consider before changing your DL/I access method from HD indexed or HIDAM to HD randomized or HDAM:

- Reassess your choice of direct-address pointers. Although all HD access methods use direct-address pointers, you may need to change the type of direct-address pointer used:
  - Because of the changing needs of your applications
  - Because pointers are partly chosen based on the type of data base you are using. For example, if you used physical twin backward pointers on root segments in your HD indexed or HIDAM data base to get fast sequential processing of roots, they will not have any use in the new data base. See chapter 4 for a description of types of pointers, their uses, and how to specify them.

- Determine which randomizing module you are going to use. Unlike HD indexed and HIDAM, HD randomized and HDAM use a randomizing module. The randomizing module generates information that determines where a data base record is to be stored. See the section in Appendix G, “Randomizing Modules and DL/I User Exit Routine Interfaces” on page G-1 called “Randomizing Modules for HD Randomized Data Bases” for information on choosing a randomizing module and how it is specified.
- Determine which HD options you're going to use. Unlike HD indexed and HIDAM, HD randomized and HDAM data bases don't have separate index data bases. Instead the data base is divided into two parts: a primary (root addressable) area and an overflow area. The root addressable area contains all root segments and is the primary storage area for dependent segments in a data base record. The overflow area is for storage of dependent segments that don't fit in the primary area. The HD options we are talking about here are the ones that pertain to choices you make about the root addressable area. These are:
  - The maximum number of bytes of a data base record to be put in the primary area when segments in the data base record are inserted consecutively (without intervening processing operations).
  - The number of blocks or CIs in the primary area.
  - The number of root anchor points (RAPs) in a block or CI in the primary area. (A RAP is a field that points to a root segment.) See chapter 4 for information on choosing the options and how they are specified.
- Reassess your choice of CI size. See the section in chapter 4 called “Selecting CI and Block Sizes” for a discussion of what to consider in choosing a CI size and how it is specified.
- Reassess your choice of data base buffer sizes and the number of buffers you have allocated. Remember that if you have changed your CI or block size, you need to allocate buffers for the new size. See the section in chapter 4 called “Buffer Pool Assignment” for a discussion of what to consider in choosing buffer number and size and how they are specified.
- Recalculate data base space. You need to do this because the changes you are making will result in different space requirements. See the section in chapter 4 called “Determining VSAM Space Requirements” for a description of how to calculate data base size.

Once you have determined what changes you want or need to make, you are ready to change your DL/I access method. To do this:

1. Unload your data base using the existing DBD and the HD unload utility.
2. Code a new DBD that reflects the changes you want or need to make.
3. If there are changes you want or need to make that are not specified in the DBD (such as changing data base buffer sizes or the amount of space allocated for the data base), make those changes. Remember that HD randomized and HDAM require only one data set, while HD indexed and HIDAM require two.
4. Perform an ACBGEN.
5. Delete the space allocated for the old VSAM data set cluster(s) and define space for the new cluster(s).
6. Reload the data base using the new DBD and the HD reload utility. Remember to make an image copy of your data base as soon as it is reloaded.

   If you are using logical relationships or secondary indexes, you will need to run additional utilities right before and after reloading your data base. The

flowchart in Figure 11-1 tells you which utilities and the order in which they must be run.

From HD Randomized to HISAM

Here are the things you need to consider before changing your DL/I access method from HD randomized or HDAM to HISAM:

- Reassess your choice of CI size and convert to logical record length and block size. See the section in chapter 4 called “Selecting a Logical Record Length” for a discussion of what to consider in choosing a logical record length and how it is specified. See the section in chapter 4 called “Selecting CI and Block Sizes” for a discussion of what to consider in choosing a block size and how it is specified.
- Reassess your choice of data base buffer sizes and the number of buffers you have allocated. Remember that if you have changed your CI or block size, you need to allocate buffers for the new size. See the section in chapter 4 called “Buffer Pool Assignment” for a discussion of what to consider in choosing buffer number and size and how they are specified.
- Recalculate data base space. You need to do this because the changes you are making will result in different space requirements. See the section in chapter 4 called “Determining VSAM Space Requirements” for a description of how to calculate data base size.

When you have determined what changes you want or need to make, you are ready to change your DL/I access method. Remember that you have to write your own unload and reload programs unless data base records in the HD randomized or HDAM data base are in physical root key sequence.

To change the access method:

1. Unload your data base using the existing DBD and:
   - Your unload program, or
   - The HD unload utility if data base records are in physical root key sequence.
2. Code a new DBD that reflects the changes you want or need to make. Remember that you will not be specifying direct-address pointers or HD options.
3. If there are changes you want or need to make that are not specified in the DBD (such as changing data base buffer sizes or the amount of space allocated for the data base), make those changes. Remember that HD randomized and HDAM require only one data set, while HISAM requires two.
4. Perform an ACBGEN.
5. Delete the space allocated for the old VSAM data set cluster(s) and define space for the new cluster(s).
6. Reload the data base using the new DBD and:
   - Your load program, or
   - The HD reload utility if data base records are in physical root key sequence.

From HD Randomized to HD Indexed

Here are the things you need to consider before changing your DL/I access method from HD randomized or HDAM to HD indexed or HIDAM:

- Determine whether you are going to set aside free space in the new data base. You can set aside periodic blocks or CIs of free space or a percentage of free space in each block or CI. This free space can then be used for inserting data base records or segments into the data base after initial load. See the section in chapter 4 called “Specifying Distributed Free Space” for a description of free space and how it is specified.
- Reassess your choice of direct-address pointers. Although all HD access methods use direct-address pointers, you may need to change the types of pointers used:
  - Because of the changing needs of your applications
  - Because pointers are partly chosen based on the type of data base you are using. For example, you may want to use physical twin forward *and* backward pointers on root segments in your new data base to get fast sequential processing of roots. See chapter 4 for a description of types of pointers, their uses, and how to specify them.
- Reassess your choice of CI size. See the section in chapter 4 called “Selecting CI and Block Sizes” for a discussion of what to consider in choosing a CI size and how it is specified.
- Reassess your choice of data base buffer sizes and the number of buffers you have allocated. Remember that if you have changed your CI or block size, you need to allocate buffers for the new size. See the section in chapter 4 called “Buffer Pool Assignment” for a discussion of what to consider in choosing buffer number and size and how they are specified.
- Recalculate data base space. You need to do this because the changes you are making will result in different space requirements. See the section in chapter 4 called “Determining VSAM Space Requirements” for a description of how to calculate data base size.

When you have determined what changes you want or need to make, you are ready to change your DL/I access method. Remember that you have to write your own unload and reload programs unless data base records in the HD randomized or HDAM data base are in physical root key sequence.

To change the access method:

1. Unload your data base using the existing DBD and:
   - Your unload program, or
   - The HD unload utility if data base records are in physical root key sequence.
2. Code a new DBD that reflects the changes you want or need to make.
3. If there are changes you want or need to make that are not specified in the DBD (such as changing data base buffer sizes or the amount of space allocated for the data base), make those changes.
4. Perform an ACBGEN.
5. Delete the space allocated for the old VSAM data set cluster(s) and define space for the new cluster(s).
6. Reload the data base using the new DBD and:
   - Your load program, or
   - The HD reload utility if data base records are in physical root key sequence
   - Remember to make an image copy of your data base as soon as it is reloaded.

If you are using logical relationships or secondary indexes, you will need to run additional utilities before reloading your data base. The flowchart in Figure 11-1 tells you which utilities and the order in which they must be run.

Hierarchical Structure Changes

There are several types of changes you might want to make that affect performance and involve changes to the structure of your data base record. They are discussed in the following sections. Note that changes involving adding and deleting segments in the hierarchy are covered in chapter 8.

Changing the Sequence of Segment Types

In HSAM and HISAM data bases, performance is best if frequently-used dependent segments are close to the root segment and infrequently-used dependent segments are toward the end of the data base record. This is true because dependent segments are located by searching sequentially through the data base record.

To change the placement of a segment type, you have to write a program to unload segments from the data base in the new hierarchical sequence. Then you need to load the segments into a new data base. Again, you have to write a program to reload.

To change the hierarchical structure, you have to:

1. Determine whether the change you are making will affect the code in any application programs. If so, make sure the code gets changed.
2. Unload your data base using your unload program and the existing DBD.
3. Code a new DBD.
4. If the change you are making affected the code in application programs, make any necessary changes to the PSBs for those application programs. Remember that if you have the DB/DC Data Dictionary, it can help you determine which application programs and PCBs may be affected by the DBD changes you have made.
5. Perform an ACBGEN.
6. For non-VSAM data sets, delete the old data base space and define new data base space. For VSAM data sets, delete the space allocated for the old cluster(s) and define space for the new cluster(s).
7. Reload your data base using your load program and the new DBD. Remember to make an image copy of your data base as soon as it is reloaded.
8. If your data base uses logical relationships or secondary indexes, you will have to run some of the reorganization utilities before and after reloading to resolve prefix information. The flowchart in Figure 11-1 tells you which utilities and the order in which they must be run.

Combining Segments

The second type of change you might want to make in the structure of your data base record is combining segment types to maximize use of space. For example, if you have a dependent segment for college classes and a segment dependent on it for instructors who teach the classes, it may be an inefficient use of space to have two segment types if, in most cases, only one or two instructors teach a class. Rather than having a separate instructor segment, you can combine the two segment types, thereby saving space.

Combining segments also requires that you write an unload and reload program. (The reorganization utilities cannot be used to make such a change.)

To change the hierarchical structure, you have to:

1. Determine whether the change you are making will affect the code in any application programs. If so, make sure the code gets changed.
2. Unload your data base using your unload program and the existing DBD.
3. Code a new DBD.
4. If the change you are making affected the code in application programs, make any necessary changes to the PSBs for those application programs. Remember that if you have the DB/DC Data Dictionary, it can help you determine which application programs and PCBs may be affected by the DBD changes you've made.
5. Perform an ACBGEN.
6. For non-VSAM data sets, delete the old data base space and define new data base space. For VSAM data sets, delete the space allocated for the old cluster(s) and define space for the new cluster(s).
7. Reload your data base using your load program and the new DBD. Remember to make an image copy of your data base as soon as it's reloaded.
8. If your data base uses logical relationships or secondary indexes, you will have to run some of the reorganization utilities before and after reloading to resolve prefix information. The flowchart in Figure 11-1 tells you which utilities and the order in which they must be run.

Chapter 12. Data Base Recovery

This chapter tells you how to plan for and accomplish recovery and restart of a damaged data base. There are three sections:

1. Recovery Plan
2. Recovery Tools and Their Use
3. Restart

The first section describes the need for a recovery plan, and guides you in the generation of a plan. The second section tells about the tools available for use in data base recovery, and describes their use. The last section discusses the considerations in planning for restarting an application after recovery, and how to accomplish the restart. See the *DL/I DOS/VS Recovery Restart Guide* for a more detailed discussion of recovery concepts and procedures.

Recovery Plan

In any data processing environment, it is inevitable that some failures or errors will occur that damage the data being maintained. Successful recovery and restart from these failures and errors can be characterized by a five step process:

- Detection of the error or failure
- Determination of the cause
- Recovery of the data
- Correction of the cause
- Restart of processing

In traditional batch file processing, normally not much thought or effort is spent in developing and implementing recovery and restart procedures. The usual recovery-restart procedure in this environment is to:

- Periodically dump the files
- Save all input since the last file dump
- When an error is detected or a failure occurs, restore the files and rerun all the jobs from the time of the file dump.

This traditional recovery approach is illustrated in Figure 12-1.

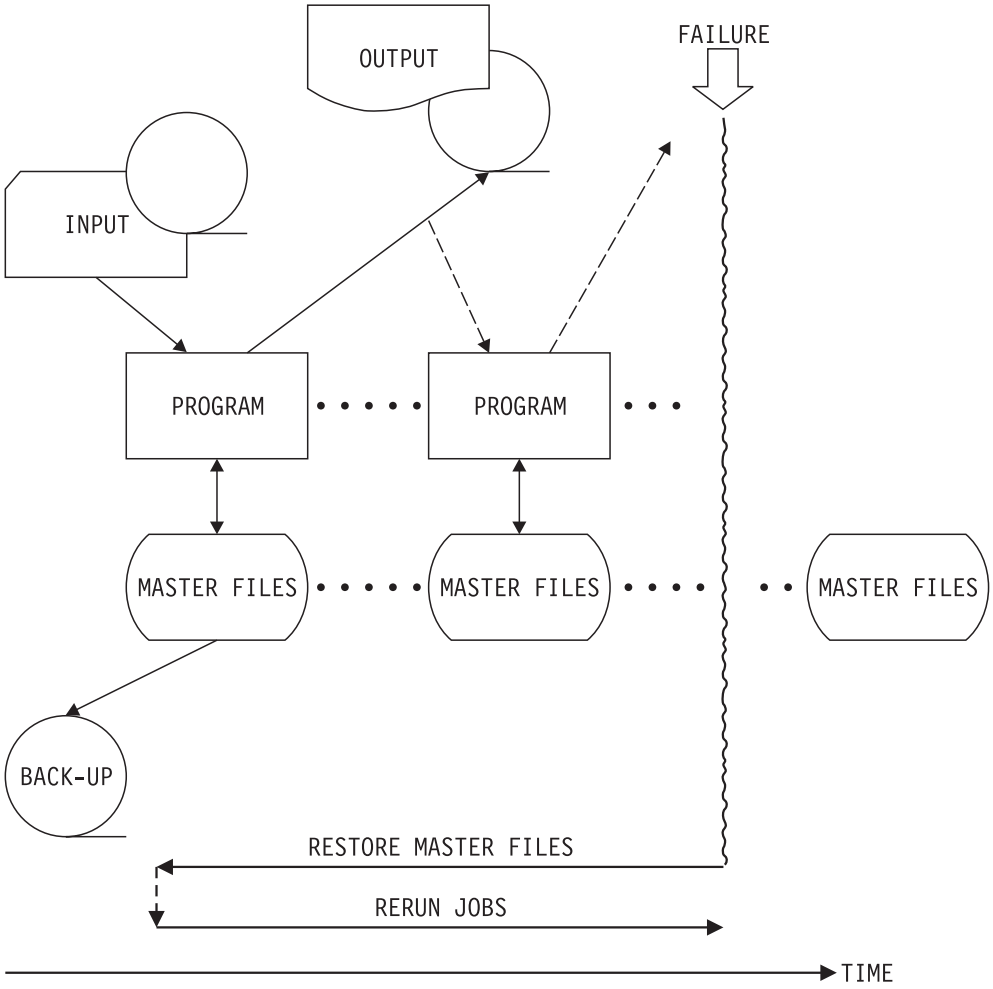


Figure 12-1. Traditional Recovery Approach

In a data base environment where timeliness of information is important, this simple procedure is often not sufficient because of the time required to dump and restore large data bases and rerun all the jobs since the last data base dump.

Error detection and correction in a data base environment is complicated by the various interrelationships within the data base and between data bases. For example, if a data base participating in a logical relationship is damaged or destroyed, simply restoring a copy of the data base from the last dump does not restore the relationships between the data bases. To recover in the traditional manner, all related data bases would have to be restored. Unless dumps of all the related data bases are taken at the same time, it is impossible to recover from the failure in the traditional manner.

In an online environment, the time to recover and restart after a failure or error becomes critical. Often, while the damage is being repaired, key applications may not be able to run, affecting many user departments. Recovery in an online environment is further complicated because input to the online programs may not be available for reprocessing. For example, in a customer service environment where the input may come from direct interaction with the customer over a telephone, the input data is often unreproducible.

DL/I provides several recovery facilities to overcome these problems in an online data base environment. Of course, they can also be used in a batch-only DL/I environment. These facilities include:

- A logging facility to record all changes made to the data bases
- An abnormal termination routine to help determine the cause of a program check or other condition that prevents a normal end of job
- A set of utility programs to correct or reconstruct data bases after damage
- A checkpoint facility to minimize reprocessing time when restarting the job.

In addition to the facilities provided by DL/I, you will have to establish standards and procedures for detecting errors and failures, and for recovery and restart. For a complete recovery and restart system, the DL/I facilities must be combined with facilities provided by VSAM and VSE, and with your own user written programs.

You should make recovery and restart planning part of the design of your data base applications. As you define and develop an application, also define and develop recovery procedures for all possible failure conditions that could affect it. It is important that recovery procedures be planned as part of the application design because they can influence design choices.

One way to start is by making a list of different types of failures:

- Power failure, “hard wait” in the operating system, and other failures where the DL/I abnormal termination routine is not executed
- Physical damage to the data base that renders the data unreadable, as in the case of disk head crash or a dropped disk pack
- Application program errors that cause the program to abnormally terminate
- Application program logic errors where the program terminates normally but updates the data base incorrectly
- VSAM catalog damage rendering the VSAM data sets defined in the catalog inaccessible
- DL/I abend conditions where the DL/I abnormal termination routine is executed. For example, “bad pointer” or no more storage space.

By reviewing the various DL/I messages found in *DL/I DOS/VS Messages and Codes*, you can identify other failure conditions and their symptoms.

When planning the recovery procedures for each of the failure conditions you define, consider the application needs and environment. For example, is the application online, batch, or MPS batch? How frequently are the data bases updated? How large are the data bases? Are logical relationships or secondary indexes used? What are the consequences of the data bases being unavailable for 10 minutes, 10 hours, 10 days?

With this type of information, you can better plan such things as:

- Frequency of data base back-up or image copy
- Log record volumes, if logging is used, and the log space requirements and change accumulation frequency
- Alternate processing plans should a lengthy recovery be required for particular types of failures
- Procedures to determine if the recovered data is correct.

When testing the application programs, you should also be testing your recovery procedures. Don't wait until the application is in production to find out if your

recovery procedures work. During the testing of your recovery procedures you can also determine recovery timings. For example, if a DL/I image copy of your test data base takes five minutes, and the planned data base will be ten times larger, then the approximate DL/I image copy time for the full data base is 50 minutes. This helps you to decide whether or not your planned frequency of image copy is reasonable. A 50 minute image copy once a day may be reasonable where a five hour image copy once a day would not. Similar times should also be developed for such operational steps as operating system IPL, CICS/VS restart, etc.

Finally, just as you must document the operation of your application programs, you must document the various recovery steps in your procedures. One way to document recovery procedures is to use the flow-chart approach. A recovery flow-chart for a batch program abend might look something like Figure 12-2.

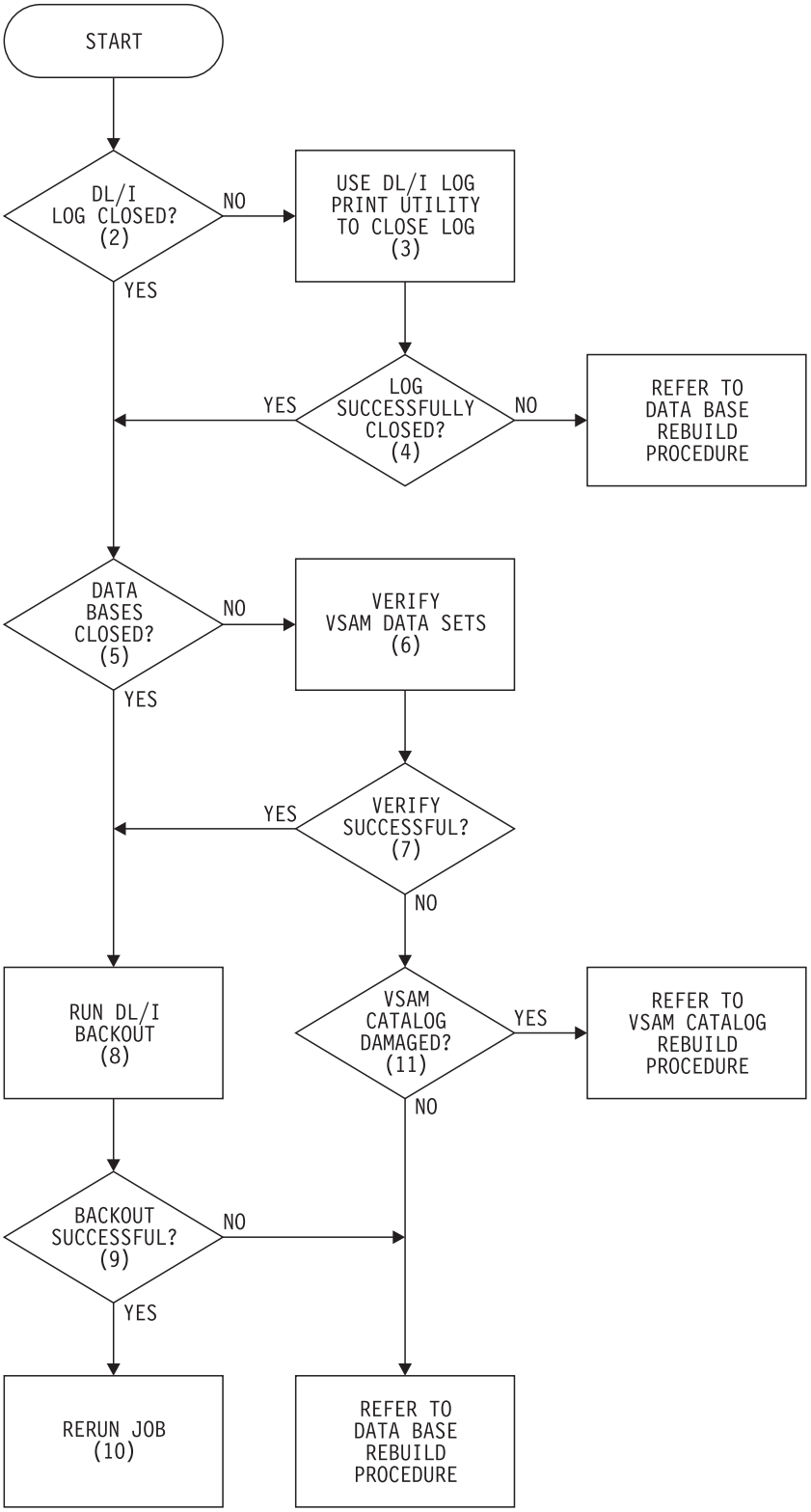


Figure 12-2. Sample Recovery Procedure Flowchart

Notice that this recovery procedure includes the use of facilities in addition to those provided by DL/I, such as VSAM VERIFY. Often, user written programs are required as part of a recovery procedure. If the abending batch program also updated non-DL/I files (VSAM, ISAM, etc.) with related information, then user

programs might be required to recover the information in them after a failure. These steps should be included in the recovery procedures.

The numbers in parentheses in the recovery flowchart refer to additional narrative documentation that should include information on such items as:

- Messages that indicate successful or unsuccessful execution of a recovery step
- Disposition of output produced by a step: whether it should be saved for later analysis or destroyed. This is especially important if sensitive or confidential data is involved.
- Instructions on how to execute the particular recovery step; including JCS, operator responses to messages, etc.

The following sections of this chapter are intended to provide an understanding of the various DL/I recovery facilities so you can construct recovery procedures suitable to your DL/I applications and environment.

Recovery Tools and Their Use

DL/I Logging Facility

DL/I provides a data base change logging mechanism that records every change made to a DL/I data base. DL/I records both “before” and “after” images of all segments changed by application programs. Application program requests may cause segment changes that are “invisible” to the program itself. For example, if an application program deletes a segment, all of the segment's children are also deleted, even if the application program is not sensitive to them. This same delete request can also cause pointers in related segments to be modified. All these changes to the data base are recorded on the log, even though they may be invisible to the application program. The use of these log records is explained in the following sections on the various DL/I recovery utilities. Note that a DL/I log is not created when a data base is initially loaded.

Before any physical change is made to the data base on disk, DL/I ensures that the log records reflecting the changes are physically recorded on the log. This is called “write ahead logging.” This write ahead logging technique is used in both batch and online environments. Thus, if a failure occurs, there can be only one of three possible relationships between the log and the data base:

1. Neither the log nor the data base reflects the latest change request made by the application program. This is the case when a failure occurs after the application program makes its change request (insert, delete, or replace) to DL/I, but before DL/I is able to write to physical storage.
2. The log reflects the latest data base change request made by the application program but the data base does not reflect the change. This is the case when a failure occurs between the physical write of the log record and the physical write of the data base record.
3. Both the log and the data base reflect the latest change request made by the application program. This is the case when a failure occurs after the physical write of the data base record.

The various DL/I recovery utilities include logic to correct the data base for any of the above three conditions.

Asynchronous Logging Option

Asynchronous logging is provided as a performance option by DL/I. Its use is restricted to batch jobs only. It can not be used in a CICS/VS or MPS environment. The potential disadvantage of its use is that, after some types of failures, the DL/I backout utility cannot be used to correct the data base damage. These conditions are explored in more detail in the section on the DL/I backout utility later in this chapter.

Generally, the performance improvements to be gained by using this option are small and you should carefully evaluate whether or not they are worth the risk of not being able to use the DL/I backout utility in the event of a failure.

There is one case where the data base can be physically updated before the corresponding log record is physically written. This can occur if you use the asynchronous logging option and a failure occurs that prevents DL/I from executing its abnormal termination routine successfully. A power failure, for example.

Logging and Performance

The performance of your application programs can be very dependent on the efficiency of the DL/I logging mechanism. You can affect the efficiency of the DL/I logging mechanism in the design of your data base application. Minimizing the number of physical writes to the log improves that efficiency. Physical writes to the log are required whenever:

1. The log buffer fills up
2. A modified data base record is about to be written back to disk and the log records corresponding to the change have not yet been written.

The second condition is known as a “force write” of the log. Physical writes to the log because of the first condition are performed asynchronously by DL/I. Force writes are, by their very nature, synchronous. Thus, the execution of your application program will be delayed until the log record is written. If your program causes many force writes to the log, you will find that the performance of your program is dependent on the speed of the log device.

There are several things you can do in designing your data base applications to minimize writing to the log:

- Avoid the use of HISAM. The HISAM access method uses VSAM logical record processing. As a consequence, DL/I has no control over when a physical write to the data base will occur. DL/I must assume that every PUT request to VSAM will result in a physical write to the data base. This causes DL/I to force write the log more often than would normally occur if an HD access method were chosen. With the HD access methods, DL/I controls when physical writes to the data base actually occur. Physical writes to the data base in the HD access methods tend to be deferred longer, thus allowing a greater opportunity for an asynchronous log write.
- Index data bases; whether HD indexed or HIDAM primary index data bases, or secondary index data bases; are also processed using VSAM logical record processing. Therefore, the same conditions exist when DL/I updates those data bases as exist for HISAM data bases. Be particularly careful with secondary indexes. Every time the source field value changes for a secondary index, DL/I writes at least three log records. If the source field is very volatile, considerable logging activity occurs.

- Consider the relative data base activity when assigning several HD data bases to the same HD buffer subpool. If two highly active data bases are assigned to the same subpool, activity in one data base can cause DL/I to write back updates to the other data base sooner than would otherwise be necessary. DL/I does this to free up buffer space in the subpool. This, in turn, could cause force writes to the log. Balancing the activity across subpools reduces the chances that this premature writing will occur.

DL/I Abnormal Termination Routines

DL/I provides abnormal termination routines to assist in an orderly shut-down of its data bases in the event of a program error. The abnormal termination routines apply to all environments: batch, MPS batch, and CICS/VS. However, successful completion of a DL/I abnormal termination routine does not imply that the data bases are intact. Some recovery processing is almost always required on any abnormal termination.

Abnormal Termination in Batch

In a batch environment the DL/I abnormal termination routine performs the following functions:

- Closes the DL/I log. The contents of the log data buffer are written onto the log and the log is closed. If the log is assigned to tape, the tape is rewound and unloaded. Successful execution of this phase of processing is indicated by message “DLZ001I” on the system console. This message is normally preceded by other DL/I messages describing why the job is being abended. If no DL/I message precedes DLZ001I, the abnormal termination routine was entered because of a program check or other condition that causes the operating system to invoke a STXIT exit (AB or PC). Possible causes of a STXIT exit be entered can be found in *VSE/Advanced Functions Application Programming: Reference*, in the section that discusses the STXIT macro.
- Writes any altered data base records still in main storage back to the data bases and closes the data bases. Closing the data bases causes the VSAM catalog entries for the DL/I data base data sets to be updated. Successful execution of this phase of processing is indicated by the “DLZ002I” message on the system console.
- Depending on the setting of the UPSI byte, optionally produces a storage dump.
- Cancels the job. Any remaining job steps in the job are not executed.

DL/I uses the operating system STXIT AB and STXIT PC macros in a batch environment to establish its abnormal termination routine. Consequently, your batch application programs should not also use the STXIT AB and STXIT PC functions. To do so would disrupt the operation of DL/I's abnormal termination routine. While the STXIT AB and STXIT PC facilities are not directly usable in COBOL programs, the application program debugging facilities provided by the COBOL compiler invokes the STXIT AB function. Therefore, your production application programs should not use these debugging facilities.

It is possible, in the batch environment, via the UPSI byte settings, to bypass the use of DL/I's abnormal termination routine and DL/I's use of STXIT AB and STXIT PC. This can be useful in testing, where the state of the data base after a program error is unimportant. If DL/I's abnormal termination routine is bypassed, the COBOL debugging aids can be used. The PL/I debugging aids cannot be used, since DL/I always resets PL/I's STXIT PCs, even if the DL/I abnormal termination

routine is bypassed, except that PL/I can get control for a formatted dump, even though PL/I uses STXIT PC. If you bypass the DL/I abnormal termination routine in a testing environment and a program error occurs, you should reload the data bases before using them further. The condition of the data bases after an error, where the DL/I abnormal termination routine was not executed, is unpredictable. Attempting to use the data bases for additional program testing without recreating them can cause additional program errors due to the incorrect status of the data bases.

Abnormal Termination in MPS Partition

The DL/I abnormal termination routine in a DL/I MPS batch program cannot be bypassed. In an MPS batch environment, the DL/I abnormal termination routine performs the following functions:

- Writes message “DLZ096I” on the system console if the abnormal termination was caused by a program check or other error that causes the operating system to invoke a STXIT exit (AB or PC). Possible causes of a STXIT exit being entered can be found in *VSE/Advanced Functions Application Programming: Reference*, in the section that discusses the STXIT macro. If DL/I's abnormal termination routine was entered directly from DL/I, rather than through a STXIT, then the “DLZ096I” message is not produced. Instead, DL/I produces messages indicating the explicit reason for the abend.
- Notifies DL/I in the CICS/VS partition that the MPS batch job is abending. If the abend condition originated in the MPS batch partition, the batch partition controller task supporting this batch partition issues a CICS/VS abend request.
- Deletes the XECB defined by this partition.
- Depending on the setting of the UPSI byte, it optionally produces a storage dump if the abnormal termination routine was entered via a STXIT AB or STXIT PC. If DL/I's abnormal termination routine was entered directly from DL/I rather than through a STXIT, a dump is always produced, regardless of the UPSI setting.
- Cancels the job. Any remaining job steps are not executed.

Because DL/I always uses STXIT AB and STXIT PC in an MPS batch environment, your application programs should not use any facilities that require STXIT AB or STXIT PC (except that a formatted dump is possible in PL/I even though STXIT PC is used). To do so would disrupt DL/I's abnormal termination routine and would cause unpredictable errors in the CICS/VS partition, possibly causing it to terminate abnormally as well.

Abnormal Termination in CICS/VS

DL/I Abnormal Task Termination: For a DL/I application task abend in CICS/VS, including the abend of an MPS batch partition controller task, DL/I's abnormal task termination routine performs the following functions:

- Cancels outstanding I/O operations for the terminating task and returns to CICS/VS so the user abnormal exit routine (if any) can be given control.
- Writes a dump of the DL/I control blocks on the CICS/VS dump data internal set. Control is then returned to CICS/VS so dynamic transaction backout can be performed.
- Issues a TERMINATE request on behalf of the abending task. This causes any data base records altered by that task, which are still in main storage and have not been backed out by dynamic transaction backout, to be written back to the data bases. The action also causes any log records reflecting these changes

to be written to the log. If PI (program isolation) was used for this task, any DL/I records or segments enqueued for the task are released.
- Writes a DL/I termination record on the log and a CICS/VS sync point record on the CICS/VS system journal.
- Releases any DL/I resources that belong to this task, such as, PST, PPST, PSB, etc.
- If the abending task is an MPS batch partition controller task, the XECBs defined by that task are deleted.
- Returns control to CICS/VS.

DL/I and CICS/VS do not terminate in the event of a DL/I application task abend. DL/I data bases are not closed on a task abend condition.

DL/I Internal Error Termination: For the condition where DL/I has detected an internal error that does not permit it to continue processing, the DL/I abnormal system termination routine performs the following functions:

- Abends all DL/I tasks which are currently waiting during the processing of a DL/I request with CICS/VS abend code of 'D062'.
- Returns an error code in the TCA or UIB to any non-HLPI DL/I tasks attempting to issue a request after this condition occurs.
- Abends any HLPI DL/I tasks which attempt to issue a request after this condition occurs with a CICS/VS abend code of 'DHTJ'.
- Writes a message to the system console and CICS master terminal
- Returns control to CICS/VS.

If dynamic transaction backout has been specified for the DL/I tasks which are abended, any changes to data bases made by these tasks will be backed out. CICS/VS remains operational so that non-DL/I tasks can continue to be processed. When a CICS/VS shut-down is later attempted, DL/I issues message “DLZ068I,” and does not attempt to close its data bases at that time. However, the DL/I log, if not assigned to the CICS/VS system journal, is closed at CICS/VS shut-down.

CICS/VS Internal Error Termination: If CICS/VS is terminating abnormally, the DL/I abnormal termination routine performs the following functions:

- If the DL/I log is not assigned to the CICS/VS system journal, the log buffer is written to permanent storage, the log is closed, and the operating system subtask for the DL/I log is detached.
- If MPS was active, it deletes all XECBs that had been defined
- Writes message “DLZ070I” to the system console
- Attempts to load and execute the DL/I online formatted dump program “DLZFSDP0”.

  Note: This program must be specified in the CICS/VS PPT for this step to execute.

- Returns control to CICS/VS.

Note that no attempt is made to close the DL/I data bases on this error condition. However, the VSAM automatic close facility will be invoked by operating system end-of-job processing to close the VSAM data sets used by DL/I. Refer to the section on VSAM considerations in this chapter for more discussion on the VSAM automatic close facility. DL/I task termination records are not written to the DL/I log on this error condition. CICS/VS does not write its task termination records on the CICS/VS system journal on this abend condition. Thus, either CICS/VS emergency

restart or the DL/I backout utility executed in batch can back out the effects of the “in-flight” DL/I tasks.

DL/I Recovery Utilities

The data base recovery system is supported by the following utility programs:

- Data base backout
- Data base data set image copy
- Data base change accumulation
- Data base data set recovery
- Log print utility
- Trace print utility.

These utility programs support the basic functions of the recovery system, which are:

- Removal of changes made to data bases by selected application programs
- Creation of dump images of data base data sets
- Accumulation of data base changes since the last complete image dump
- Restoration of a data base using a prior image copy and the accumulated changes
- Printing of the contents of DL/I log files.

Because log records are not created when initially loading a data base, and HSAM does not support inserts, deletes, or replaces, the recovery utilities do not support simple HSAM or HSAM organizations.

You can code the DL/I recovery utility job streams or, if you have a 3270-type terminal available, you can use the Interactive Utility Generation (IUG) facility to generate job streams for the utilities.

For complete information on IUG and how to use it, see *DL/I DOS/VS Interactive Resource Definition and Utilities*.

If you choose not to use IUG, *DL/I DOS/VS Resource Definition and Utilities* gives you complete information on how to code and execute the utilities necessary to perform the various recovery functions. See the *DL/I DOS/VS Recovery Restart Guide* for additional information on recovery procedures.

Data Base Backout

When the status of a data base is uncertain because a batch program that was updating it terminated abnormally, the data base backout utility can be used to remove (back out) the effects of the program. This utility reads backwards the log created during the processing of the program with the error. Using the data base log records read this way, it restores the data base to its status at the time the abnormally terminated program began processing. It also creates a log that must be used as input to any future recovery operation unless the data base data set image copy utility or reorganization unload/reload utilities are executed immediately after a successful data base backout utility execution.

All data base changes made by the program from the time it began processing until it terminated abnormally are backed out. If the program issued checkpoint requests, only the changes made since the last checkpoint are backed out. See “DL/I Checkpoint Facility” later in this chapter.

If the data bases were not closed by DL/I during abnormal termination, you must execute the VSAM access method services command, VERIFY, to update the VSAM master catalog for each file. If this is not done, an error occurs when the DL/I system attempts to open the data bases. Refer to the section "VSAM Considerations in DL/I Recovery/Restart," later in this chapter.

A log is created during backout. This log must be included in any subsequent data base recovery attempted for these data bases.

Input to the data base backout utility as illustrated in Figure 12-3 consists of:

1. One PSB and one or more DMBs loaded from a core image library during DL/I initialization
2. The input data base(s) the data base backout utility is to operate with
3. The DL/I log for the DL/I job execution that terminated abnormally.

Output from the data base backout utility as illustrated in Figure 12-3 consists of:

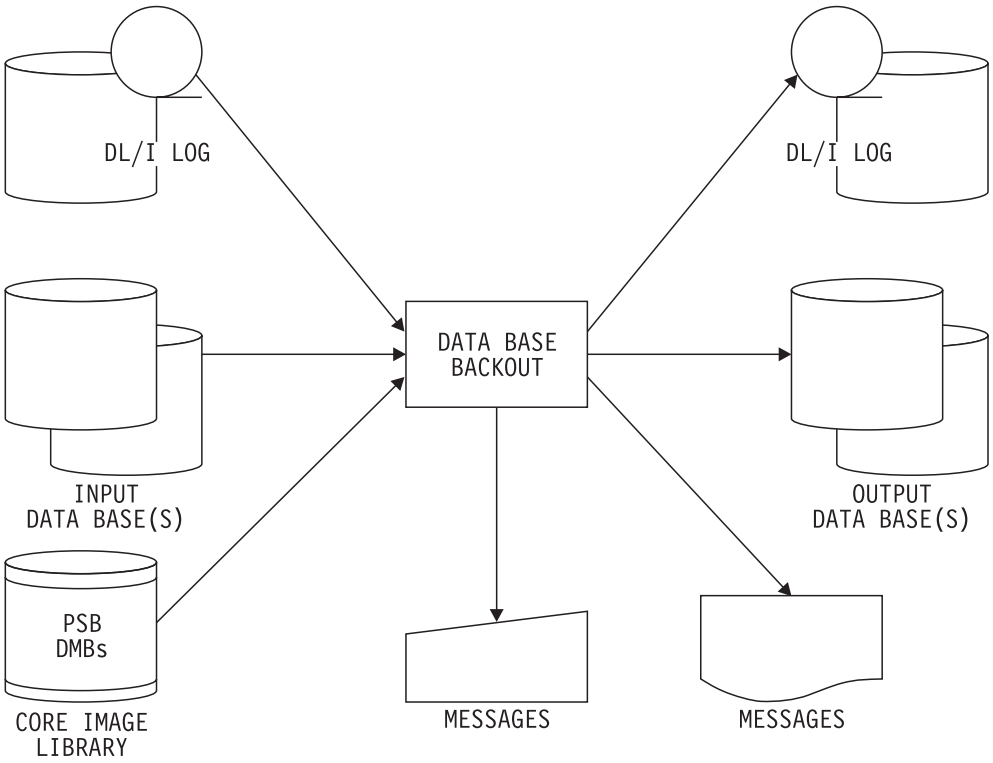


Figure 12-3. Data Base Backout Utility

1. The data bases reflecting the status prior to the DL/I job execution that terminated abnormally
2. The output DL/I log reflecting the changes made to the data base during the data base backout utility execution. This log, as well as the input log, must be used as input for any future recovery execution against the above data bases, unless the image dump utility or HISAM reorganization unload/reload utilities (HISAM or simple HISAM) are executed after the data base backout utility.
3. Messages on the SYSLST and SYSLOG devices.

For online operation, DL/I backout can be invoked automatically by CICS/VS emergency restart, or dynamic transaction backout. See *CICS/VS Recovery and Restart Guide* for more information.

Data Base Recovery

Data base recovery is accomplished using the information on the DL/I system log and periodically created copies of the data base. The DL/I system log contains records that describe the modifications made to the data base. The number of log files necessary for data base recovery depends on the frequency of data base copy execution, the volume of data base modifications, and the amount of usage of DL/I. Because information from a considerable number of log files may be necessary for data base recovery (all logs created since the last data base copy), a technique for accumulating all the latest changes to each specific file in a data base is provided. This accumulation of the latest data base modification is performed by the data base change accumulation utility. After using this utility the information necessary for data base recovery is contained in the following sources:

- Data base copy created when the data base was last dumped, through the data base data set image copy utility
- Data base change accumulation created from logs available since the last data base copy creation (not supported for simple HISAM)
- Logs employed since the last data base copy creation and not incorporated into the accumulation tape. This includes at least the log in use at the time problems were encountered with the data bases.

The data base data set recovery utility, which is the final stage of data base recovery, operates as an application program under control of the DL/I system. Recovery is done individually for each file. In most cases, a file is synonymous with a data base. This is true with HD, HDAM, HIDAM, INDEX, and simple HISAM data bases. HISAM data bases consist of two files, a KSDS and an ESDS. Thus, for HISAM, if the contents of one file are destroyed, it is not necessary to recover the complete data base. Recovery is by direct physical replacement of data within a file rather than by logical reprocessing of transactions.

The following functions, as illustrated in Figure 12-4, are required to accomplish data base recovery:

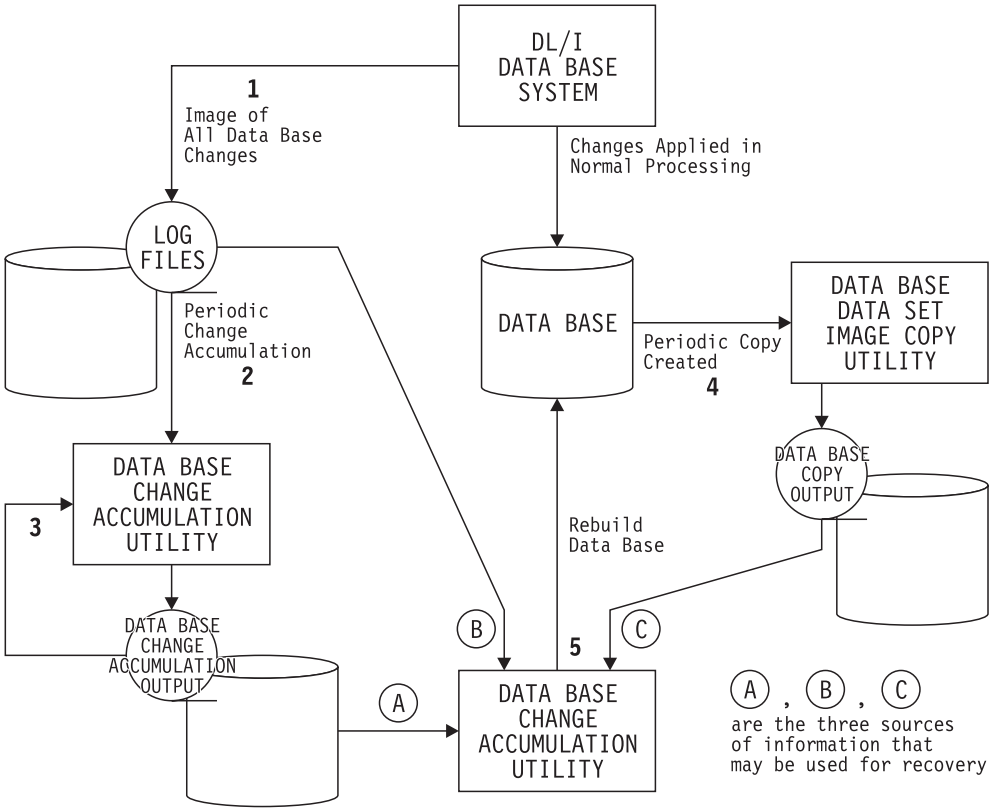


Figure 12-4. Data Base Recovery

1. Log the change data for a segment replace, insert, or delete, including the identification of the updated segment. This function is performed for all data bases.
2. Select the changed data base log records from the log file(s) and sort them in order by data base and file (not supported for simple HISAM). If the file is VSAM key sequenced (KSDS), the sort is ordered by VSAM key. If the file is VSAM entry sequenced (ESDS), the sort is by ESDS relative byte address (RBA). Selecting and sorting are performed as part of the change accumulation file creation by the data base change accumulation utility.
3. Merge the sorted, selected, changed records with the prior cumulative changes, keeping only the most recent data. Merging is performed as part of the change accumulation file creation by the data base change accumulation utility.
4. Dump the data set occasionally to provide a backup copy using the data base data set image copy utility.
5. When recovery is necessary, read the prior copy of the file to be restored and merge the cumulative changes, thereby reloading a partially restored file. Then read logs not included in the most recent cumulative changes and update the file to the point at which the error was detected. These functions are performed using the data base data set recovery utility.

Note: Items 2 and 3 are optional and are not supported for simple HISAM. All updates to the data base at recovery time can be applied from the logs rather than from the sorted cumulative changes and the logs. Occasional data base dumps reduce recovery time by reducing the number of records on the sorted change accumulation file.

Recovery of DL/I-IMS Compatible Data Bases

It is possible for you to generate a DL/I data base that is compatible with the IMS/VS data base management system. Typically, you would do this during migration from a VSE system to an OS system. During the transition period, the data base could be used on either system.

Use of a compatible data base creates a new recovery situation. The change accumulation utility produces a data set that is compatible between DL/I and IMS, but the log and image copy data sets are not compatible. Because of this, you must handle compatible data base recovery with a special procedure.

- Make an image copy of the data base under each data base manager before any updating occurs on either system
- Before switching the data base to the other system, always run the change accumulation utility if any updates were made. This summarizes the log files and ensures an up-to-date change accumulation file before the switch is made.
- If a failure occurs and recovery is necessary; use that system's image copy, the last change accumulation summary made on that system, and the log file or files made on that system since the switch was made.

This procedure does not work for simple HISAM data bases because the DL/I change accumulation utility does not support simple HISAM. In this case, a data base that is alternately accessed for update by DL/I and IMS must be recovered by some other technique. One method is to perform a dump and restore plus the application of the log file or files. Another possibility is to convert the SHISAM data base to another DL/I access method with the DL/I utilities.

DL/I Checkpoint Facility

DL/I provides a checkpoint facility to reduce the time required to back out data base changes after a failure.

Do not confuse this facility with the VSE CHKPT macro or program checkpointing facilities provided by COBOL and PL/I. The use of the DL/I checkpoint facility does not require the use of VSE checkpoints unless the MPS Restart Facility is used.

CHECKPOINT Request

The CHECKPOINT request causes a checkpoint record to be written on the DL/I log as an aid in restart processing. For batch, the data base buffers are written to secondary storage and a checkpoint log record, with a user-supplied unique checkpoint identification, is written to the DL/I log.

For MPS and online tasks running with CICS/VS journaling active, a CHECKPOINT request is, in effect, a CICS/VS sync point call with the exception that the task's PSB scheduling status is not changed. Therefore, if the task has a scheduled PSB in effect at the time the CHECKPOINT is issued, a PSB scheduling request is not required after the CHECKPOINT request. In fact, a scheduling request issued under these circumstances causes a scheduling error.

You should issue periodic CHECKPOINT requests in long-running MPS and online tasks running with program isolation to keep control block size to a minimum.

You should also not use DL/I logging for MPS and online tasks. With DL/I logging active, a CHECKPOINT request causes data base buffers to be written to secondary storage and a checkpoint log record to be written to the DL/I log, as in

the batch environment. However, these functions are not usable for performing backout because batch backout cannot be used in an online environment. Backout for an MPS or online DL/I task can only be performed using CICS/VS dynamic transaction backout, which requires that CICS/VS journaling be active. See the *DL/I DOS/VS Application Programming: High Level Programming Interface* or *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces* for additional details about the CHECKPOINT request.

DL/I Checkpoint in Batch Programs

The DL/I backout utility normally backs out all changes made to the data bases by the program in execution at the time of failure, back to the start of the program's execution. Start of program execution is indicated on the log by a scheduling record. If a long-running batch program fails towards the end of its run, a considerable number of data base changes have to be backed out, a lengthy process. By issuing the DL/I checkpoint request, “CHECKPOINT”, at intervals during your program's execution, you can shorten the time required for this recovery step. When a batch program issues a checkpoint request, DL/I performs the following functions:

1. Writes any altered data base records currently in main storage back to the data bases. This action will also force write the log records for these changed records if necessary. At this point, any position in the data base is lost.
2. Writes a checkpoint record on the log
3. Writes message “DLZ105I” to the system console indicating that a DL/I checkpoint request has been successfully executed
4. Returns control to the application program.

Thus after a checkpoint request, all data base changes are reflected on the data bases. However, the data bases are not closed on a checkpoint request. Note that after a checkpoint request your program must reestablish position in the data bases before continuing.

If your program were to fail immediately after the checkpoint request, the data base pointers would be good and the information intact. Consequently, when the DL/I backout utility encounters a checkpoint record on the log while backing out changes, it stops operation as it knows any changes recorded on the log prior to the checkpoint record are correctly written on the data bases. This then reduces the volume of data base records that must be backed out and reduces the time required to back out changes after a failure. The effect of a checkpoint request then, is to “commit” the data base changes made by your application program to the data bases even if a subsequent failure and backout occur. Figure 12-5 illustrates the relationship between the checkpoint records and the DL/I backout utility.

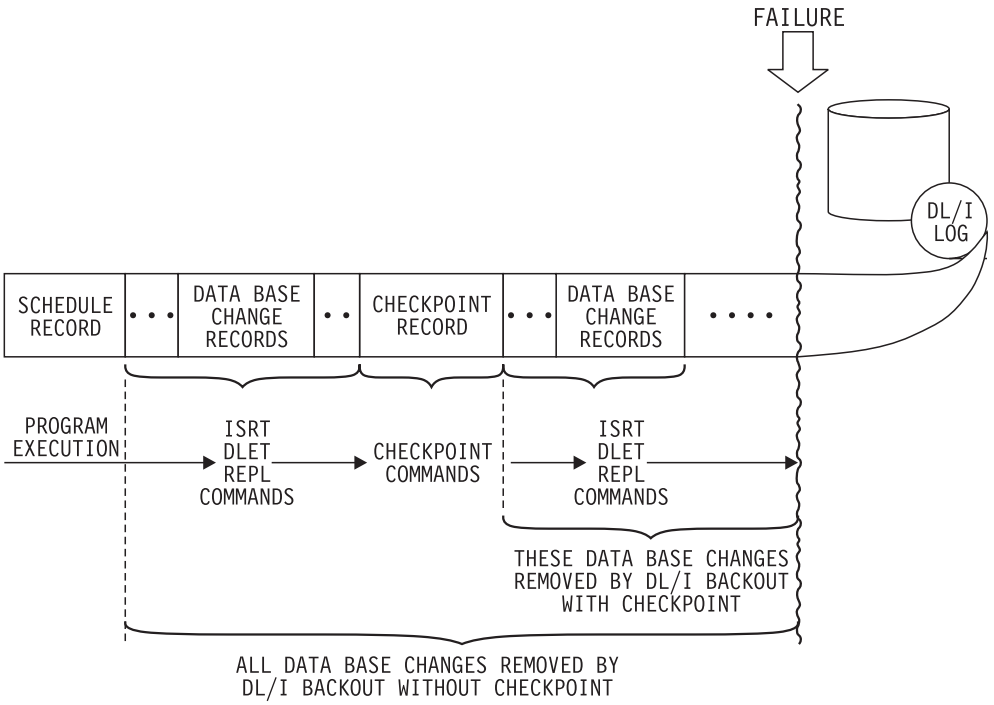


Figure 12-5. Checkpoint Records and DL/I Backout

The DL/I forward recovery utility ignores any checkpoint records on the log. Checkpoint records are not carried onto the change accumulation file created by the DL/I change accumulation utility. Checkpoint records are used only by the DL/I backout utility.

The data base changes made by your program prior to the checkpoint request are still present on the data base after your program fails and the DL/I backout utility is run. Therefore, you cannot simply rerun your program again from the beginning with all of the original input. You must have logic in your program to allow it to start at a point where the environment corresponds to that present when the last checkpoint request was issued. This may require logic in your program to reposition other files, restore internal counters and total fields, etc. To assist in this restart, each checkpoint message written on the system console contains an eight-byte identification field that your program can specify in the checkpoint request. This information can be used as input to your program when it is restarted, to assist it in re-establishing the proper environment. You must provide the mechanism to enter this information into your program for restart; DL/I does not provide this function.

The VSE CHKPT macro and similar facilities in COBOL and PL/I cannot be used in your batch DL/I programs. The operating system CHKPT facility (which is used by the COBOL and PL/I checkpoint facilities) requires that any VSAM data sets be closed before it is used. DL/I data bases are not closed by its checkpoint request, thus preventing the use of the operating system CHKPT facility.

DL/I Checkpoint in MPS Batch Programs

DL/I checkpoint should only be used in an MPS batch program operating in the following environment:

- The DL/I log is assigned to the CICS/VS system journal
- Dynamic Transaction Backout support has been generated into the CICS/VS system
- The DL/I Batch Partition Controller transaction, CSDC, has “DTB=YES” specified in its CICS/VS PCT entry.

When using the MPS Restart facility, additional environment requirements are:

1. The DL/I Master Partition Controller transaction, CSDB, must have “DTB=YES” specified in its CICS/VS PCT entry.
2. The MPS Restart purge temporary storage program, DLZMPURO, must be specified in the PCT and PPT.
3. “START=AUTO” must be specified in the System Initialization Table (SIT). Other specifications which support warm starts and emergency restarts should also be included.
4. Temporary storage queue name, DLZTSQ00, must have TYPE=RECOVERY specified in its CICS/VS TST entry.

When a DL/I checkpoint request is issued in a DL/I MPS batch program under these conditions, DL/I performs the following functions:

1. Writes any altered data base records currently in main storage back to the data bases. This action will also force write the log records for these changed data base records, if necessary. At this point any position in the data bases is lost.
2. Writes a CICS/VS sync-point record on the CICS/VS system journal
3. Dequeues any records or segments enqueued for this program if PI is being used
4. Writes message “DLZ105I” to the system console indicating that a DL/I checkpoint request has been successfully executed
5. Returns control to the application program.

If a failure occurs later, CICS/VS invokes its dynamic transaction backout facility and backs out all data base changes made since the last checkpoint request was issued.

In order to assure successful recovery and restart after a failure in an MPS batch program, the MPS Restart facility should be used. For additional information on the MPS Restart facility, refer to the “Restart” section later in this chapter.

If you do not use the MPS Restart facility, your MPS Batch Programs should have their own restart capabilities. This includes procedures to reestablish the program environment at the point of the last checkpoint request before failure.

VSAM Considerations in DL/I Recovery/Restart

Because VSAM is the primary access method used by DL/I, you should understand how VSAM and DL/I interact when failures or errors occur. Like DL/I, VSAM provides some facilities to assist in recovery. These facilities include:

- A facility to capture all changes made to the VSAM catalogs
- A mechanism to close VSAM data sets on an abnormal termination
- Utility programs to determine and correct catalog damage and reconstruct damaged data sets.

The DL/I recovery utilities normally provide all the facilities necessary to repair or reconstruct DL/I data base data sets after a failure. Therefore, the VSAM data set reconstruction utilities (REPRO being the primary one) are not normally used against the DL/I data sets. However, any catalog damage caused by a failure has to be corrected with the assistance of the VSAM facilities. DL/I provides no facilities to repair VSAM catalogs.

VSAM Catalog

The VSAM catalog contains a variety of information about its environment, including information about the volumes it owns and the data sets defined to it. The volume information includes information on:

- The VSAM space on each volume owned by the catalog, where the space is located on the volume, and when it was acquired (time stamp)
- The data sets that are assigned to particular volumes and the space they occupy.

The information about each data set that VSAM maintains in its catalog includes:

- Extents of the data set
- Volume-key range information
- Data set description (CI/CA size, allocation, key size (if KSDS), etc.).
- High RBA (Where is the current end of the data set within the space allowed to it?)
- Statistics on usage and activity (number of reads, number of updates, etc.)

The information in the catalog changes whenever:

- You DELETE, DEFINE, or extend the VSAM space on a volume
- You DELETE, DEFINE, or ALTER a data set
- A data set grows within the space allocated to it (high RBA)
- A secondary allocation is made for the data set
- The data set is accessed or updated (statistics).

The most frequent values that change in the catalog are the data set high RBA and the data set statistics. Therefore, these two values are the most likely to be in error after a failure. VSAM provides a special utility, VERIFY, to correct the high RBA in the catalog and an automatic close facility to minimize the chances of it not being updated on a failure.

The data set statistics are not essential to the integrity of the data sets so no facility is provided to recover the information in the event of a failure.

There are several things you can do in VSAM to minimize the potential for catalog and data set damage:

- Protect the master catalog from change by using user catalogs for all data set definitions. If the only objects defined in the master catalog are user catalogs, the master catalog will change very infrequently as the addition or deletion of a user catalog is normally a rare occurrence. Establishing an update password for the master catalog helps prevent unplanned master catalog changes.
- Establish separate user catalogs for each application and type of usage, i.e., test or production data sets. In this way catalog damage caused by failure in one application does not affect other applications. Because a volume can be owned by only one catalog, this may be difficult if there are many small VSAM

data sets required by a large variety of applications. At a minimum, you should try to establish separate test and production user catalogs.

- Use share option 1 or 2 for all VSAM data sets. Share option 1 is the only option that ensures full read and write integrity. Because DL/I-MPS programs residing in different partitions are really accessing the data bases through the common CICS/VS partition, VSAM share option 1 is sufficient if the data bases are not shareable. DL/I always opens its data base data sets for updates, regardless of the intent of the PSB (except for the case of read-only data sharing (PROCOPT=GO)). Because of this, share option 2 provides no functional benefit for VSAM's data sets when the data bases are not shareable.

  VSAM share option 2 is used to define data sets for shareable data bases. Specifying share option 2 alone allows the data base to be shared between subsystems on the same host system. If the data base is to be shared across host systems, both the data set and its VSAM catalog must be on devices shared between the host systems. VSAM share option 2 ensures write integrity only. Therefore, data consistency for read-only subsystems is not guaranteed.

  Share option 3 should never be used as it allows for concurrent scheduling of update jobs in different partitions with no warnings or data set integrity protection. It is especially important that share option 3 not be used with DL/I data base data sets as this could allow the high RBA value in the catalog to be updated incorrectly when the data sets are closed. Because DL/I depends on the high RBA value in the catalog for its operation, use of share option 3 can cause loss of information and internal data base pointer damage. You should only access DL/I data bases from multiple partitions using DL/I MPS.

  Share option 4 should not be used with DL/I's VSAM data sets either, as this can also result in improper DL/I operation.

Closing VSAM Data Sets

When a VSAM data set is closed, the data set statistics and the high RBA (if the data set was opened for update) are rewritten in the catalog. If, at end-of-job, the VSAM open list in the partition indicates that there are any open VSAM data sets, the VSAM close routines are invoked by the operating system job management routines. VSAM then closes the data sets, provided the necessary VSAM control blocks in the partition GETVIS area are not damaged. If the automatic close fails, the operating system writes a “4n26I” or “4n27I” message on the system console. Automatic close is invoked on a CICS/VS abend, as the DL/I data base data sets are not closed by DL/I under these conditions.

If a VSAM data set was not closed (due to a power failure or VSAM control block damage, for instance), the next time a program attempts to open a data set, VSAM returns an X‘74’ error code to the program. Under these conditions, the high RBA value in the catalog is probably incorrect. DL/I always writes a DLZ020I message (which includes the VSAM error code) on the system console and terminates if any error occurs during open. However, the data set is then closed and a subsequent open appears normal, hiding the earlier close failure. Therefore, if DL/I terminates with an open error, you should always run VSAM's VERIFY utility to ensure that the high RBA value in the catalog is correct.

Space Management Utilities IUP

The DL/I DOS/VS Space Management Utilities IUP, 5796-PKF, can help you in one area of recovery. One of the Space Management Utilities, the HD pointer checker, can be used to locate pointers in HD data bases that are in need of repair. The pointer checker utility detects data base segments that contain inaccurate physical or logical direct address pointers. A direct address pointer is the value of the relative byte number of a segment in a physical data base. The pointer checker utility reports the actual disk location of these segments so that you can take steps to repair the pointers. The utility doesn't perform the repair itself.

Pointers in an image copy of a data base can also be validated.

The pointer checker looks for:

- Missing targets (a target is the segment a pointer points to)
- Invalid targets
- Segments that are not pointed to by any other segment
- Invalid combinations of pointers to a particular target
- The presence of all required pointers to a particular target
- Correspondence between the counter field value and the actual logical child count.

For every pointer in error, the segment containing the pointer is identified. Also reported are:

- The target segment
- The cylinder/head/record/offset address of the pointer
- A snapshot of the physical block containing the pointer if the target is within the same physical block.

You should use the DL/I recovery utilities to correct the pointer errors, unless there are very few of them. In that case, time can be saved by using DITTO, a VSE utility, to make the repairs.

Restart

Once you are sure that all damage has been repaired by your recovery procedure, you are ready to restart the application that failed.

MPS Restart Facility

In an MPS batch environment, the MPS Restart facility should be used to assure successful restart if a DL/I program abnormally terminates. To invoke the MPS Restart facility, you must:

1. Use the function code, DLR, in the parameter input to DL/I when executing your MPS batch program. The DLR function code must be used instead of the DLI function code when the job is first started and not just when it it restarted.
2. Code a VSE checkpoint before each DL/I checkpoint in your application program.

If your MPS batch program abnormally terminates, the following procedure is necessary to restart it:

1. Obtain the checkpoint ID from which the job is to be restarted from the SYSLOG message issued by DL/I.

- If the individual MPS batch job failed, the message containing the correct checkpoint ID is issued at the time of failure.
- If there was a system failure, the message is issued when MPS is started again in the online partition.

2. Use the checkpoint ID on the VSE RSTRT job control statement to restart the job. The RSTRT statement is used instead of the EXEC statement when the job is resubmitted for execution.

Note: The jobname on the restart job must be the same as the name on the job that failed.

When the restart job is executed, your MPS batch program will be automatically restarted from the designated checkpoint ID.

To provide the restart capability described above, DL/I supports the use of VSE checkpoint/restart with the DL/I checkpoint facility. A VSE checkpoint writes a copy of the partition in which the checkpoint was issued onto disk or tape. The VSE RSTRT job control statement reloads this copy into the partition and passes control to the restart address that was specified when the VSE checkpoint was issued. For COBOL and PL/I programs using their respective VSE checkpoint interfaces, this corresponds to the next program statement after the one which invokes the VSE checkpoint.

MPS Restart provides the following functions:

1. Combined Checkpoint Verification

   MPS Restart verifies that a VSE checkpoint is issued immediately before each DL/I checkpoint. This is called a combined checkpoint. A VSE checkpoint may be issued in PL/I and COBOL by using the checkpoint interfaces provided by those languages. For information and examples of coding combined checkpoints, see *DL/I DOS/VS Application Programming: High Level Programming Interface* or *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces*.

2. MPS Batch Reinitialization

   The first DL/I call or command executed following a VSE restart must be a DL/I checkpoint. This will be the normal sequence when VSE checkpoints are placed immediately before each DL/I checkpoint. The DL/I checkpoint will automatically determine that a VSE restart has occurred and will reinitialize the MPS batch environment. Following successful reinitializaiton, the checkpoint call or command will return control to the application program as if from a normal checkpoint. The application program may then proceed with data base processing.

3. Correct Checkpoint ID Notification

   As each VSE checkpoint is taken, a message is issued to the system operator's console (SYSLOG) containing a checkpoint identifier (ID) which is unique to that checkpoint. This checkpoint ID is a 4-digit number between 0001 and 9999. To restart a job, the checkpoint ID for the checkpoint from which the job is to be restarted must be obtained from the messages on SYSLOG and specified as a parameter on the VSE RSTRT job control statement.

In the case of restarting a DL/I MPS batch job, the checkpoint ID for the *last* successful VSE checkpoint must be used. This is because when a failure occurs, dynamic transaction backout can only back out the DL/I data base(s) to the last successful DL/I checkpoint which was taken. Because the VSE and DL/I checkpoints are synchronized as one combined checkpoint, this corresponds to the last VSE checkpoint.

If an MPS batch job fails, it may have been some time since the last combined checkpoint was issued. To avoid having to look back through SYSLOG output for the last VSE checkpoint ID, DL/I issues a message to SYSLOG at the time of failure. This message contains the correct checkpoint ID. If the job fails before the first combined checkpoint it taken, a message will be issued indicating that the job should be started over from the beginning of the job step.

If a system-wide failure occurs while MPS batch jobs are executing, there is no opportunity for such messages at the time of failure. In this case, when MPS is started again in the online partition, DL/I will issue messages to SYSLOG containing the correct checkpoint ID for each MPS batch job which had taken a combined checkpoint before the failure occurred. Messages will not be issued for jobs which did not take a combined checkpoint before the failure.

4. Checkpoint ID Verification

   When restarting an MPS batch job, if the correct (last) VSE checkpoint ID was not specified on the VSE RSTRT job control statement, the job would be restarted from the wrong checkpoint and desynchronization between the application program and the data base would result. This could easily invalidate information written to the data base, particularly where the application program performs updates based on the information already contained in the data base.

   Because of the importance of using the correct VSE checkpoint ID to restart MPS batch jobs, DL/I will verify that the checkpoint ID used for restart is the correct one. If the checkpoint ID is not correct, DL/I will issue a message to SYSLOG and to SYSLST containing the correct checkpoint ID. The job will then be cancelled, allowing you to restart the job again, using the correct checkpoint ID which was provided in the message.

Restart Considerations

The following are special considerations when restarting MPS batch jobs:

1. The VSE restart facility requires the jobname to be the same on a restart job as it was on the job that failed.
2. During a VSE restart, a data check (tape checkpoint files) or end-of-file (disk checkpoint files) may occur if the checkpoint ID specified is greater than the actual number of checkpoints taken before the failure.
3. On a restart, parameter input is ignored by DL/I because the parameters were already read and saved when the job first started. However, if the parameter input statment was included on SYSIPT (instead of having been entered from SYSLOG) when the job first started, it is important that one also be included when the job is restarted. This is because Dl/I will attempt to position SYSIPT past the parameter input statement when the job is restarted.

Restrictions on Using VSE Checkpoint/Restart

Certain restrictions exist on the use of VSE checkpoint/restart which you should be aware of. For example, VSAM files must be closed before a VSE checkpoint is issued (DL/I data bases used by MPS programs are in the online partition and are not affected by this restriction). Also, VSE restart cannot be used to restart programs which failed because of program logic errors. This is because a copy of the program, exactly as it was before it failed, is loaded back into the partition during a restart.

For details on these and other restrictions, see *VSE/Advanced Functions Application Programming: Reference* and *VSE Advanced Functions Application Programming: User's Guide*.

Purging the Temporary Storage Queue Used by MPS Restart

Occasionally, it may be desirable to purge the CICS/VS temporary storage queue (TSQ) used by MPS Restart. Under normal operating conditions, this will not be necessary and is not recommended. However, if MPS batch jobs using MPS Restart abnormally ended and are not to be restarted, you may wish to purge the TSQ. This will eliminate the issuing of messages containing checkpoint IDs for those jobs each time MPS is started and will free up space in the temporary storage data set. Use transaction CSDP to request that the TSQ be purged. Access to this transaction should be restricted.

Purging the TSQ eliminates the entire TSQ and all the information it contained. A purge is not allowed while MPS batch jobs using MPS Restart are running. This would cause a Temporary Storage error the next time one of those jobs issued a combined checkpoint. A purge should also not be requested if any MPS batch jobs using MPS Restart abnormally ended and have not yet been restarted. This would destroy the checkpoint ID information used for verification when those jobs are restarted. If the TSQ is purged before such a job is restarted, the job may still be restarted, but checkpoint ID verification will need to be overridden by responding appropriately to the messages issued during restart. You should be aware of the possible dangers of using this procedure, because restarting from the wrong checkpoint will result in desynchronization between the application program and the data base(s).

User Written Restart Programs

To restart MPS batch programs which do not use the MPS Restart facility, as well as other normal batch and online programs, you must include logic in the program which allows it to start at a point where the environment corresponds to that present when the last checkpoint request was successfully executed before the failure. This program logic may have to include facilities to reposition non-DL/I files, restore internal counters and total fields, etc.

The checkpoint message (DLZ105I) written on the system console at the time the checkpoint was executed contains an eight-byte identification field. You can use this identification as input when you restart your program to assist it in re-establishing the proper environment.

Checkpoint message DLZ105I is not written in the online environment, so your application program must include its own provisions for identifying where restart is to begin after a CHECKPOINT request is executed.

Part 6. Appendixes

Appendix A. DL/I Virtual Storage Estimates

This appendix describes the method of making DL/I virtual storage estimates.

This appendix shows you how to make a detailed analysis of virtual storage requirements for your DL/I system. Work sheets and examples are provided. See Appendix B for a discussion of DL/I real storage requirements.

DL/I Batch System Storage Requirements

The operating system partition in which the DL/I data base batch system operates must include storage space for several major items. They are:

1. DL/I data base system modules
2. DL/I program specification blocks (PSBs) and associated blocks
3. DL/I data management blocks (DMBs) derived from data base descriptions (DBDs)
4. DL/I data base buffer pool and control blocks
5. VSE/VSAM access method modules
6. VSAM/SAM access method buffers
7. Application programs.

Each item is discussed in detail below. Figure A-1 is a worksheet for accumulating the estimated values.

The values required are:

1. The size of the virtual partition. This value is line 15 of Figure A-1.
2. The value to be used in the SIZE parameter in the job control EXEC statement. The minimum value is line 11. It is recommended that this value be increased by at least 10%.

Optionally, the reentrant modules may be placed in the operating system shared virtual area (SVA). See the sections called “Storage Layout Control (SLC) Option” in both chapter 4 and 5 for details of a list of the modules that are eligible for placement in the SVA.

Ref No.	Component	Bytes	Rounded to Next 2K Multiple
1	DL/I Initialization and Termination Routines		
2	DL/I Nucleus and Program Request Handler	No requirement	
3	PSB Size*		
4	DMB Sizes		
5	Subtotal of lines 3 and 4		
6	DL/I Buffer Pool and Control Block Size*		
7	DL/I Data Base Organization Dependent Modules*		
8	VSE/Advanced Functions SAM Modules		
9	Subtotal = DL/I System		
10	Application Program*		
11	Subtotal (Minimum SIZE parameter value) = DL/I System and Application Program		
12	VSE/VSAM Modules and Control Blocks*		
13	VSAM/SAM Buffers		
14	Subtotal of lines 12 and 13 = Access Method Modules and Buffers		
15	Total of lines 11 and 14 = DL/I System, Application Program, and Access Method Modules and Buffers		

Figure A-1. DL/I Virtual Storage Requirements Worksheet

DL/I Initialization and Termination Routines

The initialization and termination routines are always required. The approximate storage requirement for all modules is 25,000 bytes. They are normally paged out during application program execution.

DL/I Batch Nucleus and Program Request Handler

The DL/I nucleus and program request handler is required independent of the data base organizations used by the application programs and the manner in which the data bases are used. The requirement is approximately 3,750 bytes. However, this storage overlays part of the initialization routines and requires no additional storage in your computation.

DL/I Program Specification Block (PSB)

One PSB is required for execution. For the size of the PSB you can use the size as it exists in a core image library after being processed by the application control blocks creation and maintenance utility (DLZUACB0). This block can be displayed by using the VSE CSERV utility, specifying the PSB name (the name from the PSB generation, extended with at-signs (@) if necessary, with a “P” appended in the eighth position). Its size can also be found in the linkage editor map produced when the PSB is link-edited into a core image library. Additional storage must be

allocated for the PSB work areas. The size of these areas can be found in the PSB prefix section of the PSB CSERV output. The offset to the prefix is contained in the first 4 bytes of the block. Offsets 4, 8, 20, and 24 from the beginning of the PSB prefix contain the fullword sizes of the four possible work areas. In summary: the space requirement for the PSB is equal to the size of the PSB plus the total size of the PSB work areas.

DL/I Data Management Blocks (DMBs)

The data base descriptions (DBDs) must be converted to internal control blocks, called data management blocks (DMBs), for use by DL/I. The space requirements for the DMBs used during an execution of the data base system are the sum of all DMB sizes.

For the size of a DMB you can use the size as it resides in a core image library after being processed by DLZUACB0. To this size, add 168 if the data base organization is HD, HDAM, HIDAM, INDEX, or SHISAM; add 296 if the organization is HISAM. No additions are required for HSAM or SHSAM. Do not include the size of the MTMOD (DLZTAPE) or SDMODS (DLZDISKI and DLZDISK0) in the HSAM DMB size calculation.

DL/I Data Base Buffer Pool and Control Blocks

The buffer pool is subdivided into one or more subpools. The number of subpools is specified at DL/I initialization. It is based on the number of data bases required, work space needed for deleting and inserting records, and the amount of main storage to be allocated for a buffer pool.

A subpool consists of a maximum of 32 fixed-length buffers. The buffer size is consistent with the VSAM data base control interval size and may be 512, or any multiple of 512, to a maximum of 4,096. During DL/I initialization, a data base is assigned a specific subpool based on user options or on the control interval size and the number of subpools allocated. A data base may not be assigned to more than one subpool. However, multiple data bases can be assigned to the same subpool. A data base control interval size may not exceed the subpool buffer size but may be less than that size. Additionally, the size of the control blocks that manage the buffer pool must be calculated.

To calculate the DL/I data base buffer pool and control blocks size use the formula:

```
136 + ((bufno x (bufsize + 32) + 46) for each subpool)
```

where:

```
bufno = number of buffers in the subpool
bufsize = buffer size.
```

DL/I Modules--Data Base Organization Dependent

The following DL/I storage requirements depend on the data base access methods and processing options used by the application program.

Use Figure A-2 to determine these requirements. The sum of all the sizes of all user-provided randomizing routines, index exit routines, and compression routines must be added to the size calculated from the figure.

```
A set of conditions is listed for each item
below (where duplicated from item to item,
use both figures).  If all conditions are
met, the value should be included in the
estimate.  The sum of all the values selected
provides the total loaded module requirements.
Do not select a given entry more than once.
   1. Basic code (required)             3,800
   2. Any data base PCB except         25,000
      PROCOPT=L for HSAM, HISAM,
      HD indexed, or HIDAM
   3. Any PROCOPT = I, L, or A or if    8,500
      HSAM
   4. Any PROCOPT = D, R, or A         14,500
   5. Any primary or secondary indexes  3,400
      and PROCOPT = I,R,D,A, or L
   6. Any data base except HSAM         7,500
   7. Any HD type data base and         4,800
      PROCOPT = I, R, D, A, or L
   8. Any HD type data base and        29,000
      PROCOPT = L
   9. Any PROCOPT = I, R, D, or A       3,500
      and logging is desired           _______
   TOTAL if all conditions are met    100,000
Notes:
  L = Load (also includes LS)
  I = Insert
  R = Replace
  D = Delete
  A = All = G, I, R, D (not for simple HSAM
      and HSAM).
```

Figure A-2. Storage Estimate for DL/I Data Base Organization Dependent Modules

VSE Modules--Access Method Dependent

The VSE data management access method modules used in the DL/I system environment are VSAM and SAM. Figure A-3 shows how they are used.

Access Method	DL/I Use	Space Requirements (Approximate)
VSAM	VSAM GET/PUT	240,000 bytes
SAM	HSAM GET/PUT	Tape - 800 bytes DASD - 1500 bytes
	UTILITIES GET/PUT	IJJFCBZD - 700 bytes IJFUBZZN - 2400 bytes IJFSZZWN - 1400 bytes IJFFZZZN - 900 bytes

Figure A-3. Data Management Access Method Space Requirements

Note: The examples that follow assume a value of 240,000 bytes for these access method dependent modules. Since VSAM storage requirements are installation

dependent, you should determine your own VSAM requirements, and use this figure for the rest of

your calculations. See *VSE/VSAM Programmer's Reference*.

VSAM/SAM Buffers

VSAM allocates buffers for each file of a HISAM data base or for an INDEX data base. Unless you specify otherwise, three index buffers and two data buffers are allocated for the KSDS, and two data buffers are allocated for the ESDS. You can specify a larger number when loading a data base, or when updating or retrieving from one, which may give improved performance.

Two buffers are allocated for each HSAM data base. However, these buffers are not part of the DL/I buffer pool. The HSAM buffer size is equal to the HSAM record size.

DL/I Online System Storage Requirements

The DL/I storage requirements in an online partition are similar to those in a batch system. However, the following additional factors must be considered:

1. DL/I online nucleus module
2. DL/I tables (ACT, PPST, PDIR, RPDIR, DDIR)
3. DL/I partition specification table (PST)
4. Additional PSB storage.

Also, all PSBs and DBDs required are permanently loaded. These items are discussed in more detail in the following paragraphs.

DL/I Online Nucleus

This module contains scheduling and termination routines, the online program request handler, and error routines. The storage requirement is about 29,300 bytes (30,900 bytes if a remote PSB is defined in the ACT).

DL/I Tables (ACT, PPST, PDIR, RPDIR, DDIR)

These control blocks are generated by the user and link-edited with the online nucleus module. The total size of these tables depends on the specifications in the DLZACT macro and can be calculated as follows:

```
Total size = PPST + PDIR + RPDIR + DDIR + ACT
```

where:

```
PPST  = 40 + 32 x (value of MAXTASK in DLZACT macro)
PDIR  = 28 x (number of unique PSBs specified in
        DLZACT macro)
RPDIR = 16 x (number of RPSB entries specified
        in DLZACT macro)
DDIR  = 24 x (number of DBDs referenced by
        specified PSBs (directly or indirectly))
ACT   = 22 + 11 x (number of program names
                   specified in DLZACT macro)
           +  2 x (total number of PSBs
                   specified in DLZACT macro
                   including duplicates)
           +  4 x (number of HDBFR entries
                   specified in DLZACT macro)
           +  8 x (number of DBD names
                   specified in HDBFR entries)
           + 16 x (number of HSBFR entries
                   specified in DLZACT macro).
```

DL/I Partition Specification Table (PST)

A PST is allocated in CICS/VS dynamic storage each time a DL/I task is scheduled. Storage is released when a DL/I TERM call is issued by the task. The size of one PST is 1,144 bytes.

Additional PSB Storage

Each time a task schedules a read-only PSB, or an update PSB if program isolation is used, which is already in use by another task, a copy of this PSB is created in CICS/VS dynamic storage. Therefore, the size of such PSBs must be multiplied by the maximum number of tasks that could be using them concurrently.

DL/I MPS System Storage Requirements

While the use of the MPS facility slightly increases the storage requirements for the online partition, the MPS batch partitions require significantly less storage than if the same jobs were run in batch partitions.

1. MPS online partition:

   Add to the online system requirements (only if MPS is active):

   ```
   Total size = MPC + (BPC x number of active
                partitions))
   ```

   where:

   ```
   MPC = 2,850 + (TCA size + 488 transaction
         work area)
   BPC = 1,050 + ((TCA size + 256 transaction
         work area)
   ```

2. MPS batch partition:

   In each batch partition, the only DL/I component present contains initialization, termination, and program request handler modules. Its size is approximately 15,500 bytes. Use of PL/I requires space for a PCB list. Add four bytes for each PCB. Application program storage requirements must be added to this.

DL/I Batch System Storage Requirement Example

The following environment is assumed for the calculation of storage requirements in this example:

- One 20,000-byte application program
- Three data bases - HSAM, HISAM, and HD indexed organizations
- 240,000 bytes for VSAM space allocation
- One buffer subpool with 12 buffers
- Buffer size is 2K
- All DL/I functions are being used.

Figure A-4 is the worksheet referred to in the following discussion of this example. The numbers to the left of the paragraphs correspond to the reference numbers in the first column of the figure.

Ref No.	Component	Bytes	Rounded to Next 2K Multiple
1	DL/I Initialization and Termination Routines	25,000	26K
2	DL/I Nucleus and Program Request Handler	No requirement	
3	PSB Size*	3,792	
4	DMB Sizes	3,056	
5	Subtotal of lines 3 and 4	6,848	8K
6	DL/I Buffer Pool and Control Block Size*	25,142	26K
7	DL/I Data Base Organization Dependent Modules*	71,000	70K
8	VSE/Advanced Functions SAM Modules	800	2K
9	Subtotal = DL/I System		132K
10	Application Program*	20,000	20K
11	Subtotal (Minimum SIZE parameter value) = DL/I System and Application Program		152K
12	VSE/VSAM Modules and Control Blocks*	260,000	256K
13	VSAM/SAM Buffers	18,360	18K
14	Subtotal of lines 12 and 13 = Access Method Modules and Buffers	278,360	274K
15	Total of lines 11 and 14 = DL/I System, Application Program, and Access Method Modules and Buffers		426K

Figure A-4. Worksheet for DL/I Data Base System Example

1. DL/I initialization and termination routines require 25,000 bytes. Enter on line 1 of the worksheet.
2. DL/I nucleus and program request handler overlay part of the initialization routines and, therefore, require no additional storage.
3. The PSB size as determined from the core image library is 3,792. Enter this figure on line 3 of the worksheet.

4. Four DMBs are required for the three data bases referenced (including one for the index data base). Their sizes as determined from the core image library are:

   ```
   HSAM DMB  = 576
   HISAM DMB = 856
   HD DMB    = 1,312
   INDEX DMB = 312
   ```

   Adding the DMBs (576 + 856 + 1,312 + 312) = 3,056.

   Enter this figure on line 4 of the worksheet.
5. Lines 3 and 4 should be totaled and rounded to the next 2K multiple to ensure alignment on a 2K boundary.

   ```
   3,792 + 3,056 = 6,848, rounded to 8K.
   ```

   Enter this on line 5 of the worksheet.
6. One data base buffer subpool with 12 buffers is used. The largest control interval size is 2,048 bytes.

   The buffer pool size = 136 + ((12 x (2048 + 32) + 46) = 25,142, rounded to 26K.

   Enter on line 6 of the worksheet.

7. The DL/I organization module requirement is determined from Figure A-2. Item 8 is the only one not chosen. The sum of the sizes required is 71,000 bytes (70K).

   Enter on line 7 of the worksheet.
8. Because one of the data bases is HSAM, 800 bytes is entered for SAM tape support. Enter on line 8 of the worksheet.
9. Lines 1, 5, 6, 7, and 8 (rounded to 2K multiples) are added to give the total storage requirements for DL/I.

   26K + 8K + 26K + 70K + 2K = 132K

   Enter on line 9 of the worksheet.
10. The size of the application program is 20,000 bytes. This value is rounded to 20K and entered on line 10 of the worksheet.
11. Lines 9 and 10 are added to give the total storage requirements for DL/I and the application program. This value can be used for the SIZE parameter of the job control EXEC statement. It is recommended that it be increased by at least 10%.
12. VSE/VSAM modules and control blocks require 260,000 bytes.

    Enter this figure on line 12 of the worksheet.
13. VSAM/SAM buffer requirements (assuming SAM buffer for HSAM, and VSAM buffer for HISAM and HD) are:

    ```
    VSAM buffers = 2 KSDS x (2 buffers x 2,048)
                 = 8,192
                 = 1 ESDS x (2 buffers x 2,048)
                 = 4,096
                 = 2 KSDS x (3 index buffers x 512)
                 = 3,072
    SAM BUFFERS  = 2 buffers x 1,500
                 = 3,000

    8,192 + 4,096 + 3,072 + 3,000 = 18,360
    ```

    Enter on line 13 on the worksheet.

14. Lines 12 and 13 are added and rounded to the next 2K multiple to give the total of access method related modules and buffers.

    ```
    260,000 + 18,360 = 278,360, rounded to 274K.
    ```

    Enter on line 14 on the worksheet.
15. Lines 11 and 14 are added to give the total virtual storage requirements for the DL/I system, including the application program and VSE access method related modules and buffers.

    152K + 274K = 426K

    Enter on line 15 on the worksheet. This value represents the virtual storage that must be allocated (ALLOC) to the partition in which the DL/I application program will execute.

DL/I Data Base Utilities Storage Requirements

This section shows you how to estimate storage requirements for the DL/I data base utilities used for DL/I application control block (ACB) building, data base recovery, and data base reorganization/load processing functions of DL/I. The minimum value required for the SIZE parameter in the EXEC card can be calculated by subtracting the sizes of any GETVIS and VSAM items from the total size calculated for the partition. To be safe, add an additional 10% to the calculated value.

These utilities, which can be grouped as shown below, are discussed more fully in *DL/I DOS/VS Resource Definition and Utilities*.

- DL/I application control blocks creation and maintenance - DLZUACB0
- Data base recovery
  - Data base data set image copy - DLZUDMP0
  - Data base change accumulation - DLZUCUM0
  - Data base data set recovery - DLZURDB0
  - Log print - DLZLOGP0
  - Data base backout - DLZBACK0
- Data base physical reorganization
  - HISAM reorganization unload - DLZURUL0
  - HISAM reorganization reload - DLZURRL0
  - HD reorganization unload - DLZURGU0
  - HD reorganization reload - DLZURGL0
- Data base logical relationship resolution
  - Data base prereorganization - DLZURPR0
  - Data base scan - DLZURGS0
  - Data base prefix resolution - DLZURG10
  - Data base prefix update - DLZURGP0.

Application Control Blocks Creation and Maintenance Utility - DLZUACB0

The following formula can be used to estimate the total virtual storage requirements for a given execution of the DL/I application control blocks creation and maintenance utility.

```
Size in bytes = DLZUACB0 + SAM + GETVIS
                + VSAM buffer size + GETVIS
```

where:

```
DLZUACB0 = 47,000 bytes.
SAM      = Size of the SAM data management
           modules.  Refer to the value
           in Figure A-2 for IJJFCBZD.
GETVIS   = Three multiplied by the largest PSB
           specified, rounded to the nearest
           128-byte multiple, plus three multi-
           plied by the largest size of all DBDs
           referenced by any one PSB.  The sizes
           of PSBs and DBDs can be found from the
           PSB generation and DBD generation
           assembly listings.
```

Data Base Data Set Image Copy Utility - DLZUDMP0

The following formula can be used to estimate the total virtual storage requirements for a given execution of the data base data set image copy utility.

Size in bytes = DLZUDMP0 + SAM + VSAM +
VSAM buffer size + GETVIS

where:

```
DLZUDMP0 = 27,000 bytes.
VSAM     = Size of the VSE/VSAM data
           management modules.  Refer to the
           VSAM value in Figure A-3.
SAM      = Size of the SAM data management
           modules.  Refer to the values
           in Figure A-3 for IJJFCBZD and
           IJFSZZWN.
VSAM     = Space required for the I/O buffers used by
buffer     VSAM: the control interval size x number of
size       buffers.  Refer to "VSAM/SAM Buffers"
           in this appendix.  Ten data
           buffers are allocated (as well as four
           index buffers if HISAM).
GETVIS   = DMB size rounded to the nearest 128-byte
           multiple.  The DMB size formula is detailed
           under "DL/I Data Management Blocks (DMBs)"
           in this appendix.
```

Data Base Change Accumulation Utility - DLZUCUM0

The following formula can be used to estimate the total virtual storage requirements for a given execution of the data base change accumulation utility.

```
Size in bytes = DLZUCUM0 + SAM + GETVIS + SORT
                ⌈                          ⌉
                |+ VSAM + VSAM buffer size |
                ⌊                          ⌋
```

where:

```
DLZUCUM0 = 62,500 bytes.
SAM      = Size of the SAM data management
           modules.  Refer to the values
           in Figure A-3 for IJJFCBZD,
           IJFSZZWN, and IJFUZZZZ (subset of
           IJFUBZZN).
GETVIS   = 14,736 bytes of virtual address space for
                      ⌈
           work areas + |2 x maximum log buffer size
                      ⌊
                                                    ⌉
           specified on an LI input control statement.|
                                                    ⌋

           This value is required only if a buffer size
           greater than 1,024 is requested.
SORT     = Size of VSE Sort modules.
           Refer to
           DOS/VS-VM/System Product Sort Merge:
           Programmer's Guide, SC33-4044.
VSAM     = Size of the VSE/VSAM data management
           modules (required for VSAM log files only).
           Refer to the VSAM value
           in Figure A-3.
VSAM     = Space required for the I/O buffers used by
buffer     VSAM: the control interval size x number of
size       buffers (required for VSAM log files only).
           Refer to "VSAM/SAM Buffers"
           in this appendix.  Ten data buffers
           are allocated.  The control interval size
           is 1,024.
```

Data Base Data Set Recovery Utility - DLZURDB0

This utility executes under the control of DL/I. Therefore, the virtual storage requirements should be calculated using the DL/I data base worksheet provided in this appendix. The reference item numbers are indicated for each module component.

```
Size in bytes = DLZURDB0 + SAM + GETVIS
                           ⌈                ⌉
                           |+ VSAM buffer size|
                           ⌊                ⌋
```

where:

DLZURDB0 = 44,500 bytes (Item 10).
SAM = Size of the SAM data management modules. Refer to the values in Figure A-3 for IJJFCBID (superset of IJJFCBZD), IJFSZZWN, and IJFUZZZN.
GETVIS = 2 x maximum log buffer size specified on an LI input control statement. This value is required only if a buffer size greater than 1,024 is requested.
VSAM buffer size = Space required for the I/O buffers used by VSAM: the control interval size x number of buffers (required for VSAM log files only). Refer to "VSAM/SAM Buffers"

in this appendix. Ten data buffers
are allocated. The control interval size
is 1,024 (add to Item 13).

Log Print Utility - DLZLOGP0

The following formula can be used to estimate the total virtual storage requirements for a given execution of the log print utility:

```
                                          ⎡
Size in bytes = DLZLOGP0 + SAM            ⎢+VSAM + VSAM
                                          ⎣

                           ⎤ ⎡          ⎤
                buffer size⎥ ⎢+ GETVIS  ⎥
                           ⎦ ⎣          ⎦
```

where:

DLZLOGP0 = 19,500 bytes

SAM = Size of the SAM data management modules. Refer to the values in Figure A-3 for IJJFCBID (superset of IJJFCBZD) and IJFUZZZN (subset of IJFUBZZN).

VSAM = Size of the VSE/VSAM data management modules (required for VSAM log files only). Refer to the VSAM value in Figure A-3.

VSAM buffer size = Space required for the I/O buffers used by VSAM: the control interval size x number of buffers (required for VSAM log files only). Refer to "VSAM/SAM Buffers" in this appendix. Ten data buffers are allocated. The control interval size is 1,024.

GETVIS = ((N-10)/10) x 128 where N is the maximum number of concurrent tasks (as reflected by the log file) rounded up to a multiple of 10. This value is required for online log files only.

\+ (N x maximum log buffer size) where N=4 if log copy is requested; otherwise N=2, and maximum log buffer size is the largest size specified on an LI control statement. This value is required only if a buffer size greater than 1,024 is requested.

Data Base Backout Utility - DLZBACK0

This utility executes under the control of DL/I. Therefore, the virtual storage requirements should be calculated using the DL/I data base worksheet provided in this appendix. The reference item numbers are indicated for each module component.

$$\text{Size in bytes} = \text{DLZBACK0} + \text{SAM} \left\lceil \begin{array}{l} \\ \text{buffer size} \end{array} \right\rceil + \text{GETVIS} + \text{VSAM}$$

where:

DLZBACK0 = 11,500 bytes (Item 10).

SAM = Size of the SAM data management modules. Refer to the values in Figure A-3 for IJJFCBZD and IJFUBZZZ (subset of IJFUBZZN) (Item 8).

GETVIS = 2 x maximum log buffer size specified on an LI input control statement. This value is required only if a buffer size greater than 1,024 is requested (add to Item 12).

VSAM buffer size = Space required for the I/O buffers used by VSAM: the control interval size x number of buffers (required for VSAM log files only). Refer to "VSAM/SAM Buffers" in this appendix. Ten data buffers are allocated. The control interval size is 1,024 (add to Item 13).

HISAM Reorganization Unload Utility - DLZURUL0

The following formula can be used to estimate the total virtual storage requirements for a given execution of the HISAM reorganization unload utility.

```
Size in bytes = DLZURUL0 + DLZRULM0 + VSAM + SAM
                + VSAM buffer size + GETVIS
```

where:

```
DLZURUL0 = 24,000 bytes
DLZRULM0 = 1,100 bytes
VSAM     = Size of the VSE/VSAM data management
           modules.  Refer to the value for VSAM
           in Figure A-3.
SAM      = Size of the SAM data management
           modules.  Refer to the values
           in Figure A-3 for IJJFCBZD
           and IJFUZZZN (subset of IJFFZZZN).
VSAM     = Space required for the I/O buffers used by
buffer     VSAM: the control interval size x number of
size       buffers.  Refer to "VSAM/SAM Buffers"
           in this appendix.  Ten data
           buffers and four index buffers are allocated
           to the KSDS, and two data buffers are
           allocated to the ESDS.
```

```
                ┌                                  ┐
GETVIS   = 168 + │the following, each rounded to a  │
                 │128-byte multiple:                │
                 │a. DBD size - see assembly        │
                 │   listing of DBDGEN              │
                 │b. 48 bytes per segment           │
                 │c. Sum of:                        │
                 │   1. larger of 48 or ESDS size   │
                 │      (KSDS if SHISAM)            │
                 │   2. KSDS size                   │
                 │   3. ESDS size (0 if SHISAM)     │
                 └                                  ┘
```

HISAM Reorganization Reload Utility - DLZURRL0

The following formula can be used to estimate the total virtual storage requirements for a given execution of the HISAM reorganization reload utility.

```
Size in bytes = DLZURRL0 + DLZRRLM0 + VSAM
                + SAM + VSAM buffer size + GETVIS
```

where:

```
DLZURRL0 = 16,000 bytes
DLZRRLM0 = 900 bytes
VSAM     = Size of the VSE/VSAM data management
           modules.  Refer to the VSAM value
           in Figure A-3.
SAM      = Size of the SAM data management
           modules.  Refer to the values
           in Figure A-3 for IJJFCBZD
           and IJFUZZZN.
VSAM     = Space required for the I/O buffers used by
buffer     VSAM: the control interval size x number of
size       buffers.  Refer to "VSAM/SAM Buffers"
           in this appendix.  Ten data
           buffers and four index buffers are allocated
           to the KSDS, and ten data buffers are
           allocated to the ESDS.
GETVIS   = 40 x number of segment types, rounded
           to the nearest 128-byte multiple.
```

HD Reorganization Unload Utility - DLZURGU0

This utility executes under the control of DL/I. Therefore, the virtual storage requirements should be calculated using the DL/I data base worksheet provided in this appendix. The reference item numbers are indicated for each module component.

```
Size in bytes = DLZURGU0 + SAM + GETVIS
```

where:

```
DLZURGU0 = 45,800 bytes (Item 10).
SAM      = Size of the SAM data management
           modules.  Refer to the values
           in Figure A-3 for IJJFCBZD
           and IJFUZZZN.
GETVIS   = a. 40 x number of segment types, rounded
              to the nearest 128-byte multiple.
           b. Maximum segment size + 40, rounded
              to the nearest 128-byte multiple
              (add to item 12).
```

HD Reorganization Reload Utility - DLZURGL0

This utility executes under the control of DL/I. Therefore, the virtual storage requirements should be calculated using the DL/I data base worksheet provided in this appendix. The reference item numbers are indicated for each module component.

```
Size in bytes = DLZURGL0 + SAM
```

where:

```
DLZURGL0 = 17,200 bytes (Item 10).
SAM      = Size of the SAM data management
           modules.  Refer to the values
           in Figure A-3 for IJJFCBZD
           and IJFSBZZN.
```

Data Base Prereorganization Utility - DLZURPR0

This utility executes under the control of DL/I. Therefore, the virtual storage requirements should be calculated using the DL/I data base worksheet provided in this appendix. However, because no DL/I calls are made, the VSAM code and buffers (items 12 and 13) are never loaded and the DL/I code and buffers (items 7 and 8) are loaded but never used. The reference item number of each component of the prereorganization utility is indicated.

```
Size in bytes = DLZURPR0 + SAM + GETVIS
```

where:

```
DLZURPR0 = 9,000 bytes (Item 10).
SAM      = Size of the SAM data management
           modules.  Refer to the values
           in Figure A-3 for IJJFCBZD.
GETVIS   = Size of control file.  Allow
           approximately 4,096 bytes (Item 12).
```

Data Base Scan Utility - DLZURGS0

This utility executes under the control of DL/I. The virtual storage requirements should be calculated using the DL/I data base worksheet provided in this appendix. Include Item 8 from Figure A-2 in the size calculation of the DL/I data base dependent modules (Item 7 of the worksheet). The reference item number of each scan utility component is indicated.

```
Size in bytes = DLZURGS0 + SAM + GETVIS
```

where:

```
DLZURGS0 = 23,000 bytes (Item 10).
SAM      = Size of the SAM data management
           modules.  Refer to the values
           in Figure A-3 for IJJFCBZD
           and IJFVZZZN (subset of IJFSZZWN).
GETVIS   = (Size of the control file, rounded to next
           128-byte multiple) + (size of the longest
           logically related segment + 256, rounded to
           the next higher 128-byte multiple).  Assume
           4,096 as the control data set size, unless
           it is larger (Add to Item 12).
```

Data Base Prefix Resolution Utility - DLZURG10

The following formula can be used to estimate the total virtual storage requirements for a given execution of the data base prefix resolution utility.

```
Size in bytes = DLZURG10 + SAM + SORT + GETVIS
```

where:

```
DLZURG10 = 40,000 bytes.
SAM      = Size of the SAM data management
           modules.  Refer to the values
           in Figure A-3 for IJJFCBZD,
           IJFFZZZN, and IJFVZZZN (subset of IJFSZZWN).
SORT     = Size of VSE Sort modules.
           Refer to
           DOS/VS-VM/System Product Sort Merge:
           Programmer's Guide, SC33-4044.
GETVIS   = Size of the control file, rounded to the
           next higher 128-byte multiple.  Assume
           4,096 as the control data set size, unless
           it is larger.
```

Data Base Prefix Update Utility - DLZURGP0

This utility executes under the control of DL/I. Therefore, the total storage requirements should be calculated using the DL/I data base worksheet provided in this appendix. The reference item numbers are indicated for each data base prefix update component.

```
Size in bytes = DLZURGP0 + SAM + GETVIS
```

where:

```
DLZURGP0 = 12,800 bytes (Item 10).
SAM      = Size of SAM data management
           modules.  Refer to the values
           in Figure A-3 for IJJFCBZD
           and IJFVZZZN (subset of IJFSZZWN) (Item 8).
GETVIS   = If a secondary index data base is being
           updated, size of the longest index
           segment, rounded to the next higher
           128-byte multiple.  (Add to Item 12).
```

Appendix B. DL/I Real Storage Estimates

This appendix describes the method of making DL/I real storage estimates.

This appendix shows you how to calculate the approximate real storage requirements of DL/I and the pageable components it invokes (DOS/VSE access method modules and VSAM). (See *CICS/VS Performance Guide* for a discussion of CICS/DOS/VSE real storage requirements.)

For this purpose, the real storage requirement is considered to be the number of referenced pages during normal program execution. Initialization and termination, error, or exceptional condition processing are excluded.

DL/I will run in less real storage than the amount estimated, but only at the expense of increased paging, which results in decreased performance.

When estimating real storage requirements use peak load conditions.

The real storage required by a DL/I system is the sum of the real storage requirements of the following components:

- DL/I action modules (code)
- DL/I control blocks (PSBs, DMBs, etc.)
- DL/I buffer pool
- VSAM action modules (code)
- VSAM control blocks (AMBLs, OPWAs, etc.)
- VSAM buffers
- Application programs

and for online:

- CICS/VS management modules (code)
- CICS/VS control blocks and tables (CSA, PCT, etc.)
- CICS/VS dynamic storage.

DL/I Action Modules

The chart in Figure B-1 shows the sizes and usage of the various components of DL/I DOS/VS. To compute your DL/I real storage requirements, total the values listed for the appropriate components. Modules not shown in this chart are used only during initialization, termination, or exceptional condition processing and are not to be included. It is assumed that the eligible DL/I modules (indicated by *) have been placed in the SVA. Use of the SVA for DL/I action modules may not always be desirable in a paging environment. However, their location, whether in the CICS/VS partition or the SVA, will not significantly affect their real storage requirements. See the sections called “Storage Layout Control (SLC) Option” in both chapter 4 and 5 for more details.

Figure B-1. DL/I Action Module Real Store Requirements

DLZ...	Function	Size	Usage
DLA00*	Call Analyzer	6K	Always
DLR00*	Retrieve	23K	Always
		+6K	If LR used
DBH00*	Buffer Handler	10K	Always
DXMT0*	Index Maintenance	4K	HD indexed or HIDAM, or HD indexed or HIDAM with SI and PROCOPT≠G
DHDS0*	Space Management	8K	HD and PROCOPT≠G
DDLE0*	Load/Insert	6K	If PROCOPT=L or I
		+2K	If HD
		+2K	If HS
DLD00*	Delete/Replace	4K	If HS,
		4K	If HD, PROCOPT=R with or without LR and HS
		+2K	If SI
		+2K	If VLS
		8K	If HD, PROCOPT=D with or without SI and HS
		+4K	If LR used
		8K	If HD, PROCOPT=D, R with or without SI and HS
		+4K	If LR used
		+2K	If VLS used
QUEF0	PI Queueing Facility	4K	If PI used
DSEH0	Workfile Generator	26K	If PROCOPT=L and LR or SI used
RDBL0	DB Logger (DL/I) (see note 1)	6K	If logging to DL/I
RDBL1	DB Logger (CICS/VS)	4K	If logging to CICS/VS
BNUC0	Batch Nucleus	4K	If batch (not MPS)
ODP	CICS-DL/I Interface (see note 2)	14K+	If online
BPC00	MPS Batch Partition Controller (see note 3)	2K	1/active MPS batch partition
MPI00	MPS Batch Nucleus	6K	If MPS batch
STRB0	Batch Field Storage Manager	1K	If Boolean

Notes:
1. This module includes a 1K log buffer.
2. A large ACT will increase the size of the module.
3. In addition, this transaction will require a ICA/transaction-work-area with a real storage requirement of 2K.

HS = Hierarchical Sequential access method
HD = Hierarchical Direct access method
LR = logical relationships
SI = secondary indexing
VLS = variable length segments
PI = program isolation
* indicates modules eligible for placement in SVA

DL/I Control Blocks

Certain types of DL/I control block, such as PSBs and DMBs, are frequently referenced during DL/I execution. For this reason, their sizes must be included in the real storage requirements calculations. Any of the following methods will give values for the sizes of PSBs and DMBs:

- Use the size from the link-edit map following execution of the the ACB utility (DLZUACB0)
- Use the sizes shown in a DSERV of the core-image library
- Use a value of about 2K for each DMB and PSB.

Each group of 64 buffers in the DL/I buffer pool requires a 2K buffer prefix control block. Divide the total number of buffers in the buffer pool (not the number of subpools) by 64 and round up. Multiply this value by 2K. Add the product to the real storage requirements estimate.

Each scheduled online transaction requires a PST control block of 1,144 bytes. The required total PST storage for the DL/I peak load condition is 1,144 multiplied by the maximum number of DL/I tasks to be supported in a CICS/VS system (CMAXTSK). CMAXTSK is an ACT generation parameter. Count a PST for each MPS batch partition, since each one has an online transaction scheduled to support it.

When program isolation (PI) is used, queue space is required for the queue elements created as a byproduct of application program data base calls. PI queue space is acquired in 2K increments.

Queue space requirements for typical CICS/VS-DL/I systems are in the range of 2K to 4K. However, long-running MPS batch programs that don't issue CHECKPOINT commands can require much more queue space.

The basic DL/I control blocks; such as the SCD, PPST, PDIR, and RPDIR; are part of the DL/I nucleus modules and should not be counted here.

DL/I Buffer Pool

The buffers in the subpools are DL/I I/O areas and are frequently referenced. DL/I's buffer manager tends to distribute I/O activity evenly across all buffers in a subpool. Therefore, the size of all active subpools must be considered the normal reference set of the buffer pool.

The size of each buffer subpool equals the number of buffers specified (2 - minimum, 32 - maximum and default) multiplied by the size of the largest CI of the DMBs assigned to that subpool.

Compute DL/I buffer pool requirements for HD data bases only, not for HS. SHISAM, HISAM and INDEX data bases do not use the DL/I buffer pool except as work areas during load, insert, or delete operations. The I/O for these data bases is performed in the VSAM buffers.

VSAM Action Modules

The real storage requirement for KSDS and ESDS processing is normally 18K to 22K. If a data set is being loaded or a CA split is taking place this will increase to 32K to 44K. If VSAM is not in the SVA, these figures generally increase by about 2K due to poor page alignment within the partition.

VSAM Control Blocks

The real storage requirement for the VSAM control blocks necessary to support an ESDS in DL/I is 4K. For a KSDS it is 6K. Figure B-2 shows the VSAM control block requirements for each type of DL/I data base. Sum the VSAM control block requirements for all active data bases and add this number to the real storage requirement calculation.

Figure B-2. DL/I VSAM Control Block Requirements

DL/I Access Method	VSAM Control Block Requirements/DMB	Normal Reference Set per DMB
HD randomized or HDAM	1 ESDS	4K
HD indexed or HIDAM	1 ESDS = 1 KSDS	10K
INDEX (SI)	1 KSDS	6K
HISAM	1 KSDS + 1 ESDS	10K
SHISAM	1 KSDS	6K
DISK LOG	1 ESDS	4K

Each operating system partition using a GETVIS area (a result of specifying “SIZE” on the EXEC statement) has an anchor table at the beginning of the GETVIS area. VSAM requires a GETVIS area. The real storage requirement for the anchor table is 2K. This should be included in the real storage requirements calculation when VSAM is used.

VSAM Buffers

DL/I uses VSAM buffers for all KSDS I/O and its own buffer pool for HD ESDS I/O. No VSAM buffers are used for DL/I disk logging. HISAM ESDS I/O is performed out of the VSAM buffers. The real storage requirements of the VSAM buffer pool for each type of DL/I access method can be determined from the following equations.

Note: Although VSAM buffers are used for all KSDS and some ESDS processing by DL/I, DL/I does not use the Local Shared Resources facility of VSAM.

HD ESDS = 0K (DL/I buffer pool is used)
HS ESDS = (BUFND) x (data CI size)
All KSDS = (BUFND) x (data CI size) +
(BUFNI) x (index CI size)

The default BUFND (number of data buffers) and BUFNI (number of index buffers) values for DL/I are 2 and 3, respectively. These values can be overridden via the DL/I card for batch DL/I programs (non MPS), ACT specification for online and MPS batch programs, and the job control DLBL statement.

After calculating the real storage requirements of the VSAM buffers for each data set, round each data set's buffer requirements to a page boundary before adding it to the total real storage requirements figure. For example, a KSDS with .5K index CI size and 2K data CI size requires 5.5K of buffer space from equation (3) above. This should be rounded up to 6K.

Application Programs

The real storage requirements for each application program must also be estimated. For batch application programs an approach similar to that taken for VSAM and DL/I can be used: calculate the requirements for executable code, control blocks (DTFs), and I/O areas separately, then total these numbers. Note that the DL/I language interface normally links at the end of the application program. This may add 2K to the program's real storage requirements.

The real storage requirements for online application programs can be estimated by considering them in two parts: code and transaction storage. Since online programs tend to be small, their normal reference set is usually equal to their virtual address size. When estimating online program sizes, be sure to include the size of all BMS maps used by the programs. If CICS/VS statistics are available, an “average” program's size can be derived by using the PPT statistics and a DSERV of the core-image library.

Normally an online application program's transaction storage will be in the range of 2K to 6K.

Once you have computed the size of the average online application program (code + transaction storage), multiply this by the CMAXTSK value to determine the real storage requirements for peak DL/I load conditions.

DL/I Real Storage Requirements Example

The following DL/I online environment is assumed for the calculation of real storage requirements in this example:

- One active MPS partition
- Two HIDAM data bases with 2K data CI size, .5K index CI size
- One HDAM data base with 2K data CI size
- One subpool per data base, 6 buffers per subpool
- Default VSAM buffer specifications (3 index, 2 data) for HIDAM KSDSs
- Logical relationships
- DL/I logging to CICS/VS journal
- Program isolation.

System Component	Real Storage Requirement
DL/I Action Modules	
Call Analyzer	6K
Batch Field Storage Manager	1K
Retrieve	29K
Buffer Handler	10K
Index Maintenance	4K
Space Management	8K
Load/Insert	8K
Delete/Replace	12K
PI Queueing Facility	4K
DB Logger (CICS/VS)	4K
CICS/VS-DL/I Interface	14K
Batch Partition Controller	2K
MPS Batch Nucleus	6K
DMBs (2-INDEX, 2-HIDAM, 1-HDAM)	10K
PSBs (1/DB @ 2K each)	6K
Buffer Pool Prefix	2K
PSTs (4 @ 1,144 bytes each)	6K
PI Queue Space	2K
Buffer Pool (6/DB @ 2K each)	36K
VSAM	
Action Modules	18K
Control Blocks	
1 HDAM DB	4K
2 HIDAM DBs	20K
Buffers	
2 HIDAM KSDS	12K
	224K

Appendix C. Example of Storage Requirements for DL/I Documentation Aid

The provided job stream (A.DLZDATAB) acquires two public DBSPACEs for the DL/I DA SQL/DS tables. The size (PAGES =) of the DBSPACE acquired in this job stream example may exceed, or be too small, for your installation. The actual size depends on the number of, and the complexity (number of SEGM, LCHILD, FIELD, etc. statements) of the DBD and PSB definitions to be stored in the tables. In most cases the size of the DBSPACE specified in the job stream should be sufficient.

You can determine your actual DBSPACE requirements by comparing the number of, and the complexity of, your DBDs and PSBs in your installation to the number and definitions of the DBDs and PSBs described in this example to acquire DBSPACE.

The specifications for the DBSPACE in this example are:

- NHEADER = Default value of 8 header pages is taken. (Each header page requires 2K bytes.)
- PAGES = 512 pages is specified for this DBSPACE. (Each page is 4096 bytes.)
- PCTINDEX = Default of 33% of the DBSPACE is taken.
- PCTFREE= 18% of each page to be kept empty when data is inserted into the DBSPACE.
- LOCK = DBSPACE lock size is specified.
- STORPOOL = Not specified. DPSPACE to be acquired from any storage pool.

For further information about acquiring DBSPACE refer to the *SQL/DATA System Planning and Administration* manual. For further information about the ACQUIRE DBSPACE command refer to *SQL/Data System Application Programming* manual.

Estimating DBSPACE Size by Comparison to the Example

Data Base Descriptions

Based on the properties specified for the DBSPACE described previously, approximately ten complete sets of the following DL/I Data Base Descriptions can be stored before the space is exhausted for the DBD information.

1. Four physical data bases:
   - Data base description 1:
     - Segments (Number of SEGM statements):.....9
     - Secondary Indexes (Number of LCHILD/XDFLD statements):.....11
     - Logical relationships (Number of LCHILD statements):.....1
     - Fields (Number of FIELD statements):.....189
   - Data base description 2:
     - Segments (Number of SEGM statements):.....8
     - Primary Index (Number of LCHILD statements):.....1
     - Secondary Indexes (Number of LCHILD/XDFLD statements):.....1
     - Logical relationships (Number of LCHILD statements):.....2
     - Fields (Number of FIELD statements):.....136
   - Data base description 3:
     - Segments (Number of SEGM statements):.....9
     - Logical relationships (Number of LCHILD statements):.....4
     - Fields (Number of FIELD statements):.....161
   - Data base description 4:
     - Segments (Number of SEGM statements):.....9
     - Primary Index (Number of LCHILD statements):.....1
     - Fields (Number of FIELD statements):.....162
2. Four logical data bases:
   - Data base description 1:
     - Segments (Number of SEGM statements):.....23
   - Data base description 2:
     - Segments (Number of SEGM statements:.....16
   - Data base description 3:
     - Segments (Number of SEGM statements:.....23
   - Data base description 4:
     - Segments (Number of SEGM statements:.....25

The total number of of statements in the above DBDs equal 792. Ten sets of these statements, therefore, equals 7920 statements.

Note: If additional space is necessary, three more completed sets can be stored if the PCTFREE space is used. The three additional sets total 2376 (792 X 3) statements.

Program Specification Block

Based on the properties specified for the DBSPACE described previously, approximately two complete sets of the following PSBs can be stored before the space is exhausted for the PSB information.

- Three (3) Program Specification Blocks
  - Program Specification Block 1:
    - Program Control Blocks (Number of PCB statements):.....12
    - Segments (Number of SENSEG statements):.....50
    - Fields (Number of SENFLD/VIRFLD statements):.....780
  - Program Specification Block 2:
    - Program Control Blocks (Number of PCB statements):.....13
    - Segments (Number of SENSEG statements):.....81
    - Fields (Number of SENFLD/VIRFLD statements):.....1469
  - Program Specification Block 3:
    - Program Control Blocks (Number of PCB statements):.....45
    - Segments (Number of SENSEG statements):.....90
    - Fields (Number of SENFLD/VIRFLD statements):.....614

The total number of statements in the above PSBs equals 3154. Three sets of these statements, therefore, equals 9462 statements.

NOTE: If additional space is necessary, one additional set can be stored if the PCTFREE space is used.

A comparison should be made to determine if this DBSPACE is sufficient for the DBDs and PSBs you installation will be storing in the DL/I DA SQL/DS tables. If more, or less, DBSPACE is desired, the PAGES parameter may be changed accordingly.

Estimating DBSPACE Size by Calculation

A more accurate approximation for the DBSPACE needed may be obtained using the formulas in Appendix C: *Data Base Storage Estimating, Section: Estimating DBSPACE Size* in the *SQL/Data System Planning and Administration* manual.

The average row lengths (in bytes) of the DL/I DA SQL/DS tables that describe the physical characteristics of the data base are described in the following table:

SQL/DS TABLE	BYTES	DESCRIPTION
DBDBASICDATA	158	DL/I DBDGEN DBD and DATASET statement information
DBDACCESSDATA	273	DL/I DBDGEN ACCESS and/or LCHILD and XDFLD statement information for primary and secondary indexes information
DBDSEGMENTDATA	197	DL/I DBDGEN SEGM statement information
DBDLCHILDDATA	98	DL/I DBDGEN LCHILD statement information for logical relationships
DBDFIELDDATA	98	DL/I DBDGEN FIELD statement information

The average row lengths (in bytes) of the DL/I DA SQL/DS tables that describe the application programs and their use of logical data structures are described in the following table:

SQL/DS TABLE	BYTES	DESCRIPTION
PSBBASICDATA	60	DL/I PSBGEN PSBGEN statement information
PSBPCBDATA	92	DL/I PSBGEN PCB statement information
PSBSEGMENTDATA	85	DL/I PSBGEN SENSEG statement information
PSBFIELDDATA	165	DL/I PSBGEN SENFLD and/or VIRFLD statement information

Running Out of DBSPACE

If, while running the Application Control Blocks Creation and Utility (ACBGEN), a SQL/DS error occurs with a SQLCODE returned of -701, the DBSPACE is full. To solve this problem, you should check the PCTFREE value for the DBSPACE (initially set at 18% for each page). If this value is greater than 0, which indicated PCTFREE space is available, it may be reduced to a smaller value by using the ALTER DBSPACE command. This allows more data to be inserted in the DBSPACE.

If the PCTFREE value is 0, a new DBSPACE must be created. The UNLOAD and RELOAD commands of the Data Base Services Utility will transfer the information to the new, and larger, DBSPACE. Refer to the *SQL/Data System Application Programming* manual for more details.

Appendix D. A Recommended Naming Convention

This appendix summarizes a naming convention recommended in the *Standards Manual for DOS/VSE*.

The *Standards Manual for DOS/VSE* describes and recommends an installation-wide naming convention that covers all possible situations in the DOS/VSE environment. Those parts of the convention that are most significant to you in your work are summarized here.

ACRONYM	DESCRIPTION	VALID CONTENTS
S	General system	Alpha
PP	Project code	Alpha
f	Frequency of run	Numeric 0-daily 4-yearly 1-weekly 5-as needed 2-monthly 6-one time job 3-quarterly
J	Job	Alpha-assigned in sequence A-Z. If more than 26 jobs/project, divide into two projects.
n	Program number	Numeric - in sequence within job 0-9. If only one program/job, then it is always 0.
mm	Phase number	Numeric - in sequence 00-99.
r	Filename sequence number or Procedure component sequence number	Numeric - in sequence 0-9. Numeric - in sequence 0-9.
d....d	Descriptive file name	Alphameric - 1-8 character description of job

Job Name

```
     Job                          SPPfJ

EXAMPLE

     Description                  Acronym     Content
     General system               S           P
     Project code                 PP          CK
     Frequency of run             f           1
     Job                          J           A
```

Putting this all together, the job name is PCK1A.

Program Name

The program name is the phase name used at link-edit time to catalog the phase into a core image library. It is also the term generally used when referencing this program for any reason.

```
     Program                      SPPfJn00

EXAMPLE

     Description                  Acronym     Content
     General System               S           P
     Project Code                 PP          CK
     Frequency of run             f           1
     Job                          J           A
     Program                      n           1
     Constant                     00          00
```

Putting this all together, the program name is PCK1A100.

Phase Name

The phase name is the same as the program name unless there are multiple phases in a program. In that case, the last two digits are used to designate the phase number. The root or first phase is always the same as the program name. It always ends in “00.”

```
     Phase                        SPPfJnmm

EXAMPLE

     Description                  Acronym     Content
     General System               S           P
     Project Code                 PP          CK
     Frequency of run             f           1
     Job                          J           A
     Program                      n           1
     Phase number                 mm          01
```

Putting this all together, the phase name is PCK1A101. Since the last two digits are not “00,” you know that this is not the root phase of this program. The root phase would be named PCK1A100.

Additional Conventions for DL/I

All data base jobs and programs will utilize the convention described, but a modification is required to define names that identify data base related components such as DBDs, PSBs, segments, fields, filenames, and file-IDs. (DL/I terms such as DBD and PSB are defined in the glossary.)

The first three positions of these names (ZBB) will identify them as data base components. The “Z” is a constant that is only used for data bases. The “BB” is unique for each data base and will identify all elements addressing that data base.

The fourth position of the names (ZBB*C*) is a category code to identify the specific element being named. The contents of this position are as follows:

C Real logical child segment
D DBD or filename or file-ID
F Field
P PSB
Q Sequence (key) field
S Segment
U Duplicate data field in index segment
V Virtual logical child segment
W Destination parent segment
Z User data field in index segment.

The fifth thru eighth positions of a name (ZBBCXXXX) will vary depending on the type of element. The values for these positions are specified in the following sub-sections.

DBD

- ZBBDttt -

  where
  ttt=
  DBP Physical DBD
  DBL Logical DBD
  XnP Index DBD (n is 0 thru 9)

Example: Physical DBD for the Personnel/Payroll data base in the Administration System - ZPRDDBP.

Note: The DBD name can be one to seven alphameric characters. The at-sign (@) must not be used.

Filename

- ZBBDvvv -

  where
  vvv=
  DBC Cluster for physical DBD
  DBI Index for physical DBD
  DBD Data for physical DBD
  XnC Cluster for index DBD (n is 0 thru 9)
  XnI Index for index DBD (n is 0 thru 9)
  XnD Data for index DBD (n is 0 thru 9)

Example: Filename for the VSAM cluster of a physical DBD - ZPRDDBC.

Note: The filename can be one to seven alphameric characters.

File-Id

- ZBBDvvv0.d...d -

  where
  vvv=
  Same as filename value
  and
  d...d=
  Alphabetic description of file content

Example: VSAM cluster name for physical DBD - ZPRDDBC0.HIDAM.

PSB

- ZBBPnnn -

  where
  nnn=
  Numeric sequence within a data base

Example: ZPRP003.

Note: The PSB name can be one to seven alphameric characters. The at-sign (@) must not be used.

Segment

- ZBBSGddd -

  where
  Gddd=
  G—Alphanumeric segment identifier for this segment that is unique within a data base.
  ddd—Alphabetic abbreviation describing contents of this segment type. This is project/content oriented rather than dependent on the physical structure.

Example: The tax segment of the Personnel/Payroll data base - ZPRSTTAX.

Note: The segment name can be one to eight alphameric characters.

Field

- ZBBFGddd -

  where
  Gddd=
  G—Same segment identifier as segment in which field occurs.
  ddd—Alphabetic abbreviation describing content.

Example: The number-of-exemptions field in the tax segment - ZPRFTEXS.

Note: The field name can be one to eight alphameric characters.

Appendix E. Controlling the DL/I Online Systems Environment

This appendix describes the methods of controlling the DL/I online system environment. It describes the method of control using system calls in detail.

The online system environment can be controlled using system generation parameters to establish the initial configuration of the online system. Dynamic services (CICS/VS master terminal function) can be used for adjusting the system configuration.

The DL/I online system can also be controlled in various other ways. By specifying a unique DL/I online nucleus during CICS/VS system initialization (SIT parameter DL1=suffix), different configurations of the DL/I online system may be selected. By adjusting the parameters within the online nucleus generation, buffer storage allocation (BFRPOOL), program storage allocation (SLC), and DL/I system loading (MAXTASK and CMAXTSK) may be controlled.

Note: An on-line test program that you may find useful as a debugging aid is documented in the *DL/I DOS/VS Diagnostic Guide*. The program, DLZMDLI0, accepts DL/I system calls and some special calls, and displays the results on a screen. You should also note that the high level programming interface (HLPI) does not support system calls. Also, system calls may not be issued from DL/I MPS batch or DL/I batch programs.

Directly related to system performance is the number of tasks allowed to run concurrently within the online system. Optimum performance requires a detailed knowledge of the available system resources and each task's usage of these resources.

System resources consist of dynamic storage acquired from the CICS/VS storage pool for the DL/I PST. Additionally, if read-only tasks are being scheduled, the storage requirements are increased by the size of the PSB being scheduled plus its index work areas.

The deferred-open option of the DL/I entry in the CICS/VS file control table has the effect of reducing the time required for system initialization. However, if the data bases are required at a later time, invoking the dynamic-open option causes degradation of the system response due to the non-asynchronous DL/I open function. During DL/I open/close the partition loses control for the duration of the request and no task dispatching occurs. The CSMT transaction cannot be used to open data bases for which the deferred-open option has been specified. They must, instead, be opened by using a DL/I system call with the function STRT, as explained below.

A third possibility exists to control the DL/I system in an on-line environment through the use of special DL/I calls. These system calls may be issued from a special application program and allow for a more dynamic control than that provided by the methods previously described.

Note: Utilization of the system calls should be kept under control and, if possible, be restricted to one program since these calls are specific to DL/I DOS/VS and are not supported by IMS/VS. Unrestricted use could therefore make a potential migration to CICS/OS/VS with DL/I interface more difficult.

The system calls provide the ability to adjust the current maximum task value (CMXT), to start and open a DBD (STRT), and to stop and close a DBD (STOP). In addition, the tracing facility can be controlled by two system calls, TSTR (trace start) and TSTP (trace stop). Refer to the *DL/I DOS/VS Diagnostic Guide* for a description of the tracing facility. While the CMXT call may be used primarily for the purpose of balancing the DL/I online system load, the STRT and STOP calls may typically be used for closing a data base previously stopped by DL/I because of an I/O error. The user may then perform an off-line recovery procedure before making the data base available again for online processing. Great care should be exercised when invoking the open/close functions due to the non-asynchronous execution of the VSAM open/close function. When the data base calls are issued, the online system is not dispatched until the open/close function is completed. This has an adverse effect on the teleprocessing system and therefore the timing of the DL/I STRT and STOP calls should be carefully considered.

In addition, the STOP call terminates scheduling for all PSBs that reference the DBD being stopped. Those tasks that are actively scheduled on any affected PSBs are allowed to continue until they are unscheduled through either the DL/I TERM call or end-of-task processing.

DL/I System Call Formats and Returns

The DL/I system calls are only supported for CICS/VS online programs written in Assembler language. These programs must use the DL/I CALLDLI macro interface. The general structure of the call is:

```
CALLDLI  ASMTDLI,(function,parameter-list)
```

function

```
is the name of the 4-byte field containing CMXT,
STRT, STOP, TSTR, or TSTP.
```

parameter-list

```
is the name of a 64-byte field containing
function parameters and the interface work area.
```

Note: For information about the Call interface and Assembler coding in the DL/I environment, see *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces*.

The following is a description of the parameter requirements and return conditions for each call.

CMXT Call: Parameter requirements

```
bytes 0-1 of the parameter list contain
the requested new value for current maximum task
in packed decimal format.
```

Return normal

```
TCAFCTR contains X'00'
bytes 0-1 of parameter list are unchanged
bytes 2-3 of parameter list contain the previous value of
current maximum task.
```

Return abnormal

```
TCAFCTR contains X'08'
this indicates that the requested value was negative, zero,
or exceeded the MAXTASK specification in the ACT.
```

STRT Call: Parameter requirements

```
bytes 0-7 contain the DMB name, consisting of the DBD
generation name extended to seven characters with at-signs (@),
if necessary, and with the alphabetic character D as the eighth
character.
```

Return normal

```
TCAFCTR contains X'00';
bytes 8-11 contain the value of register 15 returned by
VSAM open/close,

bytes 12-15 contain the address of the ACB for this data
base (if HISAM contains HISAM KSDS),

bytes 16-19 contain the address of the ACB for the
ESDS if the data base organization is HISAM.
```

Return abnormal

```
TCAFCTR contains X'01'; data base previously started
TCAFCTR contains X'02'; DBD is unusable
TCAFCTR contains X'03'; TESTCB failed
TCAFCTR contains X'08'; DBDNAME is invalid.
```

STOP Call: Parameter requirements

```
bytes 0-7 contain the DMB name, consisting of the DBD
generation name extended to seven characters with at-signs
(@), if necessary, and with the alphabetic character D as
the eighth character.
```

Return normal

```
TCAFCTR contains X'00';
bytes 8-11 contain the value of register 15 returned by
VSAM open/close,

bytes 12-15 contain the address of the ACB for this
data base (if HISAM contains HISAM KSDS),

bytes 16-19 contain the address of the ACB for the
ESDS if the data base organization is HISAM.
```

Return abnormal

```
TCAFCTR contains X'01'; data base previously stopped
TCAFCTR contains X'02'; DBD is unusable
TCAFCTR contains X'03'; TESTCB failed
TCAFCTR contains X'08'; DBDNAME is invalid.
```

Note: It is the user's responsibility to verify the contents of the return code from VSAM open/close stored in bytes 8-11 of the parameter list. If an error has occurred, the user may use the VSAM SHOWCB macro using the ACB(s) addresses returned.

TSTR Call: Parameter requirements

```
bytes 0-7 contain the name of the trace module to be used.
This program name must have been placed in the processing
program table (PPT) prior to execution of CICS/VS.
```

Return normal

```
TCAFCTR contains X'00'.
```

Return abnormal

```
TCAFCTR contains X'01'; tracing already active
TCAFCTR contains X'02'; trace module not found
TCAFCTR contains X'04'; GETMAIN failed for trace table.
```

TSTP Call: Parameter requirements

```
none; the work area must, however, be provided.
```

Return normal

```
TCAFCTR contains X'00'.
```

Return abnormal

```
TCAFCTR contains X'01'; tracing was not active.
```

Scheduling the DL/I System Call

To provide the system programmer with a method of controlling the use of the DL/I system calls, a special scheduling call must be issued prior to invoking them. If a system call is issued with an invalid password, the violating task is abnormally terminated with the abend code DLPV.

The following is the format of the call and an explanation of its return codes:

```
CALLDLI  ASMTDLI,(function,psbname,password(,uibparm))
```

function

is the name of a field containing the 4-byte constant 'PCBb'

psbname

is the name of a field containing the 8-byte constant 'SYSTEMDL'

password

is the name of a field containing an 8-byte constant equal to the password generated during the online nucleus generation.

uibparm

is the address of a fullword in which DL/I returns the address of the user interface block (UIB).

Upon return from the call, the field TCAFCTR or UIBFCTR (if the UIB is used) contains a 1-byte return code indicating the following:

X'00'

indicates the task may proceed with the system calls.

X'08'

indicates the system call interface is active. Only one task may be active on this interface at a time.

To terminate activity on the system call interface, the standard DL/I termination call may be used (TERM).

DL/I System Call Examples

CMXT Call Example

Using the CMXT call, the following shows a method of modifying the current maximum task value.

```
MODCMXT   CSECT
            •
            •
          CALLDLI ASMTDLI,(SCHED,SYSPCB,PASSWORD)   SCHEDULE SYSTEM CALL
          CLI     TCAFCTR,0           DID SCHEDULING SUCCEED
          BNE     SCHEDERR            IF NOT GO PROCESS ERROR
            •
            •
          DFHSC   TYPE=GETMAIN,       GET PARAMETER STORAGE        *
                  NUMBYTE=72,                                      *
                  CLASS=USER

          L       SAACBAR,TCASCSA     FIND PARAMETER STORAGE
          MVC     NEWCMXT,=PL2'5'     SET TO MODIFY CMXT TO 5

          CALLDLI  ASMTDLI,(FUNCTION,NEWCMXT)  ISSUE CMXT SYSTEM CALL

          CLI      TCAFCTR,0          ENSURE NORMAL COMPLETION
          BNE      CMXTERR            IF NOT CHECK RETURN CODE
            •
            •
            •

SCHEDERR  DS       0H                 PROCESS SCHEDULING ERROR
            •
            •
            •
SCHED     DC       CL4'PCB'           SCHEDULING CALL CONSTANT
SYSPCB    DC       CL8'SYSTEMDL'      SPECIAL SYSTEM SCHEDULING REQUEST
PASSWORD  DC       CL8'MYPASS'        PASSWORD SPECIFIED IN NUCLEUS
            •
            •
FUNCTION  DC       CL4'CMXT'          MODIFY CURR MAX TASK VALUE REQ
          COPY     DFHSAADS
NEWCMXT   DS       PL2                VALUE CMXT IS TO BE ADJUSTED TO
OLDCMXT   DS       PL2                PREVIOUS VALUE OF CMXT
          DS       CL60               WORKAREA FOR CALL PROCESSOR
            •
            •
            •
```

STRT and STOP Call Example

The following illustrates a method of issuing a STRT open DBD system call. The STOP close DBD system call is the same as this except that the FUNCTION DC is coded FUNCTION DC CL4'STOP'.

```
MODDBD    CSECT
           •
           •
           •
          CALLDLI ASMTDLI,(SCHED,SYSPCB,PASSWORD)    ENABLE SYSTEM CALL
          CLI     TCAFCTR,0           DID SCHEDULING SUCCEED
          BNE     SCHEDERR            IF NOT, PROCESS ERROR
           •
           •
           •
          DFHSC   TYPE=GETMAIN,       GET PARAMETER STORAGE          *
                  NUMBYTE=72,                                        *
                  CLASS=USER
           •
           •
           •
          MVC     DMBNAME,=CL8'DMB1@@@D'  SET DMB NAME FOR STRT CALL

          CALLDLI ASMTDLI,(FUNCTION,DMBNAME)  ISSUE STRT CALL TO SYSTEM
           •
          CLI     TCAFCTR,0               IF OPEN NOT ISSUED
          BNE     STRTERR                 CHECK WHY NOT
           •
           •
           •
          L       WORK2,VSAMRET           GET OPEN R15 FOR VSAM RETURN
          LTR     WORK2,WORK2             TEST OPEN RETURN CODE
          BNZ     OPENERR                 DO SHOWCB IF OPEN BAD
           •
           •
           •
SCHED     DC      CL4'PCB'                SCHEDULING CALL CONSTANT
SYSPCB    DC      CL8'SYSTEMDL'           SPECIAL SYSTEM SCHEDULING REQ
PASSWORD  DC      CL8'MYPASS'             PASSWORD AS SPECIFIED IN NUC
           •
           •
FUNCTION  DC      CL4'STRT'                 START/OPEN DATA BASE REQUEST

          COPY    DFHSAADS
DMBNAME   DS      CL8                       DMB TO BE MODIFIED
VSAMRET   DS      F                         OPEN REGISTER 15 RETURN CODE
ACBADR1   DS      F                         POINTER TO ACB
ACBADR2   DS      F                         POINTER TO SECOND ACB IF HISAM
```

TSTR and TSTP Call Example

The following shows a method of issuing a TSTR call to start tracing. The TSTP call is the same except that no module name is required and that the FUNCTION DC is coded FUNCTION DC CL4'TSTP'.

```
STTRC     CSECT
           •
           •
           •
          CALLDLI ASMTDLI,(SCHED,SYSPCB,PASSWORD)    ENABLE SYSTEM CALL
           •
           •
          DFHSC   TYPE=GETMAIN,       GET PARAMETER STORAGE          *
                  NUMBYTE=72,                                        *
                  CLASS=USER
          L       SAACBAR,TCASCSA     FIND PARAMETER STORAGE

          MVC     MODNAME,=CL8'TRACEMOD'   SET MODULE NAME

          CALLDLI ASMTDLI,(FUNCTION,MODNAME)   ISSUE CALL TO SYSTEM
           •
          CLI     TCAFCTR,0           ENSURE NORMAL COMPLETION
          BNE     TRCERR              IF NOT, CHECK RESPONSE
           •
           •
           •
SCHED     DC      CL4'PCB'              SCHEDULING CALL CONSTANT
SYSPCB    DC      CL8'SYSTEMDL'         SPECIAL REQUEST
PASSWORD  DC      CL8'MYPASS'           PASSWORD SPECIFIED IN NUCLEUS
           •
           •
           •
FUNCTION  DC      CL4'TSTR'             START TRACE REQUEST

          COPY    DFHSAADS

MODNAME   DS      CL8                   TRACE MODULE NAME
          DS      CL56                  WORKAREA
           •
           •
           •
```

Appendix F. Incompatibilities Between DL/I and IMS

This appendix describes incompatibilities between DL/I and IMS.

The following incompatibilities are known to exist between DL/I and IMS.

INCOMPATIBILITY	RESOLUTION
1. Log Files	User must process the log file(s) in the creating system.
2. Image Copy Files	User must run Image Copy under system he will also recover under.
3. Index Format	User must specify DOSCOMP in OS DBDGEN.
4. Control Interval Format	User must specify IMSCOMP in DOS DBDGEN when going from IMS to DL/I
5. Field Level Sensitivity	Basic support is compatible. If DL/I extensions are used, changes may be required to application programs and data base definitions.
6. HLPI	Supported only if using IMS with CICS/VS. Otherwise, user must rewrite application programs in CALL interface.
7. HD Unload in DOS/ HD Reload in OS	Only one header record on DL/I Unload - requires specification of BLKSIZE=13000 and LRECL=6500 on OS DD card.
8. HISAM Unload in DOS/ HISAM Reload in OS	User must use HD Unload/Reload.
9. HD or HISAM Unload OS/Reload DOS	Function not supported.
10. Unload/Reload Disk Files	Tape files must be used across systems.
11. CHKP CALL Parameters	User must change CHKP CALLs in all programs. If the MPS Restart facility was used, all VSE checkpoints must also be removed.
12. MODEL=3330-II	User must change data base definition for data bases on this device.
13. RPG-II Support	Not supported in IMS. User must rewrite RPG applications in another language.
14. FBA devices	Device type not supported by IMS. User

	must switch to different device types.
15. Default PSB on Scheduling CALL	DL/I default PSB name on scheduling CALL is name of first PSB associated with application in DL/I ACT. Default name for IMS is the name of application associated with this task in CICS/VS PCT.
16. CALLDLI MF=E Assembler Language DL/I programs	Not supported in IMS.
17. IMF and IUG	Not supported in IMS.
18. Disk Logging	Not supported in IMS. When IMS is used in conjunction with CICS/OS/VS, the user can log data base modifications to the CICS/VS system journal which can exist on disk. However, the IMS utilities that process log files do not support disk input. In order for the IMS utilities to process a disk CICS/VS system journal, the journal must be transferred to tape.
19. ACT	Not supported in IMS. Similar function provided by the CICS/VS generation macro DFHDLPSB.
20. NI Status Code	DL/I requires the user to manually correct a data base following an NI status code. IMS automatically fixes this error condition.
21. CI Sizes	VSE/VSAM supports use of CI sizes not available with OS VSAM. If a non-supported CI size is used, then the data base must be unloaded, redefined, and reloaded when migrating to IMS.
22. ACCESS Macro	Not supported in IMS. DL/I DBD's must be re-coded using the DL/I equivalent definitions supported by IMS (LCHILD, XDFLD, NAME=/SK, /CK).
23. Extended Remote PSB	Not supported in IMS. User must synchronize local and remote data base activity.
24. Selective Unload	Not supported in IMS. User must write an application to simulate function.
25. CMF Hooks (DL/I PA II extensions)	Not supported in IMS.
26. Run and Buffer Statistics	Not supported in IMS.
27. Checkpoint Option for HD Unload	Not supported in IMS.

28. Rewind Option	Not supported in IMS, but available through OS Job Control Language.
29. PL/I Options	Not supported in IMS.
30. Documentation Aid	Not supported in IMS.
31. MPS Restart	Not supported in IMS.
32. Variable-Length Index Source Segments	Not supported in IMS.

Appendix G. Randomizing Modules and DL/I User Exit Routine Interfaces

This appendix gives information about randomizing modules for the HD randomized access method (with examples), and describes the DL/I user exit routine interfaces.

Randomizing Modules for HD Randomized Data Bases

The DL/I HD randomized access method requires that you supply a randomizing module for generating the addresses of the data base records. It does this by using an algorithm to convert the root key value of the root segment of each record into a relative block number and anchor point number. One or more of these modules can be used within your DL/I system. However, a particular data base has only one randomizing module associated with it. A generalized module, which uses parameters supplied during DBD generation to perform randomizing for a particular data base, can be written to service several data bases.

After a randomizing module has been compiled and tested, and before its use by the DL/I system, it must be cataloged and link-edited into a core image library. Each randomizing module must have a unique name that does not conflict with the name of an existing module in the core image library.

The name given to the load module used for randomizing functions with a specific data base should also appear in the DBD generation associated with the data base.

All randomizing modules are loaded from a core image library during DL/I initialization. DL/I initialization obtains the name of the randomizing module from the direct algorithm communication table (DACT) within the DMB. This block is constructed by the application control blocks creation and maintenance utility from parameters specified in the associated DBD.

When an application program issues a GET UNIQUE, a qualified GET NEXT, or an INSERT request that operates on a root segment of an HD randomized data base, your randomizing module is invoked. In some cases, a DELETE request can also result in a call to the randomizing module. The segment selection argument and the segment I/O work area in the data base request, which relate to the sequence field of a root segment, provide the primary input parameters to the randomizing module.

If the data base request consists of a root segment insert, the segment selection argument consists of only the segment name. In this case, the field value is obtained from the segment I/O area provided in the insert request. (For additional information, see *DL/I DOS/VS Application Programming: High Level Programming Interface* or *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces*.

The field value parameter is supplied to the randomizing module for conversion to a relative block number and anchor pointer number within the data base. In addition

to the field value parameter supplied by the application program, parameters from the DBD generation associated with the data base being used are available to the randomizing module in the DACT. The DACT is contained in the DMB, and its address is passed to the module each time a conversion is requested.

When a randomizing module is invoked for the purposes of conversion, control is passed from the DL/I data base retrieve function module, DLZDLR00.

The following DSECT defines the format of the DACT:

```
DMBDACS    DSECT
DMBDANME   DS    CL8   NAME OF ADDR ALGORITHM LOAD MODULE
DMBDAKL    DS    0CL1  EXECUTABLE KEY LENGTH OF ROOT
DMBDAEP    DS    A     EP OF ADDR LOAD MODULE
DMBDASZE   DS    H     SIZE OF THIS DSECT
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 OVERFLOW 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. A save area address is provided in register 13. Upon return to DL/I, registers must be restored. The register contents on entry are as follows:

```
Register  Meaning or Content

0         DMB address.
1         Address of the DACT.
7         PST address.
9         Address of first byte of key field value
          supplied by application program.
13        Save area address.
          The first three words in the
          save area must not be changed.
14        Return DL/I address.
15        Entry point address of randomizing module.
```

Internal DL/I control blocks that are of value to a randomizing routine are:

- The partition specification table (PST)
- The data management block (DMB)
- The physical segment description block (PSDB) for the root segment
- The first field-description block (FDB), which is the root segment key field format description.

The examples that follow later in this section show how DSECTs of these blocks can be assembled using the macro DLZIDLI.

The result of a randomizing module conversion must be in the form BBBR where:

```
BBB
   is the three-byte binary number of the block (VSAM control
   interval) into which a root segment is to be inserted
   or from which it is to be retrieved.
R
   is the one-byte binary number of the appropriate anchor
   point within a relative block (VSAM control interval)
   within an HD randomized ESDS.
```

This result must be placed in the CSECT addressed by register 1 in the four-byte fixed name DMBDACP. If the result exceeds the contents of the field DMBDABLK, the result is changed to the highest block and last anchor point of that block.

Notes:

1. With the exception of the DMBDACP field, DL/I internal control block fields inspected by the randomizing routine must not be modified.
2. The PST (partition specification table) contains the field PSTCNVB as an intermediate workfield for randomizing module use. The field is eight bytes long but not aligned on any boundary.

Randomizing Module Examples

The following four randomizing module examples are provided. You can use one of them, or use them to help you write one of your own. Each of the first three examples uses one of the following techniques:

- Modulo or division method
- Binary halving method
- Hashing method.

These three randomizing modules convert a root segment key field value to a relative block number and anchor point number in an HD randomized data base. The relative block number can range from 1 to 2^{24}-1. The anchor point number can range from 1 to 255.

The fourth example is a generalized randomizing module for any HDAM randomized data base. Its use is preferrable to the first 3 because of its level of sophistication. For this randomizer, the relative block number can range from 2 to 2^{24}-1. The anchor point number can range from 1 to 256.

Modulo or Division Method Example

Figure G-1 illustrates use of the modulo or division method for randomizing. This method is based on the principle that the remainder of a division can range only from 0 to the divisor minus 1. Thus, any number divided by 4 can only yield a remainder of 0, 1, 2, or 3. To determine the base location for a root segment, multiply the number of blocks in the root segment addressable area by the number of anchor points per block. This is effectively the number of base locations for root segments in the root segment addressable area. Then, divide the root segment key field value by the result of the multiplication. The remainder indicates the appropriate base location.

To convert the base location to relative block and anchor point number, divide the base location by the number of anchor points per block. This last division leaves the relative block number as the quotient and the anchor point number as the

remainder. Since both numbers are relative to zero both must be incremented by 1 to yield the correct block and anchor point.

Example:

Assume

```
a. root segment addressable area is 50 blocks
b. 2 anchor points per block
c. root segment key value is 23
```

Result

```
a. number of base locations = 50 x 2 = 100
b. appropriate base location = 23/100 = 23
   (the remainder)
c. appropriate block = 23/2 = 11 (the quotient),
   appropriate anchor point = 1 (the remainder)
d. adjust both numbers by one; thus, relative
   block = 12 and anchor point = 2
```

Notice that external keys 123, 223, 323, etc., are synonyms. As the number of base locations is increased, the distance between root segments increases. This may waste direct access space. However, the number of synonyms decreases as the number of base locations approaches or exceeds the largest key value. If the root segment key field value is numeric and the number of base locations equals or exceeds the largest key value, no synonyms are produced.

```
HDC1     TITLE 'DL/I, SAMPLE HDAM ADDRESS CONVERSION/RANDOMIZATION'
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                     *
*                S A M P L E   C O N V E R S I O N   P R O G R A M *
*                                                                     *
*             THIS CSECT CONVERTS AN EBCDIC NUMERIC KEY TO A RELATIVE*
*        BLOCK AND ROOT ANCHOR POINT. THIS RESULT IS OBTAINED AS      *
*        FOLLOWS  RECNO= MOD(KEY,DMBDABLK*DMBDARAP)                   *
*                 BLOCK= RECNO/DMBDARAP+1                             *
*                 RAP  = MOD(RECNO,DMBDARAP)+1                        *
*             THE CSECT ASSUMES THAT THE EXTERNAL KEY IS 9 BYTES OR   *
*        LESS.  NON-NUMERIC CHARACTERS ARE VALID, HOWEVER ONLY THE    *
*        FOUR LOW ORDER BITS WILL BE USED.                            *
*                                                                     *
*        CALLING SEQUENCE                                             *
*              R0 - DMB                                               *
*              R1 - DMBDACS                                           *
*              R7 - PST                                               *
*              R9 - KEY ADDRESS                                       *
*        ON RETURN                                                    *
*              DMBDACP - BBBR                                         *
*                                                                     *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
DLZHDC10 CSECT
         STM   14,12,12(13)        SAVE
         USING PST,R7
         USING DMBDACS,R1
         USING DLZHDC10,R15
         XC    PSTCNVB(8),PSTCNVB  INIT FOR CVB
         IC    R5,DMBDAKL          GET EXECUTABLE KEY FLD LENGTH
         EX    R5,PACK
         SR    R4,R4
         OI    PSTCNVB+7,X'0F'     FORCE SIGN
         CVB   R5,PSTCNVB
         L     R6,DMBDABLK         HIGHEST BLOCK NUMBER DIRECTLY ADDR
         MH    R6,DMBDARAP         HIGHEST RECORD NUMBER
         DR    R4,R6
         LR    R5,R4               RECNUM
         SR    R5,R4
         LH    R6,DMBDARAP
         DR    R4,R6
         LA    R4,1(,R4)           ROOT ANCHOR POINT
         LA    R5,1(,R5)           BLOCK
         SLL   R5,8
         OR    R4,R5               BBBR
         ST    R4,DMBDACP          RESULT
         LM    14,12,12(13)        RESTORE
         BR    R14                 RETURN
PACK     PACK  PSTCNVB(8),0(0,R9)  PACK KEY
         PRINT NOGEN
         DLZQUATE
         DLZIDLI   DMBBASE=0,PSTBASE=0
         END
```

Figure G-1 (Part 1 of 2). Randomizing Module - Modulo or Division Method

Notes:

1. Non-numeric keys are allowed only when the four low-order bits (4-7) of that character are numeric (0-9). If this restriction is not observed, a program check will occur at the CVB instruction.
2. If assembled with option ALIGN, the CVB instruction (PSTCNVB) causes error message IPK182 to occur. This message can be ignored, however, as the instruction will execute correctly.

Figure G-1 (Part 2 of 2). Randomizing Module - Modulo or Division Method

Binary Halving Method Example

Figure G-2 illustrates the use of the binary halving method. This method is an attempt to distribute root segments across the root segment addressable area according to the bit pattern of a root segment key field value after it has been converted to a binary value. This distribution is performed as follows: A result register is set to 0. After a key field value has been converted to binary, the number of base locations (number of blocks in the root segment addressable area times number of anchor points per block) is computed and divided by 2. The low-order bit of the converted key field value is tested. If it is equal to 1, the current number of base locations is added to the result register. If the low-order bit is 0, no addition to the result register is performed.

The number of remaining base locations is again divided by 2 and the quotient tested. If it is nonzero, the next higher bit position in the converted key field is tested for a 1 or 0 and the appropriate action taken. This process continues until the number of remaining base locations divided by 2 yields a quotient of 0. At this point, the appropriate base location is in the result register. To produce the proper relative block number and anchor point number, this value is divided by the number of anchor points per block. The division yields a quotient of relative block number and remainder of anchor point number. As in the previous module, the results are both relative to 0 and must be incremented by 1 to yield the appropriate values.

Example:

Assume

```
a. 10 blocks in root segment addressable area
b. 2 anchor points per block
c. root segment key field value of 29
```

After initialization:

```
Converted   No. of remaining   Result
key         base locations     register

1 1 1 0 1   (10x2)/2 = 10      0

After bit tested

. . . . x   10                 10
. . . x .    5                 10
. . x . .    2                 12
. x . . .    1                 13
```

At this point, the number of remaining base locations is reduced to 0. Hence the appropriate base location is 13. To get the actual relative block number and root anchor point, divide 13 by 2 and add 1 to both the quotient and the remainder to yield a relative block number of 7 and an anchor point number of 2.

Notice that the number of base locations determines when testing ceases. Hence, in this example, all key field values ending in the same four bits are synonyms. Additional bits of the key are tested when the number of base locations exceeds another power of two. If the number of base locations is not a power of two, some of the base locations will never be used.

The major advantage of this method is that the relative order of root segment placement is disturbed very little when the number of base locations is changed.

```
HDC2     TITLE 'DL/I, SAMPLE HDAM BINARY HALVING ADDRESS CONVERSION'
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                 B I N A R Y    H A L V I N G    C O N V E R T     *
*              THIS CSECT DETERMINES THE RELATIVE BLOCK AND ROOT     *
*        ANCHOR POINT BY A BINARY HALVING TECHNIQUE. THIS APPROACH   *
*        IS SLOWER THAN THE MODULO SCHEMES, BUT IT DOES TEND TO KEEP *
*        THE SAME PHYSICAL SEQUENCE WHEN THE NUMBER OF ADDRESSABLE   *
*        BLOCKS IS CHANGED. SINCE THE ROUTINE USES SHIFTS ON INTEGER *
*        NUMBERS, SOME RECORD NUMBERS WILL BE INACCESSIBLE IF THE    *
*        TOTAL NUMBER OF DIRECTLY ADDRESSABLE RECORDS (BLOCKS*ROOT   *
*        ANCHOR POINTS) IS NOT A POWER OF 2. THIS CSECT ASSUMES      *
*        AN INTERNAL KEY OF 9 BYTES OR LESS.                         *
*                                                                    *
*        CALLING SEQUENCE                                            *
*             R0 - DMB                                               *
*             R1 - DMBDACS                                           *
*             R7 - PST                                               *
*             R9 - KEY ADDRESS                                       *
*        ON RETURN                                                   *
*             DMBDACP - BBBR                                         *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
DLZHDC20 CSECT
         STM   14,12,12(13)
         USING PST,R7
         USING DMBDACS,R1
         USING DLZHDC20,R15
         XC    PSTCNVB(8),PSTCNVB  INIT FOR CVB
         IC    R5,DMBDAKL          GET EX KEY LENGTH
         EX    R5,PACK
         OI    PSTCNVB+7,X'0F'     FORCE SIGN
         CVB   R2,PSTCNVB
         L     R4,DMBDABLK
         MH    R4,DMBDARAP         HIGHEST RECORD IN RANGE
         SR    R5,R5               CLEAR RESULT REG
CVTLP    SRL   R4,1                CUT RANGE IN HALF
         LTR   R4,R4               RANGE EXHAUSTED
         BZ    XIT                 YES
         SR    R3,R3               NO
         SRDL  R2,1                TEST MASK FOR 1
         LTR   R3,R3
         BZ    CVTLP               NO ONE
         BXH   R5,R4,CVTLP         ONE - ADD IN RANGE
XIT      DS    0H
         LH    R6,DMBDARAP
         DR    R4,R6
         LA    R4,1(,R4)           ROOT ANCHOR POINT
         LA    R5,1(,R5)           BLOCK
         SLL   R5,8
         OR    R4,R5
         ST    R4,DMBDACP          RESULT
         LM    14,12,12(13)
         BR    R14
PACK     PACK  PSTCNVB(8),0(0,R9)  PACK KEY
         PRINT NOGEN
         DLZQUATE
         DLZIDLI   DMBBASE=0,PSTBASE=0
         END
```

Figure G-2 (Part 1 of 2). Randomizing Module - Binary Halving Method

Notes:

1. Non-numeric characters are valid in the internal key. However, only the four low-order bits are used. Non-numeric keys are allowed only when the four low-order bits (4-7) of that character are numeric (0-9). If this restriction is not observed, a program check will occur at the CVB instruction.
2. If assembled with option ALIGN, the CVB instruction (PSTCNVB) causes error message IPK182 to occur. This message can be ignored, however, as the instruction will execute correctly.

Figure G-2 (Part 2 of 2). Randomizing Module - Binary Halving Method

Hashing Method Example

Figure G-3 illustrates the hashing method of randomizing. This method uses a shift and add technique to develop a 31-bit binary number that should have a fairly even distribution from 0 to 2^{31}. The number is developed as follows: The result register is initialized to zero. The first character of a key field value is added to the result register and the register contents are then shifted right three hexadecimal digits. The bits of the register shifted right and out of the register are then added back to the register containing the previous shift result. This partial result is tested to be odd or even. If it is odd, the contents of the register are complemented. The original character is then added to the register. The result register content is again shifted right three hexadecimal digits. The bits of the register shifted right and out of the register are then added back to the result register. This process is repeated for each character in the key field value. Instead of starting with a zero content in the result register, the result of the previous content is used. When the key field value characters are exhausted, the result is shifted right one position to guarantee a 31-bit positive result.

Example:

Assume key field value = ABCD

```
Key Character  Result Register

A              0C100000 After test for complement
               0C10C100 After completion of A

B              1C20C10C After test for complement
               1CE1C20C After completion of B

C              2CF1CE1C After test for complement
               EDF2CF1C After completion of C

D              FE0EDF2C After test for complement
               FF0FE0ED After completion of D

               7F87F076 After shift to force
                        positive number
```

The result can then be used as input to the modulo or binary halving technique. The latter technique is used in this example.

```
HDC3     TITLE 'DL/I, SAMPLE HDAM HASHING ADDRESS CONVERSION'
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                  S A M P L E   H A S H I N G   T E C H N I Q U E*
*                                                                     *
*            THIS CSECT IS A ONE METHOD OF HASHING AN EXTERNAL KEY   *
*        INTO A 31 BIT BINARY NUMBER WHICH CAN THEN BE USED AS INPUT  *
*        TO THE BINARY HALVING ADDRESSES RESOLUTION OR A MODULO SCHEME*
*        TO DETERMINE THE BLOCK AND ROOT ANCHOR POINT.                *
*             THIS ROUTINE PLACES FEW RESTRICTIONS ON THE EXTERNAL   *
*        KEY E.G. IT CAN BE 156 BYTES LONG, IT CAN CONTAIN ANY BIT    *
*        PATTERN. THE KEY SHOULD BE LONGER THAN 3 CHARACTERS TO INSURE*
*        SOME SPREADING, HOWEVER IT WILL WORK ON SHORTER KEYS.        *
*                                                                     *
*        CALLING SEQUENCE                                             *
*              R0 - DMB                                               *
*               1 - DMBDACS                                           *
*               7 - PST                                               *
*               9 - KEY ADDRESS                                       *
*        ON RETURN                                                    *
*               DMBDACP - BBBR                                        *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
DLZHDC30 CSECT
         STM    R14,R12,12(R13)
         USING  DLZHDC30,R15
         USING  DMBDACS,R1
         SR     R12,R12
         BCTR   R12,0            SET TO ALL FF S
         SR     R11,R11
         LA     R9,0(,R9)        CLEAR ANY HIGH ORDER BITS
         SR     R7,R7            INIT
         IC     R7,DMBDAKL           FOR
         AR     R7,R9                   LATER
         LA     R6,1                         BXLE
         SR     R2,R2
LOOP     DS     0H
         IC     R11,0(,R9)       GET GROUP OF 8 BITS
         ALR    R2,R11           ADD TO HASH
         SR     R3,R3
         SRDL   R2,12            BREAK UP CHAR PATTERNS
         OR     R2,R3            ADD INTO HIGH PORTION
         STC    R2,DMBDACP       COMPLEMENT
         TM     DMBDACP,X'01'              ON
         BZ     PASS                         MODERATELY
         XR     R2,R12                       CHANGING
PASS     SR     R3,R3                      BIT
         ALR    R2,R11           DO SECOND PASS
         SRDL   R2,12            WITHOUT
         OR     R2,R3            COMPLEMENT
         BXLE   R9,R6,LOOP       EXHAUST KEY
         N      R2,NOSIGN        FORCE POSITIVE 31 BIT RESULT
*        USE R2 AS INPUT TO HALVING OR MODULO SCHEME - HALVING SHOWN
         L      R4,DMBDABLK
         MH     R4,DMBDARAP      HIGHEST RECORD IN RANGE
         SR     R5,R5            RESULT REG
```

Figure G-3 (Part 1 of 2). Randomizing Module - Hashing Method

```
CVTLP    SRL    R4,1             CUT RANGE IN HALF
         LTR    R4,R4            RANGE EXHAUSTED
         BZ     XIT              YES
         SR     R3,R3            NO
         SRDL   R2,1             TEST MASK FOR ONE
         LTR    R3,R3
         BZ     CVTLP            NO ONE
         BXH    R5,R4,CVTLP      ONE - ADD IN RANGE

XIT      LH     R6,DMBDARAP
         DR     R4,R6
         LA     R4,1(,R4)        ROOT ANCHOR POINT
         LA     R5,1(,R5)        BLOCK
         SLL    R5,8
         OR     R4,R5
         ST     R4,DMBDACP       RESULT
         LM     R14,R12,12(R13)
         BR     R14              RETURN
         DS     0F
NOSIGN   DC     X'7FFFFFFF'
         PRINT  NOGEN
         DLZQUATE
         DLZIDLI    DMBBASE=0
         END
```

Figure G-3 (Part 2 of 2). Randomizing Module - Hashing Method

Generalized HDAM Randomizer Example

Figure F-4 illustrates use of the generalized randomizing method for HDAM data bases. If root keys are unique and totally random storage is desired, this method can be used for *any* HDAM data base without performing an analysis of key distributions.

This randomizing routine works with a maximum of 16 bytes of a key at a time. Its characteristics are:

- It is reentrant.
- Keys can contain any of the 256 System/370 characters; key length can be from 1 to 256 bytes.
- It converts *any* key distribution (with unique key values) to a totally random address distribution.
- It never returns an address in block 1, which is always a bit map block in HDAM. The user can specify any number of blocks and RAPs.
- The number of blocks must be in the range between 2 and 2^{24}-1; the number of RAPs must be in the range of 2 to 2^{31}-1 when RAPs are multiplied by blocks.
- It allows the insertion of a dummy root at the highest block-RAP to ensure the formatting of the entire root addressable area at load time.

The generalized HDAM randomizer routine uses byte substitution as well as logical and arithmetic operations to convert the key to a randomized four-byte binary number. It contains a translation table for use in byte substitution and operates on keys in groups of three characters. The translation process is iterative, with results from treating the first three and next-to-last characters combined with results from

the second, third, fourth, and second-from-last, third, fourth, fifth, and third-from-last, etc. The logic can be illustrated with a sample key consisting of the characters

```
123456 (i.e., X'F1F2F3F4F5F6')
```

On the first pass, two work areas are established. The first contains X'F2F3' (characters 2 and 3); the second contains X'F1F2F3'. Bytes are substituted into the first work area by using the fifth byte of the translation table as the starting point (five is one less than the key length), so that the F7th and F8th bytes of the table will be used (5+F2, 5+F3). Bytes are substituted into the second area by using F5 as the starting point of the table (X'F5' is the next-to-last key character). The results of the two translations are then multiplied together.

On the second pass, the characters X'F3F4' (3rd and 4th key characters) are entered into the first work area, and translated beginning at location 4 of the table (key length less 2). The characters X'F2F3F4' are exclusive-or(ed) with the results of the previous work area substitution, and the results are translated beginning at location F4 of the table (because F4 is the second-from-last key character). Again, the results are multiplied, and the product is added to the result from the first pass.

The process is repeated, beginning one character farther in the key each time and decrementing key length and character from the end until the key is exhausted. In this example, the conversion number for key 123456 is X'45683199'.

The converted number is translated twice with the zero point of the translation table being determined by the results from the conversion routine. The top bit is set to zero to ensure a positive number.

Then, the maximum number of blocks minus one is multiplied by the number of RAPs. The result of this is multiplied by the translated key.

After adjustment to ensure Block 1 is not used, the result is stored in DMPDACP.

```
HDC4     TITLE 'GENERALIZED HDAM RANDOMIZING MODULE'                * 00300016
         SPACE 2                                                    * 00350016
********************************************************************** 00400016
*            S A M P L E   H D A M   R A N D O M I Z E R           * 00450016
********************************************************************** 00500016
*        THIS MODULE CAN BE USED AS THE RANDOMIZING MODULE FOR      * 00550016
*        ANY HDAM FILE.                                             * 00600016
*                                                                   * 00650016
*********************   VALID RANGES  ******************************** 00700016
*                                                                   * 00750016
*        1) KEYS -- ACCEPTS ANY KEY TYPE OF ANY LENGTH 1-256 BYTES. * 00800016
*                                                                   * 00850016
*        2) DBD SPECS - DBD MUST:                                   * 00900016
*             SPECIFY NUMBER OF BLOCKS IN RANGE 2 TO (2**24)-1      * 00950016
*             SPECIFY NUMBER OF RAPS SUCH THAT                      * 01000016
*             RAPS X BLOCKS IN RANGE 2 TO (2**31)-1                 * 01050016
*                                                                   * 01100016
*        THIS MODULE IS RE-ENTRANT.                                 * 01150016
*                                                                   * 01200016
*        IF CALLED THROUGH DL/I R7 MUST POINT TO PST ON ENTRY.      * 01250016
*        THE FIRST THREE WORDS OF THE PST WILL BE USED AS WORKSPACE. * 01300016
*        IF CALLED FROM A USER PROGRAM R7 MUST POINT TO A WRK AREA  * 01350016
*        OF 3 WORDS.                                                * 01400016
*                                                                   * 01450016
*        THIS MODULE WILL NOT RANDOMIZE TO BLOCK 1 WHICH IS ALWAYS A * 01500016
*        BIT MAP.  LATER BIT MAP BLOCKS ARE NOT AVOIDED, BUT THIS   * 01550016
*        SHOULD HAVE NO APPRECIABLE EFFECT ON PERFORMANCE.          * 01600016
*                                                                   * 01650016
*        IF DATA BASE IS RELOADED USING NEW BLK OR RAP ALLOCATION   * 01700016
*        THE LOGICAL PHYSICAL SEQUENCE WILL REMAIN THE SAME         * 01750016
*        THUS ELIMINATING UNNECESSARY SEEKS.                        * 01800016
*                                                                   * 01850016
*        ANY KEY STARTING X'FF' IS RETURNED AS HIGHEST BLOCK-RAP    * 01900016
*        TO GIVE USER A TECHNIQUE TO FORCE TOTAL FORMATTING         * 01950016
*        DURING LOAD.                                               * 02000016
*                                                                   * 02050016
*        REGARDLESS OF INPUT DISTRIBUTION OUTPUT ADDRESSES SHOULD   * 02100016
*        HAVE A POISSON DISTRIBUTION                                * 02150016
*                                                                   * 02200016
********************************************************************** 02250016
*        R E G I S T E R S   A T   E N T R Y                        * 02300016
*                                                                   * 02350016
*        1.  IF CALLED FROM DL/I INTERFACE.                         * 02400016
*            R0 - CONTAINS DATA MANAGEMENT BLOCK ADDRESS (DMB)      * 02450016
*            R1 - CONTAINS DMBDACS CSECT ADDRESS.                   * 02500016
*            R7 - CONTAINS PARTITION SPECIFICATION TABLE ADDR (PST) * 02550016
*            R9 - CONTAINS ADDRESS OF FIRST BYTE OF KEY FIELD VALUE * 02600016
*                 SUPPLIED BY AN APPLICATION PROGRAM                * 02650016
*           R13 - CONTAINS A SAVE AREA ADDRESS.  THE FIRST THREE    * 02700016
*                 WORDS IN THE SAVE AREA MUST NOT BE CHANGED.       * 02750016
*           R14 - CONTAINS THE RETURN ADDRESS TO IMS/VS.            * 02800016
*           R15 - CONTAINS THE ENTRY POINT ADDRESS OF THE           * 02850016
*                 RANDOMIZING MODULE.                               * 02900016
```

Figure G-4 (Part 1 of 5). Randomizing Module - Generalized HDAM

```
*                                                                     * 02950016
*       2.  IF MODULE IS CALLED DIRECTLY FROM A USER PROGRAM,         * 03000016
*           THE REGISTERS MUST CONTAIN THE FOLLOWING INFORMATION:     * 03050016
*                                                                     * 03100016
*       R1 - CONTAINS THE ADDRESS OF THE RDMVTAB CSECT.               * 03150016
*            THIS CSECT IS MAPPED BY THE DMBDACS DSECT AS FOLLOWS:    * 03200016
*            DMBDANME DS    2F  NOT USED BY THIS MODULE               * 03250016
*            DMBDAKL  DS    0CL1  EXECUTABLE KEY LENGTH OF ROOT.      * 03300016
*                                 KEY LENGTH - 1.  (X'00'-X'FF')     * 03350016
*            DMBDAEP  DS    A   EP OF ADDR LOAD MODULE (THESE TWO     * 03400016
*            DMBDASZE DS    H   SIZE OF THIS CSECT (FIELDS NOT USED   * 03450016
*                                              (BY MODULE.)           * 03500016
*            DFMDARAP DS    H   NO OF ROOT ANCHOR PTS/BLOCK           * 03550016
*            DFMBABLK DS    F   NO OF HIGHEST BLOCK DIRECTLY ADDR     * 03600016
*                     DS    2F  NOT USED BY MODULE                    * 03650016
*            DMBDACP  DS F    BLOCK-RAP (BBBR) RETURNED BY MODULE     * 03700016
*                                                                     * 03750016
*       R7 - CONTAINS THE ADDRESS OF A 3-WORD WORK AREA.              * 03800016
*       R9 - CONTAINS ADDRESS OF FIRST BYTE OF KEY.                   * 03850016
*       R13 - CONTAINS ADDRESS OF SAVE AREA.                          * 03900016
*       R14 - CONTAINS RETURN ADDRESS TO IMS/VS.                      * 03950016
*       R15 - CONTAINS ENTRY POINT OF RANDOMIZING MODULE.             * 04000016
*                                                                     * 04050016
********************************************************************* 04100016
*            C A L C U L A T I O N   E X A M P L E                    * 04150016
********************************************************************* 04200016
*       CALCULATING RESULTS OF PACKING DENSITY.                       * 04250016
*       1001  BLOCKS IN ROOT ADDR AREA                                * 04300016
*       5     ROOT ANCHOR POINTS PER BLOCK                            * 04350016
*       9000  RECORDS                                                 * 04400016
*       EACH BLOCK LARGE ENOUGH TO HOLD 10 RECORDS                    * 04450016
*                                                                     * 04500016
*       ROOT ADDRESS CAN HOLD (1000 - 1) X 10 =10000 RECORDS.         * 04550016
*       PACK DENSITY = 90 PERCENT.                                    * 04600016
*       PROBABLE OVERFLOW FROM A BLOCK IS PROB R GT 10 WHEN           * 04650016
*            MU = 9 ON POISSON TABLE = .2940                          * 04700016
*                                                                     * 04750016
*       IF FILE IS VERY DYNAMIC IN DELETE AND INSERT THE PROB THAT    * 04800016
*       REQUIRED REC ACCESSED VIA OVERFLOW CHAIN IS -                 * 04850016
*            (1/2) X (1/NUM RT ANCPTS) X .0294                        * 04900016
********************************************************************* 04950016
        EJECT                                                         * 05000016
DLZHDC40 CSECT                                                        * 05050016
        STM   R14,R12,12(R13)     SAVE REGISTERS                      * 05100016
        SPACE 2                                                       * 05150016
        USING DLZHDC40,R15        ESTABLISH BASE REG FOR PGM          * 05200016
        B     ENDCR               BRANCH AROUND COPY RIGHT            * 05250016
        DC    C'5746-XX1 COPYRIGHT IBM CORP 1973, 1981'               * 05300016
        DC    C'LICENSED MATERIAL - PROGRAM PROPERTY OF IBM'          * 05350016
        DLZID MOD=DLZHDC40,VR=16                                      * 05400016
ENDCR   DS    0H                                                      * 05450016
```

Figure G-4 (Part 2 of 5). Randomizing Module - Generalized HDAM

```
         USING DMBDACS,R1          ESTABLISH BASE REG FOR PARMLIST   * 05500016
         USING PST,R7              ESTABLISH BASE REG FOR PST        * 05550016
         LA    R7,PSTCNVB          LOCATE 3 WORDS WORK AREA          * 05600016
         ICM   R12,B'1111',8(R7)   SAVE PST FIELD                    * 05650016
         EJECT                                                       * 05700016
*********************************************************************** 05750016
* IF KEY STARTS X'FF' RETURN HIGHEST BLOCK-RAP                       * 05800016
*********************************************************************** 05850016
         CLI   0(R9),X'FF'         IS FIRST BYTE OF KEY X'FF'?       * 05900016
         BNE   NORMKEY             NO...GO PROCESS NORM KEY          * 05950016
         MVC   DMBDACP(3),DMBDABLK+1  YES...STORE HIGHEST BLOCK      * 06000016
         MVC   DMBDACP+3(1),DMBDARAP+1  STORE HIGHEST ANCH PT NUMBER * 06050016
         B     GOBACK              RETURN TO CALLING MODULE          * 06100016
         SPACE 1                                                     * 06150016
*********************************************************************** 06200016
*              I N I T   F O R   W H O L E   K E Y                   * 06250016
*********************************************************************** 06300016
NORMKEY  DS    0H                                                    * 06350016
         SR    R5,R5               ACCUMULATION REG FOR KEY          * 06400016
         SR    R8,R8               KEY LEN REG                       * 06450016
         IC    R8,DMBDAKL          LOAD EXECUTABLE KEY LENGTH        * 06500016
         LA    R11,0(R9,R8)        ADDR LAST BYTE IN KEY             * 06550016
         SR    R10,R10             RT HAND BYTES WORK REG.           * 06600016
         SR    R6,R6               ACCUM REG FOR RT HAND BYTES       * 06650016
         XC    0(12,R7),0(R7)      CLEAR WORK AREA                   * 06700016
         SPACE 1                                                     * 06750016
*********************************************************************** 06800016
*        IF KEY ONLY 1 OR 2 CHARS THEN SPECIAL ACTION                * 06850016
*********************************************************************** 06900016
         SH    R8,=H'1'            IS KEY LENGTH GT 1?               * 06950016
         BP    CONV                YES...GO TO CONVERSION ROUTINE    * 07000016
*********************************************************************** 07050016
*        CODE FOR KEYS OF 1 OR 2 CHARS                               * 07100016
*********************************************************************** 07150016
         LA    R8,1(R8)            RESTORE KEY LENGTH OF 1           * 07200016
         EX    R8,MVE              MOVE KEY INTO WORK AREA           * 07250016
         MVC   6(2,R7),2(R7)                                         * 07300016
         LA    R8,1                INCREMENT KEY LENGTH              * 07350016
         B     C2                  BRANCH TO SPECIAL CONV RTN        * 07400016
         EJECT                                                       * 07450016
*********************************************************************** 07500016
*        LOOP TO CONVERT EACH POSITION                               * 07550016
*********************************************************************** 07600016
CONV     DS    0H                                                    * 07650016
         MVC   2(2,R7),1(R9)       MOVE FIRST TWO BYTES OF KEY       * 07700016
         XC    6(3,R7),0(R9)       START CONVERSION                  * 07750016
         LA    R9,1(R9)            INCREMENT TO NEXT KEY CHAR        * 07800016
         BCTR  R11,0               DEC LAST KEY VALUE ADDR           * 07850016
C2       DS    0H                                                    * 07900016
         IC    R10,0(R11)          INSERT KEY CHARACTER              * 07950016
         AR    R6,R10              STORE RIGHT HAND BYTE             * 08000016
         LA    R4,TRANTAB(R8)      STORE TABLE INCR ADDRESS          * 08050016
         TR    2(2,R7),0(R4)       TRANSLATE KEY BYTES               * 08100016
         LA    R4,TRANTAB(R10)     UPDATE TABLE ENTRY                * 08150016
         TR    6(3,R7),0(R4)       TRANSLATE AGAIN                   * 08200016
```

Figure G-4 (Part 3 of 5). Randomizing Module - Generalized HDAM

```
         L     R3,0(R7)            LOAD CONV BYTES                  * 08250016
         M     R2,4(R7)            CONTINUE CONVERSION              * 08300016
         ALR   R5,R3               STORE CONV KEY BYTES             * 08350016
         BCT   R8,CONV             CONTINUE CONV  OF REMAINING K    * 08400016
         EJECT                                                      * 08450016
*********************************************************************** 08500016
*        DEVELOP BLOCK AND RAP USING RANGE RATIO METHOD              * 08550016
*********************************************************************** 08600016
END      DS    0H                                                  * 08650016
         ST    R5,0(R7)            STORE CONV BYTES INTO WORK AREA  * 08700016
         N     R6,LOWBYTE          CLEAR ALL EXCEPT LOW-ORDER BYTE  * 08750016
         LA    R4,TRANTAB(R6)      LOAD ADDR OF BEG TR BYTE         * 08800016
         TR    0(4,R7),0(R4)       TRANSLATE BYTES                  * 08850016
         IC    R6,8(R7)            STORE CONVERTED BYTE             * 08900016
         LA    R4,TRANTAB(R6)      STORE NEW TR BYTE ADDR           * 08950016
         TR    0(4,R7),0(R4)       TRANSLATE                        * 09000016
         NI    0(R7),X'7F'         FORCE TOP BIT ZERO               * 09050016
         L     R5,0(R7)            LOAD FIRST FOUR BYTES            * 09100016
         SPACE 1                                                    * 09150016
         L     R2,DMBDABLK         STORE NO. OF BLOCKS              * 09200016
         BCTR  R2,0                SUBTRACT 1 FROM COUNT            * 09250016
         MH    R2,DMBDARAP         NUMBER RAPS X (BLOCKS - 1)       * 09300016
         MR    R4,R2               R5 DEFINED AS 0-1 RANGE WITH ZERO * 09350016
         SRDL  R4,31               BEFORE BIT1. AFTER MULT EXTRACT  * 09400016
         LH    R3,DMBDARAP         ...THE NUMERIC PART              * 09450016
         DR    R4,R3               THEN DIVIDE BY RAPS              * 09500016
         SLL   R5,8                GIVING RAP IN R4 AND BLOCK IN R5 * 09550016
         LA    R4,513(R4)          ADD 2 TO BLOCK AND 1 TO RAP      * 09600016
         ALR   R5,R4               STORE RAP AND BLOCK IN R5        * 09650016
         ST    R5,DMBDACP          STORE IN DMB                     * 09700016
         SPACE 1                                                    * 09750016
GOBACK   DS    0H                                                  * 09800016
         STCM  R12,B'1111',8(R7)   RESTORE PST FIELD                * 09850016
         LM    R14,R12,12(R13)     RESTORE CALLERS REGISTERS        * 09900016
         BR    R14                                                 * 09950016
         SPACE 1                                                    * 10000016
MVE      MVC   2(1,R7),0(R9)       EXECUTE INSTRUCTION              * 10050016
         EJECT                                                      * 10100016
         TITLE 'TABLES AND CONSTANTS'                               * 10150016
         DS    0F                                                   * 10200016
TRANTAB  DC    X'5134DB02B01376F51CB28B2317A5CBC6'                  * 10250016
         DC    X'3155E426A0190D3264B457771DBC89C9'                  * 10300016
         DC    X'44A4DAE2C8395C4DB5D7B9D2FB0B6BA7'                  * 10350016
         DC    X'CE930965D9A2923E7324084B299B21BA'                  * 10400016
         DC    X'874EC2B359CA974AFEFAF0CD159A2740'                  * 10450016
         DC    X'A33C1A864828DFA86118200085FD0FC5'                  * 10500016
         DC    X'49EE99D4BB947C07359E9CDD529525FF'                  * 10550016
         DC    X'71EAB71453507211AEAFB1783880EF98'                  * 10600016
         DC    X'567F5F0CE6F70ED570AAE5F4605443A6'                  * 10650016
         DC    X'EC91E8AC74E32F62668A84CF90C12CC3'                  * 10700016
         DC    X'F1C0EBE72D6379B65806D67E45F88C9D'                  * 10750016
         DC    X'3B6DC76E884FE068044C10305AD3AB7A'                  * 10800016
         DC    X'0ACC1E3ABF16A1338FE93F961B82F63D'                  * 10850016
         DC    X'47ADF28D42D17B372B9F81E15DDE75F9'                  * 10900016
         DC    X'7D6FBD5BBE2ED0838EF36A120341366C'                  * 10950016
```

Figure G-4 (Part 4 of 5). Randomizing Module - Generalized HDAM

```
         DC    X'5E05A94667B822FC6901DCD82AC41FED'                  * 11000016
         DC    X'5134DB02B01376F51CB28B2317A5CBC6'                  * 11050016
         DC    X'3155E426A0190D3264B457771DBC89C9'                  * 11100016
         DC    X'44A4DAE2C8395C4DB5D7B9D2FB0B6BA7'                  * 11150016
         DC    X'CE930965D9A2923E7324084B299B21BA'                  * 11200016
         DC    X'874EC2B359CA974AFEFAF0CD159A2740'                  * 11250016
         DC    X'A33C1A864828DFA86118200085FD0FC5'                  * 11300016
         DC    X'49EE99D4BB947C07359E9CDD529525FF'                  * 11350016
         DC    X'71EAB71453507211AEAFB1783880EF98'                  * 11400016
         DC    X'567F5F0CE6F70ED570AAE5F4605443A6'                  * 11450016
         DC    X'EC91E8AC74E32F62668A84CF90C12CC3'                  * 11500016
         DC    X'F1C0EBE72D6379B65806D67E45F88C9D'                  * 11550016
         DC    X'3B6DC76E884FE068044C10305AD3AB7A'                  * 11600016
         DC    X'0ACC1E3ABF16A1338FE93F961B82F63D'                  * 11650016
         DC    X'47ADF28D42D17B372B9F81E15DDE75F9'                  * 11700016
         DC    X'7D6FBD5BBE2ED0838EF36A120341366C'                  * 11750016
         DC    X'5E05A94667B822FC6901DCD82AC41FED'                  * 11800016
LOWBYTE  DC    X'000000FF'                                          * 11850016
         SPACE 1                                                    * 11900016
         LTORG                                                      * 11950016
         PRINT NOGEN                                                * 12000016
         DLZIDLI DMBBASE=0,PSTBASE=0   CREATE DL/I DOS/VS DSECTS    * 12050016
         DLZQUATE                      DL/I DOS/VS                  * 12100016
         END                                                        * 12150016
```

Figure G-4 (Part 5 of 5). Randomizing Module - Generalized HDAM

Segment Edit/Compression Routine Interface

Register Contents on Entry

The following information is provided to the segment edit/compression routine when a segment is to be processed:

- Register 1 contains the address of the PST
- Register 2 contains the address of the first byte of the segment to be processed (source address)
- Register 3 contains the address of the first byte of the work area into which the segment is to be moved (destination address)
- Register 4 contains the address of the PSDB. From this block, the FDBs can be located, as required
- Register 5 contains the address of the segment compression control section that is discussed later
- Register 6 contains a function code
- Register 13 contains the address of an 18-fullword save area where the system's registers must be stored. The first three words of this save area must remain unchanged.
- Register 14 contains the return address to DL/I
- Register 15 contains the specified entry point into the segment edit/compression routine.

All DL/I control blocks provided to the segment edit/compression routine are for reference only; no data may be changed, including the segment at the source area address. The only modification allowed is the alteration of the segment during the move operation from the source to the destination address.

Segment Edit/Compression Routine Functions

When the routine is invoked, register 6 contains one of the following codes, which specifies the function to be performed:

0 - Segment to be compressed.

The source address points to a segment image as it appears in the application program input/output area.

4 - Segment to be expanded.

The source address points to a segment to be expanded into the layout required by the application program.

12 - Control is obtained for algorithm preprocessing.

Control is obtained immediately after the data base is opened. Registers 1, 2, and 3 contain zeros.

16 - Control is obtained for algorithm postprocessing.

Control is obtained immediately prior to the data base being closed. Registers 1, 2, and 3 contain zeros.

The function codes 12 and 16 may only be used when the INIT parameter in the SEGM statement is specified. With these codes, the open/close module relinquishes control to the initialization/termination subroutines immediately after data base open and immediately before the data base is closed. Any processing required for the data base that cannot be directly related to any one segment can be accomplished at this time.

Segment Compression

The compression routine must move the segment, in variable length format, from the source address to the destination address, performing the compression during the move operation. The first data field of the segment at the destination address must be a two-byte binary value describing the segment data length. All positions up to and including the end of the sequence field must be unchanged. DL/I then processes the condensed segment through the buffer pool to secondary storage.

Segment Expansion

During the segment retrieval process, the retrieve module examines the application program request. If the segment can be accepted without analysis of a data field, control is transferred to the compression routine with function code 4, and the segment is expanded into a form to be presented to the user. If data field analysis is necessary to satisfy the qualified segment selection of the DL/I call, expansion of each segment takes place. When the correct segment is found, it is passed to the user.

The format of the segment presented to the routine is that of a variable length segment: a two-byte binary field describing the segment length followed by the appropriate data. It is the responsibility of the called routine to expand the segment properly at the destination address into correct variable length format. The routine must not change the segment up to and including the end of the sequence field. As can be seen from the preceding discussion, the use of segment compression can involve a significant amount of processing before the requested segment is presented. Therefore, the data base application should be thoroughly examined before segment compression is implemented.

Segment Compression Table

To assist the user in providing parameters to his compression algorithm, the DBD block has a segment compression table, in the form of a CSECT, appended to it. One CSECT, with a name equal to that of the segment name, is developed for each segment identified by the user as having a compression routine associated with it.

Upon entry to the compression routine, register 5 contains the address of the CSECT of the segment being processed. The DSECT shown below can be used to address the fields of this CSECT.

<table>
<tr><td colspan="3">Segment Name</td></tr>
<tr><td colspan="3">Compression Routine Name</td></tr>
<tr><td colspan="3">Entry Point of Compression Routine</td></tr>
<tr><td>Flag
Byte</td><td>Sequence Field
Length (executable)</td><td>Sequence Field
Offset</td></tr>
<tr><td colspan="2">Maximum Segment Length</td><td>CSECT Length</td></tr>
</table>

```
DMBCPAC  DSECT        COMPRESSION DSECT
DMBCPCNM DS   CL8     SEGMENT NAME
DMBCPCSG DS   CL8     COMPRESSION ROUTINE NAME
DMBCPEP  DS   A       EP OF SEGMENT COMPRESSION RTN
DMBCPFLG DS   XL1     FLAG BYTE
DMBCPSEQ EQU  X'08'   SEGMENT HAS SEQUENCE FIELD DEFINED
DMBCPNIT EQU  X'01'   INITIALIZATION PROCESSING REQUIRED
DMBCPSQF DS   XL1     EXEC LENGTH OF SEQ FIELD IF DEFINED
DMBCPSQL DS   H       SEQUENCE FIELD OFFSET FROM BEGINNING
*                       OF SEGMENT (LENGTH FIELD)
DMBCPSGL DS   H       MAXIMUM SEGMENT LENGTH
DMBCPLNG DS   H       TOTAL LENGTH OF CSECT
```

User Field Exit Routine Interface

DL/I provides the addresses of both the physical segment and the application's view to the user through the parameter list described below. Because the order in which fields are processed is arbitrary, the user written routine should not rely on the contents of other fields in the application program's view during retrieves, or fields in the physical view during stores.

The conversion status code indicates problems detected during automatic data format conversion. If the user routine corrects the problem, it should reset the code to blank. Setting the code to a non-blank results in the termination of the request with a status code of Kx, where “x” is the code set by the user routine.

Upon entry to the user field exit routine, register 15 contains a pointer to the entry point, register 14 contains the return address, register 13 contains a pointer to a standard format register save area, and register 1 points to a parameter list. The format of this list is:

NAME	DISP	LENGTH	CONTENTS	DESCRIPTION
FERPEC	0	1		ENTRY CODE
FERPGET			G	GET
FERPPUT			P	PUT
FERPFNCT	1	1		FUNCTION CODE
FERPRET			G	RETRIEVE
FERPINS			I	INSERT
FERPREP			R	REPLACE
FERPCSC	2	1		CONVERSION STATUS CODE
FERPCSOK			(BLANK)	OK
FERPCSNT			A	NUMERIC TRUNCATION ERROR
FERPCSCT			B	CHARACTER TRUNCATION ERROR
FERPCSFE			C	FORMAT ERROR
FERPCSTC			D	TYPE CONFLICT
	3	1		RESERVED
FERPDSA	4	4		PHYSICAL SEGMENT ADDRESS (IF VARIABLE LENGTH, POINTS TO TWO BYTE LENGTH FIELD)
	8	2		RESERVED
FERPPFL	10	2		PHYSICAL FIELD LENGTH
FERPPFA	12	4		PHYSICAL FIELD ADDRESS
FERPUSA	16	4		USER SEGMENT ADDRESS
	20	2		RESERVED
FERPUFL	22	2		USER FIELD LENGTH
FERPUFA	24	4		USER FIELD ADDRESS
FERPFSBA	28	4		FSB ADDRESS
FERPUWA	32	32		USER WORK AREA

Secondary Indexing Exit Routine Interface

Upon entry to the secondary indexing exit routine, registers must be saved. A save area address is provided in register 13 for this purpose. The first three words of the save area must remain unchanged.

The register contents upon entry to the exit routine are:

Register	Meaning or Content
1	PST address
2	Address of (proposed or existing) index pointer segment
3	Address of index maintenance routine CSECT shown below
4	Address of the indexed source segment
13	Save area address
14	Return to DL/I DOS/VS address
15	Entry point address of the exit routine.

Upon return to DL/I, registers 1 through 14 must be restored. Register 15 must contain a return code of either 0 or 4.

The following action occurs for the functions and conditions indicated:

Insert: A new index source segment is passed to the user routine. If the user routine determines that a secondary index entry should exist for this segment, it should return a value of 0 to DL/I in R15. DL/I will then insert an entry in the secondary index for this segment. If the user routine determines that no secondary index entry should exist for this segment, it should return a value of 4 in R15. In this case, DL/I will not insert an entry into the secondary index.

Delete: An old index source segment (the one about to be deleted) is passed to the user routine. If the user routine determines that a secondary index should exist for this segment

(when the segment was inserted an entry was created), the user routine should return a value of 0 in R15. DL/I will then delete the secondary index entry for this segment. If the user routine determines that a secondary index entry should not exist (when the segment was inserted, no secondary index was created), the user routine should return a value of 4 in R15. In this case, DL/I will make no attempt to delete a secondary index entry for this segment.

Replace: For replace calls, DL/I calls the user routine twice. The first time is for the old version of the source segment, that is, the version before it is replaced. The user routine sees a copy of the “before” image of the source segment being replaced. DL/I operation at this point is the same as for deletion. The second time the user routine is called, DL/I passes a copy of the new version of the source segment, that is, the one that exists in the user's area. DL/I operation at this point is the same as for insertion.

Note that it is possible for logic errors in the user exit to create problems in the secondary index/data base relationship. For example, an insert of a segment causes a secondary index entry to be created. On subsequent deletion of the source segment, a logic error in the user routine prevents the secondary index entry from being deleted. The result is an entry in the secondary index for which there is no corresponding source segment.

User exit routines containing the following types of logic are most likely to create secondary index/data base relationship problems:

- The user routine makes its decision based on data in the segment other than the secondary index source field and this data is changed by some programs without changing the index source field.
- The user routine makes its decisions based on information external to the segment, such as current date. In this case, the conditions when the segment was created are different than when the segment is replaced or deleted.

Upon entry to the exit routine, register 3 contains the address of the secondary indexing exit routine CSECT. The DSECT shown below can be used to address the fields of this CSECT.

If an index target segment is indexed more than once and the same user routine is used for both indexes, the routine can determine which index is currently being operated on by the name of the XDFLD that is present in the index maintenance CSECT.

Index Target Segment Name	
XDFLD Field Name	
Exit Routine	
Exit Routine Entry Point Address	
Length of CSECT	Reserved
Reserved for Initialization	

```
DMBXMPRM   DSECT       INDEX MAINTENANCE PARMS DSECT
DMBXMSGN   DS   CL8    NAME OF INDEX TARGET SEGMENT
DMBXMXDN   DS   CL8    NAME OF XDFLD
DMBXMXNM   DS   CL8    NAME OF USER EXIT ROUTINE
DMBXMXEP   DS   A      ENTRY POINT ADDR OF EXIT RTN
DMBXMPLN   DS   H      LENGTH OF INDEX PARMS CSECT
           DS   H      RESERVED
DMBXMRES   DS   F      RESERVED FOR INITIALIZATION
```

Glossary

A number of terms and phrases used in describing DL/I DOS/VS are either new to most readers, or have new meanings. To improve the readability and your understanding of this and other DL/I DOS/VS publications, the significant and important terms are defined in the following Glossary. Some of the definitions refer to the representative DL/I hierarchical structure shown in Figure X-1.

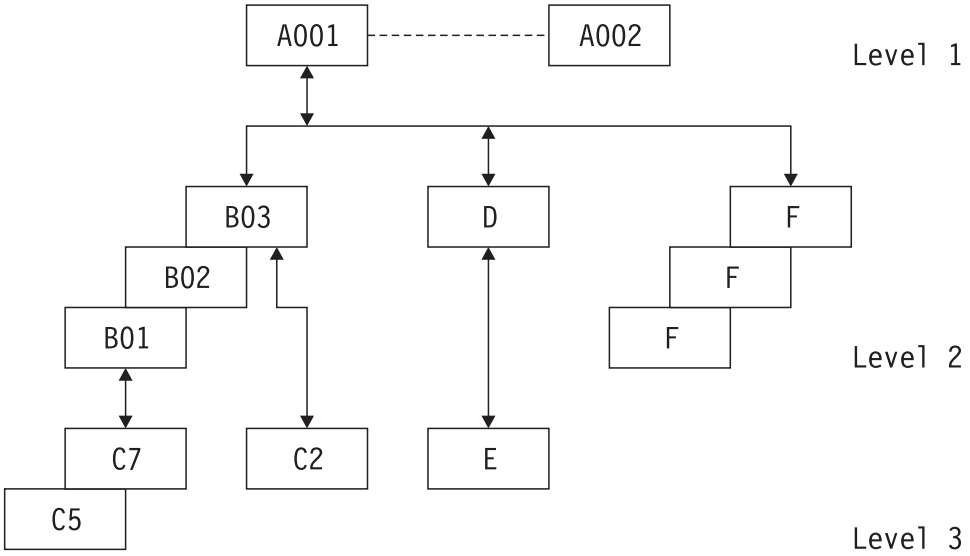


Figure X-1. Representative DL/I Hierarchical Structure

- Each block represents a segment.
- The segment names are A through F.
- The numbers represent the sequence fields (keys).
  If no number is present, the segment has no key.
- The lines connecting the segment blocks show the hierarchical paths.

ACB. (1) Application control blocks (DL/I) (2) Access method control block (VSAM)

ACBGEN. Application control block generation

access method control block (ACB). A control block that connects a program to a VSAM data set.

ACT. Application control table

addressed direct access. A VSAM term for the retrieval or storage of a data record identified by its relative byte address. (See also *keyed direct access*, *addressed sequential access*, *keyed sequential access*, and *relative byte address*).

addressed sequential access. The retrieval or storage of a VSAM data record relative to the previously retrieved or stored record. (See also *keyed sequential access*, *addressed direct access*, and *keyed direct access*).

application control blocks. The control blocks created from the output of DBDGEN and PSBGEN, e.g., a DMB of an internal PSB created by the ACB utility program.

application control block generation (ACBGEN). The process by which application control blocks are created.

application control table (ACT). A DL/I online table describing those CICS application programs that utilize DL/I.

argument. Information, such as names, constants, or variable values included within the parentheses in a DL/I command.

backout. The process of removing all the data base updates performed by an application program that has terminated abnormally. See also *dynamic backout*.

batch checkpoint/restart. The facility that enables batch processing programs to synchronize checkpoints and to be restarted at a user-specified checkpoint.

batch processing. A processing environment in which data base transactions requested by applications are accumulated and then processed periodically against a data base.

CA. control area

call. The instruction in the COBOL, PL/I, or Assembler program that requests DL/I services. For RPG II, see RQDLI command. See also *command*.

control area (CA). A collection of control intervals. Used by VSAM to distribute free space.

checkpoint. A time at which significant system information is written on the system log, and optionally, the system shut down.

child. Synonymous with *child segment*.

child segment. A segment one or more levels below the segment which is its parent, but with a direct path back up to the parent. Depending on the structure of the data base, a parent may have many children; however, a child has only one parent segment. Referring to Figure X-1:

- all the B, C, D, E, and F segments are children of A-001.
- C-5 and C-7 are children of B-01 (and A-001), but not children of the other B segments.
- B-02 has no children.

See also *logical child* and *physical child*.

CI. control interval

combined checkpoint. In MPS batch programs, a VSE checkpoint followed immediately by a DL/I CHKP command. The two checkpoints together form one logical checkpoint, and allow the use of the VSE restart facility to restart MPS batch programs.

command. The statement in DL/I High Level Programming Interface (HLPI) that requests services for application programs written in COBOL or PL/I. See also *call*.

command code. An optional addition to the SSA that provides specification of a function variation applicable to the call function.

concatenated key. The key constructed to access a particular segment. It consists of the key fields, including that of the root segment and successive children down to the accessed segment.

control interval (CI). (1) A fixed length amount of auxiliary storage space in which VSAM stores records. (2) The unit of information transmitted to or from auxiliary storage by VSAM.

data base (DB). (1) (ISO)[1] A set of data, part of the whole of another set of data, and consisting of at least one file, that is sufficient for a given purpose or for a given data processing system. (2) A collection of data records comprised of one or more data sets. (3) A collection of interrelated or independent data items stored together without unnecessary redundancy to serve one or more applications. See *physical data base* and *logical data base*.

data base administration (DBA). The tasks associated with defining the rules by which data is accessed and stored. The typical tasks of data base administration are outlined in chapter 1.

data base administrator (DBA). A person in an installation who has the responsibility (full or part time) for technically supporting the use of DL/I.

data base definition (DBD). A description of the physical characteristics of a DL/I data base. One DBD is generated and cataloged in a core image library for each data base that is used in the installation. It defines the structure, segment keys, physical organization, names, access method, devices, etc., of the data base.

data base integrity. The protection of data items in a data base while they are available to any application program. This includes the isolation of the effects of concurrent updates to a data base by two or more application programs.

data base organization. The physical arrangement of related data on a storage device. DL/I data base organizations are hierarchical direct (HD) and hierarchical sequential (HS). See *hierarchical direct organization* and *hierarchical sequential organization*.

data base record. A collection of DL/I data elements called segments hierarchically related to a single root segment.

data base reorganization. The process of unloading and reloading a data base to optimize physical segment adjacency, or to modify the DBD.

data communication (DC). A program that provides terminal communications and automatic scheduling of application programs based on terminal input. For example, CICS/DOS/VS.

---

[1] International Organization for Standardization, Technical Committee 97/Subcommittee 1

data dictionary. (1) A centralized repository of information about data, such as its meaning, relationship to other data, usage, and format. (2) A program to assist in effectively planning, controlling, and evaluating the collection, storage, and use of data. For example, DOS/VS DB/DC Data Dictionary.

data field. Synonymous with *field*.

data independence. (1) The concept of separating the definitions of logical and physical data such that application programs do not depend on where or how physical units of data are stored. (2) The reduction of application program modification in data storage structure and access strategy.

data management block (DMB). The data management block is created from a DBD by the application control blocks creation and maintenance utility, link edited, and cataloged in a core image library. The DMB describes all physical characteristics of a data base. Before an application program using DL/I facilities can be run, one DMB for each data base accessed, plus a PSB for the program itself, must be cataloged in a core image library. The DMBs and the associated PSB are automatically loaded into main storage from the core image library at the beginning of the application program execution (their loading is controlled by the parameter information supplied to DL/I at the beginning of program execution).

data set. A named organized collection of logically related records. They may be organized sequentially, as in the case of DOS/VSE SAM, or in key entry sequence, as in the case of VSE/VSAM. Synonymous with *file*.

data set group (DSG). A control block linking together a data base with the data sets comprising this DL/I data base.

DB. Data base.

DBA. (1) Data base administration. (2) Data base administrator.

DBD. Data base description.

DBDGEN. Data base definition generation -- the process by which a DBD is created.

DB/DC. Data base/data communication.

DC. Data communication.

dependent segment. A DL/I segment that relies on at least the root segment (and other dependent segments) for its full hierarchical meaning. Synonymous with *child segment*.

destination parent. The physical or logical parent segment reached by the logical child path.

device independence. The concept of writing application programs such that they do not depend on the physical characteristics of the device on which data is stored.

DIB. DL/I interface block.

DL/I interface block (DIB). Variables automatically defined in an application program using HLPI to receive information passed to the program by DL/I during execution. Contrast with *PCB mask*.

direct access. The retrieval or storage of a VSAM data record independent of the record's location relative to the previously retrieved or stored record. (See also *address direct access* and *keyed direct access*).

DMB. Data management block.

DSG. Data set group.

DTF. Define the file -- a control block that connects a program to a SAM data set.

dynamic backout. A process that automatically cancels all activities performed by an application program that terminates abnormally.

entry sequenced data set (ESDS). A VSAM data set whose records are physically in the same order as they were put in the data set. It is processed by addressed direct access or addressed sequential access and has no index. New records are added at the end of the data set.

ESDS. Entry sequenced data set

exclusive intent. The scheduling intent type that prevents an application program from being scheduled concurrently with another application program. See *scheduling intent*.

FDB. field description block

field. (1) a unique or nonunique area (as defined during DBDGEN) within a segment that is the smallest unit of data that can be referred to. See also *key field*. (2) Any designated portion of a segment.

field level sensitivity. The ability of an application program to access data at the field level. See *sensitivity*.

forward. Movement in a direction from the beginning of the data base to the end of the data base, accessing each record in ascending root key sequence, and accessing the dependent segments of each root segment from top to bottom and from left to right.

Referring to Figure X-1, forward accessing of all segments shown would be in the following sequence:

A-001, B-01, C-5, C-7, B-02, B-03, C-2, D, E, F, F, F, A-002.

FSA. free space anchor point

FSE. free space element

free space. Space available in a VSAM data set for inserting new records. The space is distributed throughout a key sequenced data set (KSDS) or left at the end of an entry sequenced data set (ESDS).

free space anchor point. A fullword at the beginning of a control interval pointing to the first free space element in this CI.

free space element. In HD data bases, the portions of direct access storage not occupied by DL/I segments are called and marked as free space elements.

HD. Hierarchical direct

HDAM. Hierarchical direct access method

HIDAM. Hierarchical indexed direct access method

HIDAM index. A data base that consists of logical DL/I records each containing an image of the key field of a HIDAM root segment. A HIDAM index data base consists of one VSAM KSDS (keyed sequenced data set).

hierarchical direct access method (HDAM). Provides for direct access to a DL/I data base in the HD organization. Segments are stored in VSAM control intervals and are referenced by a relative byte address. Root segments are accessed through a randomizing routine. An HDAM data base consists of one VSAM entry sequenced data set (ESDS).

hierarchical direct organization. An organization of DL/I segments of a data base that are related by direct addresses and may be accessed through an HD randomizing routine or an index.

hierarchical indexed direct access method (HIDAM). Provides for indexed access to a DL/I data base in the HD organization. Segments are stored in VSAM control intervals and are referenced by a relative byte address. Root segments are accessed through a HIDAM index data base. A HIDAM data base consists of one VSAM Entry Sequenced Data Set (ESDS).

hierarchical indexed sequential access method (HISAM). Provides for indexed access to a DL/I data base. A HISAM data base consists of one VSAM key sequenced data set (KSDS) and one VSAM entry sequenced data set (ESDS).

hierarchic sequence. The sequence of segment occurrences in a data base record defined by traversing the hierarchy from top to bottom, front to back, and left to right.

hierarchical sequential access method (HSAM). The segments of a DL/I HSAM physical data base record are arranged in sequential order with the root segments followed by the dependent segments. HSAM data bases are accessed by the DOS/VSE sequential access method (SAM).

hierarchical sequential organization. An organization of DL/I segments of a data base that are related by physical adjacency.

hierarchy. (1) An arrangement of data segments beginning with the root segment and proceeding downward to dependent segments. (2) A "tree" structure.

HISAM. Hierarchical indexed sequential access method

High level programming interface (HLPI). A DL/I facility providing services to application programs written in either COBOL or PL/I Optimizer language through commands.

HLPI. High level programming interface

HS. Hierarchical sequential

HSAM. Hierarchical sequential access method

index data set. An ordered collection of DL/I index entries consisting of a key and a pointer used by VSAM to sequence and locate the records of a key sequenced data set (KSDS). Organized as a balanced tree of levels of index.

index pointer segment. The segment that contains the data and pointers used to index the index target segments.

index record. A system-created collection of VSAM index entries that are in collating sequence by the key in each of the entries.

index segment. The segment in the index data base that contains a pointer to the segment containing data (the indexed segment). Synonymous with *index pointer segment*.

index source segment. The segment containing the data from which the indexing segment is built.

index set. The set of VSAM index levels above the sequence set. An entry in a record in one of these levels contains the highest key entered in an index

record in the next lower level and a pointer that indicates the record's physical location.

index target segment. The segment pointed to by a secondary index entry, that is, by an index pointer segment.

indexed segment. A segment that is located by an index. Synonymous with *index target segment*.

intersection data. Any user data in a logical child segment that does not include the logical parent's concatenated key.

key. (1) The field in a segment used to store segment occurrences in sequential order. (2) A field used to search for a segment. (3) Synonymous with *key field* and *sequence field*.

Note: A segment may or may not have a key, that is, a sequence field. All root segments, except for HSAM and simple HSAM data bases, must have keys. DL/I insures that multiple segments of the same type that have keys are maintained in strict ascending sequence by key. The key may be located anywhere within a segment; it must be in the same location in all segments of the same type within a data base. The maximum sizes for keys are 236 alphameric characters for root segments, and 255 for all dependent segments. Keys provide a convenient way to retrieve a specific occurrence of a segment type, maintain the uniqueness and sequential integrity of multiples of the same segment type, and determine under which segment of a group of multiples new dependent segments are to be inserted. Keys should normally be prescribed for all segment types; the exception being if there will never be multiples of a particular type or if a particular segment type will never have dependents.

keyed direct access. The retrieval or storage of a data record by use of an index that relates the record's key to its physical location in the VSAM data set, independent of the record's location relative to the previously retrieved or stored record. See also *addressed direct access*, *keyed sequential access*, and *addressed sequential access*.

keyed sequential access. The retrieval or storage of a VSAM data record in its collating sequence relative to the previously retrieved or stored record, by the use of an index that specifies the collating sequence of the records by key. See also *addressed sequential access*, *keyed direct access*, and *keyed sequential access*.

level. Level is the depth in the hierarchical structure at which a segment is located. Roots are always the highest level and the segments at the bottom of the structure are the lowest level. The maximum number of levels in a a DL/I data base is 15. For purposes of documentation and reference, the levels are numbered from 1 to 15, with the root segments being level number 1. Referring to Figure X-1:

- Three levels are shown.
- The A segments (roots) are at the highest level (level 1).
- The C and E segments are at the lowest level (level 3).

local system. (1) A specific system in a multisystem environment. Contrast with *remote system*. (2) The system in a multisystem environment on which the application program is executing. The local application may process data from data bases located on both the same (local) system and and another (remote) system.

logical. When used in reference to DL/I components, logical means that the component is treated according to the rules of DL/I rather than physically as it may exist, or as it may be organized, on a physical storage device. For example, a logical DL/I record (a root segment and all of its dependent segments grouped) might be contained on several physical records or blocks on a storage device; and because of prior insertions and deletions, the segments might be in different physical sequence than that by which they are retrieved logically for the application program by DL/I.

logical child. A pointer segment that establishes an access path between its physical parent and its logical parent. It is a physical child of its physical parent; it is a logical child of its logical parent. See also *logical parent* and *logical relationship*.

logical data base. A data base composed of one or more physical data bases representing a hierarchical structure derived from relationships between data segments that can be different from the physical structure.

logical data base record. (1) A set of hierarchically related segments of one or more segment types. As viewed by the application program, the logical data base record is always a hierarchic tree structure of segments. (2) All of the segments that exist hierarchically dependent on a given root segment, and that root segment.

logical data structure. A hierarchic structure of segments that is not based solely on the physical relationship of the segments. See also *logical relationships*.

logical parent. The segment a logical child points to. A logical parent segment can also be a physical parent. See also *logical child* and *logical relationship*.

logical relationship. A user defined path between two segments; that is, between logical parent and logical

child, which is independent of any physical path. Logical relationships can be defined between segments in the same physical data base hierarchy or in different hierarchies.

logical twin. All occurrences of one type of logical child with a common logical parent. Contrast with *physical twin*. See also *twin segment*.

MPS. Multiple partition support

MPS Restart facility. The capability to restart an MPS batch job when a system or application program failure occurs using VSE checkpoint/restart with the DL/I checkpoint facility.

multiple partition support (MPS). Multiple partition support provides a centralized data base facility to permit multiple applications in different partitions to access DL/I data bases concurrently. MPS follows normal DL/I online conventions in that two programs cannot both update the same segment type in a data base concurrently. However, two or more programs can retrieve from a data base while another program updates it. If one program has exclusive use of a data base, no other program can update it or retrieve from it.

multiple SSA. A series of segment search arguments (SSAs) included in a DL/I call to identify a specific segment or path. See also *segment search argument*.

object segment. The segment at the lowest hierarchical level specified in a particular command. See also *path call*.

option. A command keyword used to qualify the requested function.

parent. Synonymous with *parent segment*.

parentage. Establishment in a program of a particular parent as the formal beginning point for the use of the GNP (get next in parent) or GHNP (get hold next in parent) functions. Parentage can only be established by issuing successful GU, GHU, GN, or GHN commands/calls.

parent segment. (1) A segment that has one or more dependent segments. Contrast with *child*. (2) A parent is the opposite of a child, or dependent segment, in that dependent segments exist directly beneath it at lower levels. A parent may also itself be a child. Referring to Figure X-1:

- A-001 is the parent of all B, C, D, E, and F segments.
- D is a parent of E; yet a child of A.
- B-02 is not a parent.
- None of the level-3 segments are parents.

path. The chain of segments within a record that leads to the currently retrieved segment. The formal path contains only one segment occurrence from each level from the root down to the segment for which the path exists. The exact path for each retrieved segment is returned in the PCB by means of the following of its nine fields:

Field 2 Segment hierarchy level indicator

Field 6 Segment name feedback area

Field 7 Length of key feedback area

Field 9 Key feedback area, containing the concatenated keys in the path

Referring to Figure X-1:

- The path to C-5 is A-001, B-01.
- The path to C-7 is the same as the path to C-5.
- There is no path to A-002 because it is a root segment.

path call. (1) The retrieval or insertion of multiple segments in a hierarchical path in a single call, by using the D command code and multiple SSAs. (2) The retrieval, replacement, or insertion of data for multiple segments in a hierarchical path in a single command, by using the FROM or INTO options specifying an I/O area for each parent segment desired. The object segment is always retrieved, replaced, or inserted.

PCB. Program communication block

PCB mask. A skeleton data base PCB in the application program by which the program views a hierarchical structure and into which DL/I returns the results of the application's calls.

physical child. A segment type that is dependent on a segment type defined at the next higher level in the data base hierarchy. All segment types, except the root segment, are physical children because each is dependent on at least the root segment. See also *child segment*.

physical data base. An ordered set of physical data base records.

physical data base record. A physical set of hierarchically related segments of one or more segment types.

physical data structure. A hierarchy representing the arrangement of segment types in a physical data base.

physical parent. A segment that has a dependent segment type at the next lower level in the physical data base hierarchy. See also *parent*.

physical segment. The smallest unit of accessible data.

physical twins. All occurrences of a single physical child segment type that have the same (single occurrence) physical parent segment type. Contrast with *logical twins*. See also *twin segment*.

PI. Program isolation

pointer. A physical or symbolic identifier of a unique target.

program communication block (PCB). Every data base accessed in an application program has a PCB associated with it. The PCB actually exists in DL/I and its fields are accessed by the application program by defining their names within the application program as follows:

- COBOL - The PCB names are defined in the linkage section.
- PL/I - The PCB names are defined under a pointer variable.
- RPGII - The PCB names are automatically generated by the translator, or may be defined by the user.
- Assembler - The PCB names are defined in a DSECT.

There are nine fields in a PCB:

1. Data base name
2. Segment hierarchy level indicator
3. DL/I results status code
4. DL/I processing options
5. Reserved for DL/I
6. Segment name feedback area
7. Length of key feedback area
8. Number of sensitive segments
9. Key feedback area

program isolation (PI). A facility that isolates all data base activity of an application program from all other application programs active in the system until that application program commits, by reaching a synchronization point, that the data it has modified or created is valid.

This concept makes it possible to dynamically backout the data base activities of an application program that terminates abnormally without affecting the integrity of the data bases controlled by DL/I. It does not affect the activity performed by other application programs processing concurrently in the system.

position pointer. For most call functions a position pointer exists before, during, and after the completion of the function. The pointer indicates the next segment in the data base that can be retrieved sequentially. It is normally set by the successful completion of *all* call functions.

Referring to Figure X-1:

- If A-001 has just been retrieved, it points to B-01.
- If a new segment C-6 has just been inserted, it points to C-7.
- If the D segment has been deleted (E will be deleted along with it), it points to the first F segment.
- If the last F segment has just been retrieved, it points to A-002.

During PSB generation it is possible to specify either single or multiple positioning.

program specification block (PSB). A PSB is generated for each application program that uses DL/I facilities. The PSB is associated with the application program for which it was generated and contains a PCB for each data base that is to be accessed by the program. Once it is generated, the PSB is cataloged in a core image library, and subsequently processed by a utility along with the associated DBDs to produce the updated PSB and DMBs; all of these are cataloged in a core image library for subsequent use by the application program during execution.

PSB. Program specification block

PSBGEN. PSB generation -- the process by which a program specification block is created.

qualified call. A DL/I call that contains at least one segment search argument (SSA). See also *segment search argument*.

qualified segment selection. The identification of a specific occurrance of a given segment type in a command, by using the WHERE option in the command for the desired segment. Contrast with *qualified SSA*.

qualified SSA. A qualified segment search argument contains both a segment name that identifies the specific segment type, and segment qualification that identifies the unique segment within the type for which the call function is to be performed. See also *segment search argument* and *multiple SSA*.

RAP. Root anchor point

RBA. Relative byte address

read-only intent. The scheduling intent type that allows a program to be scheduled with any number of other programs except those with exclusive intent. See *scheduling intent*.

record. A data base record is made up of at least a unique root segment, and all of its dependent segments. See also *data base record*.

Referring to Figure X-1: A-001, B-01, C-5, C-7, B-02, B-03, C-2, D, E, F, F, F constitute a data base record.

relative byte address (RBA). The displacement of a stored record or control interval from the beginning of

the storage space allocated to the VSAM data set to which it belongs.

remote system. In a multisystem environment, the system containing the data base that is being used by an application program resident on another (local) system. Contrast with *local system*.

root anchor point (RAP). A DL/I pointer in an HDAM control interval pointing to a (chain of) root segment(s).

root segment. The highest level (level 1) segment in a record. A root segment must have a key unless the organization is HSAM or simple HSAM. The sequence of the root segments constitutes the fundamental sequence of the data base. There can be only one root segment per record. Dependent segments cannot exist without a parent root segment; but a root segment can exist without any dependent segments.

RQDLI command. The instruction in the RPGII program used to request DL/I services.

scheduling intent. An application program attribute defined in the PSB that specifies how the program should be scheduled if multiple programs are contending for scheduling. See *exclusive intent, read-only intent*, and *update intent*

search field. In a given DL/I call, a field that is referred to one or more segment search arguments (SSAs).

secondary index. Secondary indexes can be used to establish alternate entries to physical or logical data bases for application programs. They can also be processed as data bases themselves. See also *secondary index data base*.

secondary index data base. An index used to establish accessibility to a physical or logical data base by a path different from the one provided by the data base definition. It contains index pointer segments.

secondary key. A data element or combination of data elements within a data aggregate that identifies those occurrences of the aggregate that have a property named by the key. Used to locate those occurrences. See *primary key*.

segment. A segment is a group of similar or related data that can be accessed by the application program with one I/O function call. There may be a number of segments of the same type within a record.

segment name. A segment name is assigned to each segment type. Segment names for the different segment types must be unique within a data base. Synonymous with *segment type*.

segment occurrence. One instance of a set of similar segments.

segment search argument. (SSA). Describes the segment type, or specific segment within a a segment type, that is to be operated on by a DL/I call. See also *multiple SSA, qualified SSA, and unqualified SSA*.

segment selection. The specifying of parent and object segments by name in a command. Selection may be either qualified or unqualified. Contrast with *segment search argument*.

segment type. Different segment types may have different lengths, but within each single type, all segments must be the same length (unless variable length segments have been specified by DBA).

Referring to Figure G-1, there are six different types of segments: A through F. Synonymous with *segment name*.

sensitivity. (1) A DL/I capability that ensures that only data segments or fields predefined as "sensitive" are available for use by a particular application program. The sensitivity concept also provides a degree of control over data security, inasmuch as users can be prevented from accessing particular segments or fields from a logical data base. (2) Sensitivity to the various segments and fields that constitute a data base is controlled, on a program-by-program basis, when the PSB for each program is generated. For example, a program is said to be sensitive to a segment type when it can access that segment type. When a program is not sensitive to a particular segment type, it appears to the program as if that segment type does not exist at all in the data base. Segment sensitivity applies to types of segments, not to specific segments within a type, and to all segment types in the path to the lowest level sensitive segment type.

sequence field. Synonymous with *key field*.

sequence set. The lowest level of a VSAM index. It immediately controls the order of records in a key sequenced data set (KSDS). A sequence set entry contains the key of the highest keyed record stored in a control interval of the data set and a pointer to the control interval's physical location. A sequence set record also contains a pointer to the physical location of each free control interval in the fan-out of the record.

sequential processing. Processing or searching through the segments in a data base in a forward direction. See also *forward*.

simple HISAM. A hierarchical indexed sequential access method data base containing only one segment type.

source segment. A segment containing the data used to construct the secondary index pointer segment. See also *secondary index data base*.

source segment. A segment containing the data used to construct the secondary index pointer segment. See also *secondary index data base*.

SSA. segment search argument.

status code. Each DL/I request for service returns a status code that reflects the exact result of the operation. The first operation that a program should perform immediately following a DL/I request is to test the status code to ensure that the function requested was successful. Following a command, the status code is returned in the DIB at the label DIBSTAT. Following a call, the status code is returned in field 3 of the PCB.

sync(h) point. Synonymous with *synchronization point*.

synchronization point. A logical point in time during the execution of a application program where the changes made to the data bases by the program are committed and will not be backed out if the program subsequently terminates abnormally. Synonymous with *sync point* or *synch point*.

A synchronization point is created by:

- a DL/I CHECKPOINT command or CHKP call
- a DL/I TERMINATE command or TERM call
- a CICS/VS sync point request
- an end of task (online) or an end of program (MPS-batch).

system view. A conceptual data structure that integrates the individual structures of local views into an optimum arrangement for physical implementation as a data base. See *local view*.

transaction. A specific set of input data that triggers the execution of a specific process or job.

twin segments. All child segments of the same segment type that have a particular instance of the same parent type. See also *physical twins* and *logical twins*.

twins. Synonymous with *twin segments*.

unqualified call. A DL/I call that does not contain a segment search argument.

Unqualified segment selection. The identification of a given segment type in a command without specifying a particular occurrence of that segment type (without using the WHERE option). As a general rule, unqualified segment selection retrieves the first occurrence of the specified segment type. Contrast with *unqualified SSA*.

unqualified SSA. An unqualified SSA contains only a segment name that identifies the specific type of segment for which the I/O function is to be performed. As a general rule, the use of an unqualified SSA retrieves the first occurrence of the specified type of segment. See also *segment search argument*

update intent. The scheduling intent type that permits application programs to be scheduled with any number of other programs except those with exclusive intent.

UPSI. User program switch indicator. A special 8-bit byte that allows each bit to be programmed by the user as "1" or "0". Bits may be read by a program to determine what the user wants to do.

Index

Special Characters

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

Communicating Your Comments to IBM

IBM Data Language/I Disk Operating System/
Virtual Storage (DL/I DOS/VS)
Data Base Administration
Version 1 Release 7

Publication No. SH24-5011-01

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of the book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF form and either send it postage-paid in the United States, or directly to:

  IBM Deutschland Entwicklung GmbH
  Department 3248
  Schoenaicher Strasse 220
  D-71032 Boeblingen
  Federal Republic of Germany

- If you prefer to send comments by FAX, use this number:
  - (Germany): 07031-16-3456
  - (Other countries): (+49)+7031-16-3456

- If you prefer to send comments electronically, use this network ID:

  INTERNET: s390id@de.ibm.com

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies.

Readers' Comments — We'd Like to Hear from You

IBM Data Language/I Disk Operating System/
Virtual Storage (DL/I DOS/VS)
Data Base Administration
Version 1 Release 7

Publication No. SH24-5011-01

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	□	□	□	□	□

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	□	□	□	□	□
Complete	□	□	□	□	□
Easy to find	□	□	□	□	□
Easy to understand	□	□	□	□	□
Well organized	□	□	□	□	□
Applicable to your tasks	□	□	□	□	□

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? □ Yes □ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SH24-5011-01

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Entwicklung GmbH
Department 3248
Schoenaicher Strasse 220
D-71032 Boeblingen
Federal Republic of Germany

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

File Number: S370/S390-50
Program Number: 5746-XX1

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SH24-5011-01