**Program Product**

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

**Guide For New Users**

IBM

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

**Program Product**

## Guide For New Users

**Program Number 5746-XX1**

**IBM**

## Summary of Amendments
## for SH24-5001-2
## Version 1.5

This edition has been revised to include the following DL/I DOS/VS functional enhancements.

**Field Level Sensitivity**

This feature allows you to select fields from within a physical segment definition to build a new view of the segment for exclusive use by a particular application program.

**Extended Logical Relationships**

Extended logical relationships removes or changes some of the rules and restrictions concerning an application's view of a data base structure.

**Unique Segment Support**

A new keyword (NOTWIN) is added to the POINTER parameter on the SEGM statement to allow a segment to be limited to a single occurrence per parent.

**Sample Application Update**

The customer data base for the Sample Application is updated to show an example of field level sensitivity. Source code for this sample application in COBOL, PL/I, and RPG II is shipped with this version.

**DL/I DOS/VS—IMS/VS Compatibility**

DL/I DOS/VS users planning future migration to IMS/VS are cautioned that the VIRFLD statement and some options of the SENFLD statement (PSB generation), and some options of the SEGM and FIELD statements (DBD generation) are not supported by IMS/VS. See the *Utilities and Guide for the System Programmer* for details.

**Miscellaneous**

Several sections in this manual have been enhanced to include additional information for increased understanding. This manual also includes some miscellaneous corrections and updates.

---

## Summary of Amendments
## for SH24-5001-1
## Version 1.4

This edition has been revised to include the following DL/I DOS/VS functional enhancements.

**RPG II Support**

Application programs written in RPG II can now access DL/I data bases in a manner similar to programs written in COBOL, PL/I, and Assembler language.

**Prefix Resolution Improvement**

The prefix resolution utility now passes an actual maximum record length, instead of a maximum possible record length, to the DOS/VS or DOS sort/merge program.

**Extended DL/I Call Interface**

This support, along with CICS/VS high level language support, eliminates the need for application programs to reference internal CICS/VS control blocks. A new parameter has been added to the PCB call to obtain the address of the DL/I User Interface Block. This control block contains the information previously returned in the TCA.

This enhancement is required for application programs written in RPG II. It may also be used in programs written in COBOL, PL/I and Assembler.

**Intersystem Communication**

CICS/VS intersystem communication support enables DL/I application programs to access a data base that is resident on another CPU.

**High Level Language Debugging for PL/I**

This support PL/I allows diagnostic information to be supplied by both PL/I and DL/I. It is designed only for batch and MPS batch execution of DL/I, and does not require any changes to the PL/I code.

**Online Sample Application Update**

The customer data base for the Online Sample Application now includes a variable length segment. A sample segment edit/compression exit routine has been added to show how variable length segments can be used.

**Performance Improvements**

This edition also contains information on these performance improvements:

- Batch Partition Controller (BPC)
  The BPC program has been changed from non-reenterable to quasi-reenterable. This reduces the working set requirements of the BPC.

- Log Buffer
  When using CICS/VS journaling, a blocksize larger than 1024 bytes can now be specified in the CICS/VS jounal control table. The maximum blocksize is 32767 bytes.

- Program Isolation
  Two enhancements have been added to program isolation. They are the "O" procopt and MPS Batch Notification.

**Miscellaneous**

Several sections in this manual have been enhanced to include additional information for increased understanding. This manual also includes some miscellaneous corrections and updates.

---

DL/I DOS/VS (Data Language/I Disk Operating System/Virtual Storage) is a data base management control system that improves an installation's ability to implement and maintain batch processing applications. DL/I DOS/VS permits the writing of data independent applications and provides program and data base integrity. The DL/I DOS/VS system supports application programs written in COBOL, PL/I, RPG II, and Assembler language. DL/I DOS/VS executes as an application program under DOS/VS.

DL/I DOS/VS permits concurrent scheduling of multiple programs requesting DL/I DOS/VS services, thereby allowing access by more than one user to the same or different data bases at the same time. Application programs may utilize this concept in conjunction with CICS/VS (Customer Information Control System/Virtual Storage) to access DL/I DOS/VS data bases in a teleprocessing environment.

This publication is intended primarily for first time users of DL/I DOS/VS. It provides the information the user needs to design and implement a basic DL/I data base system. This includes assistance for the user in developing his application programs.

This publication is a starter document. It is not documentation for a subset (reduced function) or a simplified version of DL/I. It is a systematic approach to guide the user in designing simple data base structures, and controlling access to the data contained in these structures. Through extensive use of examples and references to the sample application program provided with DL/I, this publication guides the user through the basic and most often needed DL/I facilities.

This manual describes the operation and maintenance of DL/I applications from the viewpoint of both data base administration and application programming. The topics covered are designed to:

- Reinforce the user's knowledge of data base concepts, and the functions available in DL/I. The new user is expected to be familiar with the DL/I DOS/VS *General Information manual* before using this manual.

- Describe organizing, creating, and maintaining data bases.

- Guide the user in writing data base application programs.

- Provide workable examples for setting up a specific data base application such as the online order entry and inquiry system provided as a sample application with DL/I.

Because the features and facilities of DL/I are presented in this publication so the user will need to make few references to other DL/I DOS/VS publications, this publication repeats certain information that is also presented in other DL/I DOS/VS publications. However, references are made to the other publications in the DL/I DOS/VS library to assist the user in locating specific additional information that may be needed for unique application requirements.

The following IBM publications provide additional details about DL/I DOS/VS:

*DL/I DOS/VS General Information,* GH20-1246

*DL/I DOS/VS Application Programming Reference Manual,* SH12-5411

*DL/I DOS/VS Utilities and Guide for the System Programmer,* SH12-5412

*DL/I DOS/VS System/Application Design Guide,* SH12-5413

*DL/I DOS/VS Messages and Codes,* SH12-5414

References are made in this publication to CICS/VS. More information about CICS/VS can be found in the *Customer Information Control System/Virtual Storage (CICS/VS) General Information Manual,* GC33-0066.

Because of the special nature of DL/I DOS/VS as a functional subset of IBM's Information Management System (IMS/VS), some IMS or OS specific terms are retained in DL/I DOS/VS documentation. These terms are used for ease of reference to corresponding IBM documentation and to facilitate subsequent upgrading to an upward-compatible IBM system.

All further references in this manual to DL/I DOS/VS are shortened to DL/I.

# Contents

## Introduction

Data Language/I DOS/VS (DL/I) is a data base management system that improves the DOS/VS user's ability to implement and maintain batch and/or teleprocessing data processing applications. DL/I helps to reduce data processing costs by:

- Reducing application program maintenance.

- Reducing application programmer time required to implement new applications, including teleprocessing applications.

- Reducing the cost of converting to new hardware (for example, from 2314 to 3340).

- Reducing the number of programs and/or data files required to implement applications.

- Reducing the number of files in which data is repeated.

## Potential Users of DL/I

The DOS/VS user who is modifying existing applications and/or adding new applications may be faced with some of these situations:

- Data is duplicated on multiple data files with different formats for different applications.

- Programmers are spending a significant amount of time updating existing application programs to handle changes to record layouts or I/O device characteristics; often, even though program logic is not affected by these changes.

- Changing applications make it desirable to move data files from one storage device to another (tape to disk), or from one access method to another (sequential to direct).

- Programmer productivity is hindered by a limited knowledge of specific device characteristics (for example, optimum block size for indexed sequential processing) or specific access methods.

- Batch applications must be expanded smoothly and easily to teleprocessing applications.

DL/I is a control system designed to assist the user with these needs.

## General System Description

DL/I has the following characteristics:

- It runs in a user program partition under DOS/VS.

- It provides a subset of the batch data management facilities offered by IMS/VS. Application programs

are upward compatible through DL/I for easy growth.

- It includes four file organizations: Sequential, Indexed Sequential, Direct, and Indexed Direct. The user may choose the organization best suited for each data file, and later change to another organization as his application needs change, without reprogramming.

- User application programs may be written using Assembler, COBOL, PL/I, or RPG II.

- DL/I includes an interface module that allows transaction processing programs accessing data bases to run in the teleprocessing environment provided by CICS/VS. This interface module interprets requests for data, but does not alter the system in any way.

- DL/I enables application programs executing in different partitions to access the same data base concurrently. This capability, multiple partition support, permits, for example, online applications to issue inquiries to a data base while a batch program updates it.

- The DL/I data structure handles variable occurrences of fixed length data without wasting secondary storage space. For example, a customer master file containing purchase order information does not require reserved space for the maximum number of line items possible in a single purchase order.

- The complex and changing details of data access are concentrated within DL/I. Only one person or group within the data processing department needs in-depth education on the specifics of device characteristics and access methods.

- It provides for the separation of application program logic from device oriented details. This means that movement of data from one device to another (for example, tape to 3330, to 3350) does not affect the application program. This is called *device independence*.

- It provides for the separation of application logic from data organization. For example, data files may be expanded to contain additional data, or changed from indexed sequential to indexed direct organization without affecting existing application programs. If existing programs do not reference newly defined data, there is no need to recompile the application program. This is called *data independence*.

- Both DL/I data bases and other DOS/VS files may be accessed by the same application program.

## Program Structure

The following program modules are required to execute a DL/I application program:

- The user application program containing DL/I calls.

- For each application program, a PSB (program specification block) that identifies each DL/I data base used by this program and describes how each can be processed by this program.

- For each DL/I data base, a data base description block that describes the physical data base structure, the file organization, and the device on which the data base resides.

- The DL/I processing modules.

  These modules are stored in the core image library. For online execution, the CICS/VS system control functions load the modules as required.

## Data and Device Independence

The separation of the application program from data base oriented logic allows both data and device independence.

Data independence means:

- Adding new types of data to existing data bases with no application program recompile

- Optimizing system performance by varying record size, blocking factor, space allocation, and access method with no application program recompile

- Allowing programs to refer to the same data by the same name

- Reducing programming maintenance caused by changes in existing data format.

Device independence means:

- Data bases can be moved from tape to disk access methods with no application program recompile

- Device changes from 2400 to 2314, to 3330, to 3340, to 3350 or FBA (or any combination of these) can be made with no application program recompile.

## Program Execution

DL/I acts as an interface between the application program and the DOS/VS data management routines. DL/I is actually the main program in the DOS/VS partition, and the user written COBOL, PL/I, RPG II or Assembler program is treated as a subroutine. The application program communicates with DL/I via the DL/I language interface. Program requests to DL/I are issued by using a standard DL/I call statement or an RQDLI command in RPG II. These call statements provide for reading, deleting, adding, and changing segments in the data base. (A segment consists of one or more logically associated data fields, and is of fixed or variable length.) Feedback information is provided by DL/I after every call indicating successful or unsuccessful completion, and complete identification of the data base segment retrieved or processed.

The relationships between DL/I and the application program are illustrated in Figure 1-1.

1. DOS/VS loads DL/I and gives control to DL/I. DL/I loads the PSB and analyzes the data base requirements of this application program as defined in the PSB. DL/I then loads and initializes the DMBs required.

2. DL/I loads and gives control to the application program.

3. The application program processes segments in the data base through calls to DL/I.

4. The DL/I call analyzer decodes the call parameters into specific data base actions.

5. The DL/I action modules translate the data base calls into I/O requests appropriate to each data base.

6. DL/I determines which access method is required for the data base and optionally logs any changes.

7. The DOS/VS SAM or VSAM routines read and write data in the data base files.

8. Changes are made to the data base or data is returned as requested by the application program.

9. DL/I returns the requested data or a status code to the application program.

10. When the application program reaches end of job, control is returned to DL/I.

11. DL/I closes the data bases and returns control to DOS/VS.

## Utility Programs

DL/I supplies a number of utility programs that provide for the reorganization and recovery of a data base file. These utilities are used to improve DL/I performance and to facilitate future expansion. The use of the reorganization and recovery utilities is discussed in Chapters 6 and 7 of this manual.
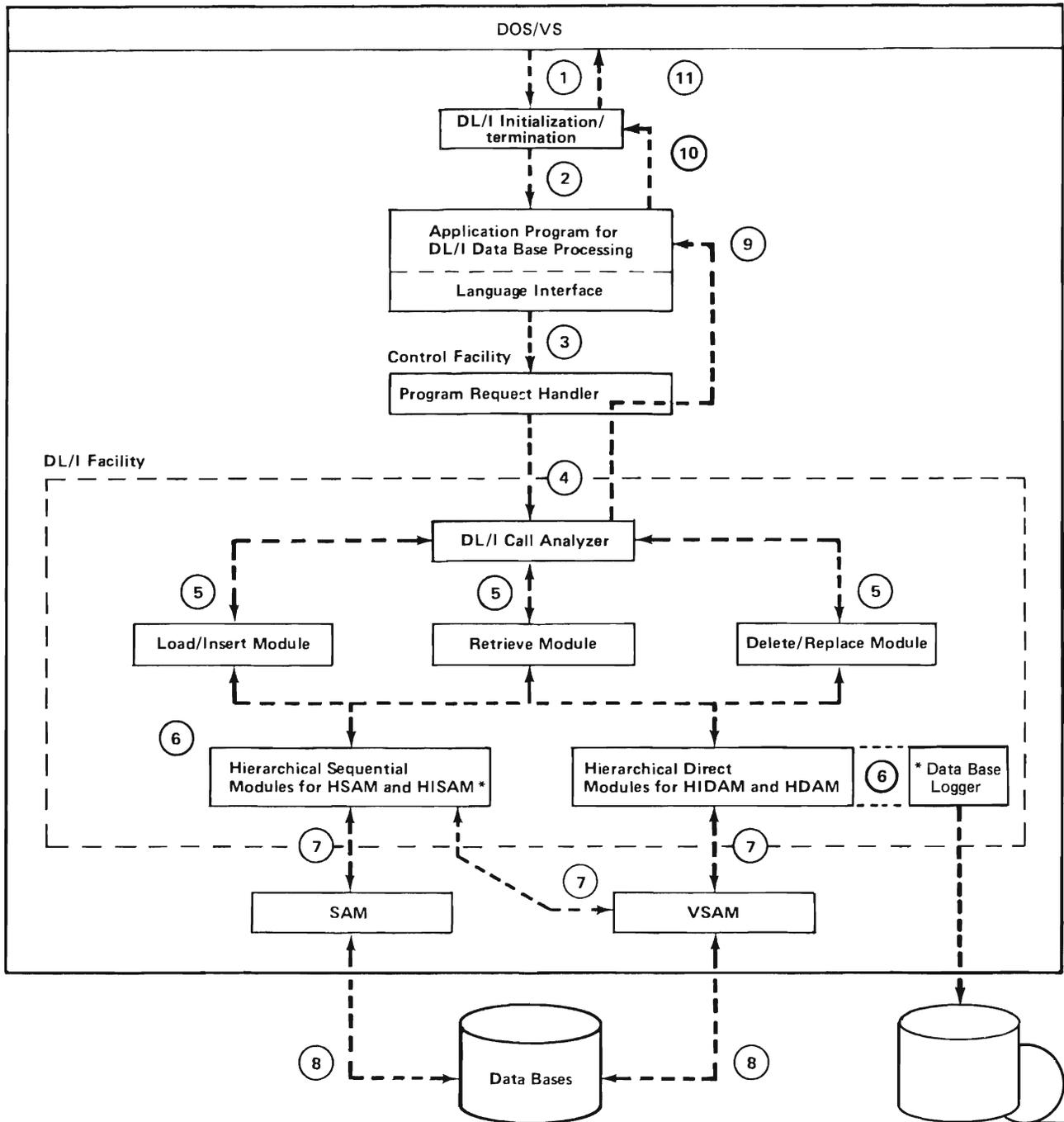
Figure 1-1. DL/I Batch System

## File Integrity and Recovery

As a user option, all modifications to any data base used in the DL/I environment can be recorded on the DL/I log. If multiple executions are performed concurrently without using MPS (multiple partition support), a DL/I log is associated with each partition. With MPS, one central DL/I log is used. No attempt should be made to access the same data base from more than one DOS/VS partition, unless MPS is used.

Data base logging provides the DL/I system with a record of all modifications to all data bases used during a data base execution and, in the online system, can be carried out in either of two ways. One is to use the standard DL/I log feature as it is used in the batch system. The other possibility is to assign the DL/I log to the CICS/VS system journal. In this case, the log file is shared between DL/I log records and any other system-provided or user-provided CICS/VS journal records. The DL/I log may, in either case, be used in conjunction with the DL/I data base recovery utilities to rebuild

a data base. The *Utilities and Guide for the System Programmer* provides additional detail on the use of data base log information for recovery.

### Online Environment

The CICS/VS interface module provided with DL/I allows DL/I VSAM data bases to be processed by CICS/VS application programs written in COBOL, Assembler, PL/I, or RPG II. The CICS/VS application program issues DL/I calls to process DL/I data base records.

All functions available to a batch application program are also provided to online transactions except for the loading of data bases and DL/I utility functions.

DL/I with CICS/VS controls access by multiple transaction processing programs to the same data, so that a single data base can concurrently be updated by any number of transaction processing programs.

## Data Base Facility

### Data Base Concepts

All data files within your organization are candidates for inclusion in a data base. A data base can be defined as a nonredundant collection of interrelated data items processable by one or more applications. DL/I, as a data base management facility, provides a structure for this data, and makes it easier to store and retrieve these items.

#### Segments

The segment is the unit of data processed by DL/I. The segment is processed with DL/I call statements. DL/I provides the application program independence from access methods, from physical storage organizations, and from the characteristics of the devices on which the data of the application is stored. This independence is provided by a symbolic program linkage and by data base descriptions external to the application program. A reduction in application program maintenance is a natural result of this separation.

#### Segment Sensitivity

An important capability of DL/I that permits development of a multi-application data base is the concept of segment sensitivity. Each application program using the data base can be sensitive to its own unique subset of the data base segments. Segment sensitivity is defined in the PSB that the application program uses during execution.

#### Field Level Sensitivity

In addition to segment sensitivity, the user can specify fields from within the physical segment definitions to build a new view of the segment for exclusive use by a particular application program. Field level sensitivity is defined in the PSB used by the application program during execution.

## Logical Data Base Structure

Each record in the data base (except for HSAM, SHSAM, and HDAM), must contain a *key* identifying that record. Data base records are variable in length and contents, as required, and normally contain all the data logically related to a particular key. Data base records are presented to the application programmer in a hierarchical segmented structure as illustrated in Figure 1-2. Logically related fields within the records are grouped together into *segments*. Segments themselves are related hierarchically, that is, some segments are dependent on the existence of a segment at a higher level.

The first segment in a data base record contains the key of the data base record and is called the *root segment*. There can be only one root segment per data base record. Segments at lower levels may be of any type, in any combination, and may occur any number of times, within the limits of the DL/I architecture. All DL/I calls issued by the application programs relate to retrieving, deleting, inserting, or replacing a segment in a data base record. As shown in Figure 1-2, level 3 segments such as segment type C are dependent for their existence on the level 2 segment type B and can not be present if the corresponding type B segment is not present in the data base record. Segment type C is called the *child* of segment type B. Thus, segment B is
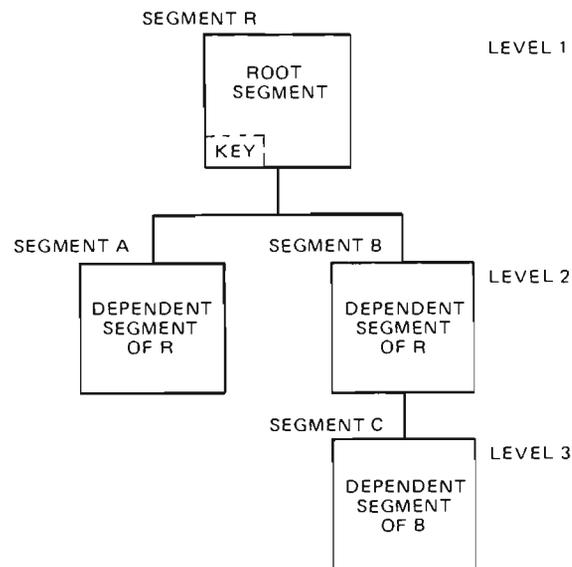


Figure 1-2. Hierarchical Data Structure

the *parent* of segment C. Each segment type can be fixed or variable length, contains logically associated fields, and has a 1 to 8 character name, such as C, which is used to reference the segment type.

### File Record Layout

A customer file record layout might appear as shown in Figure 1-3.

| CUSTOMER NUMBER | NAME | ADDRESS | SHIP-TO LOCATION | | |
|---|---|---|---|---|---|

| CUSTOMER ORDERS | | | etc. | |
|---|---|---|---|---|

Figure 1-3. Customer File Record Layout

Figures 1-4 and 1-5 illustrate the format and contents of a simple customer data base record using a hierarchical data structure. The use of multiple occurrences of a segment type is illustrated by the presence of two ORDER ITEM segments for the September CUSTOMER ORDER segment. At times a segment type may have zero occurrences. The hierarchical sequence of segments is top to bottom and left to right. Thus, the sequential retrieval for the data base structure shown in Figure 1-4 is:

1. CUSTOMER segment for Company Z.

2. CUSTOMER LOCATION segment for Southeastern Region.

3. CUSTOMER ORDER segment of Southeastern Region segment for September.

4. ORDER ITEM segment-1 for this order.

5. ORDER ITEM segment-2 for this order.

6. CUSTOMER ORDER segment of Southeastern Region segment for October.

7. ORDER ITEM segment for this order.

8. CUSTOMER LOCATION segment for Northwestern Region.

9. CUSTOMER ORDER segment of Northwestern Region segment for April.

10. ORDER ITEM segment for this order.

11. CREDIT STATUS segment.

### Rules for Data Base Structures

The rules concerning data base structures are:

- Any number of data bases may be defined.

- 1 to 20 data bases can be used by any one application program.

- A data base may consist of 1 to n data base records.

- A data base record may consist of 1 to 255 segment types (in Figure 1-4 there are 5 segment types). The segment type, CUSTOMER, is the root segment.

- Within a data base record each segment type may have 0 to n occurrences, except the root segment which can occur only once.

- A data base record may have a maximum of 15 segment levels. (The example in Figure 1-4 has 4 segment levels.)

- A dependent segment can occur only if its parent exists in the data base record.

- Each segment type has a 1 to 8 byte alphameric name and can be either fixed, or for HD organization, variable length.

- Application programs sensitive to a dependent segment must also be sensitive to its parents all the way up to and including the root segment.

- The key of the data base record is the sequence field of the root segment. It must be a fixed length field within the root segment. This key field is used by the application program to directly access data base records. *A key field of all binary 1s is reserved for use by DL/I only.*

- Although it is not required, any dependent segment which itself has children should contain a unique sequence field. The sequence field is user data within the segment that is unique for each segment within a parent. This field is used to identify a segment, and to determine where new segments are inserted. Dependent segments may or may not have a sequence field. If no sequence field is defined, segment sequence is controlled by user specified rules.

### Adding New Segment Types

The modification of the data base structure requires a new DBD, which replaces the existing DBD in the core image library.

New segment types may be added to an existing data base without affecting existing programs as long as the associated PSBs are not affected.

For HSAM and HISAM data bases, if the new segments being defined are at the "end" (that is to the right and bottom of existing segments in the hierarchy) no further action is required.

If the new segment type being defined is within the existing hierarchy, the data base must be reloaded.

CUSTOMER

| CUSTOMER NUMBER | NAME | ADDRESS |
|---|---|---|

CUSTOMER LOCATION

| LOCATION NUMBER | NAME | ADDRESS |
|---|---|---|

CREDIT STATUS

| CREDIT LIMIT | CREDIT BALANCE |
|---|---|

CUSTOMER ORDER

| ORDER DATE | ORDER NUMBER | REFERENCE DATA |
|---|---|---|

ORDER ITEM

| INVENTORY ITEM NUMBER | LINE ITEM NUMBER | QUANTITY ORDERED | QUANTITY SHIPPED | QUANTITY BACK ORDERED | ITEM AMOUNT |
|---|---|---|---|---|---|

Figure 1-4. Customer Data Base

| | | | | | | |
|---|---|---|---|---|---|---|
| 000003 | Company Z, Inc. | 6 Hyde Street | | | | CUSTOMER |
| 000010 | Southeastern Region | 715 Roundtrip Place | | | | CUSTOMER LOCATION |
| 770920 | 100700 | Third 1977 Order | | | | CUSTOMER ORDER |
| 000300 | 10 | 000080 | 000040 | 000040 | 000000000160 | ORDER ITEM |
| 000400 | 01 | 000050 | 000050 | 000000 | 000000000750 | ORDER ITEM |
| 771015 | 161293 | Fourth 1977 Order | | | | CUSTOMER ORDER |
| 000300 | 10 | 000080 | 000040 | 000040 | 000000000160 | ORDER ITEM |
| 000011 | Northwestern Region | 1220 North Ave. | | | | CUSTOMER LOCATION |
| 770415 | 100012 | First 1977 Order | | | | CUSTOMER ORDER |
| 000400 | 01 | 000050 | 000050 | 000000 | 000000000750 | ORDER ITEM |
| 000000100000 | 000001500000 | | | | | CREDIT STATUS |

Figure 1-5. Customer Data Base -- Sample Record

This simple task can be accomplished using the following procedure:

- Use the DL/I utility to unload the old data base.

- Create the new DBD.

- Use the DL/I utility to reload the new data base.

### Application Program Data Base Processing Functions

DL/I provides a set of functions that allows the application programmer to access and process data base records. Your application programmer issues a standard DL/I statement, referred to as a *call* statement, from his COBOL, PL/I, or Assembler language programs. For RPG II, your application program issues the RQDLI (request DLI) command to access the data base. Specific details regarding the coding of DLI calls are included in Chapter 4.

Data base records can be processed sequentially, skip sequentially, or in random order. If sequential or skip sequential techniques are used, the program can interchangeably use a tape or a disk data base.

The DL/I call statements allow the application programmer to:

- Retrieve a unique segment (GET UNIQUE)
- Retrieve the next sequential segment (GET NEXT)
- Retrieve the next sequential segment within the same parent (GET NEXT WITHIN PARENT)
- Replace the data in the existing segment (REPLACE)
- Delete an existing segment (DELETE)
- Insert a new segment (INSERT)
- Write a checkpoint record to the DL/I log (CHECKPOINT).

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

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

The DL/I calls are independent of the data base access method.

### *Physical Data Base Structure*

Six access methods are available for processing DL/I data bases. In all instances, the logical data structure presented to the application programmer is identical. The six access methods are:

- Simple hierarchical sequential access method (SHSAM)
- Hierarchical sequential access method (HSAM)

SHSAM and HSAM use the DOS/VS Sequential Access Method (SAM) to access physical storage.

- Simple hierarchical indexed sequential access method (SHISAM)

- Hierarchical indexed sequential access method (HISAM)

SHISAM and HISAM use the DOS/VS Virtual Storage Access Method (VSAM). A HISAM data base is composed of one key sequenced file (KSDS) and one entry sequenced file (ESDS). A SHISAM data base consists of only a key sequenced file (KSDS).

- Hierarchical direct access method (HDAM)

  HDAM consists of one entry sequenced file (VSAM ESDS).

- Hierarchical indexed direct access method (HIDAM)

  HIDAM consists of one key sequenced file (VSAM KSDS) and one entry sequenced file (VSAM ESDS).

### *Basic Segment Types in a Hierarchical Data Structure*

Figure 1-6 shows the segment types and how they are related in a hierarchical data structure. The segment types are:

- *Root Segment:* This segment is at the top of the structure. Each root segment has a key field which serves as the unique identifier of that root segment, and as such, of that particular data base record. The key field for this root segment is the customer number.

- *Dependent Segment:* The dependent segment relies on some higher level segment for its full meaning and identification.

  A *parent/child* relationship exists between a segment and its immediate dependents.

- *Twin Segment:* Multiple occurrences of a particular segment type under the same parent are called twin segments.

### *Sequence Fields and Access Paths*

To identify and provide access to a particular data base record and its segments, DL/I uses *sequence fields*. Each segment normally has one of its fields denoted as the sequence field. Although not required, it is a good practice to make sequence fields unique in value for each occurrence of a segment type below its parent occurrence. However, not every segment type need have a sequence field defined. Particularly important is the sequence field for the root segment, because it serves as the identification for the data base record. DL/I provides a fast, direct access path to the root segment of the data base record based on this sequence field.

Note: The sequence field is often referred to as the *keyfield* or simply *key*.
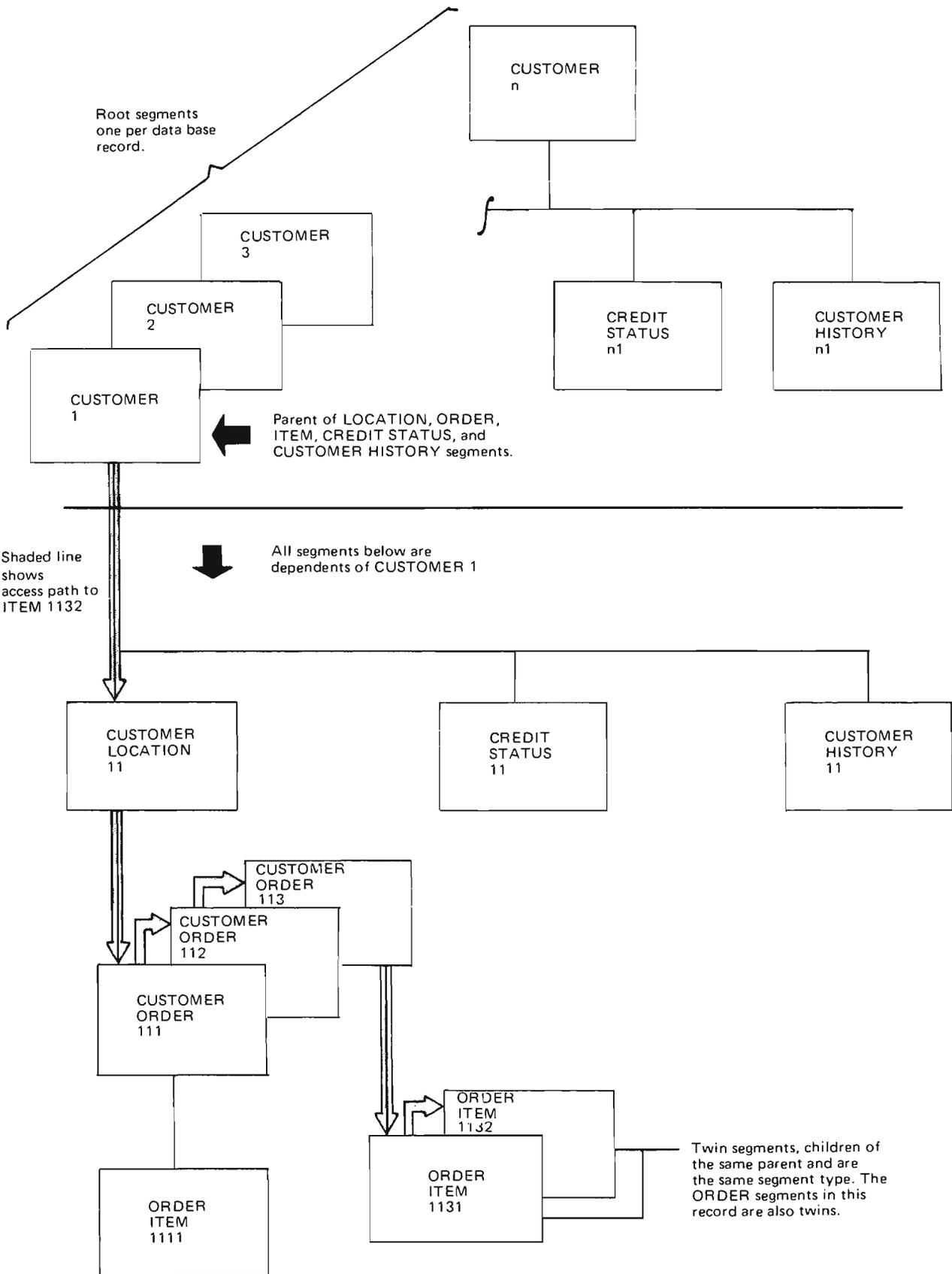
Figure 1-6. Segment Types and Their Relationships in a Hierarchical Data Structure

Figure 1-6 shows as a shaded line an example of an access path to the ORDER ITEM 1132 segment. It must always start with the root segment. This is the access path as used by DL/I. The application program, however, can directly request a particular ORDER ITEM segment of a given CUSTOMER ORDER to a given CUSTOMER LOCATION for a specific CUSTOMER in a single DL/I request by specifying a sequence field value for all four segment levels.

### Logical Relationships

In addition to the basic DL/I facilities discussed so far, DL/I provides a facility to interrelate segments from different hierarchies, or within the same hierarchy. In doing so, new hierarchical structures are defined that provide additional access capabilities to the segments involved. The segments can belong to the same or different data bases. A new data base can be defined called a *logical data base*. This logical data base allows presentation of a new hierarchical structure to the application program.

The basic mechanism used to build a logical relation is to create a dependent segment as a *logical child* that points to a second parent, the *logical parent*.

In Figure 1-7, the logical child segment ORDER ITEM exists only once, yet participates in two hierarchical structures. It has a *physical parent*, CUSTOMER ORDER, and a *logical parent*, INVENTORY ITEM. The data in the logical child segment, if any, is called *intersection data*.

By defining two additional logical data bases, two new logical data structures as shown in Figure 1-8 can be made available for application program processing. The ORDER ITEM/INVENTORY ITEM segment in Figure 1-8A, is a *concatenated segment*. It consists of the logical child segment plus the logical parent segment. The ORDER ITEM/CUSTOMER ORDER segment of Figure 1-8B is also a concatenated segment, but it consists of the logical child segment plus the physical parent segment. Logical children with the same logical parent are called *logical twins*. In this case, all ORDER ITEM segments which point to the same INVENTORY ITEM segment are logical twin segments. As can be seen in Figure 1-7, this logical child has two access paths. One via its physical parent, the *physical* access path, and one via its logical parent, the *logical* access path. Both access paths are maintained by DL/I and can be concurrently available to one program. When the logical child segment has two access paths as in Figure 1-7, the logical relationship is called *bidirectional*. DL/I also provides for *unidirectional* logical relationships in which case the logical child segment can be accessed only via its physical parent.

Because the DL/I logical relationship function may not be required for your first DL/I application, we will

deal with it separately in this manual. To show the use of logical relationships, we will use phase 2 of the sample application as described in Chapter 2.

### Secondary Indexing

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

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

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

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

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

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

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

Because you might not need the secondary index function for your initial data base requirements, we separate its discussion throughout the manual. The use
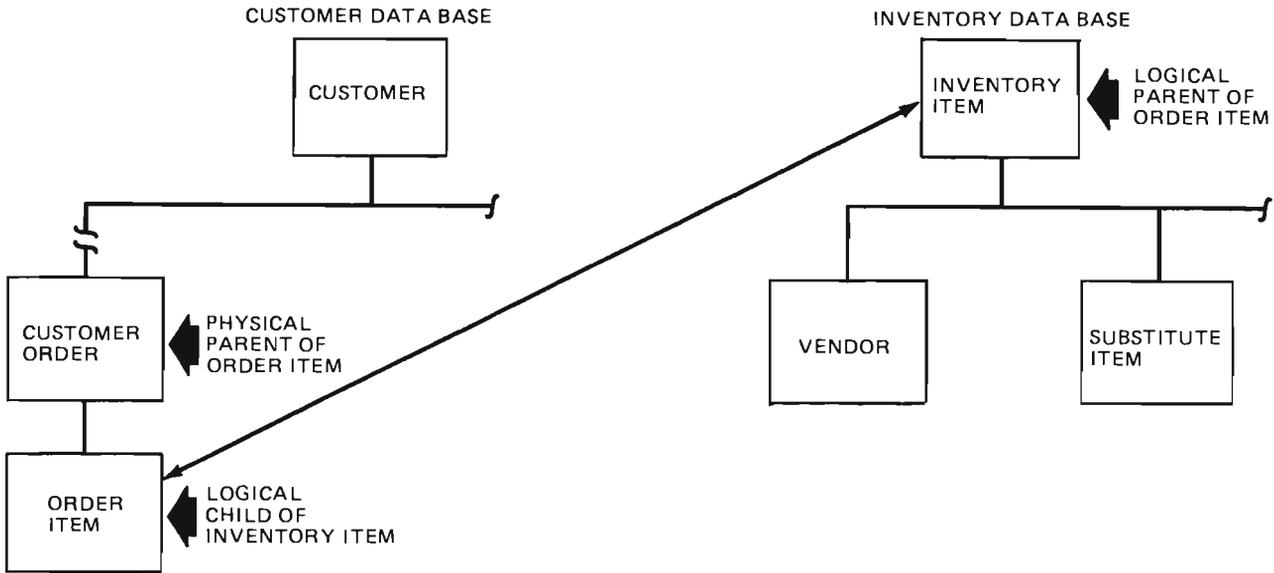
Figure 1-7. Two Logically Related Data Bases, CUSTOMER and INVENTORY



A. LOGICAL DATA STRUCTURE -
   CUSTOMER ORDER - INVENTORY ITEM

B. LOGICAL DATA STRUCTURE -
   INVENTORY ITEM - CUSTOMER ORDER
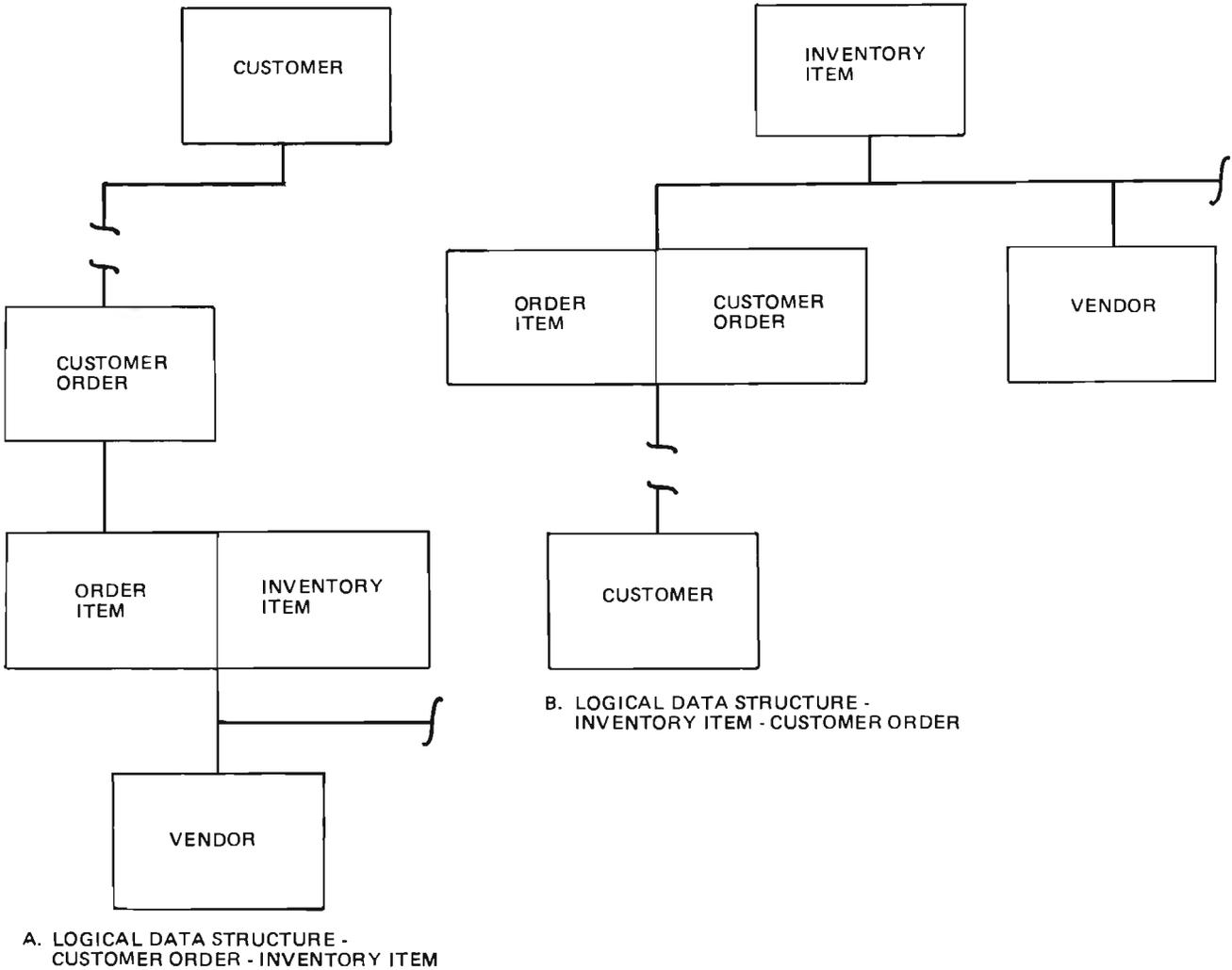
Figure 1-8. The Logical Data Bases After Relating CUSTOMER and INVENTORY Data Bases
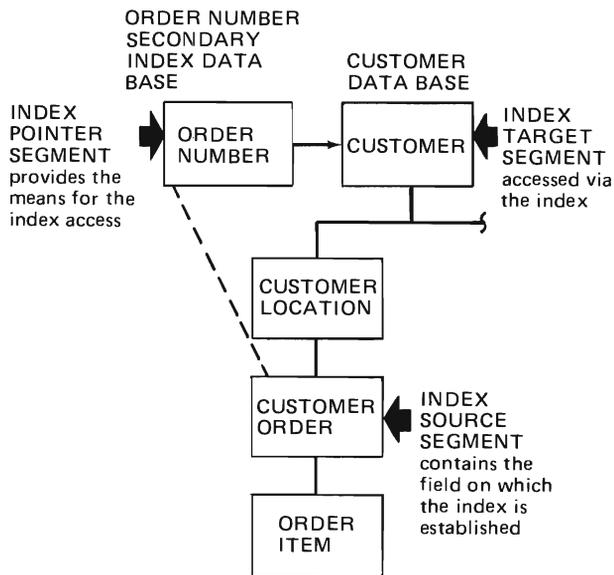
Figure 1-9. A Data Base and its Secondary Index

of secondary indexing is shown in the phase 3 sample application as described in Chapter 2.

# Data Base Definition

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

## DBD (Data Base Description)

The first level is called the DBD (data base description). It describes most of the file characteristics you must put into every non-data-base DOS/VS program. Each DBD is created from statements you provide. The statements define the hierarchical data structure and physical organization of the data base. These statements are assembled as the DBD generation procedure.

The DBD describes the contents of the data base, the names of the segments, their hierarchical relationship, and the physical organization and characteristics of the file. You can think of the DBD as the master description of everything that is in the data base.

The DBD provides DL/I with the mapping from the application data structure of the data base used in the application program to the physical organization of the data used by DOS/VS. The data structure can be re-mapped into a different physical organization without application program modification. Other application data can also be added to this data base and not require a change to the original application programs. The concept of the DBD reduces application program maintenance caused by changes in the data requirements of the application. The three types of DBDs are:

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

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

- The *index DBD* allows the definition of an alternate access path into a physical or logical DBD.

The process of generating a DBD is called *data base description generation* (DBDGEN).

## PSB (Program Specification Block)

The second level of data base definition, the PSB (program specification block), defines the application data structure for each application program. It is created from statements you provide for each of your application programs. The PSB defines which segments of the data base a specific program requires (the application data structure required by that application program). A PSB contains one or more *PCBs* (program communication blocks), one for each hierarchical data structure the program intends to use. Each PCB defines the hierarchical (sub)structure the program *sees* from the physical or logical data base. It specifies for each segment the kind of access allowed by the program (read only, update, insert, and delete). There is at least one PSB for every program that uses the data. An on-line program may use more than one PSB; more than one program may use the same PSB. You can think of the PSBs as describing the logical data needed for the program (usually a subset of the entire data base). The process of generating a PSB is called *program specification block generation* (PSBGEN).

# User Responsibilities

## System Installation

The user of DL/I has two primary responsibilities:

1. The development of data processing applications that use DL/I. This includes application programs, as well as backup and recovery procedures using the DL/I utilities.

2. The structuring of his data processing environment:

- Data Bases
- Batch Processing Programs
- CICS/VS as the teleprocessing support for transaction processing programs

## *Data Base Administration*

The centralization of data and control of access to this data is essential to a data base management system. One of the advantages of this centralization is the availability of consistent data for more than one application. This dictates a tighter control of that data and its usage. Responsibility for an accurate implementation of control lies with the data base administration function. Although the data base administration function is usually performed by a person called the data base administrator, this function may actually be performed by a group of individuals with experience in both application and system programming. The duties of the data base administrator are to:

- Identify, define, implement, and maintain data base specifications
- Control and monitor the use of data base information
- Integrate application requirements for common information
- Provide for efficient application migration from a batch to online environment
- Establish a reliable and efficient data base operating environment
- Identify data base security requirements
- Monitor and evaluate performance

The data base administration function can be separated into three general areas:

- Data Base Analysis
- Data Base Management
- Data Base Operations

### Data Base Analysis
Responsibilities:

- Design data base structures that will be easy to program, and use available hardware resources efficiently.
- Establish data base recovery and reorganization procedures.
- Authorize and control use of data bases.
- Establish a data base environment for testing use.

### Data Base Management
Responsibilities:

- Create and maintain a data element dictionary. This dictionary should contain each identifiable data element together with its attributes, source, edit and integrity responsibility, and a cross reference to all programs and data bases that use it.
- Determine file organization schemes.
- Evaluate how and by whom data is used. On operational data bases, a use profile should be maintained to determine if design decisions remain valid.
- Define, code, execute, and control all PSB and DBD generations.

### Data Base Operations
Responsibilities:

- Monitor all operational data base activity. The foremost goal is to preserve the integrity of the data base system.
- Based on results of the monitoring function, make recommendations for changes to the data base/data communications environment, configuration, or procedures that will improve performance, recoverability, and integrity.
- For online system operation, initialize, terminate, monitor, and control the online data base/data communications environment.
- Assist in the procedures required to properly recover from a compromised data base, should this occur.

# Project Approach

The implementation of a DL/I application is most successfully done using the project approach. With this approach, you assure that adequate planning is done in a timely manner, stating all the necessary steps for the design, test and installation of the application. For more complex applications, you may want to try using a project team with a definition of the tasks and responsibilities of all parties involved, if possible.

## *Project Cycle*

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

- Preliminary investigation
- Planning
- Design
- Implementation

- Testing
- Operation

Figure 1-10 shows the relative manpower requirements for each of the phases.

Following is a brief introduction to each of the phases:

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

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

*Planning:* A project plan is established. A Project team is formed and the tasks and responsibilities of individuals and departments are defined. 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, detail designs are worked out together with a test plan.

*Test:* Both unit test and integrated system tests are performed, resulting in the acceptance test.

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

*Administration:* Another important aspect is project administration. The timely and accurate planning for and establishing of standards and guidelines is mandatory for an efficient project implementation and later maintenance. Most organizations already have standards which should be ex-

tended 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.

### Gross PERT Chart

Figure 1-11 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 PERT chart (Figure 1-11).

*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 and people involved for each activity. Figure 1-12 shows an example of such a Gantt chart.
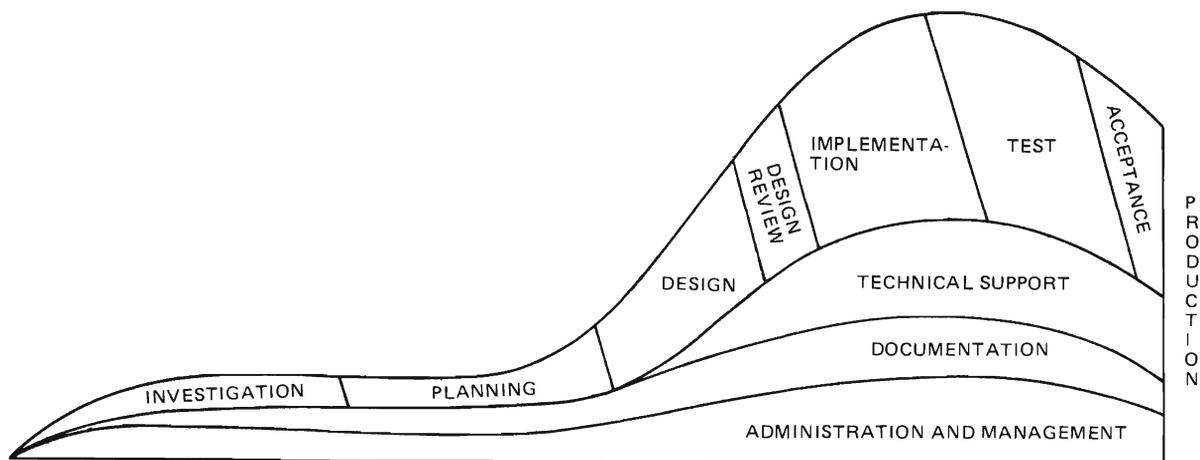


Figure 1-10. The Project Cycle

The Gantt chart should clearly state the actual days/months 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 or PL/I 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/programs are established for the conversion of existing data files.

*Recovery and Reorganization (300-440-650/640-700):* A timely plan for recovery and reorganization can avoid later redesigns and reprogramming. These procedures, although rarely needed, are vital to the data base integrity and availability. Therefore, a thorough test plan must be made and carried out before

production starts. The production staff should be carefully trained in problem determination and the secure and accurate execution of such procedures. An incomplete treatment of this topic is the most common source of problems 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):* Detail flow. charts are established. The data base call sequences are defined in a standard fashion.

*Test Plan (400-600):* A detail 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:
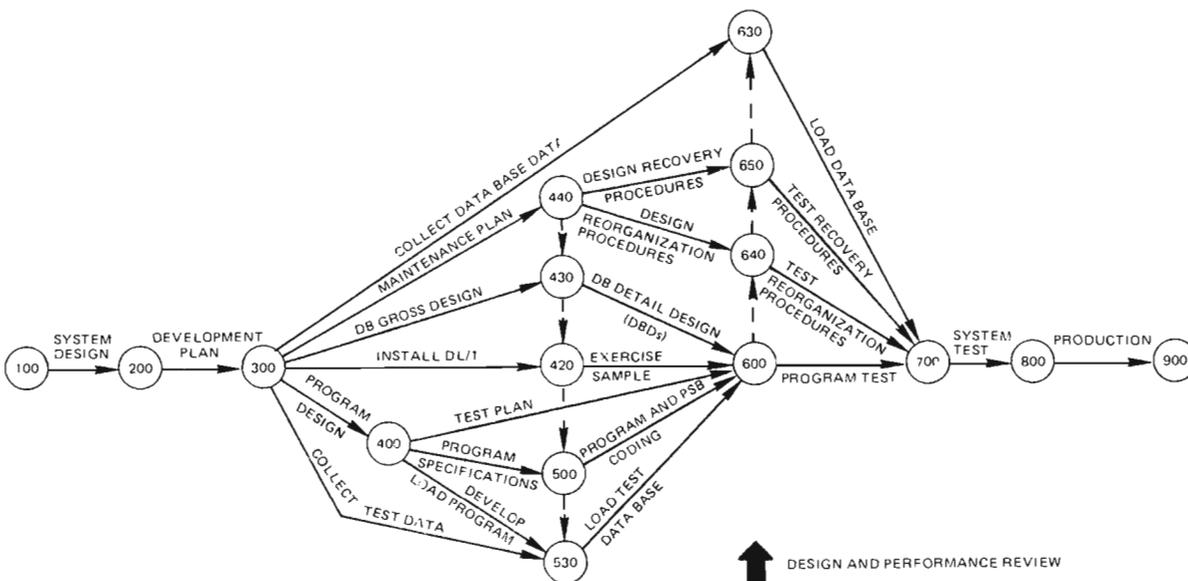


Figure 1-11. DL/I Installation Plan PERT Chart

- 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 to future trouble shooting, application modification, and application extensions.

# Implementation Overview

Based on the information presented in this chapter, the following steps are necessary to implement a data base:

- Define the application requirements.

- Design the physical data base.
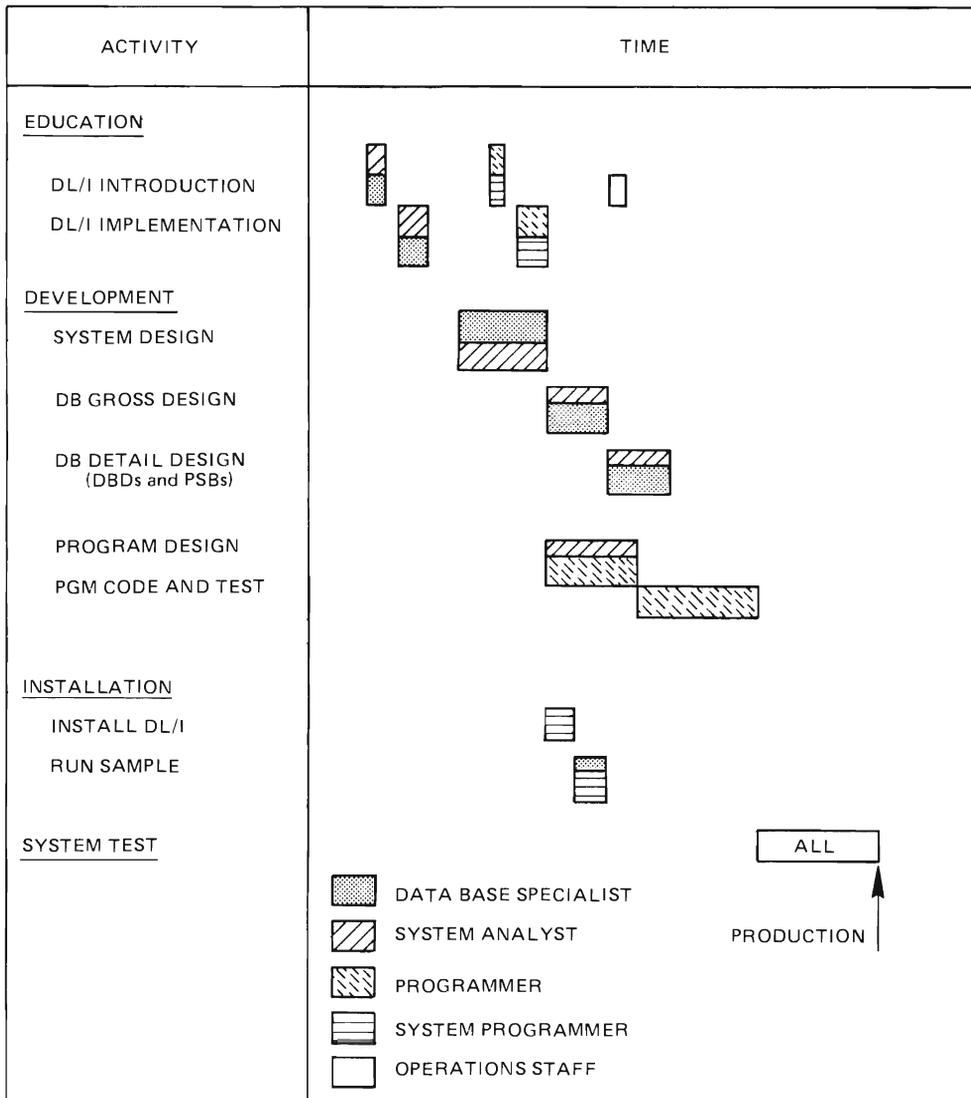


Figure 1-12. Sample Gantt Chart

- Define the logical relationships.
- Design the logical data bases.
- Define the secondary indexes.
- Code the DBDs of the physical and logical data bases.
- Code the PSBs.
- Use the DBDs and PSBs to build the control blocks (ACB Generation).

- Define VSAM data sets for the physical data bases.
- Load the physical data bases (User Written Application).
- Use the DL/I utilities to resolve the logical relationships between the data bases.
- Execute the applications.

As in almost any system implementation, the design is the most challenging task to be performed. Yet, a designer is often bound to a time limit and does not know all future requirements. To cope with these problems, a designer needs a good plan and proper techniques.

DL/I itself is not an application. It is a data management control system that provides the method of constructing data base/data communication applications. To simplify the use of this manual as a tool to guide you in your data base design, a sample application is used throughout this manual as a base for all the examples. This sample is intended to guide you in a normal sequence through all the steps needed for successful implementation of an application using DL/I.

The sample application is an online Customer Order Processing Program using DL/I in conjunction with CICS/VS. However, the examples for, and discussion of, data base and application program design are also valid for batch processing considerations. Any material presented that applies to online considerations only is clearly defined.

The sample application uses two data bases: Inventory, and Customer. The Inventory data base is designed first based on existing (non-data-base) ISAM and/or VSAM files already in use at the installation (for example, Inventory Master, Item Location, and Vendor). This data base is used initially for batch applications, but the installation has plans to eventually relate this data base to another data base (Customer), using logical relationships to eliminate redundant data. To gain an alternate path to retrieving the data, the installation will also use secondary index data bases.

Finally, the data bases will be placed in an online environment using the DL/I interface to CICS/VS.

The fact that the examples used in this manual are directed towards a specific application should not preclude your using DL/I for other applications. Actually, the basic data structure and processing shown in these examples can readily be adapted to other applications.

Installing a data base management system involves two separate processes:

- Data base design
- Data base implementation.

Data base design is a user process of determining which data base structures will satisfy the organization's application program data needs while satisfying an organization's data security, integrity, and redundancy objectives.

# Chapter 2: Data Base Design

Data base implementation is a user process of creating, tuning, and maintaining data bases. This process includes the selection of DL/I access method options, storage allocation, and other performance and tuning options. The implementation process is fully described in the next chapter.

This chapter introduces the concepts, techniques and guidelines for the designing of DL/I data structures. It is aimed at those individuals who are designing their first DL/I data base.

**Data Base Design Objectives**
Just as reasons for installing a data base management system vary among users, data base design objectives will also vary. Some objectives of a data base design are to:

- Provide an access path to the stored data required by an application.

- Isolate current applications from the impact of future applications on the same data base.

- Support data security objectives

- Support data redundancy objectives

- Support multiple applications, making trade-offs in the best interest of the organization as a whole.

Each user must determine the applicability and priority of the design objectives to the current design effort. The more limited and simple the objectives the more simple the task of data base design.

In addition to the guidelines provided in this chapter, the data base design process may also be accomplished by using one of the application development aids available that support data base design. Examples are:

- DOS/VS DBDA (Data Base Design Aid) - PP number 5748-XX4

  The DBDA is a collection of programs that assist in performing a major portion of the data base design process. The DBDA uses your installation's input that describes the data base requirements and produces a structural model of the data base by mapping the data elements into segments and a hierarchical structure that shows the minimum set of relationships required by an integrated data base (one which serves many application programs).

- DOS/VS DB/DC Data Dictionary - PP number 5746-XXC

The DB/DC dictionary is a collection of programs that provide a data definition interface, store the data definitions, provide displays and reports of the defined data and, upon request, produce definitions for the DL/I control block generation process.

Data base design aids are also available for certain industry applications. If your installation intends to use an IBM Chain File Bridge PP or some of the industry application programs available, you should investigate the data base design aids offered by these programs.

Because data base design is an area where there has been little standardization, there has been no consistent vocabulary for describing the concept involved. The reader who intends to utilize one of the above facilities may wish to skip the rest of this chapter.

## About This Chapter
This chapter consists of three different sections:

- *Section 1. The DL/I Sample Application* introduces the sample applications in detail. It sets the requirements and the environment for the actual data base design process. It provides the background for the examples used in the two following sections.

- *Section 2. The DL/I Data Base Facility* introduces the functions of DL/I available to the data base designer.

- *Section 3. The Data Base Design Process* introduces the concepts, techniques and guidelines for the designing of data bases with DL/I. It is aimed at those individuals who are designing their first data bases with DL/I.

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

- Phase 1: Basic data bases

- Phase 2: Data bases with logical relationships

- Phase 3: Data bases with secondary indexes

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

## Section 1: DL/I Sample Application
The sample application documented in this manual is for a fictitious company (a wholesale distribution firm) that offers a wide variety of electronic components. The components are purchased from various vendors and sold to customers. Most customer orders arrive by telephone. Because of this and the growth in numbers of orders and variety of items, an upgrade of the existing inventory control and customer order applications became necessary. It was decided to build a new system which integrated these applications utilizing the DL/I data base approach.

Some objectives for the new application were:

- Implement:

  - Inventory control with its associated purchase order processing

  - Customer order processing

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

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

- Provide accurate entry of both purchase orders and customer orders with respect to items in stock

- Provide a base for online processing of orders and inquiries

The implementation of this system will be the common thread throughout the examples used in this manual.

### Inventory Data Base
Information about items in stock is managed by the inventory control department. All data will be stored in the Inventory data base. This data base consists of one record for each item the company stocks. Each record identifies:

- Standard information for all items

- Stock location information for those items that are in stock

- Purchase information for those items that need restocking

### Customer Data Base
Information about customer orders is managed by the sales department. All order data will be stored in the Customer data base. It consists of one record for each customer order. Each record identifies:

- Standard information for each order and customer

- Order detail information for each ordered item

- Shipment information for this order

A link is required to the Inventory data base because it is necessary to know which parts are on order by each customer and vice versa.

## Naming Conventions Used in the Sample Application

The naming conventions used in the sample application observe the following format:

A_B C D E_F_G H

→ Reserved for DOS/VS, CICS/VS, and DL/I

→ Description (within application)

→ Application (within application area)

→ Category of name (DBD, PSB, etc)

→ General Application Area

**Example:**

S_T D C X_1_C b

→ INDEX 1 VSAM CLUSTER

→ CUSTOMER

→ DBD
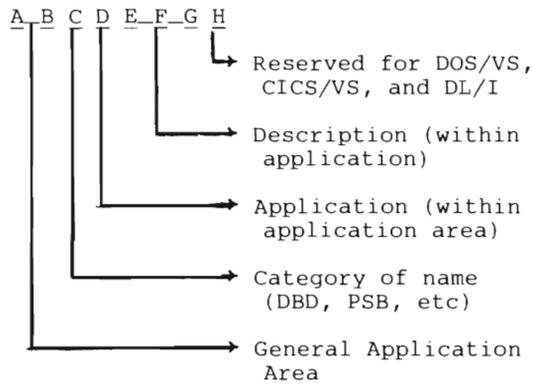
→ SAMPLE TRANSACTION
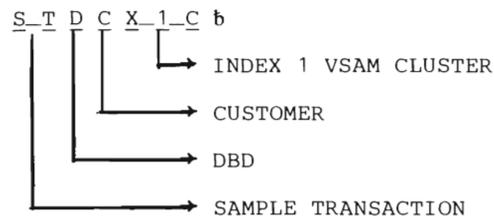
**Thus:** The name STDCX1C represents the VSAM cluster definition for Index 1 of the CUSTOMER data base within the Sample Transaction set of applications.

### Naming Conventions - Application Area

ABCDEFGH
ST - Sample Transaction

### Naming conventions - Categories

```
ABCDEFGH
A - not used
B - PSB
C - Real Logical Child
D - DBD
E - not used
F - Field
G - Length Field (Variable
    Length Segment)
H - Segment Search Argument
I - Indexing Segment
J - not used
K - Concatenated Key Field
L - Logical (Concatenated) Segment
M - not used
N - not used
O - not used
P - Destination Parent Segment
Q - Sequence Field
R - Index Search Field
S - Segment
T - not used
U - Index Duplicate Data Field
V - Virtual Logical Child
W - not used
X - Indexed Field
Y - Indexing Field
Z - Index User Data Field
```

### Naming Conventions - Applications

```
ABCDEFGH
C - Customer
I - Inventory
```

### Naming Conventions - DBD

ABCDEFGH

→ DBD TYPE

'D' - DBD

```
DBP - Physical DBD
DBL - Logical DBD
XnP - Index DBD
      (n = 0-9)
DBC - Physical DBD
      Cluster (VSAM)
DBI - Physical DBD
      Index (VSAM)
DBD - Physical DBD
      Data (VSAM)
XnC - Index DBD
      Cluster (VSAM)
XnI - Index DBD
      Index (VSAM)
XnD - Index DBD
      Data (VSAM)
```

## Sample Application Description - Phase 1

The phase 1 data base is the Inventory data base. It contains the data the installation needs to monitor stock status and to process customer orders. The data base contains four segment types as shown in Figure 2-1.
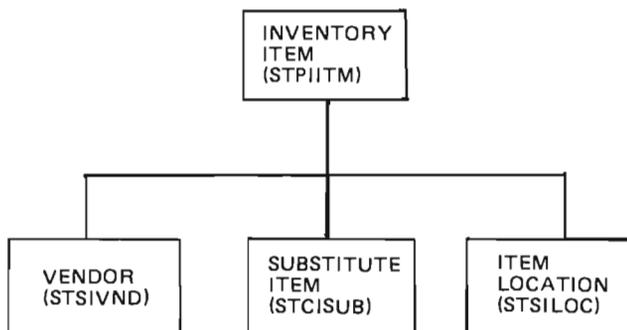


Figure 2-1. Inventory Data Base

The segments and the data elements they contain are:

### Inventory Item Segment (STPIITM): contains the item number, description, quantity on hand, quantity on order, unit price, and unit of issue.

| Name | Description | Length (bytes) |
|---|---|---|
| STQIINO | Item Number | 6 (key) |
| STFIIDS | Description | 25 |
| STFIIQH | Quantity on hand | 6 |
| STFIIQO | Quantity on order | 6 |
| STFIIQR | Quantity reserved | 6 |
| STFIIPR | Unit price | 6 (3 dec. places) |
| STFIIUN | Unit of issue | 1 |

### Vendor Name Segment (STSIVND): contains the vendor number, name, and three lines of address.

| Name | Description | Length (bytes) |
|---|---|---|
| STQVVNO | Vendor Number | 6 (key) |
| STUVVNM | Vendor Name | 25 |
| STFVVA1 | Loc. Address Line 1 | 25 |
| STFVVA2 | Loc. Address Line 2 | 25 |
| STFVVA3 | Loc. Address Line 3 | 25 |

### Substitute Item Segment (STCISUB): This segment contains the number of the item (if any) that can be substituted for the item referenced in this record. The field is:

| Name | Description | Length (bytes) |
|---|---|---|
| STQCCNO | Sub. Item Number | 6 (key) |

### Inventory Location Segment (STSILOC): This segment contains the Inventory location number for the item, and the quantity. The fields are:

| Name | Description | Length (bytes) |
|---|---|---|
| STQILNO | Inventory Loc. No. | 6 (key) |
| STFILQT | Quantity | 6 |

## Sample Application Description - Phase 2

The second data base is the Customer data base. For phase 2, this data base will be related to the Inventory data base using logical relationships. Details on how this is done are presented later in this chapter. The customer database contains the customer information the installation needs to begin processing a customer order, such as customer name, address, order information, and credit status. It contains six segment types as shown in Figure 2-2.
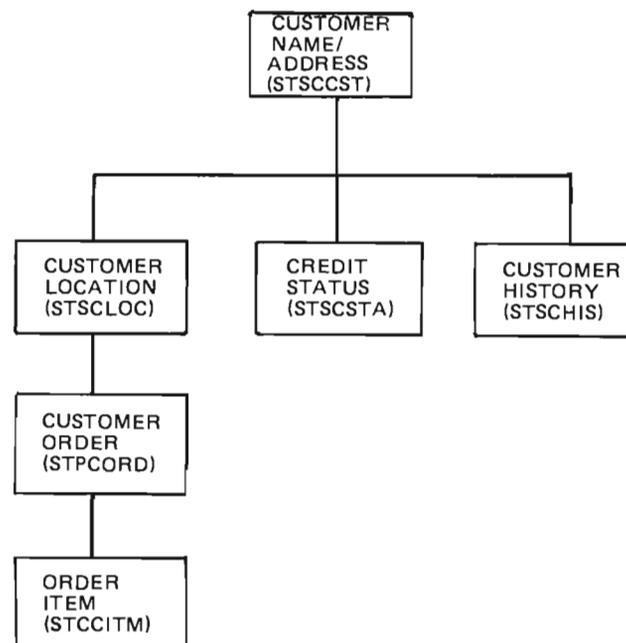


Figure 2-2. Customer Data Base

The segments and the data elements they contain are:

### Customer Name and Address Segment (STSCCST): contains the customer number, customer name, and a three line address.

| Name | Description | Length (bytes) |
|---|---|---|
| STQCCNO | Customer Number | 6 (key) |
| STUCCNM | Customer Name | 25 |
| STFCCA1 | Cust. Address Line 1 | 25 |
| STFCCA2 | Cust. Address Line 2 | 25 |
| STFCCA3 | Cust. Address Line 3 | 25 |

*Customer Location Segment (STSCLOC):* similar in format to the Customer Name and Address segment. It provides multiple 'Ship to' locations for a customer. It contains the location number, location name, and three lines of address.

| Name | Description | Length (bytes) |
|---|---|---|
| STQCLNO | Location Number | 6 |
| STFCLNM | Location Name | 25 |
| STFCLA1 | Loc. Address Line 1 | 25 |
| STFCLA2 | Loc. Address Line 2 | 25 |
| STFCLA3 | Loc. Address Line 3 | 25 |

*Customer Order Segment (STPCORD):* A segment exists for each active (open) order. This segment contains totals and reference information unique to the order. Fields are: Order Date, Order Number, Order Reference Data, Item Count, and Total Order Amount.

| Name | Description | Length (bytes) |
|---|---|---|
| STQCODN | Order Date (yr-mo-day) and Order Number | 12 |
| STFCORF | Order Reference Data | 25 |
| STFCOIC | Order Item Count | 2 |
| STFCOAM | Order Amount | 12 |

*Order Item Segment(STCCITM):* One segment exists for each line item of the order. It contains quantity and amount fields unique to the line item. The fields are: Inventory Item Number, Line Item, Quantity Ordered, Quantity Shipped, Quantity Back Ordered, and Order Amount.

| Name | Description | Length (bytes) |
|---|---|---|
| STKCIIN | Inventory Item Number | 6 |
| STQCILI | Line Item Number | 2 |
| STFCIQO | Quantity Ordered | 6 |
| STFCIQS | Quantity Shipped | 6 |
| STFCIQB | Quantity Back Ordered | 6 |
| STFCIAM | Item Amount | 12 |

*Customer Status Segment (STSCSTA):* contains information pertaining to the credit status of the customer. This information is placed in a separate segment so that access to it can be restricted to those who are authorized to use it. This data security is provided using the segment sensitivity feature of DL/I. The segment contains two fields: Credit Limit and Credit Balance.

| Name | Description | Length (bytes) |
|---|---|---|
| STFCSCL | Credit Limit | 12 |
| STFCSBL | Credit Balance | 12 |

*Customer History Segment (STSCHIS):* this segment is similar in format to the Customer Open Order Item segment. It is used to retain summary information about previous (closed) orders. This segment is defined as a variable length segment to provide flexibility in recording the order status field, STFCLOS, while optimizing storage requirements for the segment. The first field in a variable length segment is used to record the length of the segment. See "Variable Length Segments" later in this chapter for details.

| Name | Description | Length (bytes) |
|---|---|---|
| STGCSL | Segment Length | 2 |
| STQCHDN | Order Date (yr-mo-day) and Order Number | 12 |
| STFCHRF | Order Reference Data | 25 |
| STFCHIC | Order Item Count | 2 |
| STFCHAM | Order Amount | 12 |
| STFCLOS | Order Status | 77 |

## Sample Application Description - Phase 3

The phase 3 data base environment includes the addition of secondary indexes to the customer and inventory data bases. This is done in the sample application to allow alternate access paths to the data as required for the online order/inquiry system. Details on how this is done are included later in this chapter.

## DL/I Sample Programs

In DL/I Version 1.3, several new sample data bases and sample programs were added to demonstrate the use of DL/I with logical relationships and secondary indexes, and to allow you to test DL/I in an online environment. The sample programs are used to load, access, and print or display the contents of the Customer and Inventory data bases as described in this manual for the Phase 3 environment. All DBD, PSB, and ACB generation control statements are included for the physical, logical, and secondary index data bases.

The sample jobstream also includes the access method services DEFINE commands for VSAM and the utilities used to create the secondary indexes and resolve the logical relationships. The sample application programs are:

- DL/I Online Sample Load Program - DLZSAM40

  This program loads the Customer and Inventory data bases for the DL/I online sample program.

- DL/I Online Sample Print Program - DLZSAM50

  This program prints the Customer and Inventory data bases as loaded by DLZSAM40.

- DL/I-CICS/VS Sample Online Application - DLZSAM60

  This program is an interactive DL/I-CICS/VS online application designed to allow customer order inquiry and customer order entry to the Customer and Inventory data bases defined for this sample application.

The online sample application also includes a program that defines the format of the displays to the 3270

screen as used by DLZSAM60. See Chapter 8, "DL/I Online Sample Application," for more information.

# Section 2: DL/I Data Base Facility

This section of Chapter 2 provides an introduction to DL/I functions and their use. It is the main source of reference for the data base administrator. This section is subdivided into two parts. The first part provides the necessary insight into DL/I for doing the data base design. The second part provides details for the implementation of the data base(s). Each part has three sections. These sections cover the following main data base facilities:

- Physical data bases and access methods

- Logical relationships

- Secondary indexes

## *Physical Data Bases and Access Methods*

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

The access methods are:

- Simple Hierarchical Indexed Sequential Access Method (Simple HISAM)

- Hierarchical Indexed Sequential Access Method (HISAM)

- Hierarchical Indexed Direct Access Method (HIDAM)

- Hierarchical Direct Access Method (HDAM)

- Simple Hierarchical Sequential Access Method (Simple HSAM)

- Hierarchical Sequential Access Method (HSAM)

The data base type, its access method, and structure are defined in the DBD (data base description). To use a data base in an application program, you must provide a PSB (program specification block). The PSB specifies the data base(s) to be used and the kind of usage required. DBDs and PSBs are created during *data base description generation* (DBDGEN) and *program specification block generation* (PSBGEN) respectively. This is discussed in detail later in this chapter.

Before discussing each of the access methods further, this section will first elaborate on some of the basic DL/I concepts that were introduced in Chapter 1.

## *DL/I Data Base Record*

The DL/I data base record as shown in Figure 2-3 consists of one root segment and a number of dependent segments. Each dependent segment can have a variable number of occurrences below its parent occurrence.

In its most elementary form, this record could be stored in one or more physical records. In principal, the segments would be stored in their hierarchical sequence, as shown in Figure 2-4.

Note that Figure 2-4 is a simplification. In reality DL/I uses more elaborate storage organizations to allow for efficient replacement, insertion, and deletion of segment occurrences. Generally available functions include for example:
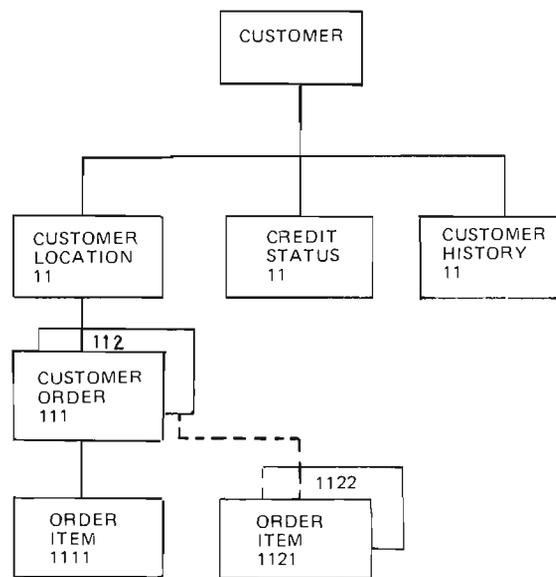


Figure 2-3. A DL/I Data Base Record
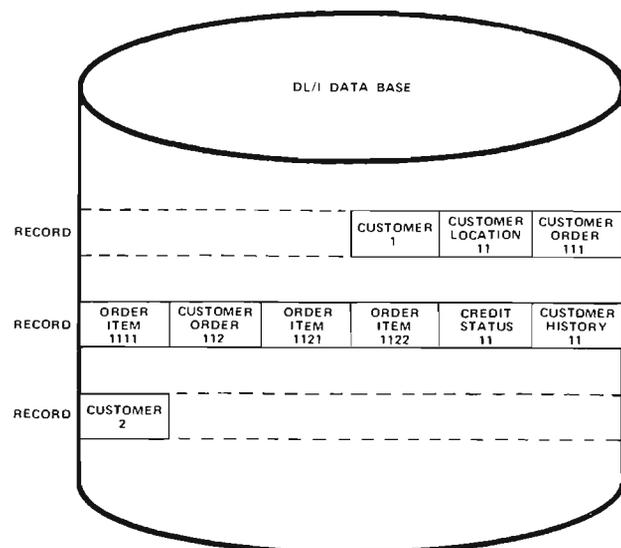


Figure 2-4. A DL/I Data Base Record in Physical Storage

- Space reuse of deleted segments
- Direct or key-sequenced access for the root segment based on the root segment sequence field (=key field).

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

### Segment Format

A segment in a DL/I data base record consists of a prefix and data portion. The prefix contains the system data used by DL/I and is not presented to application programs. The data portion contains the user data as seen by the application program. The prefix of a segment contains a segment code, a delete byte, and optional pointers. Figure 2-5 illustrates the segment format. The one byte *segment code* is used to identify each segment stored in a DL/I data base. It is the first byte of the prefix. The second byte is the *delete byte*. It is used to maintain the status of a segment within the data base.

Note: SHSAM and SHISAM data bases can contain only one segment type. (The root segment for the data base record.) These data base organizations do not contain segment prefixes.

Figure 2-5 shows that segments also contain a pointer area. Pointers are used in HDAM and HIDAM data bases for linking the segments within one data base record in their hierarchical order. Pointers are also used to link segments involved in logical relationships, and to implement index pointing.

The segment types in each data base are coded in hierarchical sequence from 1, the root segment, up to 255, as shown in Figure 2-6.
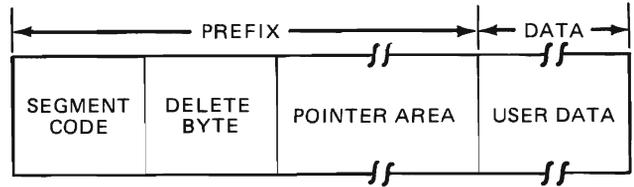


Figure 2-5. Segment Format

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

### Concatenated Key

The *concatenated key* of a segment consists of all sequence fields from the root down the hierarchical path to and including the sequence field of the segment itself as shown in Figure 2-7.

### Calls and Data Base Positioning

To help gain a better understanding of each particular data base organization, a basic description of the DL/I calls used to process segments in a data base follows.
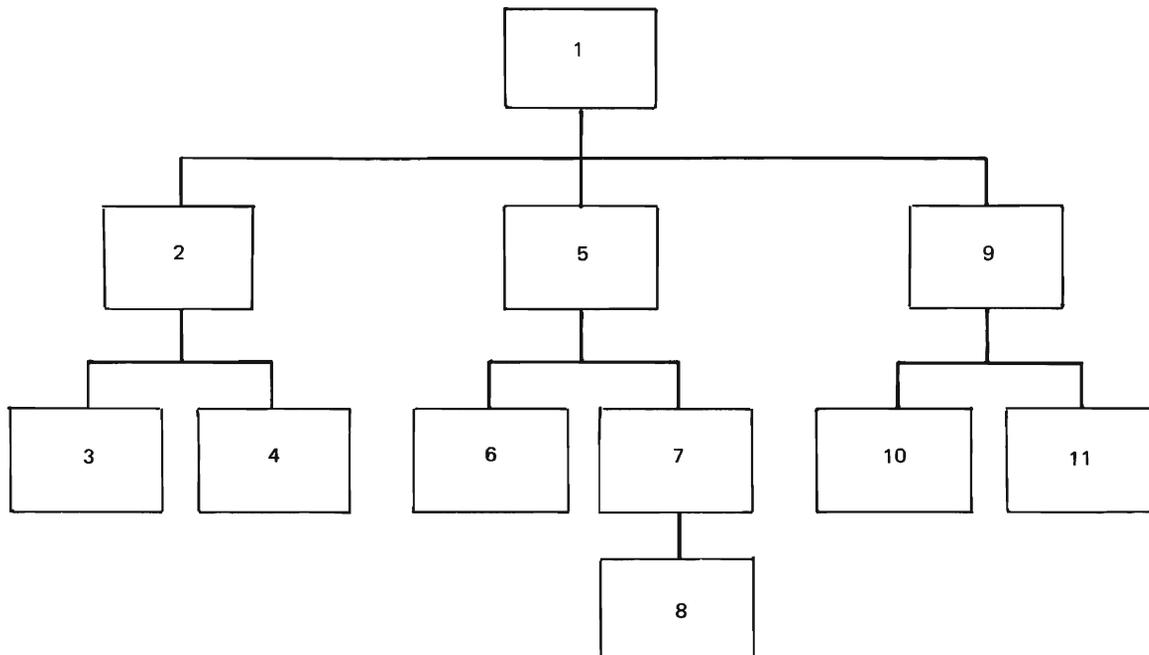


Figure 2-6. Segment Types Numbered in Hierarchical Sequence
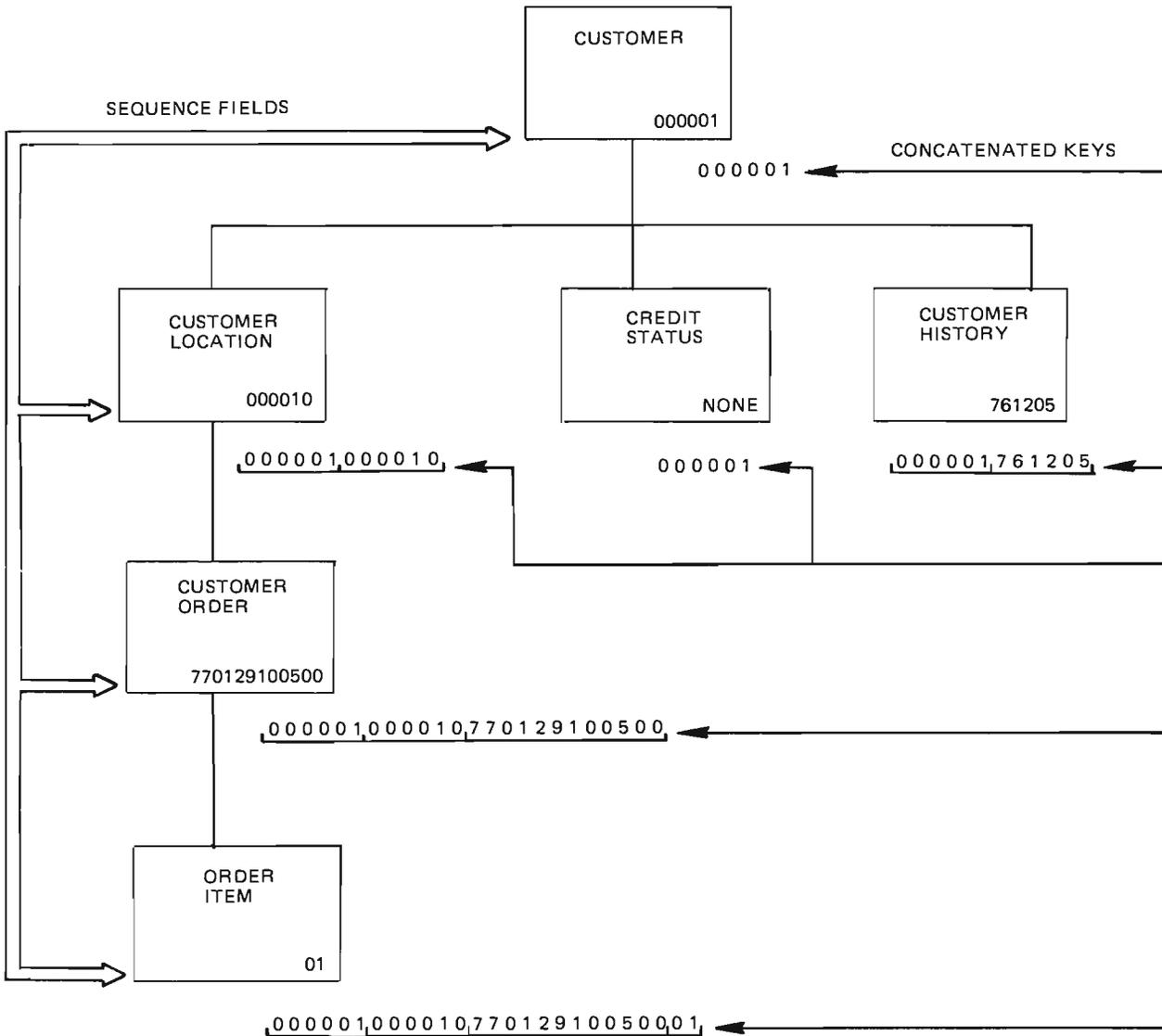
Figure 2-7. Concatenated Keys

The segments in a DL/I data base are processed through calls issued by an application program. Calls are issued to get, insert, delete, or replace a segment or a path of segments. A call references a parameter list which includes all data required by DL/I to complete the call. Included in the list are a function code and, optionally, one or more SSAs (segment search arguments). The function code states the call to be performed, and the SSAs define the segments along the hierarchical path down to, and including the segment to be processed. A call is *unqualified* when no SSA is included with the call, and is *qualified* when one or more SSAs are included. A brief description of SSAs follows. For more detailed information, refer to Chapter 4, "Processing Data Bases".

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

The basic DL/I calls are:

- GU (get unique) call is used to retrieve a specific segment or path of segments from a data base. At the same time it establishes a position in a data base from which additional segments can be processed in a forward direction.

- GN (get next) call is used to retrieve the next desired segment or path of segments from a data

base. The get next call normally moves forward in the hierarchy of a data base from current position. It can be modified to start at an earlier position than current position in the data base through a *command code,* but its normal function is to move forward from a given segment to the next desired segment in a data base. Command codes are discussed in detail in Chapter 4.

- GNP (get next within parent) is used to retrieve the next desired segment or path of segments within established parentage. Parentage must have been established by a successful GU, GHU (get hold unique, see following text), GN, or GHN (get hold next, see following text) call either immediately before this call, or at some prior time, provided no other call that changes parentage has intervened. A GNP call or GHNP call does not establish parentage.

- GHU (get hold unique), GHN (get hold next), or GHNP (get hold next within parent indicates the intent of the user to issue a subsequent delete or replace call. A get hold call must be issued to retrieve the segment before issuing a delete or replace call.

- ISRT (insert) call is used to insert a segment or a path of segments into a data base. It is used to initially load segments in data bases, and to add segments in existing data bases.

  To control where occurrences of a segment type are inserted into a data base, the user normally defines a unique sequence field in each segment. When a unique sequence field is defined in a root segment type, the sequence field of each occurrence of the root segment type must contain a unique value. When defined for a dependent segment type, the sequence field of each occurrence under a given physical parent may contain a nonunique value. If no sequence field is defined, a new occurrence is inserted according to rules specified by the user when the data base is defined.

- DLET (delete) call is used to delete a segment from a data base. When a segment is deleted, its dependents, if any, are also deleted. This call must be preceded by a get hold call.

- REPL (replace) call is used to replace the data in the data portion of a segment or path of segments in a data base. Sequence fields cannot be changed with a replace call. This call must be preceded by a get hold call.

- CHKP (checkpoint) causes a checkpoint record to be written on the DL/I log as an aid in restart processing.

## SSA (segment search argument)

An SSA specifies the conditions that a segment must meet to satisfy a call. An SSA can contain three parts. As a minimum, it contains the name of the segment type. Optionally, an SSA can also contain command codes and/or qualification statements. Command codes, when used, specify a functional variation of the call. Qualification statements identify, through field values, the segment occurrence of the specified segment type. A qualification statement contains a field name, relational operator, and comparative value. When occurrences of the segment type are searched by DL/I, the specified field is compared to the comparative value in accordance to the relational operator specified. If only the name of the segment type is specified, the first encountered occurrence of that type will satisfy the call.

## *VSAM (Virtual Storage Access Method)*

VSAM is very flexible in that this single access method can be used to process data sets organized in several different ways. Two of these data sets are called the ESDS (entry-sequenced data set), and the KSDS (key-sequenced data set). The primary difference between these data sets is the sequence in which records are stored in them.

In a KSDS, records are stored logically in order of collating sequence of the contents of a key field. This field is part of the data content of each record. It appears in the same position of each record in the data set. The key field contains a unique value, such as customer number or order number, which determines the record's collating position in the data set.

A KSDS has an index which is used to locate the record's physical position in the data set. Each entry in the index couples a key of a record with its location in the data set. This key is the highest key value in that section of the data set.

In an ESDS, the records are stored physically in the order in which they are entered into the data set, that is, their entry sequence. The data content of an ESDS record has no effect on the position in which it is stored. New records are simply stored at the end of the data set. VSAM does not maintain an index for an ESDS.

### Cluster Concept

In VSAM (for a KSDS), both the index component and the data component can be treated as independent data sets. You can give each component a name. For example, you could name the index of a payroll data set PAYDEX and the data part PAYDAT.

**Note:** The index component of a KSDS is a VSAM index. It is *not* the primary or secondary index you can define for DL/I data bases.

Thus, it is possible to process the data portion separately from the index portion and vice versa. In DL/I, you will be treating the index and data as a single data set with its own name. In VSAM, this combination is called a cluster and the name that is given to the combined components (index and data) is called a cluster name. For example, you could give the payroll data set a cluster name of PAYROLL. This is the name you use as the file-ID in a DLBL statement to process the payroll data set as a single functional unit.

```
// DLBL PAYFILE,'PAYROLL',,VSAM
```

This concept of a cluster is carried over to the ESDS. It is considered by VSAM to be a cluster without the index component. To be consistent, the ESDS is given a cluster name, just as the KSDS, which is normally used as the file-ID when processing the data set.

In VSAM it is necessary to define a cluster before it can be used as a dataset. A DL/I data base that is physically stored as a VSAM KSDS and/or ESDS must be defined as a cluster to VSAM. VSAM clusters are defined with the access method services DEFINE command.

## Data Base Access Methods

### Simple HSAM
The simple HSAM data base consists of root segments only. Segments contain data only and are placed sequentially in a physical record of a DOS/VS Sequential Access Method (SAM) file on DASD or tape. Any DOS/VS SAM file defined with RECFORM=FIXUNB may be defined as a simple HSAM data base.

### HSAM
Figure 2-8 shows the HSAM physical storage of the logical data structure. HSAM uses the DOS/VS Sequential Access Method (SAM) data management facility for DASD and TAPE files. Segments which contain DL/I prefix information and data are placed sequentially in a physical record until the remaining space in the record will not hold the next segment to be stored. The next segment is then placed in the next physical record. Unused space at the end of a physical record is filled with binary zeros.

### Call Functions
- GET: SHSAM and HSAM will accept GET functions

- INSERT: SHSAM and HSAM will accept ISRT functions on initial load only. Inserts to an HSAM data base must be in sequence.

- DELETE: SHSAM and HSAM will *not* accept a DLET function.

- REPLACE: SHSAM and HSAM will *not* accept a REPL function.

### Simple HISAM
The simple HISAM data base access method may be used for indexed sequential access to a root segment only data base. Because of this, there is no segment prefix needed. Each segment contains only data and constitutes one record of a DOS/VS VSAM key sequenced file (KSDS). This makes it possible to process a non DL/I KSDS as a DL/I data base with full DL/I function. The main use of SHISAM is as a migration tool to DL/I for existing KSDS files. It is not recommended for new data bases. Any fixed length KSDS may be defined as a simple HISAM data base.

### HISAM
The HISAM data base access method is used for indexed sequential access. Data management capabilities are provided by DOS/VS VSAM. HISAM requires a KSDS and an ESDS.

One KSDS record is allocated to each DL/I data base record. Each segment contains DL/I prefix information and data. A root segment and as many dependent segments of the DL/I data base record as can be accomodated are placed in the KSDS record.

If additional space is required for storage of dependent segments of a DL/I data base record, one or more ESDS records is used. Direct addresses relate the KSDS record and all ESDS records for one DL/I data base record. The ESDS records together form a VSAM entry sequenced data set. Figure 2-9 presents the HISAM physical storage of the logical data structure.

A VSAM control interval (KSDS or ESDS) consists of one or more logical records. KSDS control intervals may contain several logical records, each of which relates to a different data base record. KSDS and ESDS records may differ in size, however the KSDS record must be large enough to contain at least the root segment plus prefix. The ESDS logical record length must be large enough to contain the largest dependent segment plus prefix and it must be at least as large as the KSDS record.

### Considerations of HISAM and HSAM
In deciding whether to use HISAM or HSAM, the HSAM restrictions must first be considered. Since HSAM is used to reference a sequential file, data cannot be added, deleted, or replaced in an existing HSAM data base. DELETE and REPLACE calls are not valid for HSAM. INSERT calls are invalid except when loading the data base.

HSAM is useful for processing existing sequential files and archival storage of data bases.

DATA BASE RECORD

```
                    ┌─────────────┐
                    │ CUSTOMER    │
                    │ 1           │
                    └─────────────┘
                           │
        ┌──────────────────┼──────────────────┐
  ┌─────────────┐   ┌─────────────┐     ┌─────────────┐
  │ CUSTOMER    │   │ CREDIT      │     │ CUSTOMER    │
  │ LOCATION    │   │ STATUS      │     │ HISTORY     │
  │ 11          │   │ 11          │     │ 11          │
  └─────────────┘   └─────────────┘     └─────────────┘
        │
   ┌──────112──────┐
   │ CUSTOMER      │
   │ ORDER         │
   │ 111           │
   └───────────────┘
        │
   ┌─────1112─────┐      ┌──────1123──────┐
   │ ORDER        │      │ ORDER    1122  │
   │ ITEM         │      │ ITEM           │
   │ 1111         │      │ 1121           │
   └──────────────┘      └────────────────┘
```
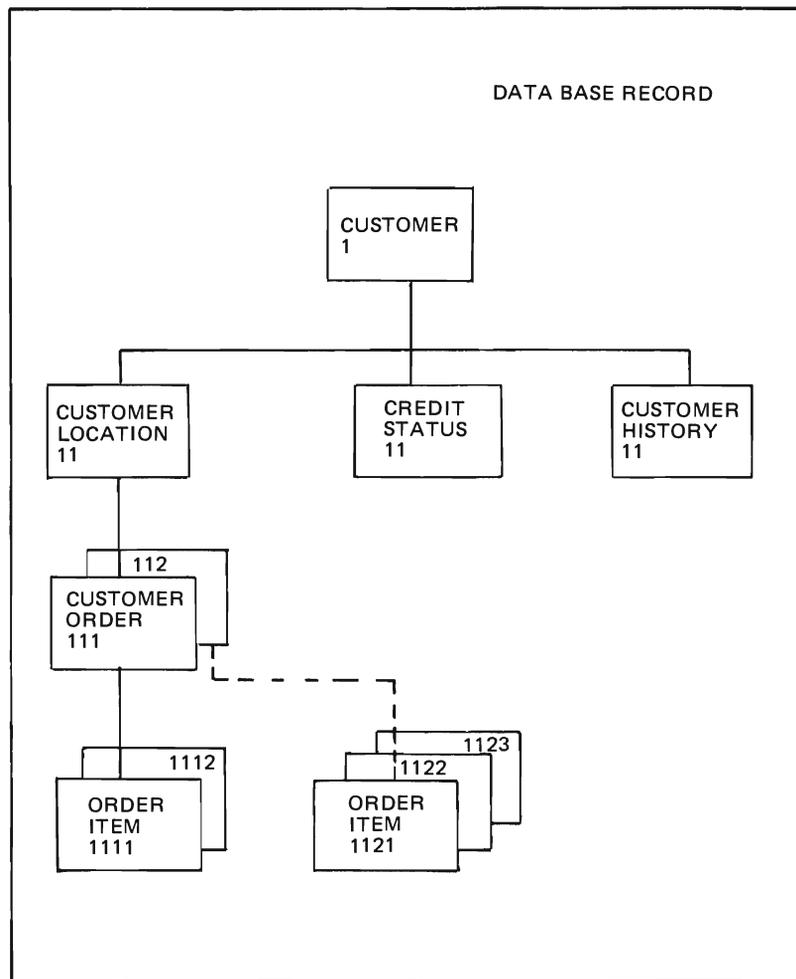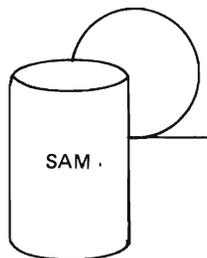
| | | | |
|---|---|---|---|
| PHYSICAL RECORD 1 | CUSTOMER 1 | CUSTOMER LOCATION 11 | CUSTOMER ORDER 111 | ORDER ITEM 1111 |
| PHYSICAL RECORD 2 | ORDER ITEM 1112 | CUSTOMER ORDER 112 | ORDER ITEM 1121 | ORDER ITEM 1122 |
| PHYSICAL RECORD 3 | ORDER ITEM 1123 | CREDIT STATUS 11 | CUSTOMER HISTORY 11 | |

Figure 2-8. HSAM Physical Storage of a Logical Data Structure
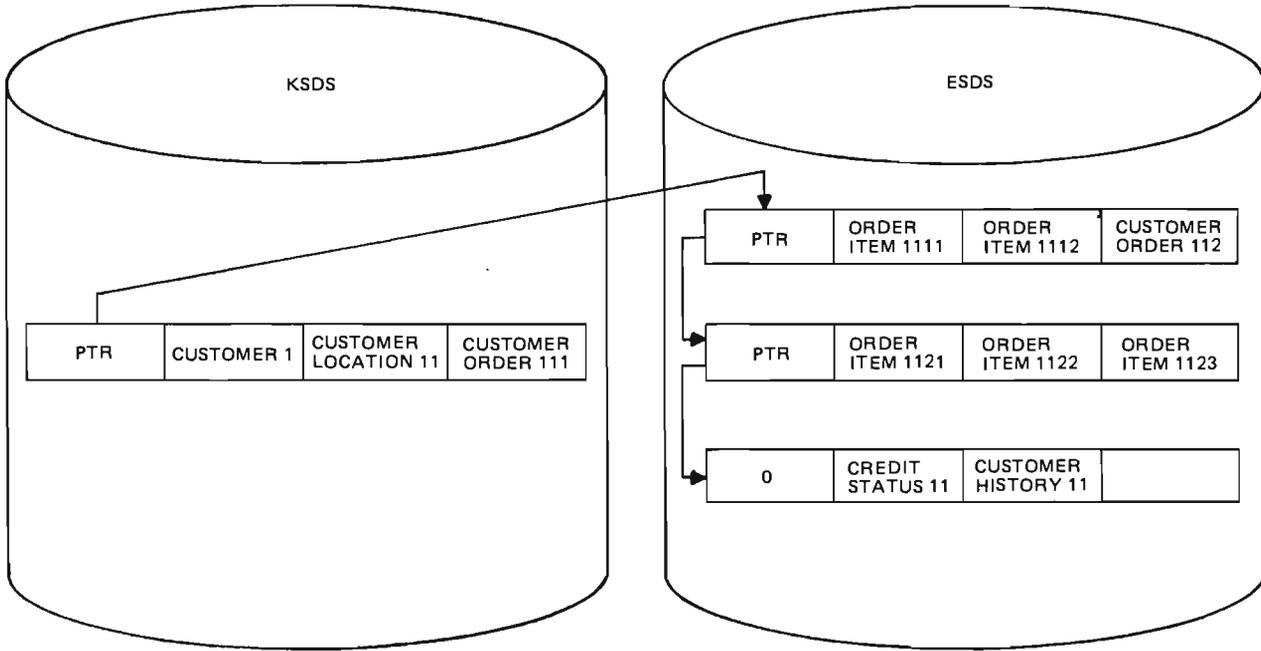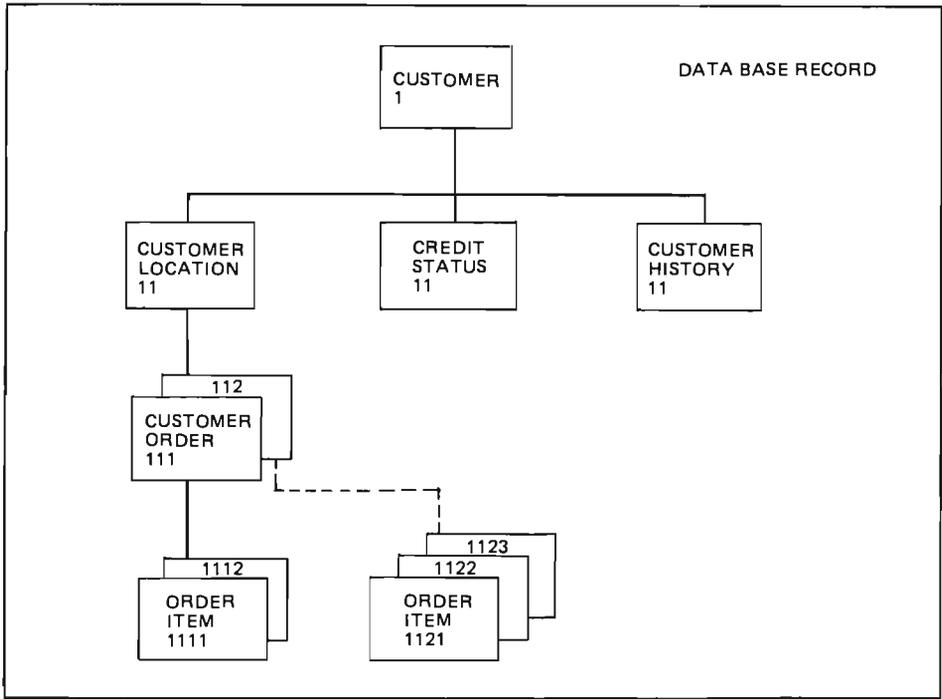
Figure 2-9. HISAM Physical Storage of a Data Base Record

HISAM data bases have these limitations:

- No support for variable length segments, secondary indexes, or logical relationships.

- Less efficient use of DASD space than HD organizations (for example, no space is reclaimed on delete processing). Space is reclaimed during a reorganization.

## HDAM and HIDAM

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

### HDAM and HIDAM Characteristics

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

The segment storage organization used for HDAM and HIDAM data bases is essentially the same. The primary difference, at the access method level, between HDAM and HIDAM data bases is that access to occurrences of the root segment type is through a randomizing module for an HDAM data base, and through an index for a HIDAM data base. To access a given root segment in an HDAM data base, the randomizing module examines the key of the root, and through hashing or some other arithmetic technique, computes the address of the root and passes it to DL/I. To access the same root in a HIDAM data base, an index must be searched by DL/I to find the address of the root. By using a randomizing module to locate root segments, the need for I/O operations required to search the index is eliminated.

**HDAM:** Figure 2-10 shows that an HDAM data base consists of one ESDS. To access the data in an HDAM data base, DL/I uses a randomizing module. This module converts a sequence field value, supplied by an application program for root segment insertion into or retrieval from an HDAM data base, into an address for the root segment.

The ESDS is divided into two areas:

- The *root addressable area*: This is the first of *n* control intervals/blocks in the data set. You define *n* in your DBD (data base description).

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

The root addressable area is used as the primary storage area for segments in each data base record. The overflow area is used for overflow storage. Since data base records vary in length, a parameter (in the DBD) is used to control the amount of space used for each data base record in the root addressable area. This parameter limits the number of segments of a data base record that can be consecutively inserted into the root addressable area. When consecutively inserting a root and its dependents, each segment is stored in the root addressable area until the next segment to be stored causes the total space used to exceed that specified. The total space used for a segment is the combined lengths of the prefix and data portions of the segment. When exceeded, that segment and all remaining segments in the data base record are stored in the overflow area. Note that this parameter controls only segments consecutively inserted in one data base record. Consecutive inserts are inserts to one data base record with no intervening call to process a segment in a different data base record.
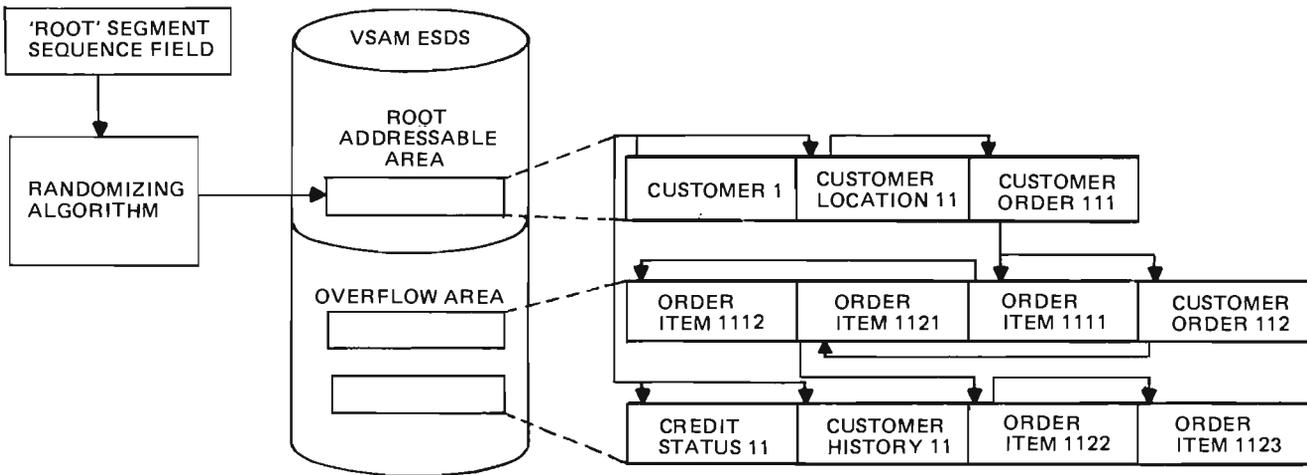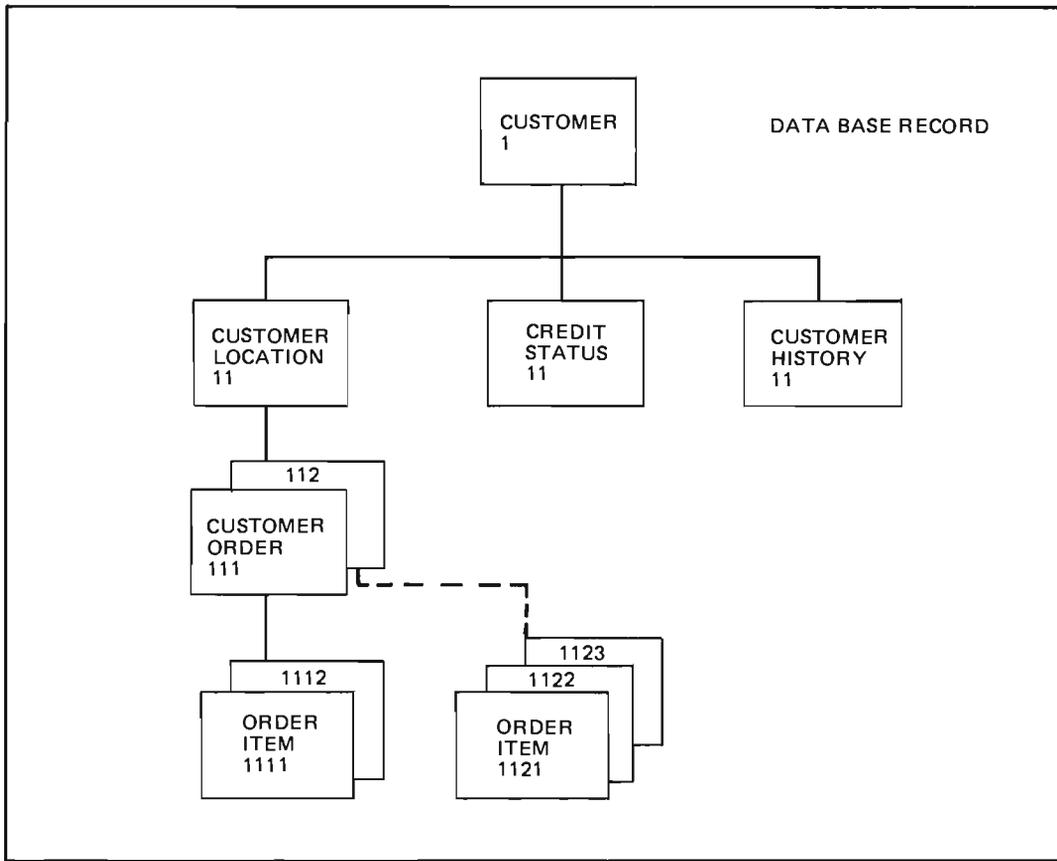
Figure 2-10. HDAM Data Base Record In Physical Storage

**HIDAM:** A HIDAM data base in auxiliary storage is actually comprised of two data bases that are normally referred to collectively as a HIDAM data base. When defining each during DBDGEN, one is defined as the HIDAM primary index data base and the other is de-

fined as the main HIDAM data base. In the following discussion the term 'HIDAM data base' refers to the main HIDAM data base.

The HIDAM primary index data base is used to locate the data base records stored in a HIDAM data base.

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

The HIDAM primary index data base consists of a KSDS; its only data (and key) is the sequence field of the root segment. The main HIDAM data base is an ESDS. The segment storage organization in this ESDS is comparable to the one in the HDAM ESDS. Figure 2-11 shows the lay-out of the HIDAM data base.
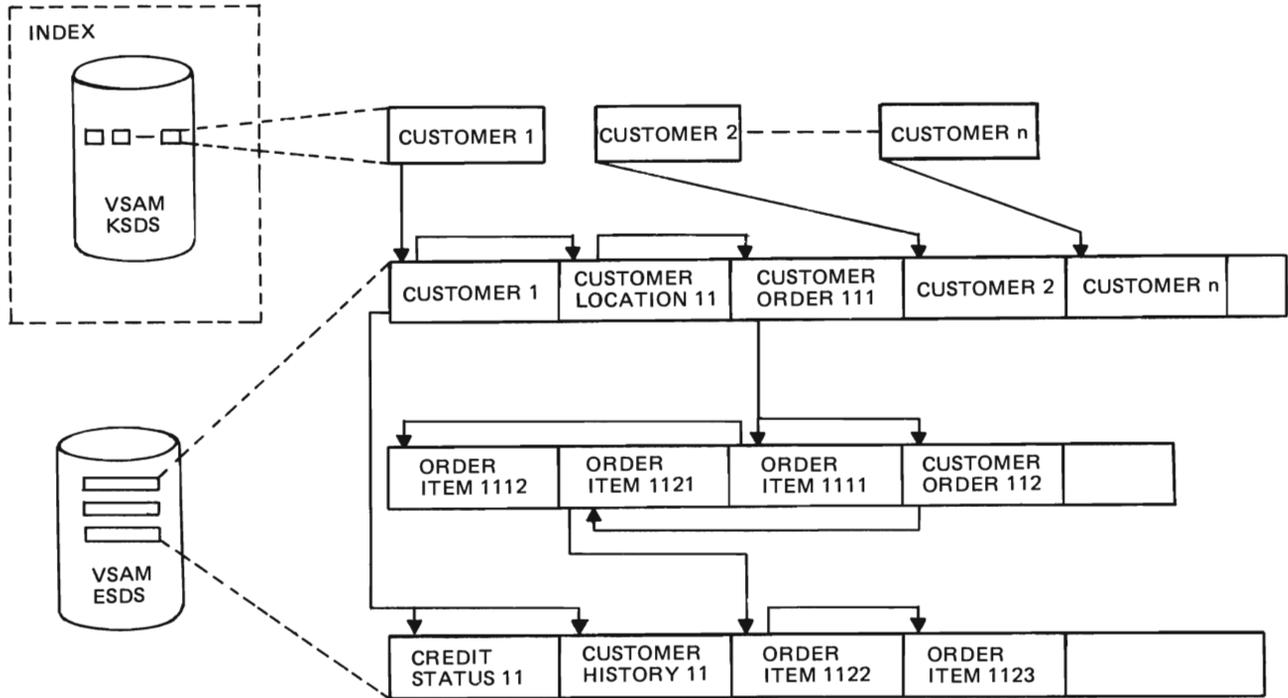


Figure 2-11. HIDAM Data Base Record in Physical Storage

## Inserts and Deletes in HDAM and HIDAM

The techniques used to insert or delete segments are the same for both HDAM and HIDAM data bases. The techniques involve use of bit maps and free space elements. These system fields are used by DL/I to find space when inserting a segment, or to record free space when a segment is deleted. Normally, the space a segment occupies is immediately freed up after the delete of the segment. You need only be aware of these system maintained fields when doing control interval blocksize calculations because they are allocated within your selected control interval blocksize.

DL/I allows you to specify free space for the ESDS at data base load time (initial load or reload during reorganization). This feature, distributed free space, allows

segments to be loaded as close to related segments as possible. Distributed free space is specified in the data base description.

For a primary index KSDS, free space can be assigned with the VSAM access method services DEFINE command. In theory, you can also specify free space in the DBD for an HDAM data base. This is, however, not recommended because it might conflict with the randomizing module algorithm.

## Direct Access Pointers in HDAM and HIDAM

Refer to Figure 2-12 for the following description of pointers.
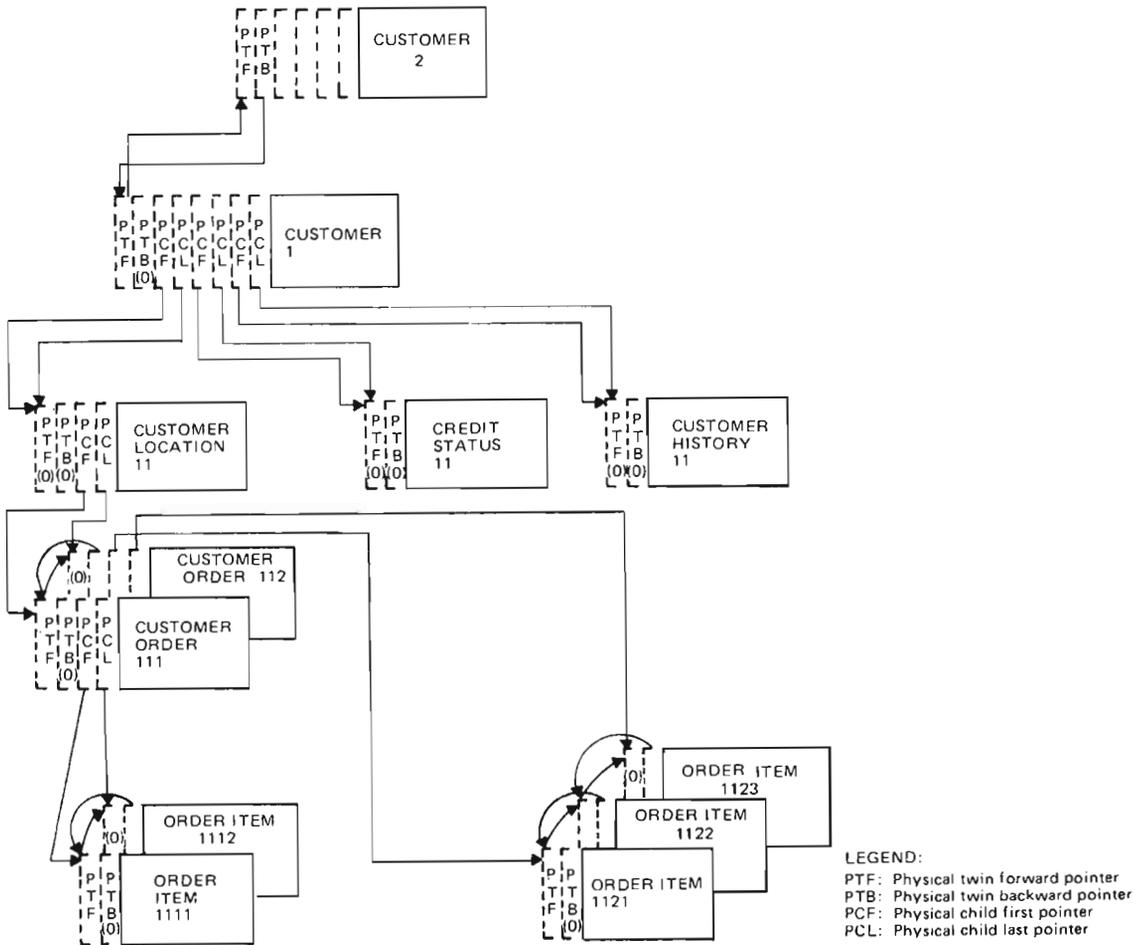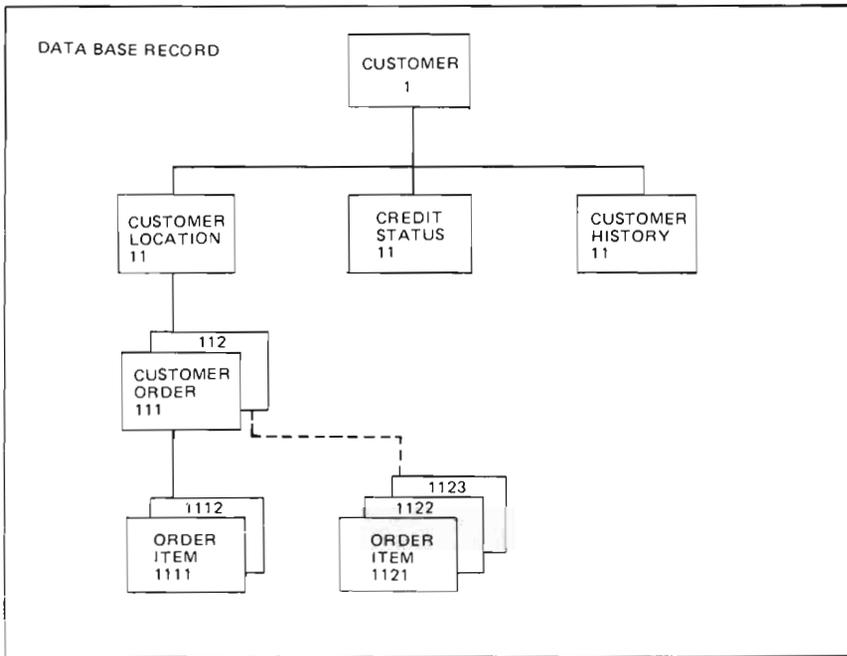
Figure 2-12. Direct Access Pointers in HDAM and HIDAM

*Physical Child/Physical Twin Pointers:* Every parent segment in the data base has a pointer to the first occurrence of each of his child segment types. This is the *physical child first pointer*. Optionally, per child segment type, there is also a pointer to the last occurrence of that child segment type, the *physical child last pointer*. This physical child last pointer will improve segment insert performance of that child if that segment has no sequence field defined.

Every segment in a HIDAM or HDAM data base has a pointer in its prefix which points to the next occurrence of this segment under the *same parent*. (If it is the last occurrence under the parent, this pointer is zero.) This pointer is named the *physical twin forward pointer*.

Optionally, you can also select a pointer in each segment prefix which points to the previous segment occurrence under the same parent. This is the *physical twin backward pointer*. This pointer is useful for delete processing.

When physical twin forward and backward pointers are specified for the root segment type of a HIDAM data base, they enable sequential processing across data base records without intervening references to the HIDAM index. When only physical twin forward pointers are specified for the root segment type of a HIDAM data base, sequential processing across data base records requires intervening references to the HIDAM index.

## Logical Relationships

### *Why Logical Relationships*
We have so far addressed only single hierarchical data structures. Often, especially with different applications, several DL/I data bases are needed. In addition, there is often a requirement to access the *same data* in *different hierarchical structures* and different data bases.

This can create problems of:

- Consistency - if stored more than once how to update at same time.

- Data redundancy - if large data elements were stored many times this could consume excessive external storage.

- Access of data - which access path should be used to access the appropriate copy of the data.

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

DL/I's logical relationships provide this function. The basic linkage is always between two segments. However, the linkage can extend to several data bases. The resulting compound data structure will always be presented as a single hierarchical data structure to a particular application. The basic mechanism of the DL/I logical relationship is the connection of a segment to two parents in two different hierarchical structures. All segments below the root segment must have a physical parent. By giving a segment a logical parent, that segment (and its dependents) now belongs to two different hierarchical structures. This enables the definition of a new hierarchical structure which contains segments from both related structures. Such a definition is called a *logical data base*. Once this logical data base is defined, DL/I automatically maintains the relationship between the two data bases.

### *Building Logical Relationships*
**Segment Types Involved in Logical Relationships**
Figure 2-13 shows that three segment types are needed to establish a logical relationship.

PHYSICAL DATA BASES                    LOGICAL DATA BASE
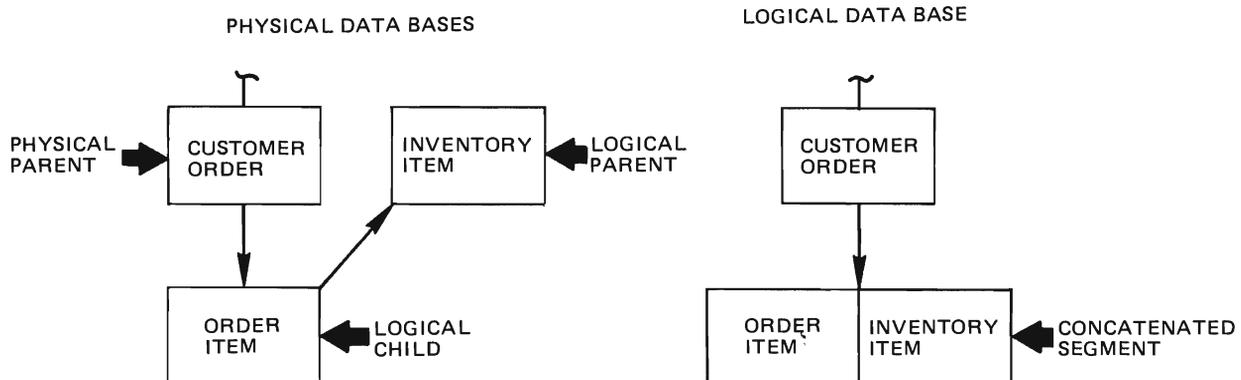


Figure 2-13. Segment Types Involved in Logical Relationships

The segment types are:

- *Logical Child*: This segment has two parents. A *logical parent* and a *physical parent*. The logical child segment and its dependents, if any, are accessable via both parents. The access path via its physical parent is called the *physical access path*. The access path via its logical parent is called the *logical access path*. When presented to the user a logical child segment contains the concatenated key of the logical parent followed by user data, if any. The user data in the logical child is called *intersection data*. It consists of data unique to the intersection of the two parents. The *logical parent concatenated key* (LPCK) is always presented with the intersection data whenever the logical child is accessed via its physical path (see Figure 2-14).

| PREFIX | LPCK | INTERSECTION DATA |
|--------|------|-------------------|

|◀ - - TO/FROM USER'S I/O AREA - -▶|

Figure 2-14. Logical Child Segment Format

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

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

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

Logical relationships between HDAM and HIDAM data bases are implemented using direct address pointers, which are all 4-byte relative byte address pointers similar to other pointers in HDAM and HIDAM.

### The Virtual Logical Child Segment (VLC)

In order to define the relationship between the logical parent and its logical children, DL/1 uses a special segment type. It is called the *virtual logical child* and is defined as a dependent of the logical parent segment. It does not exist on DASD. Its only purpose is to provide a mechanism to define the logical parent's view of the data in the logical child. It controls the access from the logical parent to the logical child. It is used to define the sequencing of the logical child segment when that logical child segment is accessed via its logical parent. The virtual logical child is said to be *paired* with the real logical child. Because the logical child can be accessed as a dependent of the logical parent as well as the physical parent, the logical relationship is bidirectional. See Figure 2-15.

PHYSICAL DATA BASES                          LOGICAL DATA BASES

Figure 2-15. Virtual Paired Bidirectional Logical Relationship

KEY:

PP - Physical parent pointer
LP - Logical parent pointer
LCF - Logical child first pointer

When accessed, the virtual logical child contains the concatenated key of the physical parent of the real logical child, plus the intersection data of the real logical child. So the virtual logical child ORDER ITEM, in Figure 2-15 contains the key of the CUSTOMER ORDER segment plus the user data of the real ORDER ITEM segment.

### Destination Parent

With bidirectional pairing, DL/I refers to the parent that is other than the one used to access the logical child, as the *destination parent*. When the logical child is accessed from its physical parent, the logical parent concatenated key (LPCK) is returned. When the logical child is accessed from its logical parent, the physical parent concatenated key is returned. Therefore, the logical child always starts with the *destination parent concatenated key* (DPCK).

## *Logical and Physical Data Bases*

The *physical data bases* used to implement a logical relationship must be HDAM or HIDAM data bases. Figure 2-16 shows the physical data bases of the phase 2 sample environment. The INVENTORY ITEM segment in the inventory data base is the logical parent of the ORDER ITEM segment in the customer data base. Note that the INVENTORY ITEM segment in the inventory data base is also a physical and logical parent of the SUBSTITUTE ITEM segment in the *same* data base. In this example, the first occurrence of INVENTORY ITEM is the physical parent of the logical child segment, SUBSTITUTE ITEM, and the second occurrence is the logical parent of SUBSTITUTE ITEM. In either case, the virtual logical child is not shown in Figure 2-16. However, the virtual logical child segments will appear in the DBDs as discussed later.

CUSTOMER DATA BASE
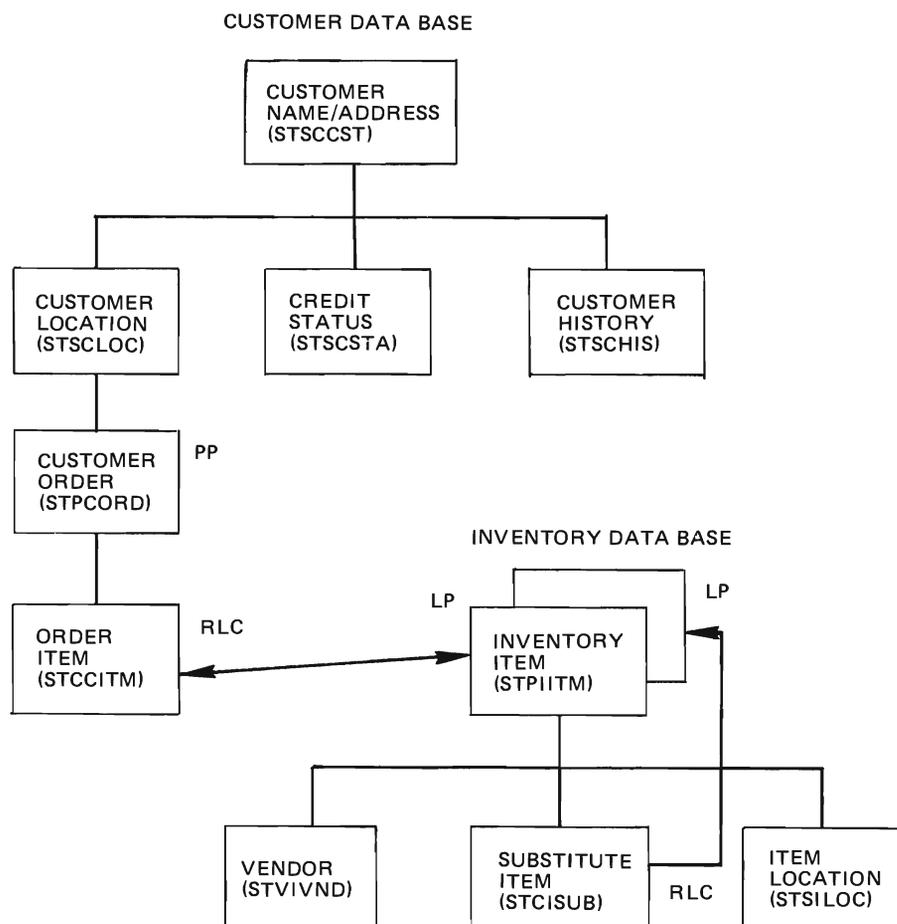


Figure 2-16. The Phase 2 Physical Data Bases

2-16

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

### Concatenated Segment

All segments in the logical data base stem from one segment in one of the physical data bases, except when the logical child is accessed. Whenever the logical child is accessed in a logical data base, it is *concatenated* with the destination parent segment. See Figure 2-17. The destination parent is the parent of the logical child in the access path other than the one from which you came.

| LOGICAL CHILD | | DESTINATION PARENT |
|---|---|---|
| DPCK | INTERSECTION DATA | |

Figure 2-17. Concatenated Segment Format

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

- When accessing the concatenated segment from its physical parent, it consists of:

    1. The real logical child which consists of the concatenated key of the logical parent and the data of the real logical child segment, if any.

    2. The logical parent segment itself.

- When accessing the concatenated segment from the logical parent, it consists of:

    1. The virtual logical child which consists of the concatenated key of the physical parent and the data of the real logical child segment, if any.

    2. The physical parent itself.

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

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

Figure 2-18 shows the two logical data bases used in the sample application. These two data bases are defined using the related physical data bases of Figure 2-16.

LOGICAL INVENTORY DATA BASE

```
                        ┌──────────────┐
                        │  INVENTORY   │
                        │    ITEM      │
                        └──────┬───────┘
       ┌───────────────┬───────┴───────┬────────────────┐
┌──────────┬────────┐ ┌────────┐ ┌──────────┬──────────┐ ┌──────────┐
│  ORDER   │CUSTOMER│ │ VENDOR │ │SUBSTITUTE│INVENTORY │ │  ITEM    │
│  ITEM    │ ORDER  │ │        │ │  ITEM    │  ITEM    │ │ LOCATION │
│  (LC)    │ (DP)   │ └────────┘ │  (LC)    │  (DP)    │ └──────────┘
└──────┬───┴────────┘            └────┬─────┴──────────┘
       │                     ┌────────┴────────┐
┌──────────────┐        ┌──────────┐    ┌──────────┐
│  CUSTOMER    │        │ VENDOR   │    │  ITEM    │
│  LOCATION    │        │ (FOR     │    │ LOCATION │
└──────┬───────┘        │SUBSTITUTE│    │ (FOR     │
       │                │ ITEM)    │    │SUBSTITUTE│
┌──────────────┐        └──────────┘    │ ITEM)    │
│  CUSTOMER    │                        └──────────┘
│  NAME/       │
│  ADDRESS     │
└──────────────┘
```

LOGICAL CUSTOMER DATA BASE

```
                 ┌──────────────┐
                 │  CUSTOMER    │
                 │  NAME/       │
                 │  ADDRESS     │
                 └──────┬───────┘
       ┌────────────────┼────────────────┐
┌──────────────┐ ┌──────────────┐ ┌──────────────┐
│  CUSTOMER    │ │  CREDIT      │ │  CUSTOMER    │
│  LOCATION    │ │  STATUS      │ │  HISTORY     │
└──────┬───────┘ └──────────────┘ └──────────────┘
       │
┌──────────────┐
│  CUSTOMER    │
│  ORDER       │
└──────┬───────┘
       │
┌──────────┬──────────┐
│  ORDER   │INVENTORY │
│  ITEM    │  ITEM    │
│  (LC)    │  (DP)    │
└──────┬───┴──────────┘
   ┌───┴────────┐
┌────────┐ ┌──────────┐
│ VENDOR │ │  ITEM    │
│        │ │ LOCATION │
└────────┘ └──────────┘
```

Figure 2-18. Phase 2 Logical Data Bases

These logical data bases will be used by the sample application programs.

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

## *Logical Relationship Design Rules*

In constructing logical relationships with DL/1, two sets of rules must be observed. One set for constructing the physical data bases and the second set for constructing the logical data bases. Note that a logical data base can be defined only if the underlying physical data base(s) are properly defined.

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

### Rules for Defining Logical Relationships in Physical Data Bases

- Logical Child:

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

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

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

- Logical Parent:

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

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

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

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

- Physical Parent:

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

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

### Rules for Defining Logical Data Bases

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

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

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

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

5. The dependents of a concatenated segment are:

   - The dependents of the logical child and/or,

   - The physical dependents of the destination parent.

   - The physical parents (and their dependents) up to the root of the destination parent in destination parent to root order.

The above three groups of dependents should not be intermixed, nor should the relative order of the segments within the groups be changed. However, you can start with any one of the groups.

**Notes:**

- Because of the virtual logical child concept, paths are bidirectional and can be intermixed.
- All segments of related data bases are available as long as you follow the above rules.

Figure 2-19 shows some examples of logically related physical data bases and their associated logical data bases. These examples are not representative for a typical DL/1 application. They merely show the different possible combinations.
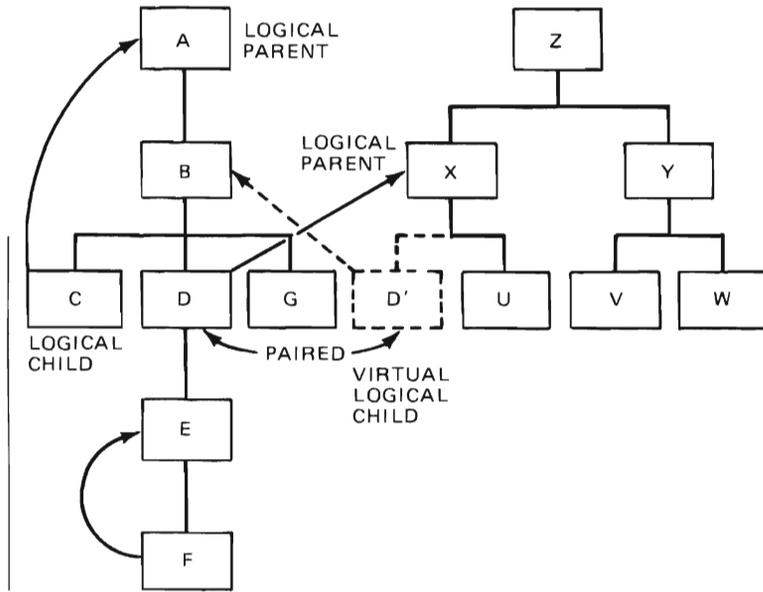
Figure 2-19. Using Multiple Logical Relationships (Part 1 of 2)
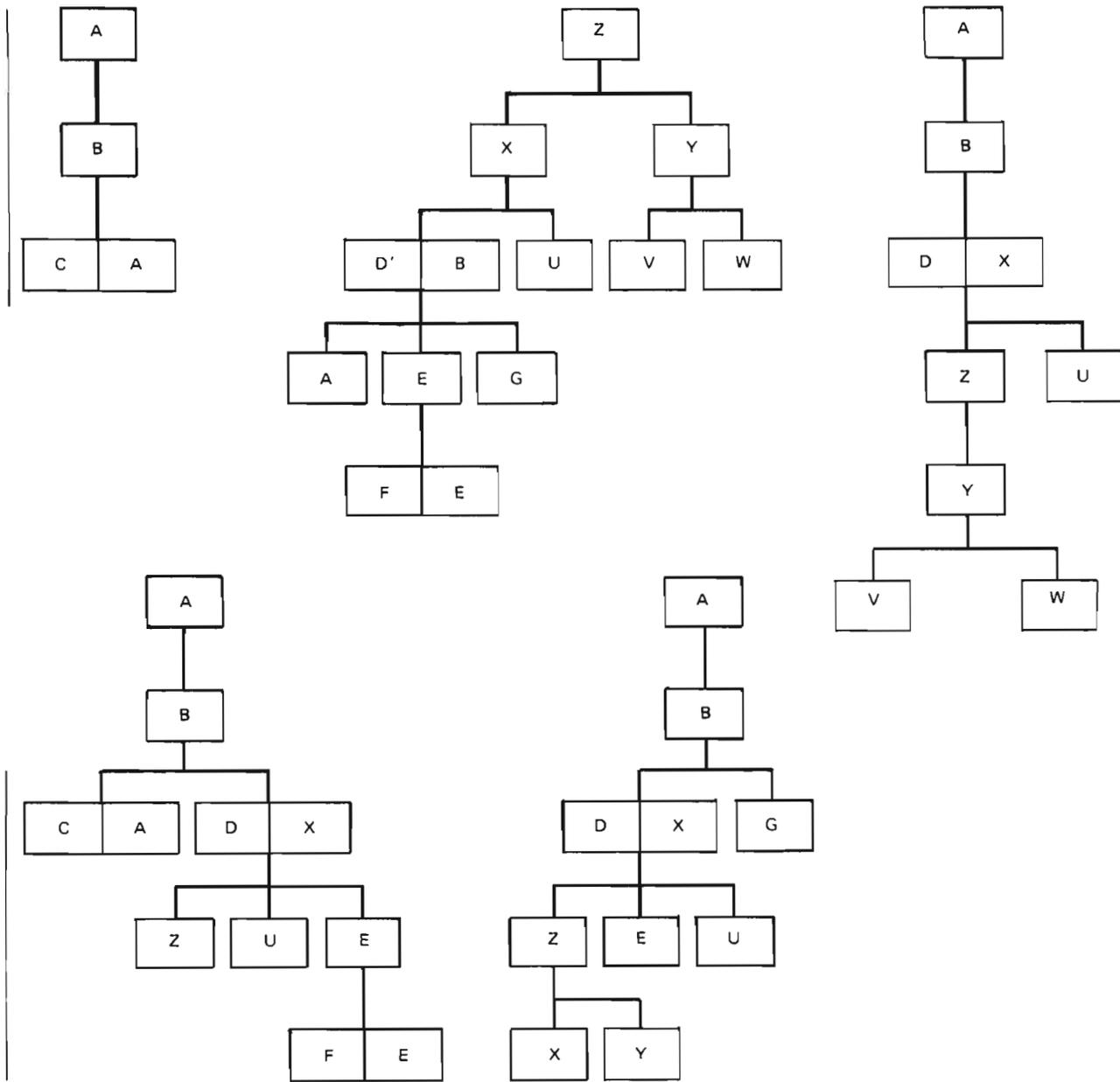
Figure 2-19. Using Multiple Logical Relationships (Part 2 of 2)

## Processing Logically Related Segments

The segments involved in logical relationships can be accessed through their physical data bases, and insert, delete, or replace operations may be performed through either access path. In order to avoid contradictory conditions, for instance a logical child pointing to a deleted logical parent, such updates are performed according to rules specified by the user in the DBDs for the physical data bases. Three modes, called physical (P), logical (L), and virtual (V) can be specified for each of the three update functions. (See Chapter 3 for details on how to code these rules.)

In general, the physical rule places restrictions on update requests through the logical data bases and requires that appropriate updates have previously been performed in the physical data bases. The logical rule removes some of these restrictions, while the virtual rule is the least restrictive for updates through logical data bases.

A detailed discussion of these rules is contained in the *System/Application Design Guide,*, and a general discussion follows.

Consider the virtual paired bidirectional logical relationship of Figure 2-20.
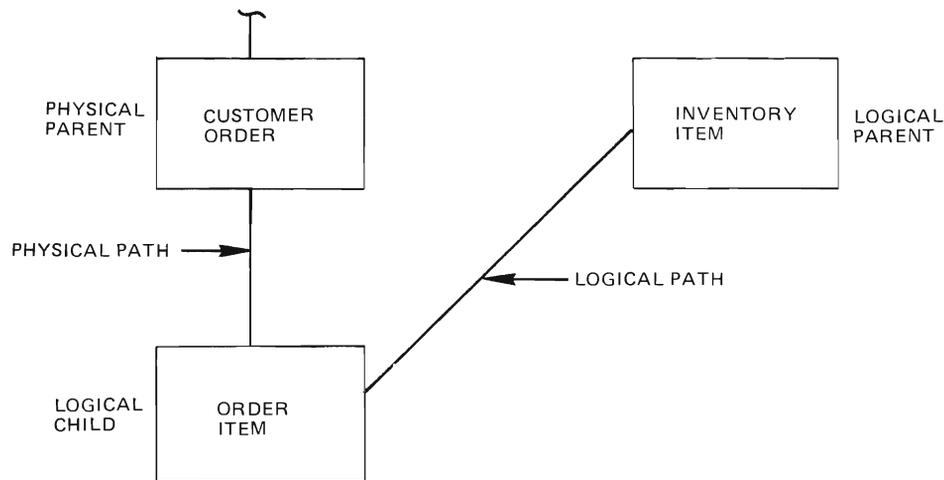


Figure 2-20. Virtual Paired Bidirectional Logical Relationship

The rules for the logical relationships are coded as xxx, where the first x is for insert, the second x is for delete, and the third x is for replace. The value of x can be specified as P (Physical), or L (Logical), or V (Virtual).

### Physical Parent Segment (CUSTOMER ORDER)

The rules for the physical parent are as follows:

*Insert Rule:* Affects the insertion of the logical child and the creation of the physical parent as a result of the insert of the logical child segment from the logical path.

P (Physical)

The physical parent must exist in the data base before the logical child may be inserted from the logical path. The destination parent portion of the concatenated segment (the physical parent) is ignored.

**L (Logical)**

The physical parent need not exist prior to insertion of logical child from the logical path. If the physical parent exists, the destination parent portion of the concatenated segment (the physical parent) is ignored. If the physical parent does not exist, the destination parent portion of the concatenated segment is used to create a physical parent segment in the data base.

**V (Virtual)**

Same as (L) above except that if the physical parent already exists, the destination parent portion of the concatenated segment will replace the current physical parent segment. Use caution when implementing this rule.

*Delete Rule:* This rule does not apply to the physical parent.

*Replace Rule:* Affects the replacement of the physical parent (destination parent) whenever a replace call is issued against a logical child concatenated segment with physical parent data as part of its content. It also affects the replacement of the logical child segment if this portion of the concatenated segment is altered. This rule is usually coded as P for the physical parent segment.

**P (Physical)**

Any replace of a logical child concatenated segment that contains a changed physical parent (destination parent) is not allowed. If the concatenated segment contains both the logical child and a changed physical parent, neither will be replaced. If the concatenated segment contains a physical parent that is not changed, the replace is allowed. However, only the logical child portion of the concatenated segment is replaced.

**L (Logical)**

A replace of the logical child concatenated segment that contains a physical parent (destination parent) is allowed even if the physical parent portion is changed, however, the physical parent portion is *not* replaced. If the concatenated segment contains both the logical child and the physical parent, only the logical child will be replaced. Be careful about using this rule for the physical parent.

**V (Virtual)**

A replace of a logical child concatenated segment that contains a changed physical parent is allowed and the physical parent is replaced as well as the logical child. Use caution when implementing this rule for the physical parent.

## Logical Parent (INVENTORY ITEM)

The rules for the logical parent are usually coded as PPP.

*Insert Rule:* Affects insertion of the logical child from the physical path and creation of the logical parent segment as a result of the insertion of a logical child concatenated segment from the physical path.

**P (Physical)**

The logical parent must exist in the data base before the logical child may be inserted from the physical path. The logical child concatenated segment need not contain the logical parent and if the logical parent is present, it is ignored.

**L (Logical)**

The logical parent need not exist prior to insertion of the logical child from the physical path. If the logical parent already exists, the logical parent portion of the concatenated segment (destination parent) is ignored. If the logical parent segment does not currently exist, the logical parent portion of the concatenated segment is used to create a logical parent segment.

**V (Virtual)**

Same as (L) above except that if the logical parent already exists, the logical parent portion of the concatenated segment will replace the current logical parent segment. Use caution when implementing this rule for the logical parent.

*Delete Rule:* Affects deletion of a segment and all its dependents.

**P (Physical)**

All of the logical parent's logical children must have been deleted from their physical path before a delete will be allowed against the logical parent segment. The logical parent segment can be deleted only from its physical path by a delete call issued to its physical path.

**L (Logical)**

The logical parent may be deleted from its physical path at any time, but will remain available from any of its logical paths. If the logical parent is deleted by its physical path, and then all of its logical children are deleted from their physical paths, the logical parent is removed.

**V (Virtual)**

Same as (L) above except that when all of the logical parent's logical children are deleted from their physical and logical paths, the logical parent is automatically deleted from its physical path, and will be removed from the data base. Use cau-

tion when implementing this rule for the logical parent.

*Replace Rule:* Assuming the replace rule for the logical child has been satisfied, this rule affects the replacement of the logical parent when the logical parent is the destination parent of the logical child concatenated segment. The replace rule can also affect the replacement of the logical child when the concatenated segment contains both the logical child and the logical parent.

P (Physical)
Any replace of the logical child segment when the concatenated segment contains a logical parent that has been altered is not allowed.

L (Logical)
Any replace of the logical child when the concatenated segment contains a logical parent (altered or not) is allowed. However, the logical parent is *not* replaced.

V (Virtual)
Same as (L) above except that the logical parent will be replaced.

**Logical Child (ORDER ITEM)**
The rules for the logical child segment are usually coded as VVV, VLV, or VPV.

*Insert rule:* Has no meaning for the logical child. Code P, L, or V.

*Delete Rule:* Affects the deletion of the logical child and indirectly the deletion of the physical parent from its physical path.

P (Physical)
The logical child must have been deleted from its logical path before it can be deleted from its physical path. This effectively keeps the physical parent from being deleted until all of its logical children have been deleted from their logical parents.

L (Logical)
The logical child can be deleted from either path and will remain available on the other path.

V (Virtual)
The logical child can be deleted from either path and as a result will be deleted from both paths.

*Replace Rule:* Affects the replace of a logical segment. This must be coded as V.

V (Virtual)
Segment can be replaced from either path.

## Logical Relationships Implementation Technique
The following pointers are used by DL/I to implement logical relationships. These pointers are maintained in the segment prefix in the same way as the previously discussed physical child and physical twin pointers. Detailed guidelines for the selection and implementation of these pointers are included in Chapter 3, Data Base Implementation.

### Pointers Used for Logical Relationships in HDAM/HIDAM

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

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

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

*Logical Twin Forward Pointer (LTF):* The logical twin forward pointer is within the prefix of the logical child segment and links all logical child occurrences of a particular logical parent. This pointer is required.

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

*Physical Parent Pointer (PP):* DL/I uses a physical parent pointer in the prefix of the logical child to locate

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

# DL/I Secondary Indexes

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

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

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

## *When to Use Secondary Indexes*

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

For batch processing, compare the costs of full or partial data base scans plus subsequent sort of the output versus the cost of using secondary indexes. For online data base processing, the choice is easier. Online response requirements normally do not allow for full data base scans and sorts.

## *Segment Types Involved in Secondary Indexes*

The segment types and associated terms involved in secondary indexes are (See Figure 2-21):

- Secondary Indexes

  A secondary index is comprised of an index pointer segment type defined in a secondary index data base.

- Index Pointer Segment

  A segment defined in a secondary index data base that contains data and a pointer to the index target segment. It controls the secondary indexing process.

- Index Target Segment

  The segment that is pointed to by an index pointer segment.

- Index Source Segment

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

- Secondary Processing Sequence

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

Although secondary indexes can be used in programs which use only logical data bases, their implementation is strictly on the physical data base level. Figure 2-22 shows the physical data bases of the phase 3 sample environment. The only difference from phase 2 is the addition of the secondary index data bases. The secondary index provides an alternate processing sequence. For example, by utilizing secondary index data bases, an application program can process the Customer data base in either order number or name sequence.

PHYSICAL OR LOGICAL DATA BASE

SECONDARY
INDEX DATA BASE

Usually the
root segment;
can be a
dependent
segment.

INDEX TARGET
SEGMENT

INDEX POINTER
SEGMENT

Can be the
same segment
as index
target segment,
or as shown,
a dependent
of the index
target segment.

INDEX SOURCE
SEGMENT

The content of the specified search
field in each index source segment
is duplicated in the respective index
pointer segment generated from each
index source segment.

Figure 2-21. Segment Types Associated with a Secondary Index

CUSTOMER DATA BASE
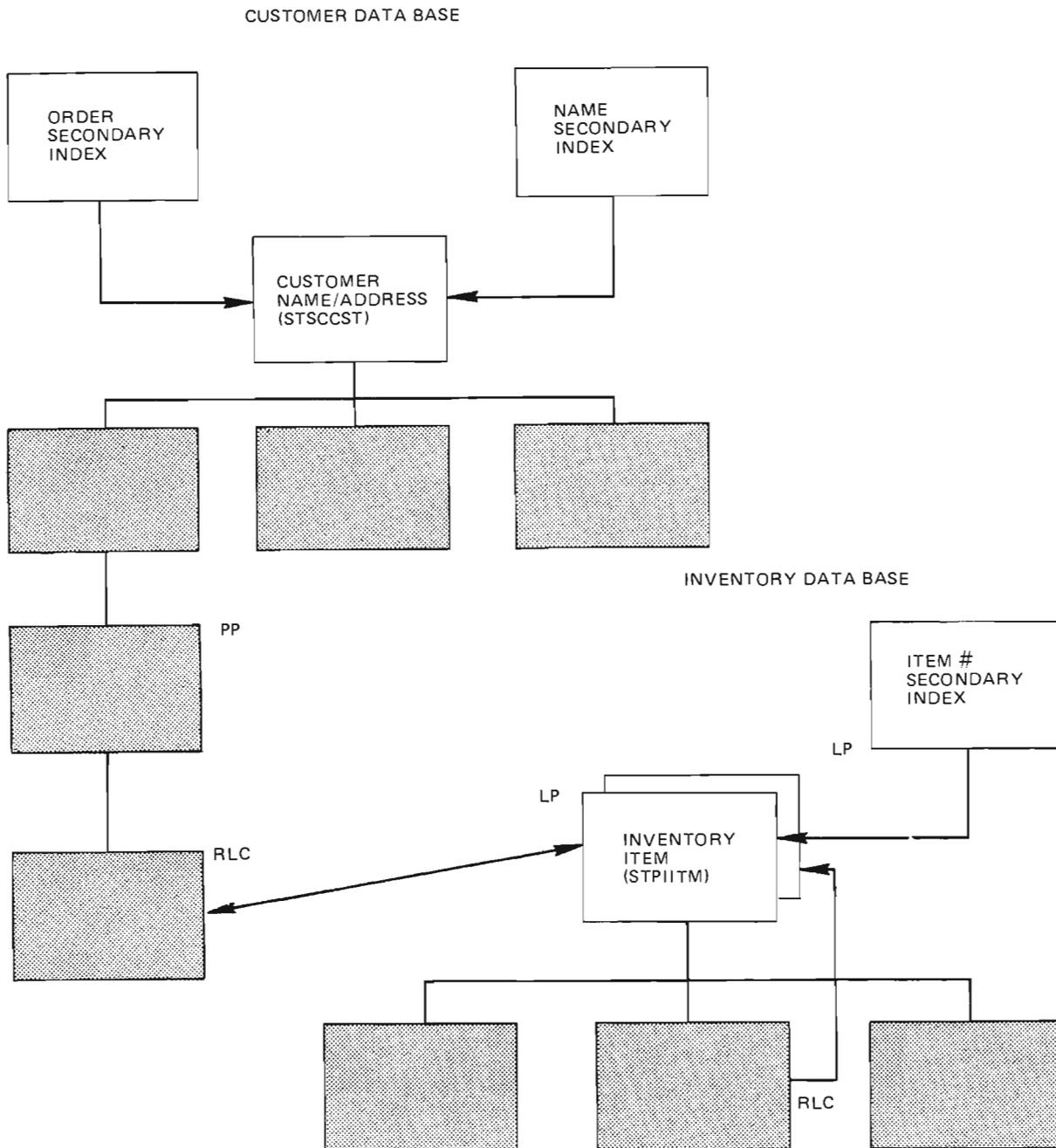


Figure 2-22. Phase 3 Physical Data Bases

## *Design Rules for Secondary Indexing*

Several rules should be observed when designing basic secondary indexes:

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

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

3. A secondary index can be used with a logical DBD, but the index target segment must be the root segment of the physical data base.

## *Implementation Technique*

In discussing secondary indexes, we have to distinguish between two different data base types. The first is the *indexed data base.* This data base contains the index source and index target segments. It is an HDAM or HIDAM data base. The second is the *secondary index*

*data base.* This data base contains the *index pointer segments* that contain pointers in their prefix to the index target segments. An INDEX data base consists of a single KSDS. Figure 2-23 shows the physical format of the KSDS logical record for the INDEX data base.
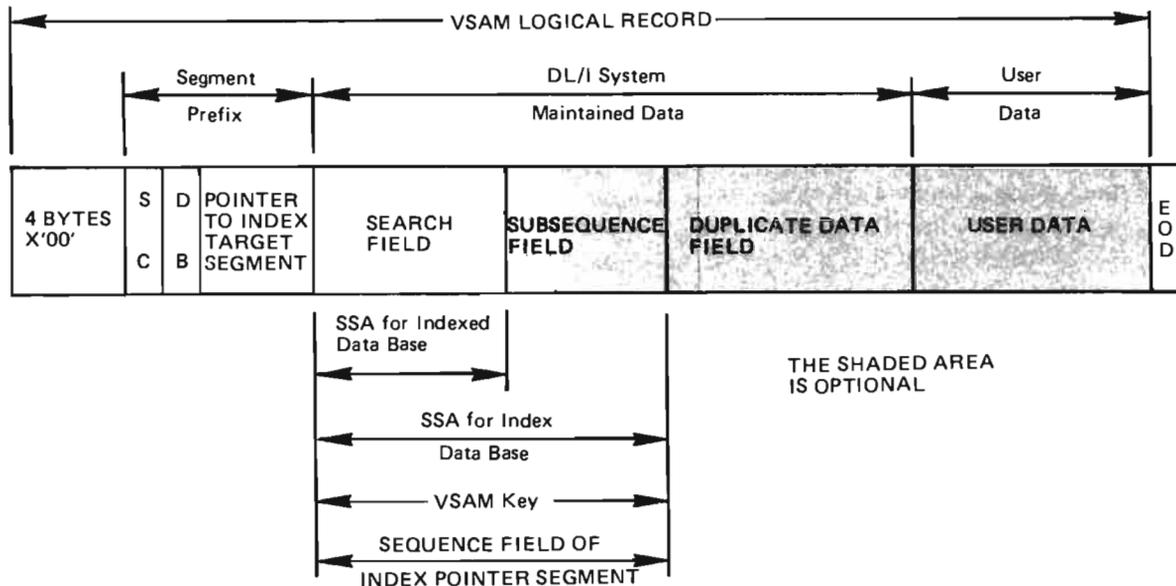


SC   = Segment Code
DB   = Delete Flag Byte
SSA for indexed Data Base   — The search field is used in qualified SSAs for the secondary data structure.

SSA for Index Data Base   — The search and subsequence fields are the sequence field of the index pointer segment and are used in qualified SSAs when the secondary index is processed as a data base itself.

EOD  = End of Data

Figure 2-23. Logical Record Format for the Index Pointer Segment

### Index Pointer Segment

The index pointer segment contains a:

- Reserved Area - 4 bytes

- Segment Code - 1 byte field (X'01')

- Delete byte that controls the delete status of the index pointer segment

- Pointer to the index target segment (4 bytes)

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

- Subsequence field (n bytes), optional. It is required if the search fields in the index pointer segments are non-unique. Its sole use is to provide a unique key for the KSDS logical record.

- Duplicate Data Field. This field is optional. It is of use only when the index data base is processed as a data base itself. The data from the indexed data base that is also included in the index data base is automatically maintained by DL/I. If a duplicate data field is changed in the indexed data base, it is also changed in the index data base.

- User data field. You can include any additional data desired in index pointer segments by specifying a length for the index pointer segment that is sufficient to include the additional data. This additional data is available to you when processing the secondary index as a data base itself. Remember, however, that initial loading of additional data, and maintenance of the additional data when reorganizing an indexed data base is a *user responsibility*.

During reorganization of an indexed data base, the secondary index(es) for the data base are re-created. When the secondary index is re-created, any additional user data that exists in the original secondary index is lost.

- End-of-data field - 1 byte (X'00')

### *Creating a Secondary Index*

Secondary indexes are created with the standard DL/I data base reorganization utilities, see Chapter 6. No user programming is needed to create a secondary index. Existing programs need not be changed unless they want to use the secondary index.

## Variable Length Segments

Variable length segments enable you to vary the amount of storage space used to store the different occurrences of the same segment type. They are intended for use by application programs that process variable length text or descriptive data. In addition, in some cases, they can be used to enhance utilization of secondary storage. You can vary the space for each occurrence of a segment type between a maximum and minimum number of bytes through a 2-byte size field loaded with each segment occurrence. You specify the maximum and minimum number of bytes for a variable length segment type during DBD generation.

The size field for a variable length segment is loaded with each segment to inform DLI of the length of data in the segment. Because the size field is in the data portion of a segment, the data length must be included in the length of the size field itself. In addition, if a sequence field is defined in the segment type, the minimum length specified must include at least the 2-byte size field and the length of all the data to the end of the sequence field.

When initially loading occurrences of a variable length segment type, the space used to store the data portion of a segment occurrence is the minimum length specified at DBD generation or the length specified in the size field of the segment occurrence, whichever is greater. The application program can then either increase or decrease the length of the data in the segment by replacing the data and changing the size field accordingly. When the data in an existing segment is replaced with data that is greater in length and the space is allocated for the existing segmnet is not sufficient for the new data, the prefix and data portions of the segment are separated to obtain space for the new data.

A variable length segment must not be a logical child segment or an index source segment, and may reside in a HDAM or HIDAM data base.

Chapter 3 of this manual contains the details you need to specify a variable length segment during DBD generation. A variable length segment is also included in the customer data base of the online sample application. If you need additional information about variable length segments, see the *System Application Design Guide.*

## Segment Edit/Compression Exit

The segment edit/compression exit facility of DL/I enables you to supply a routine to edit a segment during its movement between the application program I/O area and the data base buffer pool. You can use your routine to encode data for security purposes, to format data to be used by application programs, and to compress a segment to eliminate redundant characters. Segments to be processed by an edit/compression routine must be variable length. Application programs are never aware of the operation of the edit/compression routine.

A segment compression/expansion routine is provided with the online sample application program as an example of one way to use this facility to optimize space needed to store individual occurrences of variable length segments. This routine is documentd in Chapter 8 of this manual.

General considerations that apply to using the segment edit/compression exit facility are:

- All segment editing takes place only on variable length segmets described in a physical data base.

- Neither the relative position nor the contents of the key field (if one exists), can be changed by the routine.

- If the user routine 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 user routine cannot employ DOS/VS system macros such as STXIT.

Chapter 3 of this manual shows how to specify this facility for variable length segments during DBD generation. If you need additional information, see the *System Application Design Guide.*

## Field Level Sensitivity

Field level sensitivity allows you to specify only those fields in the physical definition of a given segment that are needed in the application program's view of that segment. You may also specify the locations of the chosen fields in the application's view of the segment. These field locations may be the same or different from their locations within the physical definition. This makes it possible for different application programs to have entirely different views of the same segment. This specification, done during PSB generation, enables DL/I to automatically map the chosen fields from the physi-

cal segment into the application program's view during execution.

Field level sensitivity also provides these capabilities:

- Virtual Fields
  You can identify fields for the application program's view of a segment that do not exist in the physical segment.

- Automatic Data Format Conversion
  DL/I automatically changes the format of the physical data to a format you specify for a given application program.

- User Field Exit Routine
  DL/I will give control to a user-written routine each time a given field is retrieved or stored.

- Dynamic Segment Expansion
  You can add fields to a segment without reloading the data base or re-compiling other application programs that access the segment.

## *Virtual Fields*

During PSB generation, 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" later in this section for further details.

## *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 in the SENFLD macro 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) |

Notes:

- 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. That is, if a field in a physical segment is specified as type 'C' and the field in that application's view is specified as type 'P', the data from the physical field is treated as though it is packed decimal. Only any necessary length adjustments are made.

## Non-supported Conversions

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 of 'KD' to be returned to the application program 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 "User Field Exit Routine" in this Chapter for additional information about resetting the conversion status code.

Additional information about field type conversion (programming considerations, status codes, etc.) is included in Chapter 3, under the description of the 'SENSEG' statement for PSB generation.

## *User Field Exit Routine*

During PSB generation, you may specify the name of a user-written field exit routine. This must be the name by which the routine is cataloged in the DOS/VS core image library. DL/I passes control to this routine whenever the associated field is referenced in either a retrieve or a store.

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 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.

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)
FERPPSL      8    2              PHYSICAL SEGMENT LENGTH
FERPPFL     10    2              PHYSICAL FIELD LENGTH
FERPPFA     12    4              PHYSICAL FIELD ADDRESS
FERPUSA     16    4              USER SEGMENT ADDRESS
FERPUSL     20    2              USER SEGMENT LENGTH
FERPUFL     22    2              USER FIELD LENGTH
FERPUFA     24    4              USER FIELD ADDRESS
FERPFSBA    28    4              FSB ADDRESS
FERPUWA     32   32              USER WORK AREA
```

## Dynamic Segment Expansion

Fields·may 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 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 sensi-

tivity, 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.

## Additional Field Sensitivity Considerations

- SSAs

  Any field to be used as a SSA 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 SENFLDs.

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

Notes:

DL/I does not take field type into consideration for compares for SSAs. As a consequence, for binary SSAs, a negative number will be larger than a positive number.

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 that sensitivity has been specified for.

Insert sensitivity of bi-directional 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.

- Key feedback area
  The information returned in the key feedback area is not converted.

- 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 as a segment search argument. 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 via 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 via a VIRFLD statement.

  - Field exit routines for overfields may 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/1 allows no conversion on overfields.

# Section 3: The Data Base Design Process

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

- Each data element is readily available by the various applications, now and in the foreseeable future.

- The data elements are efficiently stored on secondary storage.

- Controlled access is enforced for those data elements with specific security requirements.

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

## Concepts of Data Base Design

Because data base design is an area where there has been little formal standardization, there has been no consistent vocabulary for describing the concepts involved. This section presents some concepts and terms required to understand the remainder of the chapter.

### Entities

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

- Can be uniquely identified.

- We may now or in the future collect information about.

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

### Data Elements

A *data element* is a unit of information that specifies a fact about an entity. For example, suppose the entity is an inventory item. Item Number=200, Description=Transistor, and Quantity on hand=50 are three facts about that inventory item. Thus there are three data elements. A data element has a name and a value. A data element *name* tells the kind of fact being recorded; the *value* is the fact itself. In the above example, Item Number, Description, and Quantity on hand are data element names; 200, Transistor, and 50 are values. A value must be associated with a name to have a meaning.

An *occurrence* is the value of the data element for a particular entity. Figure 2-24 illustrates the concepts of data elements and their occurrences in recording the

facts about two entities, INVENTORY ITEMS (A) and
ORDER ITEMS (B).

| ENTITY A: INVENTORY ITEMS | | |
|---|---|---|
| DATA ELEMENT | OCCURRENCES | |
| NAME | VALUE | VALUE |
| Item Number | 200 | 300 |
| Description | Transistor | Resistor |
| Quantity on Hand | 10500 | 8000 |
| Quantity on Order | 500 | 2000 |
| Quantity Reserved | 550 | 1000 |
| Unit Price | $3.00 | $18.00 |
| Unit of Issue | 1 | 25 |

| ENTITY B: ORDER ITEMS | | |
|---|---|---|
| DATA ELEMENT | OCCURRENCES | |
| NAME | VALUE | VALUE |
| Inventory Item | 200 | 300 |
| Line Item Number | 01 | 02 |
| Quantity Ordered | 500 | 500 |
| Quantity Shipped | 500 | 500 |
| Quantity Back Ordered | 0 | 0 |
| Item Amount | $1500 | $9000 |

Figure 2-24. Concepts of Data Elements

Data elements which add information to an entity
are called *attributes*. An attribute is always dependent
on an entity. It has no meaning by itself. Depending
on its usage, an entity can be described by one single
data element or more. Ideally, an entity should be
uniquely defined by one single data element, such as
the order number of an order. Such a data element is
called *the key* of the entity. The key serves as the iden-
tification of a particular entity occurrence. It is a spe-
cial attribute of the entity. Keys are not always unique.
In such cases, entities with equal key values are called
*synonyms*. For instance, the full name of an employee
is possibly not a unique identification. In such cases,
we have to rely on other attributes such as full address,
date of employment, or an arbitrary sequence number.
A more common method is to define a new attribute,
that serves as the unique key, for example the employ-
ee number.

### Transaction

Data in itself is not the ultimate goal of a data base
management system. It is the application function
performed on the data that is more important. The best
way to represent that function is the *transaction*, which
is the smallest application unit representing a user in-
teracting with the data base. See Figure 2-25.
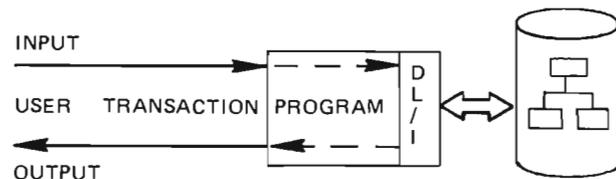


Figure 2-25. The Transaction

Transactions are processed by application programs.
In a batch system, large numbers of transactions are
accumulated (for example, all orders of a day), then
processed against the data base with a single schedul-
ing of the desired application program. Although
transactions are always distinguishable, even in batch,
we sometimes tend to think in terms of programs rather
than transactions. However, especially in a DB/DC
environment, a clear understanding of transactions is
mandatory for good data base design. The transaction
is in some way the individual usage of the application
by a particular user. As such, it is the focal point of the
DB/DC system.

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

### Access Paths

Each transaction bears in its input some kind of identi-
fication with respect to the entities used (such as the
item number when accessing an Inventory data base).
These are referred to as the *access paths* of that transac-
tion. In general, transactions require random access,
although for performance reasons, sequential access is
sometimes used. This is particularly true if the transac-
tions are batched and they are numerous, relative to
the data base size or if information is needed from most
data base records.

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

### The Transaction/Data Element Matrix

A convenient way to specify the transactions, the data
elements and their interaction is to use a
*transaction/data element matrix*, as shown in Figure
2-26.

Figure 2-26. The Transaction/Data Element Matrix

| ENTITY | DATA ELEMENTS | APPLICATION TRANSACTIONS — INVENTORY REPORT | INVENTORY INQUIRY | PURCHASE ORDERS NEW ORDER | PURCHASE ORDERS CHANGE ORDER | CUSTOMER ORDERS NEW ORDER | CUSTOMER ORDERS CHANGE ORDER | CUSTOMER ORDERS DELETE ORDER |
|---|---|---|---|---|---|---|---|---|
| INVENTORY ITEM | Item Number | (R) | [R] | (R) | (R) | R | R | |
| | Description | R | R | R | R | R | R | |
| | Quantity on Hand | R | R | R | R | R | R | |
| | Quantity on Order | R | R | U | U | R | R | |
| | Quantity Reserved | R | R | R | R | R | R | |
| | Unit Price | R | R | R | R | R | R | |
| | Unit of Issue | R | R | R | R | R | R | |
| ORDER ITEM | Inventory Item | | | | | [I] | [R] | [D] |
| | Line Item Number | | | | | I | U | D |
| | Quantity Ordered | | | | | I | U | D |
| | Quantity Shipped | | | | | I | U | D |
| | Quantity Back Ordered | | | | | I | U | D |
| | Item Amount | | | | | I | U | D |
| CUSTOMER ORDER | Order Number | | | | | [I] | [R] | [D] |
| | Reference Data | | | | | I | U | D |
| | Order Item Count | | | | | I | U | D |
| | Order Amount | | | | | I | U | D |

LEGEND:  ☐ DIRECT ACCESS PATH (KEY)
           ○ SEQUENTIAL ACCESS PATH

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

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

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

# Data Base Design Tasks

The process of designing a data base (Figure 2-27) can be generally divided into the following tasks:

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

Usually the above steps are repeated until the design satisfies the requirements. After this design process, the actual development, implementation (data base load) and production begins. During production, the system is subject to monitoring which can provide feedback for the design phase.
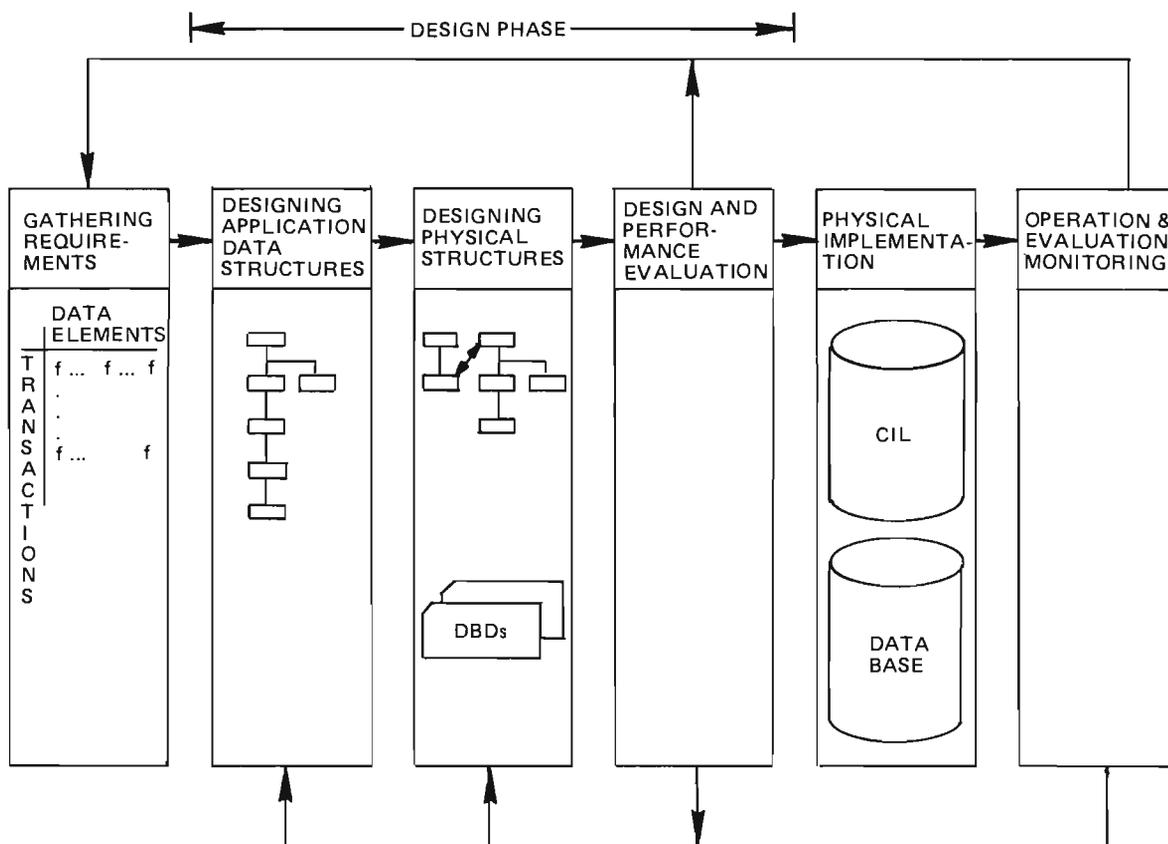


Figure 2-27. The Steps In Data Base Design

## *Gathering Requirements*

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

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

vidual applications that will use the data base being designed.

The requirement for a data base should contain:

- The data being managed, such as the entities and associated data elements.

- The relations between the entities and data elements as needed by the various users.

- The functions being performed against the data (the transactions).

- The access path as required by the transactions.

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

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

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

Additional columns could be added for miscellaneous information such as:

- Occurrence frequencies of transactions and data elements

- Size and format of data elements

- Priorities and response/turnaround time criteria

- Availability (batch or online)

- Security (who may have access to the information made available by this transaction)

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

The transaction/data element matrix, together with a detailed description of the data base and its use, constitutes the requirements for the design step. For the detailed description of the data base, its segments and fields, a documentation scheme should be established. As a minimum, forms should be used for a manual

registration of the data base, the segment layout, the fields and their attributes. It is very important to register which program uses which data elements. The next step would be to use the Assembler DSECT, COBOL COPY, or PL/I %INCLUDE facility for centralized management of segment descriptions. Ultimately, the DB/DC data dictionary system might be utilized.

### *Design the Application Data Structure*

Once the transaction/data element matrix has been built, it can be used as a guide to designing your application data structure(s). This is the logical data structure that may consist of one or more hierarchical physical data structures with the data elements arranged the way the application programmer views it.

**Segment Grouping**

In general, prior to the design of the hierarchical structure, segment design should be addressed. The process of segment design involves determining what data elements to group together to form a segment. Logically related data elements should be grouped together based upon application, usage and growth of new data elements. If you know that a future application is going to require a field that is logically associated with the other fields that form your segment, it should be placed in that segment now, even though it will not be used until the second application is implemented. Changing segment content, unlike adding new segments to a hierarchical structure, requires modifications to all application programs which utilize the segment and is thus a generally undesirable option to consider once application programming has begun. Data elements should be combined into a single segment type when they are used together. For example, name and address, or order number and order quantity, having a one-to-one relationship, should be considered candidates for inclusion in the same segment. Data elements should not be grouped together into a single segment type when they occur independently of each other, are used at different times by different application programs, or there is a large discrepancy in frequency of access. For example, if name is a highly used data element but address is a little used data element, consider the separation of name and address into different segment types, regardless of the aforementioned recommendation that logically related data elements should be placed in the same segment.

### *Design the Physical Data Structures*

In this step, the logical data structures are matched against the functions and characteristics of DL/I. Physical data base structures are defined and specified in DBDGEN control statements. The DL/I storage organization and access method is selected. Additional con-

siderations that may yield changes in the segment design are shown in Figure 2-28.

| GROUP IN ONE SEGMENT<------- >SEPARATE SEGMENTS | |
|---|---|
| Few Occurrences(<3) | Multiple Occurrences(>10) |
| Small(<20 bytes) | Large(>100 bytes) |
| High Use (Every access to record.) | Low Use (Once a Month) |
| Read-only | Update, Insert, Delete |
| General Use | Secured Use |
| Only dependent upon a single data element | Dependent upon relation of data elements |

Figure 2-28. Grouping Data Elements Into Physical Segments

The numbers shown in Figure 2-28 are not fixed. They merely provide a basis for your own estimates. Additional considerations are:

- Single verses multiple occurrences. If a data element has a high number of occurrences, it is likely to be a segment itself, especially if it is large. If it is small and highly used, then it could be stored with multiple field occurrences per segment, even in the root segment.

- If a data element needs special security, i.e., only particular applications may have access to it, it can be stored in a seperate segment together with other data elements with the same security requirements. The final result of the physical structure design steps is the data base descriptions (DBDs) and program specification blocks (PSBs) for the data bases and their processing programs.

### Selecting Data Base Access Methods
Access methods can, in general, be changed during data base reorganization without affecting application programs. Still, because the access method is one of the most critical performance factors, it should be carefully selected. This manual addresses only the considerations for the selection of HDAM, HIDAM, and SHISAM.

***When to Choose HDAM:*** HDAM is recognized in practice to be the most efficient storage organization of DL/I. It should be your first choice, especially in the online environment. HDAM's prime advantages are:

- Fast direct access (no index accesses) with few I/O operations

- Smallest working set of the six access methods

The disadvantage of HDAM is:

- Sequential access in root key order is not possible if the physical sequence of data base records in storage is not the same as the root key sequence.

This is dependent on the randomizing module and root key characteristics.

If heavy sequential processing is required and a randomizing module which maintains key sequence cannot be designed, then these techniques can be used:

- *Sort the output:* If the program is non-input driven, as is the case with many report programs, simple Get Next processing presents all the data base records in physical sequential order. The output could then be sorted in the desired order. Also, in many instances, only certain selected segments are required, so the output file of the extract can be a fairly small file.

- *Sort the input:* If there are input transactions that would normally be sorted in root key sequence they should instead be sorted in physical sequence. This can readily be done with a sort exit routine which passes each root key to the randomizing module for address calculation and then sorts on the generated addresses plus root key instead of the root key itself.

- *Build a secondary index:* A secondary index could be built with the root key as the index search argument. The cost of this should be weighed against the cost of sorting. However, the secondary index provides full generic key search capability.

***When to Chose HIDAM:*** If you cannot use HDAM for some reason, then use HIDAM (see above discussion).

***When to Choose SHISAM:*** This access method should be used only as a migration tool. That is, if your organization currently has files based on ISAM or KSDS access methods, it is not recommended for new data bases. With SHISAM, new programs can use the DL/I interface with full recovery function. Existing VSAM programs can access the data base as a regular KSDS and older ISAM-based programs can use the VSAM IIP.

### Additional Considerations
In the final steps of data base design we must look at the physical parameters closely:

- The segment length

- The number of occurrences per segment per parent

- Location of segments in the hierarchy

- Average data base record size

***Performance Aspects:*** The main measure of access performance is the number of I/O requests to satisfy the calls an application program issues. Those are mainly dependent upon the physical data base design and the

data base buffer pool size; the latter is discussed in Chapter 5. Second, the number of required DL/I calls should be considered.

Basic recommendations (HDAM and HIDAM)

* Try to locate the segments most often used together with the root segment into one control interval. The segments are initially physically stored in hierarchical sequence so the most frequently used segments should be on the left of the structure (low segment codes).

* Try to avoid long twin chains, for example, many occurrences of a particular segment under one parent.

* Inserts after initial load will first check the block of the hierarchically preceding segment for available space. If no space is found, a *bit map block* is used to search for space within plus or minus 3 cylinders. The bit map block contains one bit for each block in the data set. Bit map blocks are repeated for each n blocks; n is number of bits in a block. The bit is set to one if the corresponding block contains enough consecutive free space to hold the largest segment (including prefix) of the DBD. If no space is found, the segment is stored at the end of the data set for HIDAM and in the overflow area for HDAM.

Basic recommendation (HDAM):

* During consecutive inserts (no intervening calls) of segments of a particular data base record, the BYTES parameter in the RMNAME keyword of the DBD statement will limit the amount of data stored in the root addressable area. If the limit is reached (bytes include prefix) consecutive inserts are placed in the overflow area. Using this parameter, especially during initial load and reload, can benefit an equal distribution in the case of a large variation in data base record size.

## *Defining VSAM Clusters*

Whenever defining a VSAM cluster, you should check the DBDGEN output listing. It gives the proper access method services control statements for the definition of the KSDS (i.e. the location of the key in the KSDS record).

Always use the VSAM share option 1 and perform a LISTCAT after a DEFINE command to verify that the parameters specified in DEFINE were accepted by VSAM.

## *Data Base Design Checklist*

The following checklist gives an overview of the most important considerations/guidelines for data base design optimization. These considerations/guidelines are oriented towards performance. Sometimes, they will contradict application requirements. In such cases, a compromise must be made based on a cost/function analysis.

* Use no more complex a structure than necessary.

* Keep frequently accessed segments near the top and to the left of the hierarchy.

* Avoid widely varying segment sizes for volatile segments in the same data space.

* Check the requirement for any segment type whose relative frequency under its parent is one, or whose prefix length is greater than or equal to its data length.

* Oversegmentation results in many DL/I calls and longer reorganization times.

* Undersegmentation results in less security and less data independence.

* Avoid movement of data from one data base into another or from one part of a data base record to another.

* Avoid secondary indexing on highly volatile source segments.

* Use secondary indexing for alternate entry not sequential processing.

* If logical relationships exist, place the real logical child so that the physical path is the most active path. Also consider placing the real logical child on the longest twin chain.

* Sequencing of the logical twin chain is expensive on insert and delete processing.

* Avoid long twin chains, particularly logical twin chains.

# Chapter 3:  Data Base Implementation

## Introduction

This chapter introduces the two level data base definition language, used by data base administration, to define to DL/I the physical and logical characteristics of the data bases, and the application data structures for each application program. The first level, called the DBD (data base description), describes to DL/I the contents of the data base, the names of the segments, their hierarchical relationship, and the physical organization and characteristics of the file. The second level, the PSB (program specification block), defines the application data structure for each 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 DMB (data management control block) 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.

This chapter is divided into three sections:

* *Section 1. Data Base Description Generation:* Describes the DL/I macro instructions you must code to define your data bases. The data bases for the sample application discussed in Chapter 2 are used as examples throughout this section to guide you in determining your own data base requirements.

* *Section 2. Program Specification Block Generation:* Describes how to generate the PSBs you will need to define your application program(s) use of your data bases.

* *Section 3. Application Control Blocks Creation and Maintenance:* Describes how to create the internal control blocks, from the previously generated DBDs and PSBs, that DL/I uses to process your data bases.

## Data Base Description Generation

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

1. Basic DBDGEN for physical data bases

2. DBDGEN for logical relationships

3. DBDGEN for secondary indexes

For each data base to be used with DL/I, a data base description (DBD) must be generated. A DBD consists of a set of DL/I macro instructions, coded by you to specify the data base characteristics you need. Figure 3-1 illustrates the execution of a DBD generation. The DL/I user creates control statements that are presented to the DBD generation procedure as a normal DOS/VS problem program job. The DL/I macro instructions used for DBD generation exist in a DOS/VS source statement library. The result of a DBD generation is the creation of a DL/I DBD CSECT. The generated DBD is cataloged and link-edited into a DOS/VS core image library, for subsequent processing of the data base.

Figure 3-2 shows the sequence of the control statements in the DBD input stream.
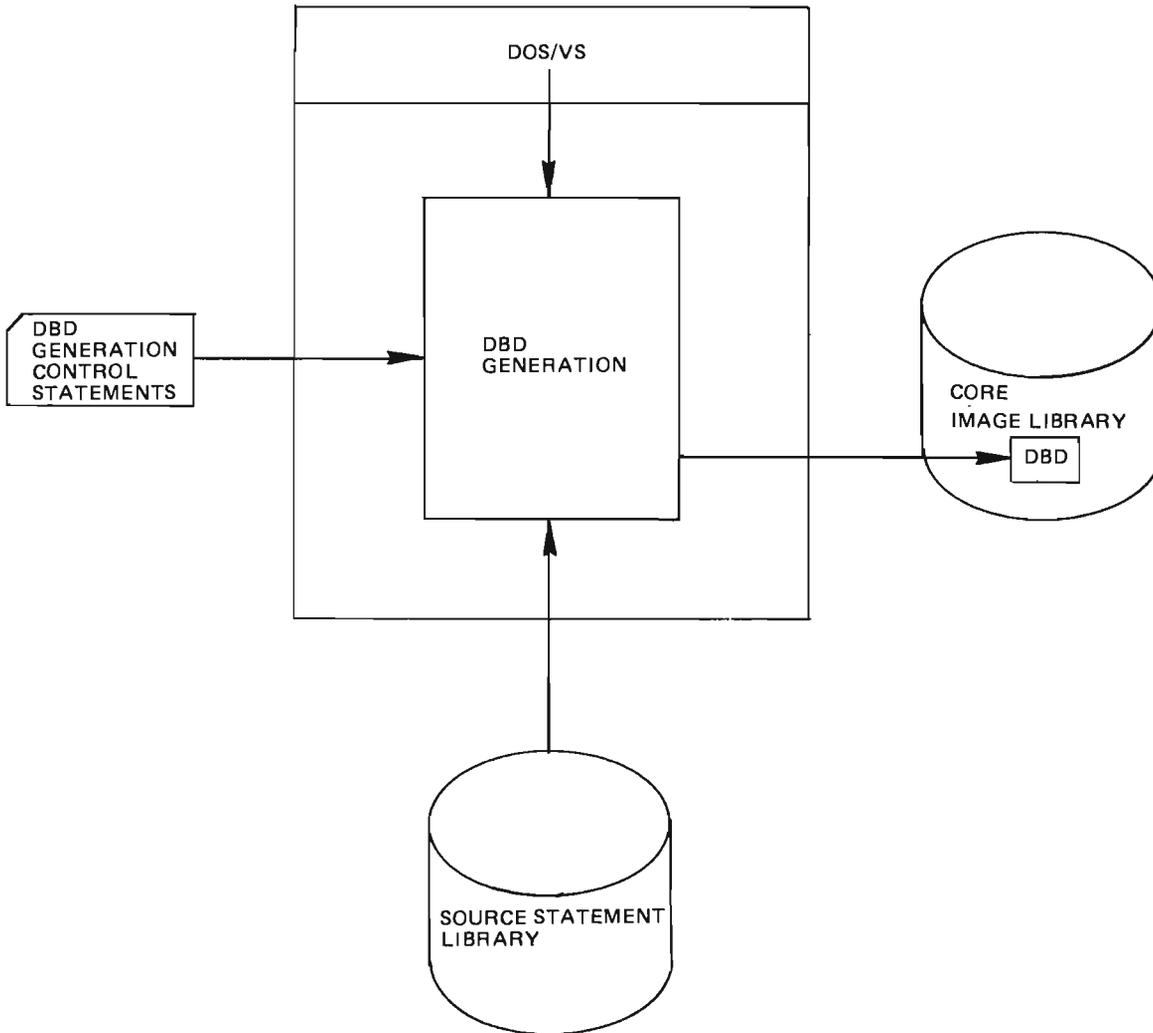
Figure 3-1. Data Base Description Generation

## DBDGEN Coding Conventions

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

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

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

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

c. The control statements are free form. Operation codes must begin after column one. Operands must follow an operation code or prior operand. The first operand must be separated from the operation code by at least one blank column. Each operand should be separated from the previous operand by a comma. Operands may be continued in subsequent statements, but must start in column sixteen on the continuation statement. A nonblank character must be coded in column 72 if a continuation statement follows.
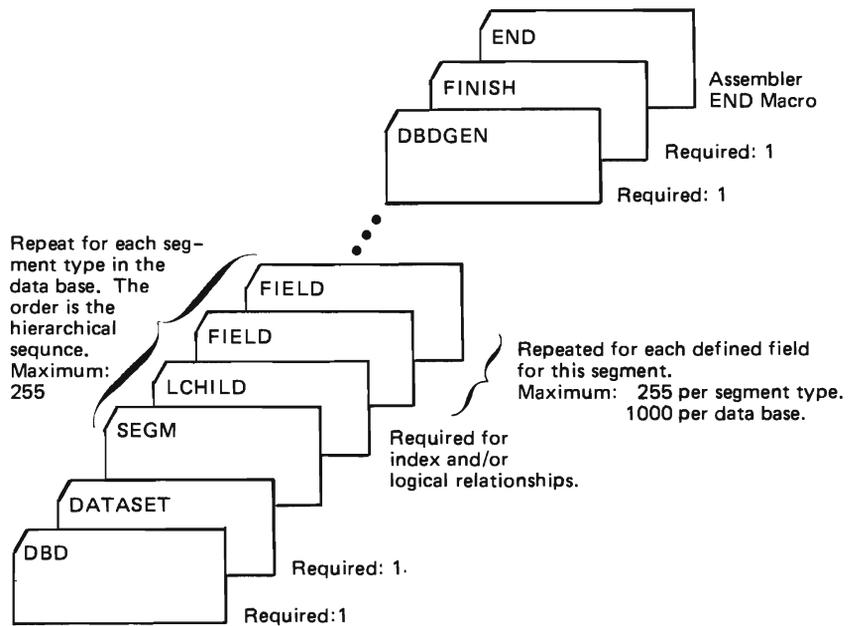
Figure 3-2. DBDGEN Input Deck Structure

| [ ] | indicates optional operands. The operand enclosed in the brackets (for example [VL]) may or may not be present, depending on whether or not the associated option is desired. If more than one item is enclosed in brackets one or none may be coded. |
|---|---|
| {} | indicates that a choice of an operand parameter must be made. One of the operand parameters from the vertical stack within the braces must be coded. |
| ,... | indicates that more than one set of parameters may be designated in the same operand. |

**Example:**

```
|← Column 1          |← Operands - Column 16
|          |←Operation - Column 10                    Column 72 →|
```

| | DBD | NAME=STDIDBP,<br>ACCESS=HDAM | * |
|---|---|---|---|

## Basic DBDGEN Control Statements Format

This section addresses the control statements required to perform the DBDGENs necessary for the sample applications. Because the purpose of this manual is to show by example the basic requirements for implementing a simple data base application, we are including only the keywords and parameters or operands of each statement as they are needed for the sample applications. (All other available keywords are mentioned only briefly.) For more detailed information about these keywords and the other options available, see the *Utilities and Guide for the System Programmer.*

Examples of the DBD statements for the sample data bases follow the discussion of the control statements.

## DBD Statement

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

| | DBD | NAME=dbdname |
|---|---|---|
| | | ,ACCESS={HDAM,RMNAME=(mod,{1    }[,rbn[,bytes]])}<br>                                 {anch}<br>    {HIDAM                                 }<br>    {INDEX                                 } |
| | | The following parameters do not apply to HDAM or HIDAM data bases<br>and therefore are given only general consideration in this manual. |
| | | {HSAM                                  }<br>{HISAM                                }<br>{SHSAM                               }<br>{SHISAM                              } |

DBD

> identifies this statement as the DBD control statement

NAME=

dbdname

> specifies the name of the DBD for this data base. This name can be from one to seven alphameric characters. However, the at-sign (@) must not be used. This name should be unique for each DBD in your installation's DL/I environment.

ACCESS=

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

HSAM

> specifies the hierarchical sequential access method.

HISAM

> specifies the hierarchical indexed sequential access method.

HIDAM

> specifies the hierarchical indexed direct access method. An INDEX DBD must be associated with any HIDAM DBD.

INDEX

> specifies the INDEX data base of a HIDAM data base. This index data base contains root segments that perform indexing to the sequence field of root segments in a HIDAM data base. This is called a primary index. This parameter is also used for secondary indexes. See "DBDGEN for Secondary Indexes" later in this chapter.

SHSAM

> specifies the simple hierarchical sequential access method. This data base consists of root segments only and does not contain segment prefixes.

SHISAM

> specifies the simple hierarchical indexed sequential access method. This data set consists of root segments only and does not contain segment prefixes.

HDAM

> specifies the hierarchical direct access method.

Notes:

- Guidelines for selecting the best access method for a particular data base are provided under the

- Parameters for the VSAM Access Method Services DEFINE command are produced in the DBDGEN output listing. These parameters must be used when defining the VSAM data set cluster. See "VSAM Requirements" later in this chapter.

RMNAME=(mod,anch,rbn,bytes)

mod

specifies the name of a randomizing module used for storing and accessing segments contained in this data base. DL/I provides several general randomizing modules (DLZHDC10, DLZHDC20, and DLZHDC30) that you can use, or you can provide your own randomizing routine. See the *Utilities and Guide for the System Programmer* for details.

anch

specifies the number of root anchor points desired in each control interval or block in the root addressable area of a HDAM data base. The default value of the parameter is one. "anch" must be an unsigned decimal integer and must not exceed 255 or be less than 1.

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

rbn

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

Note: If one of the randomizing modules supplied with DL/I will be used, this value may not be omitted. Omitting this value will cause a program check to occur in the randomizing module during load execution.

bytes

specifies the maximum number of bytes of a data base record that can be stored into the root addressable area in a series of inserts unbroken by a call to another data base record. If this parameter is omitted, no limit is placed on the maximum number of bytes of a data base record that can be inserted into this data base's root segment addressable area. "bytes" must be an unsigned decimal integer whose value does not exceed $2^{32}$-1 and is not less than 1.

## DATASET statement

This statement defines each data file that makes up the data base defined by the DBD generation. There can be only one DATASET statement for each DBD generation, and it must follow the DBD statement. The format of the DATASET statement is:

| | DATASET | DD1=fname1 |
|---|---|---|
| | | {3330}<br>{3340}<br>,DEVICE={3350}<br>{2314}<br>{TAPE}<br>{FBA }<br><br>[,BLOCK=(blk-fct-1)]<br><br>[,SCAN={cyls}]<br>{blks}<br>{<u>3</u> }<br>[,RECORD=(rec-len-1 [,rec-len-2])]<br><br>[,FRSPC=({fbff,fspf})]<br>{ <u>0</u> <u>0</u> } |
| | | The following parameters are not used for HDAM, HIDAM, or INDEX data bases and there-fore are given only general consideration in this manual. |
| | | [,DD2=fname2]<br>[,DEVADDR=(SYSnnn-1,SYSnnn-2)]<br>[,OVFLW=fname3] |

DATASET
> identifies this statement as the DATASET control statement.

DD1=fname1
> identifies the DLBL filename (1-7 characters) used in the JCL to execute DL/I application programs using the data base. It is the symbolic filename of the VSAM KSDS when ACCESS=HISAM, SHISAM, or INDEX; the VSAM ESDS when ACCESS=HDAM, or HIDAM; or the sequential input file when ACCESS=HSAM or SHSAM.

DEVICE=
> specifies the device type used for storage of this data set. TAPE may be specified only if ACCESS=HSAM or SHSAM is specified in the DBD statement.
>
> Specify DEVICE=FBA if your data base data files are to reside on FBA devices. Remember that the addressing scheme of an FBA device is different from that of other direct access storage devices. Because an FBA device is laid out as a series of fixed blocks (of 512 bytes each) starting at zero and numbered sequentially to the capacity of the device, it is addressed by block number rather than by the familiar concept of tracks and cylinders. For this reason, if an FBA device is used, the number specified in the SCAN parameter represents the number of fixed blocks to be scanned when looking for space rather than the number of cylinders (see SCAN Parameter).

BLOCK=
> specifies the control interval size (HDAM or HIDAM) to be used for each file of this data base. For SHISAM, HISAM, and INDEX data bases this parameter specifies the number of VSAM records per VSAM control interval. If this operand is not specified, the value(s) is calculated during DBD generation using a control interval size of 2048 bytes wherever possible. For HIDAM and HDAM, the parameter blk-fact-1 is the size of the VSAM ESDS control interval and must be a multiple of 512 bytes. The maximum value permitted by DL/I is 4096.
>
> In choosing the block size, the following considerations apply (Remember, the block size is the CI size):
>
> - Try to choose a CI size that allows all highly needed 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 primarily processing directly, 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.

Figure 3-3 may be helpful in calculating the size in bytes for the BLOCK and RECORD parameters of the DATASET statement.

SCAN=
> specifies the number of cylinders (for direct access devices) or blocks (for FBA devices) to be scanned in both directions when searching for available storage space during segment insertions. This operand is used only for HDAM or HIDAM data bases.
>
> If you specify *cyls*, it can be any integer from 0 to 255. Typical values are 0 to 5. The default value is 3 (suggest you start with 0). If SCAN=0 is specified, only the current cylinder is scanned for space. Scanning is performed in both directions from the current position. If space is not found for segment insertion within the bounds defined by this operand, space at the end of the data base is used.

| ACCESS METHOD | ALLOCATION IN BYTES | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| | SEGMENT PREFIX | DL/I CONTROL INFORMATION | | VSAM CI CONTROL INFORMATION | 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 |
| HIDAM | 2 + 4T + 4CI + 8C2 + 4LP1 + 8LP2 + 4LC + 4LP3 + PP | NA | 8 | 10 | 4068 | 4096 | 2048 |
| HDAM | | NA | 4+ 4RAP | 10 | 4068 | 4096 | 2048 |

**Notes:**

| | |
| --- | --- |
| CI = | VSAM Control Interval |
| T = 1 | if POINTER=TWIN |
| | if POINTER=NOTWIN |
| T = 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. |

Figure 3-3. Maximum Segment Lengths

If you specify *blks,* it can be any integer from 0 to 32767. If this parameter is omitted, a default is calculated that is approximately equal to three cylinders. If SCAN=0 is specified, only the current fixed block is scanned for space. Scanning is performed in both directions from the current position. If space is not found for segment insertion within the bounds defined by this operand, space at the end of the data base is used.

RECORD=

specifies the data management logical record length(s) to be used for each data file. This operand is optional. If omitted, the values are calculated during DBD generation. The rec-len-1 and rec-len-2 must be numeric values which are a multiple of two. The meaning of each parameter depends on the type of data base being defined.

HIDAM and HDAM

This parameter is ignored.

INDEX

The parameter rec-len-1 is the KSDS record length which is large enough to contain the index pointer segment plus the length of the prefix plus the length of the DL/I control information.

FRSPC=

specifies the amount of free space to be reserved in the data base during a load (or reload) operation for HD (HDAM or HIDAM) data bases.

*fbff*

specifies that every *n*th block is to be left free. This is a number from 0 to 100 excluding 1. Zero is the default.

*fspf*

specifies the percentage of each block to be left free. This is a number from 0 to 99. This number expresses the minimum space to be left free. Due to segment size, the actual space may be larger. Zero is the default.

Either *fbff* or *fspf* or both may be specified to achieve any combination of free and/or partially free blocks within the constraints of the parameter values.

A specification of FRSPC=(5,40) results in a data base load (or reload) in which every fifth block (5, 10, 15, etc.) would be left free and at least 40 percent of all other blocks would also be left as free space. This free space would be used at insert time to place the inserted segments as close to the related segments as possible.

The amount of free space to be reserved depends on record size and the number of inserted segments anticipated. There are no rules for determining the necessary free space. Various values will have to be experimented with to find the optimum for each data base.

**SEGM Statement**

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

```
        SEGM    NAME=seg-name1

                         {0                    }
                [,PARENT={((seg-name2[{,SNGL}))}]
                                      {,DBLE}

                [,BYTES={bytes                }]
                        {(max-bytes,min-bytes)}

                          {TWIN    }
                [,POINTER={TWINBWD}]
                          {NOTWIN  }

                        {,FIRST}
                [,RULES=({,LAST })]
                        {,HERE  }

                [,COMPRTN=(routine-name[,D,INIT])]
```

NAME=
> specifies the name of the segment being defined. The specified name is used by DL/I and application programs in all references to this segment. Duplicate segment names are not allowed within a DBD generation. The parameter seg-name-1 must be 1 to 8 alphameric characters.

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

> SNGL
> > specifies only a physical child first pointer is used in this segment's parent.

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

> DBLE should be specified if the average twin chain is more than 3 to 5 and segment has no sequence field and frequent inserts.

BYTES=
> specifies the length of the data portion of the segment in bytes. This length does not include the prefix, which is established solely by DL/I. This length cannot exceed the maximum logical record length or control interval size of the data set minus the space occupied by system fields.

> If this parameter is not specified, DL/I calculates the size of the segment based on the location and length of the fields identified as belonging to it.

> *max-bytes* specifies in bytes the maximum length of the data portion of a variable length segment type, including the 2-byte length field (see "Variable Length Segments", in Chapter 2).

> *min-bytes* specifies the minimum length (including the 2-byte length field) of the data portion of a variable length segment type. Four is the minimum specification. If you specify a minimum length greater than the actual minimum length (+2) of the data to be stored, DL/I will reserve an amount of space equal to the min-bytes specification. This, in effect, reserves free space at the end of the inserted data, and may result in more efficient processing later if the data length is increased.

**POINTER=**

(or its abbreviation PTR) specifies the fields to be reserved in the segment's prefix area. These fields are used to relate this segment to its physical twin segments and, in the case of a logical child segment, to its logical twin segments. The POINTER operand applies to HDAM and HIDAM data bases only.

- TWINBWD (twin forward and backward pointers)

  specify this parameter if:

  - No sequence field is defined and frequent inserts are expected.

  - Retrieve last plus subsequent delete is frequently used.

  - The segment is a logical child (see Phase 2).

  - It is the root segment of a HIDAM data base.

- TWIN (only twin forward pointer)

  this parameter is usually specified and is the default for a physical segment or a logical parent segment.

- NOTWIN (no twin pointer)
  specify this parameter to ensure that no more than one occurrence of this segment will exist under this parent.

Note: If you desire more details on the use and creation of pointers, see Appendix A in the *Utilities and Guide for the System Programmer.*

**RULES=**

```
  { ,FIRST}
({ ,LAST })
  { ,HERE }
```

This parameter of the RULES keyword determines where new occurrences of the segment being defined by this SEGM statement are to be inserted in the physical twin chain. This value is significant only when processing segments without a sequence field or without a unique sequence field (as indicated by the FIELD statement). It is ignored for a segment which contains a unique sequence field.

FIRST (F)

states that a new occurrence is to be inserted before the first existing occurrence of this segment type.

LAST (L)

states that a new occurrence is to be inserted after the last existing occurrence of this segment type.

HERE (H)

assumes the user has determined positioning by a previous DL/I call, and the new occurrence is inserted before the segment that satisfied the last call.

**COMPRTN=**

this keyword is used to select the segment compression option. This facility allows the reduction in length of variable length segments to increase the effective utilization of secondary storage. This operand must not be specified for virtual logical child segments or secondary index source segments.

routine-name

specifies the name of a user-supplied routine used to compress this segment. This name must be a 1- to 8-character alphameric value and must not be the same as any other name in any DOS/VS core image library that is assigned.

D

(default value) maintains upward source compatibility to IMS/VS.

INIT

indicates that initialization and termination processing control is required by the segment compression routine.

## LCHILD Statement

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

| | LCHILD | NAME=(seg-name1,db-name) |
|---|---|---|
| | | [,POINTER=INDX] |
| | | [,INDEX=fld-name] |

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

NAME=
  (seg-name1,db-name)
> seg-name1 is the name of the HIDAM root segment and db-name is the name of the HIDAM data base as coded in the DBD statement.

INDEX=
  fld-name
> specifies the name of the sequence field of the HIDAM root segment.

For the HIDAM main data base, code:

NAME=
  (seg-name1,db-name)
> seg-name1 is the name of the only segment in the primary INDEX data base for this data base, and db-name is the name of that INDEX data base.

POINTER=
  INDX
> provides for the linkage with the INDEX data base.

## FIELD Statement

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

| | FIELD | NAME=(fld-name1[,SEQ[{,U}]]) |
|---|---|---|
| | | {,M} |
| | | [,BYTES=bytes] |
| | | [,START=pos] |
| | | [,TYPE=t] |

NAME=
  fld-name1
> specifies the name of the field being defined within a segment type. The name specified can be referred to by an application program in a DL/I call SSA. Duplicate field names must not be defined for the same segment type. The fld-name1 must be a 1- to 8-character alphameric value.

**SEQ**

> the presence of the keyword SEQ as a parameter of this operand identifies this field as a sequence field in the segment type. As a general rule, a segment can have only one sequence field. If a sequence field is specified, then its value must be unique for all segment occurrences under a given parent.
>
> A unique field is optional for all dependent segment types. It must be provided for the root segment of all data bases except simple HSAM and HSAM.
>
> When no sequence field is defined for a segment, new occurrences of the segment will be inserted at the end of the physical twin chain unless changed by the RULES parameter in the SEGM statement. It is highly recommended that all segments which participate in a logical relationship have sequence fields. This includes physical and logical parents as well as logical child segments.

**U**

> indicates that only unique values of this sequence field are allowed in which case any RULES parameter in the SEGM statement is ignored.

**M**

> indicates that duplicate values of this sequence field can occur in multiple occurrences of the segment. Each new occurrence of a segment will be inserted according to the appropriate RULES operand specification (see Phase 2) or default.

**BYTES=**

specifies the length of this field in terms of bytes and must be a numeric term whose value does not exceed 256 (236 for the root segment sequence field of a simple HISAM, HISAM, HIDAM, or INDEX data base).

Notes:

- The BYTES parameter must be specified for field data types X, P, C, or Z (see TYPE parameter for field data types).
- The BYTES parameter is optional for field data types H and F. If omitted, DL/I assumes a field length of 2 bytes for type H and 4 bytes for type F.
- Do not specify the BYTES parameter for field data types E, D, or L. These data types have implicit lengths of 4, 8, and 16 respectively.

**START=**

specifies the starting position of the field being defined in terms of bytes relative to the beginning of the segment. Start position for the first byte of a segment is one (maximum 32767). Overlapping fields are permitted. If an overlapping field starts in the same position as a previously defined field, you may specify the name of the previously defined field, instead of a numeric value, to indicate the starting position (START=*fieldname*). Each field must not extend beyond the defined segment length (start position plus byte value).

If you do not specify this parameter, DL/I places this field adjacent to the end of the previous field, or if it is the first field in the segment, at the beginning of the segment (START=1). (Note that for concatenated segments, the beginning of the segment is the start of the destination parent concatenated key.)

**TYPE=**

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

'X' - hexadecimal
'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)

If this parameter is omitted, TYPE=C is assumed. It is recommended, however, that you explicitly specify the desired data type. Failure to do so could result in problems if you later decide to use the "automatic data format conversion" option of field level sensitivity (see TYPE parameter in SENFLD statement).

**Notes:**

* All DL/I calls perform field comparisons on a byte-by-byte binary basis. No check is made by DL/I to ensure that the data contained within a field is of the type specified by this operand, except when the defined field is indexed, or converted by the Field Level Sensitivity feature.

* Do not unnecessarily define fields in the DBD as this increases the size of the DBD and consequently the working set. You could include FIELD statements as comments (* in column 1) for documentation. However, be sure to define all fields that will also be defined in the SENFLD statements for PSB generation.

## DBDGEN Statement

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

| | DBDGEN | |
|---|---|---|

## FINISH Statement

This statement must be included for source-level compatibility with IMS/VS. The format is:

| | FINISH | |
|---|---|---|

## END Statement

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

| | END | |
|---|---|---|

## *Execution of DBDGEN (JCL)*

DBDGEN is run as a standard DOS/VS job. The DL/I macro instructions used for DBDGEN exist in a DOS/VS source statement library. The generated DBD is cataloged and link-edited into a DOS/VS core image library. DBDGEN requires the following job control statements:

```
// JOB        DBDGEN
// OPTION     CATAL
// EXEC       ASSEMBLY
           DBD
           DATASET
           SEGM
           LCHILD
           FIELD            DBD GENERATION CONTROL STATEMENTS
           DBDGEN
           FINISH
           END
/*
// EXEC      LNKEDT
/&
```

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

## *Examples of Physical DBDs*

Figure 3-4 shows a sample HDAM data base and the DBD statements required to assemble it. This is the Phase 1 Inventory data base of the batch sample application. The data base is assumed to reside on a 3340. If the device is other than a 3340, the DATASET statement should be changed.

```
                    ┌─────────────┐
                    │  CUSTOMER   │
                    │  NAME/      │
                    │  ADDRESS    │
                    │  (STSCCST)  │
                    └──────┬──────┘
          ┌────────────────┼────────────────┐
   ┌──────┴──────┐   ┌─────┴─────┐   ┌───────┴───────┐
   │  CUSTOMER   │   │  CREDIT   │   │   CUSTOMER    │
   │  LOCATION   │   │  STATUS   │   │   HISTORY     │
   │  (STSCLOC)  │   │ (STSCSTA) │   │  (STSCHIS)    │
   └──────┬──────┘   └───────────┘   └───────────────┘
   ┌──────┴──────┐
   │  CUSTOMER   │
   │  ORDER      │
   │  (STPCORD)  │
   └──────┬──────┘
   ┌──────┴──────┐
   │  ORDER      │
   │  ITEM       │
   │  (STCCITM)  │
   └─────────────┘
```

**Customer Data Base**

```
                    ┌─────────────┐
                    │  INVENTORY  │
                    │  ITEM       │
                    │  (STPIITM)  │
                    └──────┬──────┘
          ┌────────────────┼────────────────┐
   ┌──────┴──────┐   ┌─────┴─────┐   ┌───────┴───────┐
   │  VENDOR     │   │ SUBSTITUTE│   │   ITEM        │
   │  (STSIVND)  │   │ ITEM      │   │   LOCATION    │
   │             │   │ (STCISUB) │   │  (STSILOC)    │
   └─────────────┘   └───────────┘   └───────────────┘
```
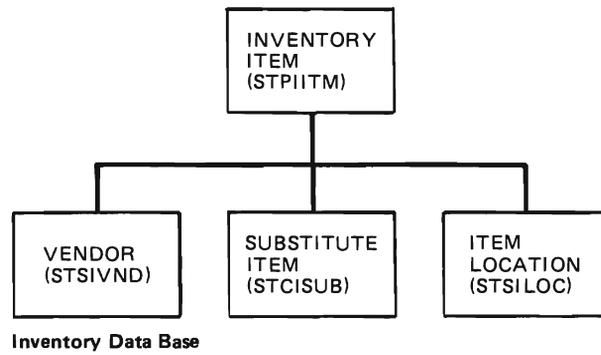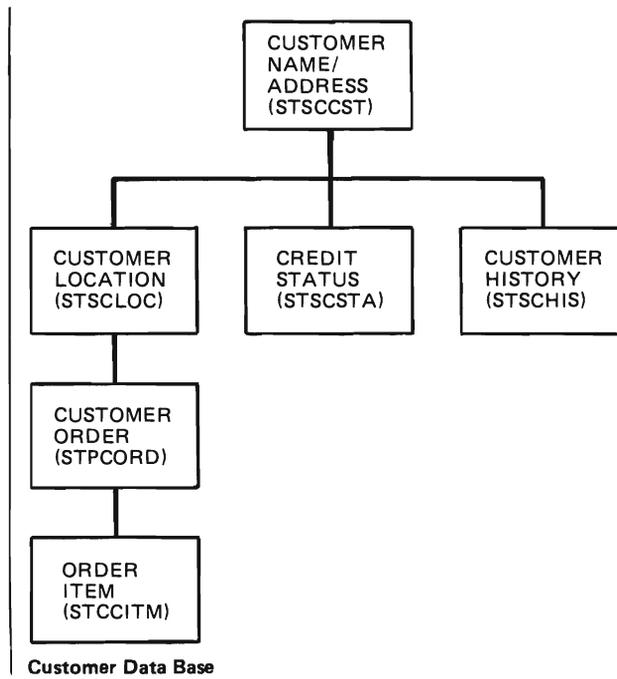
**Inventory Data Base**

Figure 3-4. DBDGEN for the Phase 1 Data Bases (Part 1 of 4)

```
// JOB STJDBDGN GENERATE DBDS FOR SAMPLE PROBLEM
// OPTION CATAL,NODECK
// EXEC ASSEMBLY
* THIS IS THE PHYSICAL DBD FOR THE CUSTOMER DATA BASE
* PHASE 1 NO LOGICAL RELATIONSHIPS, NO SECONDARY INDEXES
        PRINT NOGEN                    NO MACRO EXPANSION PRINTING
        DBD                                                           X
              NAME=STDCDBP,            DATA BASE DESCRIPTION NAME      X
              ACCESS=HDAM,             HIERARCHICAL DIRECT            X
              RMNAME=(DLZHDC10,        RANDOMIZING ROUTINE PHASENAME  X
              3,                       ROOT ANCHOR POINTS PER BLOCK   X
              100,                     ROOT ADDR. AREA HI RELATIVE BLK X
              600)                     INSERT BYTES LIMIT FOR RAA
        DATASET                                                       X
              DD1=STDCDBC,             DLBL FILE NAME                 X
              DEVICE=3340,             DISK DEVICE                    X
              BLOCK=(2048),            VSAM CONTROL INTERVAL SIZE     X
              SCAN=2                   # CYLINDERS SCAN FOR ISRT SPACE
        SEGM                                                          X
              NAME=STSCCST,            SEGMENT NAME FOR CUST NAME/ADDR X
              PARENT=0,                IT IS A ROOT SEGMENT           X
              BYTES=106,               DATA LENGTH                    X
              POINTER=TWIN             PHYSICAL TWIN FWD ONLY
        FIELD                                                         X
              NAME=(STQCCNO,SEQ,U),    UNIQUE KEY FIELD (CUST #)      X
              BYTES=6,                 FIELD LENGTH                   X
              START=1,                 WHERE IT STARTS IN SEGMENT     X
              TYPE=C                   ALPHAMERIC DATA
        SEGM                                                          X
              NAME=STSCLOC,            SEGMENT NAME CUSTOMER LOCATION X
              PARENT=STSCCST,          PARENT IS CUST. NAME/ADDR SEGM X
              BYTES=106,               FIELD LENGTH                   X
              POINTER=TWINBWD          BOTH PHYS. TWIN FWD AND BWD
        FIELD                                                         X
              NAME=(STQCLNO,SEQ,U),    UNIQUE KEY FIELD (LOCATION #)  X
              BYTES=6,                 FIELD LENGTH                   X
              START=1,                 WHERE IT STARTS IN SEGMENT     X
              TYPE=C                   ALPHAMERIC DATA
        SEGM                                                          X
              NAME=STPCORD,            SEGMENT NAME CUSTOMER ORDER    X
              PARENT=STSCLOC,          PARENT IS CUST. LOCASTION SEGM X
              BYTES=55,                DATA LENGTH                    X
              POINTER=TWINBWD          BOTH PHYS.TWIN FWD AND BWD
        FIELD                                                         X
              NAME=(STQCODN,SEQ,U),    UNIQUE KEY FIELD (DATE & ORD #) X
              BYTES=12,                FIELD LENGTH                   X
              START=1,                 WHERE IT STARTS IN SEGMENT     X
              TYPE=C                   ALPHAMERIC DATA
```

Figure 3-4. DBDGEN for the Phase 1 Data Bases (Part 2 of 4)

```
        SEGM                                                              X
              NAME=STCCITM,              SEGMENT NAME LINE ITEM            X
              PARENT=STPCORD,            PHYSICAL PARENT IS CUSTOMER ORD   X
              BYTES=38,                  DATA LENGTH                       X
              POINTER=TWINBWD            BOTH PHYS. TWIN FWD AND BWD
*
* THE FOLLOWING FIELDS ARE DEFINED TO SHOW AN EXAMPLE OF FIELD LEVEL
* SENSITIVITY.  NOTE THAT IT IS NOT REQUIRED FOR THE SEQUENCE FIELD
* TO BE DEFINED FIRST AND IF THE START PARAMETER IS NOT CODED THE FIELD
* IS ASSUMED CONTIGUOUS TO THE PRECEEDING FIELD.  SEE PSB'S STBCUSR
* AND STBCUSU FOR AN EXAMPLE OF HOW THE FIELDS ARE SELECTED BY THE
* APPLICATION PROGRAM.
        FIELD NAME=STKCIIN,             INVENTORY ITEM NUMBER             X
              BYTES=6,                   FIELD LENGTH                      X
              TYPE=C                     ALPHAMERIC DATA
        FIELD                                                             X
              NAME=(STQCILI,SEQ,U),      UNIQUE KEY FIELD (LINE #)         X
              BYTES=2,                   FIELD LENGTH                      X
              START=7,                   WHERE IT STARTS IN SEGMENT        X
              TYPE=C                     ALPHAMERIC DATA
*
        FIELD NAME=STFCIQO,BYTES=6,TYPE=C   QUANTITY ORDERED
        FIELD NAME=STFCIQS,BYTES=6,TYPE=C   QUANTITY SHIPPED
        FIELD NAME=STFCIQB,BYTES=6,TYPE=C   QUANTITY BACK ORDERED
        FIELD NAME=STFCIAM,BYTES=12,TYPE=C ITEM AMOUNT
*
        SEGM                                                              X
              NAME=STSCSTA,              SEGMENT NAME CREDIT STATUS        X
              PARENT=STSCCST,            PARENT IS CUST. NAME/ADDR SEGM    X
              BYTES=24,                  DATA LENGTH                       X
              POINTER=TWIN               PHYSICAL TWIN FWD ONLY            X
              RULES=(,FIRST)             INSERT THIS OCCURRENCE BEFORE
*                                        EXISTING OCCURENCE
*                                        OF SEGMENT
*                                        NOTE THERE IS NO KEY FIELD
*
        SEGM                                                              X
              NAME=STSCHIS,              SEGMENT NAME CUSTOMER HISTORY     X
              PARENT=STSCCST,            PARENT IS CUST. NAME/ADDR SEGM    X
              BYTES=(130,53),            SEGMENT IS VARIABLE LENGTH        X
              COMPRTN=DLZSAMCP,          NAME OF COMPRESSION ROUTINE       X
              POINTER=TWINBWD            BOTH PHYS. TWIN FWD AND BWD
        FIELD                                                             X
              NAME=(STQCHDN,SEQ,U),      UNIQUE KEY FIELD (DATE & ORD #)   X
              BYTES=12,                  FIELD LENGTH                      X
              START=3,                   WHERE IT STARTS IN SEGMENT        X
              TYPE=C                     ALPHAMERIC DATA
        DBDGEN                           REQUIRED TO MARK DBD END
        FINISH                           FOR SOURCE COMPAT WITH IMS/VS
        END
/*
// EXEC  LNKEDT
/&
```

Figure 3-4. DBDGEN for the Phase 1 Data Bases (Part 3 of 4)

```
// JOB DBDGEN
// OPTION      CATAL,NODECK
// EXEC        ASSEMBLY
* THIS IS THE PHYSICAL DBD FOR THE INVENTORY DATA BASE
* PHASE 1 NO LOGICAL RELATIONSHIPS, NO SECONDARY INDEXES
        PRINT NOGEN                   NO MACRO EXPANSION PRINTING
        DBD                                                           X
              NAME=STDIDBP,           DATA BASE DESCRIPTION NAME      X
              ACCESS=HDAM,            HIERARCHICAL DIRECT             X
              RMNAME=(DLZHDC30,       RANDOMIZING ROUTINE PHASENAME   X
              3,                      ROOT ANCHOR POINTS PER BLOCK    X
              100,                    ROOT ADDR. AREA HI RELATIVE BLK X
              400)                    INSERT BYTES LIMIT FOR RAA
        DATASET                                                       X
              DD1=STDIDBC,            DLBL FILE NAME                  X
              DEVICE=3340,            DISK DEVICE                     X
              BLOCK=(2048),           VSAM CONTROL INTERVAL SIZE      X
              SCAN=2                  # CYLINDERS SCAN FOR ISRT SPACE
        SEGM                                                          X
              NAME=STPIITM,           SEGMENT NAME INVENTORY ITEM     X
              PARENT=0,               IT IS A ROOT SEGMENT            X
              BYTES=56,               DATA LENGTH                     X
              POINTER=TWIN            PHYSICAL TWIN FWD ONLY
        FIELD                                                         X
              NAME=(STQIINO,SEQ,U),   UNIQUE KEY FIELD (ITEM #)       X
              BYTES=6,                FIELD LENGTH                    X
              START=1,                WHERE IT STARTS IN SEGMENT      X
              TYPE=C                  ALPHAMERIC
        SEGM                                                          X
              NAME=STSIVND,           AUTHORIZED VENDOR INFORMATION   X
              PARENT=STPIITM,         PARENT IS INVENTORY ITEM SEGM.  X
              BYTES=106,              FIELD LENGTH                    X
              POINTER=TWIN            PHYSICAL TWIN FWD ONLY
        FIELD                                                         X
              NAME=(STQVVNO,SEQ,U),   UNIQUE KEY FIELD (VENDOR #)     X
              BYTES=6,                FIELD LENGTH                    X
              START=1,                WHERE IT STARTS IN SEGMENT      X
              TYPE=C                  ALPHAMERIC DATA
        SEGM                                                          X
              NAME=STCISUB,           SEG. NAME FOR SUB-ITEM INFO.    X
              PARENT=STPIITM,         PARENT IS INVENTORY ITEM SEGM   X
              BYTES=56,               DATA LENGTH                     X
              POINTER=TWINBWD         BOTH PHYS. TWIN FWD AND BWD
        FIELD                                                         X
              NAME=(STQCCNO,SEQ,U),   UNIQUE KEY FIELD (SUB-ITEM #)   X
              BYTES=6,                FIELD LENGTH                    X
              START=1,                WHERE IT STARTS IN SEGMENT      X
              TYPE=C                  ALPHAMERIC DATA
        SEGM                                                          X
              NAME=STSILOC,           SEGMENT NAME INVENTORY LOCATION X
              PARENT=STPIITM          PARENT IS INVENTORY ITEM SEGM.  X
              BYTES=12,               DATA LENGTH                     X
              POINTER=TWINBWD         BOTH PHYS. TWIN FWD AND BWD
         FIELD                                                        X
              NAME=(STQILNO,SEQ,U),   UNIQUE KEY FIELD INVENTORY LOC# X
              BYTES=6,                FIELD LENGTH                    X
              START=1,                WHERE IT STARTS IN SEGMENT      X
              TYPE=C                  ALPHAMERIC DATA
        DBDGEN                        REQUIRED TO MARK DBD END
        FINISH                        FOR SOURCE COMPAT WITH IMS/VS
        END
/*
// EXEC    LINKEDT
/&
```

Figure 3-4. DBDGEN for the Phase 1 Data Bases (Part 4 of 4)

Figure 3-5 shows an example of a HIDAM version of the same data base. The difference here is that two DBDs are required, one for the index data base and one for the main data base. The index for a HIDAM data base is called a *primary* index. The DBD example in Figure 3-5 is for the primary index of the Inventory data base.

The HIDAM DBD for the Inventory data base differs from the HDAM DBD in that ACCESS=HIDAM is specified in the DATASET statement, and an LCHILD statement is added to relate this data base to the index data base. The LCHILD statement is placed after the SEGM statement for STPIITM, and identifies the segment name and the DBD name of the index data base.

The coding of a DBD for a primary index is slightly different from the coding of a DBD for a secondary index. See "DBDGEN for Secondary Indexes" later in this chapter for details.

HIDAM
PRIMARY INDEX DATA BASE
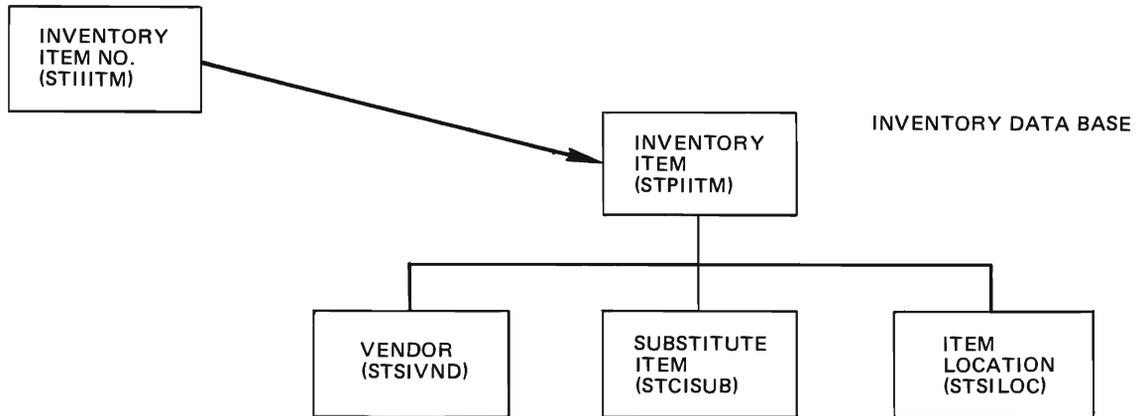
```
DBD                                DATA BASE DESCRIPTION NAME     X
        NAME=STDIX1P,              DATA BASE DESCRIPTION NAME     X
        ACCESS=INDEX               THIS IS AN INDEX
DATASET                                                           X
        DD1=STDIX1C,               DLBL FILE NAME                 X
        DEVICE=3340                DISK DEVICE
SEGM                                                              X
        NAME=STIIITM,              SEGMENT NAME OF THE INDEX      X
        PARENT=0,                  IT IS A ROOT SEGMENT           X
        BYTES=6                    LENGTH
LCHILD                                                            X
        NAME=(STPIITM,             TARGET SEGMENT ITEM NUMBER     X
        STDIDBP),                  FOUND IN INVENTORY DATA BASE   X
        INDEX=STQIINO              INDEXED FIELD NAME
FIELD                                                             X
        NAME=(STYIINO,SEQ,U),      UNIQUE KEY FIELD               X
        BYTES=6,                   FIELD LENGTH                   X
        START=1,                   WHERE IT STARTS IN SEGMENT     X
        TYPE=C                     ALPHAMERIC DATA
DBDGEN                             TO MARK END OF DBD
FINISH                            FOR SOURCE COMPAT WITH IMS/VS
END
```

Figure 3-5. Sample DBD For a HIDAM Primary Index Data Base

# DBDGEN for Logical Relationships

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

- Additional control statements and parameters can be specified in the physical DBD.

- A new type of DBD is created for the definition of the logical data base, however, this is done with an extension of the existing control statements.

The DBDGEN process itself is unchanged.

## Coding a Logical Relationship in a Physical DBD

The following control statements are unchanged:

```
DBD
FIELD
DBDGEN
FINISH
END
```

The following statements are extended:

```
SEGM
LCHILD
```

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

|  | SEGM | NAME=seg-name1 |
|---|---|---|
|  |  | ,PARENT=<br>((seg-name2,{SNGL})<br>{DBLE}<br>,(lpseg-name[,V,db-name1])) |
|  |  | ,BYTES=bytes |
|  |  | ,POINTER=({TWIN    },[{LTWIN    }])<br>{TWINBWD} [{LTWINBWD}]<br>{NOTWIN }  |
|  |  | {P}{P}{P}    {,FIRST}<br>,RULES=([{L}{L}{L}][{,LAST }])<br>{V}{V}{V}    {,HERE } |

NAME=
seg-name 1
is the name of the logical child segment.

PARENT=
seg-name 2
is the name of the physical parent segment of this logical child.

SNGL or DBLE
have the same meaning as before.

lpseg-name
is the name of the logical parent of this logical child.

db-name1
is the DBD name of the logical parent's data base.

v (default value)
maintains upward source compatibility to IMS/VS

**BYTES=**
> has the same meaning as before. Notice however that the logical child always contains the logical parent's concatenated key in the first n bytes, and its length must be included here. If you do not specify this parameter, DL/I automatically calculates a segment size large enough to contain all defined fields.

**POINTER=**

**TWIN (T)**
> the same considerations as before apply.

**TWINBWD (TB)**
> It is highly recommended that you specify TB.

**NOTWIN**
> may be specified to ensure that there will never be more than one occurrence of this segment per physical parent.

**LTWIN (T)**
> if specified, only a logical twin forward pointer is used for the logical twin chain.

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

**RULES=**

```
{P}{P}{P} {,FIRST}
{L}{L}{L}[{,LAST }]
{V}{V}{V} {,HERE }
```

The parameter values are:

- P specifies physical rule

- L specifies logical rule

- V specifies virtual rule

> The first parameter of this operand is of the format xxx, where x can be one of the characters P, L, or V. Each of the three positions can contain the same or different characters. If all three are omitted the default values are assumed. Likewise the second and third, or just the third can be omitted, in which case the default values are assumed for the omitted positions.

> In the first parameter the first value, x.., applies to SEGMENT INSERTION, the second value,.x., applies to SEGMENT DELETION, and the third value, ..x, applies to SEGMENT REPLACEMENT.

> Note: For a logical child segment type, the third value, the replace rule, must be V. Any other rule specified will be changed to V during DBD generation.

> The parameter xxx is only meaningful for physical logical child segments and for their logical parent segments if the logical relationship is unidirectional or for their physical and logical parent segments if the logical relationship is bidirectional.

> **Recommendation:** Do not use this parameter for segments that do not participate in a logical relationship.

> The following paragraphs will assist you in determining when to specify P, L, or V for the RULES= parameter.

*Insertion Rules:* The insertion rules have meaning only for the destination parent in the access path of a logical relationship. When a concatenated segment is presented for insertion into a logical data base, its destination parent portion may or may not be inserted, depending on which of the insertion rules, as shown in Figure 3-6, is specified for the destination parent.

*Deletion Rules:* When a segment is deleted all its dependent segments are also deleted. The physical deletion of a segment is caused by an explicit deletion request either for the segment itself, or for a segment on which it is physically dependent. If deletion of a logical child segment was caused by propagating an explicit deletion request across a logical relationship, then it is called a logical deletion request.

If a segment is deleted on one path, it can no longer be accessed through this path. It may, however, still be accessible through the other path. Such conditions are indicated in the delete byte of the segment prefix. The deletion rules for the logical parent and logical child are shown in Figures 3-7 and 3-8.

*Replacement Rules:* The replacement rules determine what actions take place when a concatenated segment is presented in a REPLACE call and one or both portions of it are to be altered. Replacement rules, as shown in Figure 3-9, can be specified for the destination parent portion of the concatenated segment.

RULES=

```
            {,FIRST}
(...{,LAST })
            {,HERE }
```

The second parameter of this operand determines where new occurrences of the segment being defined by this SEGM statement are to be inserted in the physical twin chain. This value is significant only when processing segments without a sequence field or without a unique sequence field (as indicated by the FIELD statement). It is ignored for a segment that contains a unique sequence field.

| IF AN INSERT CALL OF A CONCATENATED SEGMENT IS ISSUED . . . | | INSERT RULE SPECIFIED IN DP | | | | | |
|---|---|---|---|---|---|---|---|
| | | P | | L | | V | |
| AND .. | DESTINATION PARENT EXISTS IN DB | X | | X | | X | |
| | DESTINATION PARENT DOES NOT EXIST IN DB | | X | | X | | X |
| THEN | LOGICAL CHILD SEGMENT IS INSERTED | X | | X | X | X | X |
| | DESTINATION PARENT PORTION OF THE CON-CATENATED SEGMENT IGNORED | X | | X | | | |
| | DESTINATION PARENT SEGMENT IS INSERTED (*REPLACED) | | | | X* | X | X |
| | CALL IS REJECTED (IX) | | X | | | | |
| NOTE: | The insert rules are specified for destination parent only. Insert rules affect whether or not the DP portion of a concatenated segment is inserted. | | | | | | |

Figure 3-6. Insertion Rules for Logical Relationships

| | | DELETE RULE SPECIFIED IN LP | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | P (BI-DIR) | | L | | V | | |
| IF . . . | A DELETE CALL IS ISSUED FOR AN LP | X | X | X | X | X | X | |
| AND | ALL LC PDF'S ARE ON | | X | | | | | X |
| | ALL LC LDF'S ARE ON | | | | | | | X |
| | LP DEPENDENTS NOT INVOLVED IN AN LR | | | | | | | X |
| | LC DELETE RULE IS 'V' | | | | X | X | | |
| THEN | SET PDF IN LP AND LDF IN LC | | X | X | X | X | X | X |
| | SET PDF IN ALL ACCESSIBLE LC | | | | X | X | | |
| | SET LDF IN LP | | | | | X | | X |
| | CALL IS REJECTED WITH A DX STATUS CODE | X | | | | | | |

NOTE: Logical parent delete rules do not apply to destination parent unless it is also an LP.

Figure 3-7. Logical Parent Deletion Rules

| | | DELETE RULE SPECIFIED IN LC | | | | | |
|---|---|---|---|---|---|---|---|
| | | P | | L | | V | |
| IF . . . | A DELETE CALL IS ISSUED FOR AN LC | X | X | X | * | X ** | |
| AND | THE LDF OF THE LC IS ON | X | | | | | |
| | RELATIONSHIP IS BIDIRECTIONAL | X | X | X | | | X |
| THEN | SET PDF IN LC | X | | X | | X | X |
| | SET LDF IN LC | | | | X | X | X |
| | CALL REJECTED WITH A DX STATUS | | X | | | | |

NOTE: Logical child delete rules establish criteria for removal of LC. V rule is required for LC in uni-directional LR.

* delete call issued for virtual logical child

** applies to delete call for either real logical child or virtual logical child.

Figure 3-8. Logical Child Deletion Rules

| IF A REPL CALL OF A CONCATENATED SEGMENT IS ISSUED... | | REPLACEMENT RULE SPECIFIED IN DP | | | | | |
|---|---|---|---|---|---|---|---|
| | | P | | L | | V | |
| AND | THE LC IS ALTERED | X | | X | | X | |
| | LP (DEST PARENT) IS ALTERED | | X | | X | | X |
| THEN | REPLACE THE LC SEGMENT | X | | X | | X | |
| | REPLACE THE LP (DP) SEGMENT | | | | * | | X |
| | CALL IS REJECTED WITH AN RX STATUS CODE | | X | | | | |

NOTE: REPLACE RULES:
- Specified for destination parent only
- Implied rule for logical child is 'V'
- Determines ability to alter DP portion of a concatenated segment
- If SEQ field of LC or DP altered, call rejected with 'DA' status code

*LP or DP is not replaced. This is ignored by DL/I.

Figure 3-9. Replacement Rules for Logical Relationships

FIRST

> states that a new occurrence is to be inserted before the first existing occurrence of this segment type. If the segment has a nonunique sequence field, a new occurrence is inserted before all existing occurrences of the same sequence field value.

LAST

> states that a new occurrence is to be inserted after the last existing occurrence of this segment type. If the segment has a nonunique sequence field, a new occurrence is inserted after all existing occurrences of the same sequence field value. This is the default option.

HERE

> assumes the user has determined positioning by a previous DL/I call, and the new occurrence is inserted before the segment that satisfied the last call. If the segment has a nonunique sequence field and the position pointer is not pointing to occurrences of this segment type with equivalent sequence field value, a new occurrence is inserted before all existing occurrences of the same sequence field value.

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

| | SEGM | NAME=virtchild |
|---|---|---|
| | | ,PARENT=seg-name2 |
| | | ,SOURCE=((seg-name3[,D,db-name2])) |
| | | ,POINTER=PAIRED |

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

**PARENT=**
> seg-name 2
>> is the name of the logical parent, that is, the physical parent of the virtual logical child.

**SOURCE=**
> ((seg-name3,D,db-name2))
>> seg-name3 is the name of the real logical child and db-name2 is the DBD name of the data base which contains that logical child.

**PTR=**
> PAIRED
>> defines this segment as a virtual logical child.

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

```
SEGM      NAME.....,RULES=PPV
```

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

|  | LCHILD | NAME=(seg-name1,db-name) |
|---|---|---|
|  |  |             {NONE}<br>,POINTER={SNGL}<br>            {DBLE} |
|  |  | ,PAIR=virtchild |
|  |  |          {FIRST}<br>,RULES={LAST }<br>         {HERE } |

**NAME=**
> (seg-name1,db-name)
>> seg-name1 is the segment name of the logical child in the DBD whose name is db-name.

**POINTER=**

> {NONE}
> {SNGL}
> {DBLE}

NONE
> specifies a unidirectional logical relationship from the logical child to the logical parent segment. No pointer fields are reserved in the prefix of the logical parent segment, however, a 4-byte counter field will be reserved in the prefix of the logical parent segment.

SNGL or DBLE

defines a bidirectional logical relationship.

SNGL specifies that there will be only a logical child first pointer in the prefix of the logical parent.

DBLE specifies that both a logical child first pointer and last pointer will appear in the logical parent.

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

- Specify DBLE if no sequence field is defined for the virtual logical child and there are generally more than three occurrences of virtual children within a logical parent.

PAIR=

virtchild

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

RULES=

```
{FIRST}
{LAST }
{HERE }
```

See the preceding discussion of this parameter for an explanation.

## Examples of Physical DBDs With Logical Relationships

Figure 3-10 shows the two logically related physical DBDs of the phase 2 sample environment. Only those DBD statements are shown which are essential to the logical relationship function. Compare these DBDs with the ones of Figures 3-4 and 3-5.

Note the addition of the virtual logical child segment, ITEM ORDER, to the Inventory data base. Also, the SUBSTITUTE ITEM segment is now defined as a logical child to eliminate the need for redundant data. In the phase 1 Inventory data base, this segment has the same fields as the INVENTORY ITEM segment.

In the Customer data base the ORDER ITEM segment is now defined as a logical child segment.

CUSTOMER
NAME/ADDRESS
(STSCCST)

INVENTORY
ITEM
(STPIITM)

CUSTOMER
LOCATION
(STSCLOC)

CREDIT
STATUS
(STSCSTA)

CUSTOMER
HISTORY
(STSCHIS)

ITEM
ORDER
(STVICOR)

(VLC)

VENDOR
(STSIVND)

SUBSTITUTE
ITEM
(STCISUB)

(LC)

ITEM
LOCATION
(STSILOC)

CUSTOMER
ORDER
(STPCORD)

ORDER
ITEM
(STCCITM)

(LC)
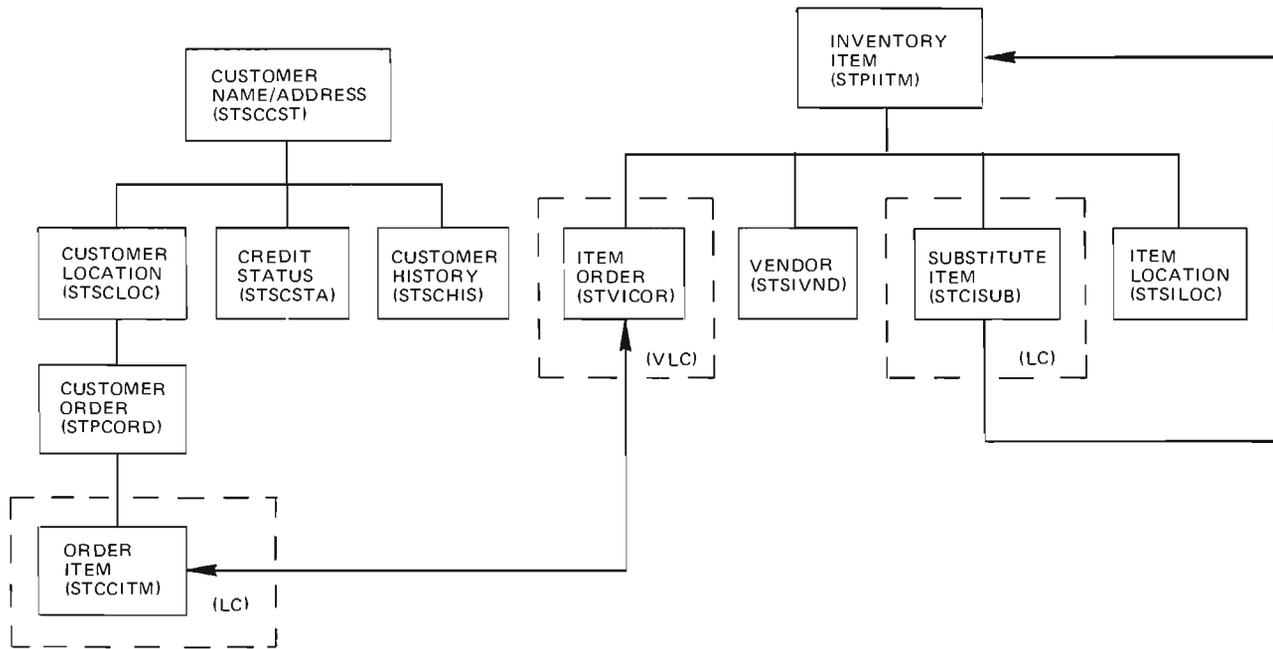
**Notes:**

1. Rules for STPIITM

   P for Insert

   When adding logical child segments to the Customer logical data base, I do not want the child added if the inventory item does not exist. If the INVENTORY ITEM segment does exist, I want the child added but I do not want the INVENTORY ITEM segment modified.

   P for Delete

   When deleting the INVENTORY ITEM segment I want to insure that no more orders are pointing to this item before allowing deletion. The only allowable path for deletion will be the physical.

   V for Replacement

   When replacing the logical child/Inventory Item concatenated segment in a logical DBD, the INVENTORY ITEM segment will be replaced if altered.

2. Rules Parameter in LCHILD macro for STCCITM

   LAST

   Implies the logical twin chain will have no required sequence. For example, when I process orders for an item, I need no specific sequence for orders.

3. POINTER=DBLE in LCHILD macro for STCCITM
   specifies that inserts last will go faster.

## Inventory Data Base

The following DBD example shows the changes made to the Inventory data base DBD of phase 1.

```
PRINT NOGEN              NO MACRO EXPANSION PRINTING
DBD                                                    X
        NAME=STDIDBP,    DATA BASE DESCRIPTION NAME    X
          •
          •
          •
DATASET                                                X
        DD1=STDIDBC,     DLBL FILE NAME                X
```

Figure 3-10. Phase 2 Physical DBDs (Part 1 of 3)

```
                        •
                        •
                        •
        SEGM                                                             X
                NAME=STPIITM,          SEGMENT NAME INVENTORY ITEM       X
                PARENT=0,              IT IS A ROOT SEGMENT              X
                BYTES=56,              DATA LENGTH                       X
                POINTER=TWIN,          PHYSICAL TWIN FWD ONLY            X
* IN THE SEGM MACRO FOR STPIITM, ADD THE RULES PARAMETER
                RULES=(PPV)            LOGICAL RELATIONSHIP RULES
* THE FOLLOWING LCHILD STATEMENT IS ADDED TO ESTABLISH A BI-DIRECTIONAL
* LOGICAL RELATIONSHIP WITH THE CUSTOMER DATA BASE VIA THE VIRTUAL
* LOGICAL CHILD SEGMENT, CUSTOMER ORDER.  THIS STATEMENT FOLLOWS THE
* SEGM STATEMENT FOR INVENTORY ITEM, THE LOGICAL PARENT SEGMENT.
        LCHILD                                                           X
                POINTER=DBLE,          BI-DIR L.R.,LCHILD FST&LST PTRS   X
                NAME=(STCCITM,         REAL LOGICAL CHILD SEGMENT NAME   X
                STDCDBP),              DATA BASE WHERE FOUND--CUSTOMER   X
                PAIR=STVICOR,          VIRTUAL LOGICAL CHILD SEG NAME    X
                RULES=LAST             REAL LOG. CHILD INSERT RULES
* THE NEXT LCHILD STATEMENT IS USED TO ESTABLISH A UNI-DIRECTIONAL
* LOGICAL RELATIONSHIP BETWEEN THE LOGICAL CHILD SEGMENT, SUBSTITUTE
* ITEM AND ITS LOGICAL PARENT, INVENTORY ITEM.
        LCHILD                                                           X
                POINTER=NONE,          UNI-DIR LOGICAL RELATIONSHIP      X
                NAME=(STCISUB,         REAL LOGICAL CHILD SEGMENT NAME   X
                STDIDBP)               D/B WHERE FOUND-ITEM-THIS ONE
        FIELD NAME=STFIIDS,BYTES=25,TYPE=C    ITEM DESCRIPTION
        FIELD NAME=STFIIQH,BYTES=6,TYPE=C     QUANTITY ON HAND
        FIELD NAME=STFIIQO,BYTES=6,TYPE=C     QUANTITY ON ORDER
        FIELD NAME=STFIIQR,BYTES=6,TYPE=C     QUANTITY ON RESERVE
        FIELD NAME=STFIIPR,BYTES=6,TYPE=C     COST PER ITEM
        FIELD NAME=STFIIUN,BYTES=1,TYPE=C     UNIT OF ISSUE
* THE NEXT SEGM STATEMENT IS ADDED TO DEFINE THE VIRTUAL LOGICAL
* CHILD, CUSTOMER ORDER.
        SEGM                                                             X
                NAME=STVICOR,          SEGMENT NAME VIRT.LCHILD ORDERS   X
                PARENT=STPIITM,        PARENT IS ITEM INFORMATION        X
                POINTER=PAIRED,        PAIRED WITH REAL LOGICAL CHILD    X
                SOURCE=((STCCITM,      REAL LOGICAL CHILD NAME           X
                D,                     REQUIRED FOR IMS/VS UPWARD COMP   X
                STDCDBP))              D/B WHERE REAL LCHILD IS FOUND
        SEGM                                                             X
                NAME=STSIVND           AUTHORIZED VENDOR INFORMATION     X
                        •
                        •
* IN THE SEGM MACRO FOR STCISUB WE MUST INDICATE THE LOGICAL PARENT.
* ADD A POINTER AND INCLUDE SOME RULES.  THIS SEGMENT IS NOW A LOGICAL
* CHILD, SO THE BYTES PARAMETER IS MODIFIED.
* THE FIELD STATEMENT FOR THIS SEGMENT AS USED IN PHASE1 IS REMOVED
* AND THE KEY FIELD FOR THE LOGICAL PARENT SEGMENT, STPIITM, IS USED
* AS DESCRIBED BELOW.
        SEGM                                                             X
                NAME=STCISUB,          SEG NAME REAL LCHILD-ITEM SUBS    X
                PARENT=((STPIITM,      PHYSICAL PARENT NAME              X
                SNGL),                 PHYS. CHILD FIRST PTR. ONLY       X
                (STPIITM,              LOGICAL PARENT SEGMENT NAME       X
                V,                     REQUIRED FOR IMS/VS UPWARD COMP   X
                STDIDBP)),             LOG.PAR.DATA BASE-ITEM-THIS ONE   X
                BYTES=6,               LENGTH OF REAL LCHILD-SEE BELOW   X
                POINTER=TWINBWD,       BOTH PHYS. TWIN FWD & BWD PTRS    X
                RULES=(PPV,            LOGICAL RELATIONSHIP RULES        X
                HERE)                  PHYSICAL INSERT RULE
```

Figure 3-10. Phase 2 Physical DBDs (Part 2 of 3)

```
* BYTES IN REAL LOGICAL CHILD'S SEGM MACRO (SEE ABOVE)
* INCLUDES THE LOGICAL PARENT'S CONCATENATED
* KEY, IN THIS CASE THE STPIITM SEGMENT'S KEY
* WHICH IS FIELD STQIINO 6 BYTES IN LENGTH;
* THE BYTES LENGTH ALSO INCLUDES ANY INTERSECTION
* DATA WHICH IN THIS CASE IS NONE.
        SEGM                                                        X
                NAME=STSILOC,         SEGMENT NAME INVENTORY LOCATION X
                •
                •
                •
        DBDGEN                        REQUIRED TO MARK DBD END
        FINISH                        FOR SOURCE COMPAT WITH IMS/VS
        END
```

## Customer Data Base

These are the changes made to the DBD of the phase 1 Customer data base.

```
* IN THE SEGM MACRO FOR STPCORD WE MUST ADD THE RULES PARAMETER
                RULES=(PPV)             LOGICAL RELATIONSHIP RULES
* IN THE SEGM MACRO FOR STCCITM WE MUST INDICATE THE LOGICAL PARENT
* ADD A POINTER AND INCLUDE SOME RULES. MODIFY THE EXISTING PARAMETERS
* AS FOLLOWS AND ADD THE RULES
                PARENT=((STPCORD),      PHYSICAL PARENT IS CUSTOMER ORD X
                (STPIITM,               LOGICAL PARENT IS ITEM INFORMAT X
                V,                      REQUIRED FOR IMS/VS UPWARD COMP X
                STDIDBP)),              LOG.PARENT IS IN INV. DATA BASE X
                POINTER=(TWINBWD,       BOTH PHYS. TWIN FWD AND BWD     X
                LTWINBWD),              BOTH LOGICAL TWIN FWD AND BWD   X
                RULES=(PPV)             LOGICAL RELATIONSHIP RULES
```

**Notes:**

1. STCCITM is now a logical child. A few points regarding its layout need to be made:

   BYTES=38 still applies

   The first 6 bytes are the concatenated key of the logical parent, STPIITM, in the Inventory data base. The remaining 32 bytes are the intersection data; data belonging to the order item to inventory item specific relationship. The key to this segment, STQCILI, is the first two bytes of this intersection data.

   The concatenated key mentioned above is not stored on disk but will be in the application I/O area as the first 6 bytes in this case.

   If this logical relationship had not been planned for earlier, the bytes parameter might have had to be changed and the segment laid out differently.

2. RULES for STCCITM - The Real Logical Child

   P for Insert
   Do not add child unless logical parent exists. Item must exist in Inventory data base if this line item is to be added to this order. The INVENTORY ITEM segment itself remains unchanged

   P for Delete
   Do not physically delete this line item unless its association with the Inventory data base is logically deleted and then only allow deletion through the physical path.

   V for Replacement
   When replacing the logical child/Inventory Item concatenated segment in a logical DBD, the INVENTORY ITEM segment will be replaced if altered.

3. Rules for STPCORD

   P for Insert
   When adding virtual logical segments to the Inventory logical data base, I do not want the child added if the order does not exist. If the CUSTOMER ORDER segment does exist, I want the child added but I do not want the CUSTOMER ORDER segment modified.

   P for Delete
   When deleting the CUSTOMER ORDER segment I want to ensure that no more items are pointing to this order before allowing deletion. The only allowable path for deletion will be physical.

   V for Replace
   When replacing the virtual logical child/CUSTOMER ORDER concatenated segment in a logical data base, the CUSTOMER ORDER segment will be replaced if altered.

Figure 3-10. Phase 2 Physical DBDs (Part 3 of 3)

## Coding a Logical DBD

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

**DBD Statement:**

| | DBD | NAME=dbdname1,ACCESS=LOGICAL |
|---|---|---|

NAME=
  dbdname1
    specifies the name of this logical DBD. It must be unique in your installation.

ACCESS=
  LOGICAL
    defines this DBD as a logical DBD

**DATASET statement**

| | DATASET | LOGICAL |
|---|---|---|

This statement is optional for logical DBDs.

**SEGM Statements:**

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

| | SEGM | NAME=seg-name1<br><br>        {0     }<br>[,PARENT={seg-name2}]<br><br>,SOURCE=((seg-name3,D,db-name1)<br>  {,(seg-name4,D,db-name2)}) |
|---|---|---|

NAME=
  seg-name1
    specifies the name of this segment.

PARENT=
  seg-name2
    specifies the name of the parent of this segment. seg-name2 must be defined previously in this DBD. This parameter may be omitted for the root segment.

SOURCE=
  ((seg-name3,D,db-name1)[,(seg-name4,D,db-name2)])
  specifies the source(s) of the defined segment. The long form is applicable only to concatenated segments.

  Nonconcatenated segments:
    seg-name3 defines the source segment. The source segment must be defined in a physical DBD whose name is db-name1.

  Concatenated segments:
  * seg-name3 defines the logical child as defined in the physical DBD. If the preceding parent segment is the physical parent or physical child of the logical child, then the name of the logical child must be coded. If the preceding parent is the logical parent, then the name of the virtual child must be coded.

  * db-name1 defines the physical DBD in which seg-name3 is defined.

  * seg-name4 defines the destination parent.

- db-name2 defines the physical DBD name of the destination parent.

## DBDGEN, FINISH, and END Statements

These should be coded as before.

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

## Example of Logical DBDs

Figure 3-11 shows the logical DBD for the phase 2 Customer data base.
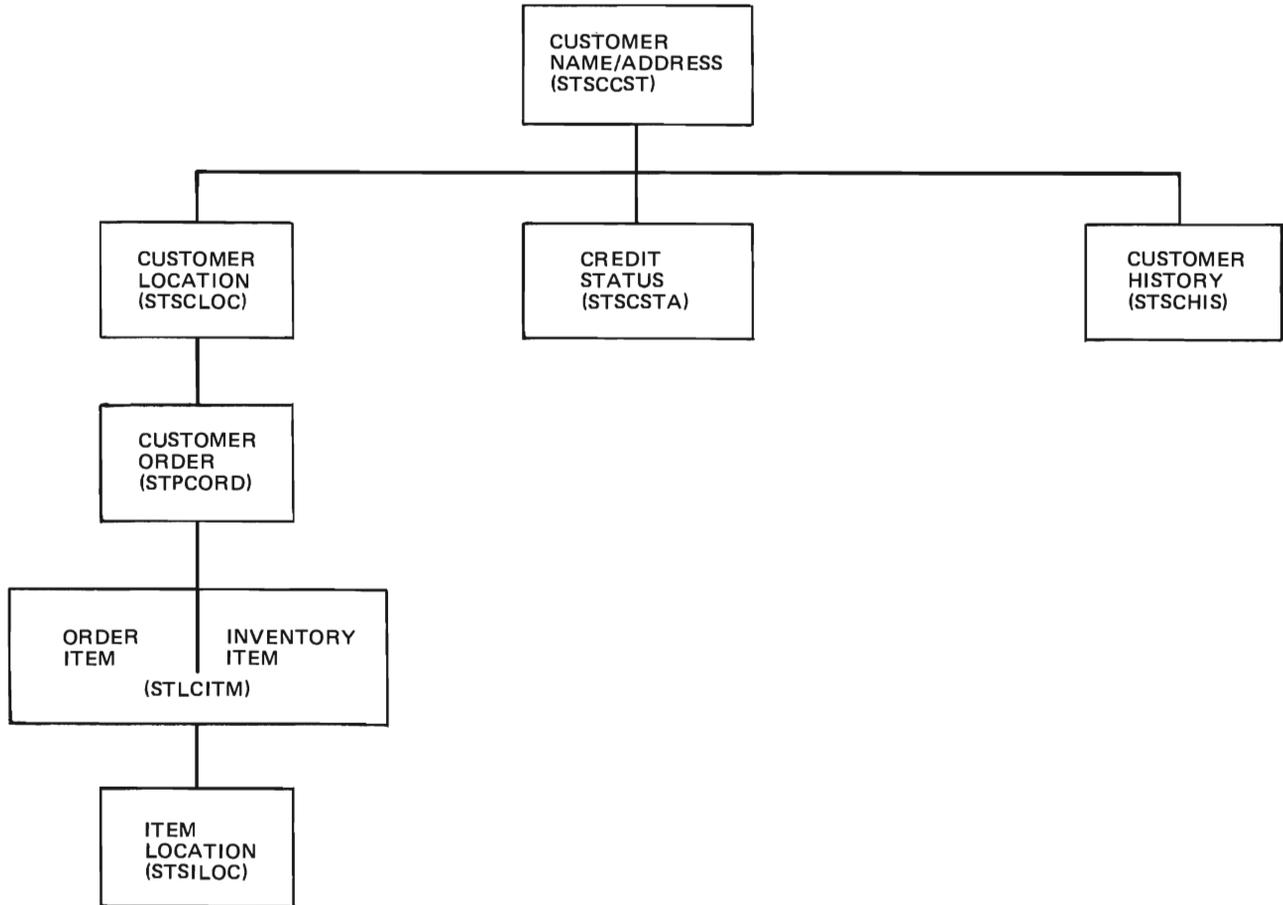


Figure 3-11. Phase 2 Logical DBD for the Customer Data Base (Part 1 of 2)

```
     PRINT NOGEN                    NO MACRO EXPANSION PRINTING
     DBD                                                              X
          NAME=STDCDBL,             LOGICAL DBD NAME                  X
          ACCESS=LOGICAL            REQUIRED
     DATASET                                                          X
          LOGICAL                   OPTIONAL
     SEGM                                                             X
          NAME=STSCCST,             SEGMENT NAME CUST NAME/ADDR       X
          PARENT=0,                 IT IS A ROOT SEGMENT              X
          SOURCE=((STSCCST,         IT IS THIS SEGMENT CUST N/A       X
          ,                         UPWARD COMPAT WITH IMS/VS         X
          STDCDBP))                 FOUND IN THE CUSTOMER DATA BASE
     SEGM                                                             X
          NAME=STSCLOC,             SEGMENT NAME CUSTOMER LOCATION    X
          PARENT=STSCCST,           PARENT IS CUST. NAME/ADDR SEGM    X
          SOURCE=((STSCLOC,         IT IS THIS SEGMENT CUST LOCATN    X
          ,                         UPWARD COMPAT WITH IMS/VS         X
          STDCDBP))                 FOUND IN THE CUSTOMER DATA BASE
     SEGM                                                             X
          NAME=STPCORD,             SEGMENT NAME CUSTOMER ORDER       X
          PARENT=STSCLOC,           PARENT IS CUST. LOCATION SEGM     X
          SOURCE=((STPCORD,         IT IS THIS SEGMENT CUST. ORDER    X
          ,                         UPWARD COMPAT WITH IMS/VS         X
          STDCDBP))                 FOUND IN THE CUSTOMER DATA BASE
     SEGM                                                             X
          NAME=STLCITM,             SEGMENT NAME LINE ITEM CONCAT.    X
          PARENT=STPCORD,           PARENT IS CUSTOMER ORDER SEGM     X
          SOURCE=((STCCITM,         PARTIALLY THE ORDER ITEM SEGM     X
          ,                         UPWARD COMPAT WITH IMS/VS         X
          STDCDBP),                 FOUND IN CUSTOMER DATA BASE       X
          (STPIITM,                 THE REST IS INVENTORY ITEM SEGM   X
          ,                         UPWARD COMPAT WITH IMS/VS         X
          STDIDBP))                 FOUND IN INVENTORY DATA BASE
     SEGM                                                             X
          NAME=STSILOC,             SEGMENT NAME INVENTORY LOCATION   X
          PARENT=STLCITM,           PARENT IS ORD. ITEM CONCAT.SEGM   X
          SOURCE=((STSILOC,         IT IS THIS SEG INVENTORY LOCN     X
          ,                         UPWARD COMPAT WITH IMS/VS         X
          STDIDBP))                 FOUND IN INVENTORY DATA BASE
     SEGM                                                             X
          NAME=STSCSTA,             SEGMENT NAME CREDIT STATUS        X
          PARENT=STSCCST,           PARENT IS CUST. NAME/ADDR SEGM    X
          SOURCE=((STSCSTA,         IT IS THIS SEGMENT CREDIT STAT    X
          ,                         UPWARD COMPAT WITH IMS/VS         X
          STDCDBP))                 FOUND IN THE CUSTOMER DATA BASE
     SEGM                                                             X
          NAME=STSCHIS,             SEGMENT NAME CUSTOMER HISTORY     X
          PARENT=STSCCST,           PARENT IS CUST. NAME/ADDR SEGM    X
          SOURCE=((STSCHIS,         IT IS THIS SEGM CUST. HISTORY     X
          ,                         UPWARD COMPAT WITH IMS/VS         X
          STDCDBP))                 FOUND IN THE CUSTOMER DATA BASE
          DBDGEN                    REQUIRED TO MARK DBD END
          FINISH                    FOR SOURCE COMPAT WITH IMS/VS
          END
```

Figure 3-11. Phase 2 Logical DBD for the Customer Data Base (Part 2 of 2)

Figure 3-12 shows the logical DBD for the phase 2 Inventory data base.
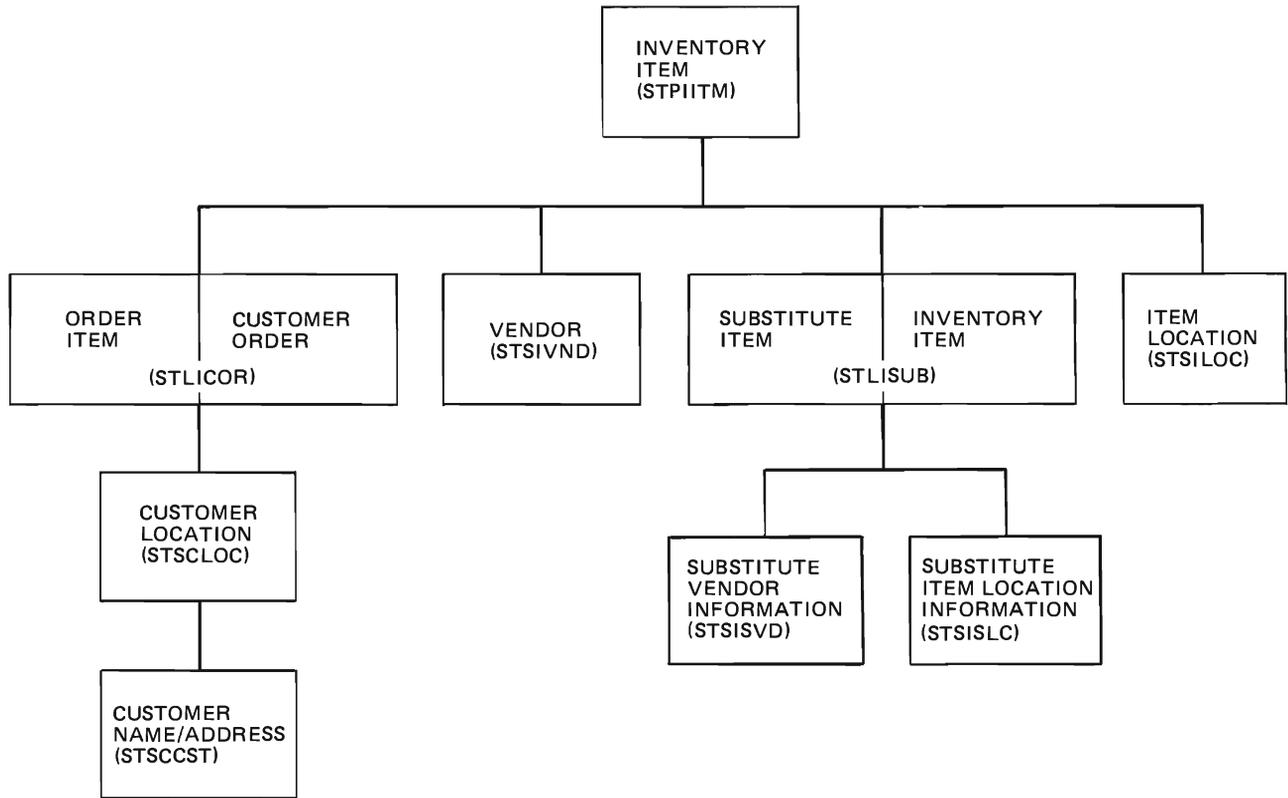


Figure 3-12. Phase 2 Logical DBD for the Inventory Data Base (Part 1 of 2)

```
PRINT NOGEN                     NO MACRO EXPANSION PRINTING
DBD                                                                 X
        NAME=STDIDBL,           LOGICAL DBD NAME                    X
        ACCESS=LOGICAL          REQUIRED
DATASET                                                             X
        LOGICAL                 OPTIONAL
SEGM                                                                X
        NAME=STPIITM,           SEGMENT NAME INVENTROY ITEM         X
        PARENT=0,               IT IS A ROOT SEGMENT                X
        SOURCE=((STPIITM,       IT IS THIS SEGMENT INV. ITEM        X
                                UPWARD COMPAT WITH IMS/VS           X
        ,
        STDIDBP))               FOUND IN INVENTORY DATA BASE
SEGM                                                                X
        NAME=STLICOR,           SEGMENT NAME ORDER CONCATENATED X
        PARENT=STPIITM,         PARENT IS INVENTORY ITEM SEGM       X
        SOURCE=((STVICOR,       PARTIALLY THE VIRT.LOG.SEGM         X
                                UPWARD COMPAT WITH IMS/VS           X
        ,
        STDIDBP),               FOUND IN INVENTORY DATA BASE        X
        (STPCORD,               THE REST THE ORDER SEGMENT          X
                                UPWARD COMPAT WITH IMS/VS           X
        ,
        STDCDBP))               FOUND IN THE CUSTOMER DATA BASE
SEGM                                                                X
        NAME=STSCLOC,           SEGMENT NAME CUSTOMER LOCATION      X
        PARENT=STLICOR,         PARENT IS ORDER SEGM CONCAT         X
        SOURCE=((STSCLOC,       IT IS THIS SEGM CUST. LOCATION      X
                                UPWARD COMPAT WITH IMS/VS           X
        ,
        STDCDBP))               FOUND IN THE CUSTOMER DATA BASE
SEGM                                                                X
        NAME=STSCCST,           SEGMENT NAME CUST. NAME/ADDR        X
        PARENT=STSCLOC,         PARENT IS CUSTOMER LOCATION         X
        SOURCE=((STSCCST,       IT IS THE CUSTOMER NAME/ADDR        X
                                UPWARD COMPAT WITH IMS/VS           X
        ,
        STDCDBP))               FOUND IN THE CUSTOMER DATA BASE
SEGM                                                                X
        NAME=STSIVND,           AUTHORIZED VENDOR INFORMATION       X
        PARENT=STPIITM,         PARENT IS INVENTORY ITEM SEGM.      X
        SOURCE=((STSIVND,       IT IS THE VENDOR SEGMENT            X
                                UPWARD COMPAT WITH IMS/VS           X
        ,
        STDIDBP))               FOUND IN INVENTORY DATA BASE
SEGM                                                                X
        NAME=STLISUB,           SEGMENT NAME ITEM SUBS. CONCA       X
        PARENT=STPIITM,         PARENT IS INVENTORY ITEM SEGM       X
        SOURCE=((STCISUB,       PARTIALLY THE ITEM SUB SEGM         X
                                UPWARD COMPAT WITH IMS/VS           X
        ,
        STDIDBP),               FOUND IN INVENTORY DATA BASE        X
        (STPIITM,               THE REST IS INVENTORY ITEM SEGM X
                                UPWARD COMPAT WITH IMS/VS           X
        ,
        STDIDBP))               FOUND IN INVENTORY DATA BASE
SEGM                                                                X
        NAME=STSISVD,           SUBSTITUTE VENDOR INFORMATION       X
        PARENT=STLISUB,         PARENT IS SUBSTITUTE ITEM SEGM.  X
        SOURCE=((STSIVND,       IT IS THE VENDOR SEGMENT            X
                                UPWARD COMPAT WITH IMS/VS           X
        ,
        STDIDBP))               FOUND IN INVENTORY DATA BASE
SEGM                                                                X
        NAME=STSISLC,           SUBSTITUTE INVENTORY LOCATION       X
        PARENT=STLISUB,         PARENT IS SUBSTITUTE ITEM SEGM.  X
        SOURCE=((STSILOC,       IT IS THE INVENTORY LOCAT. SEGM X
                                UPWARD COMPAT WITH IMS/VS           X
        ,
        STDIDBP))               FOUND IN INVENTORY DATA BASE
SEGM                                                                X
        NAME=STSILOC,           SEGMENT NAME INVENTORY LOCATION X
        PARENT=STPIITM,         PARENT IS INVENTORY ITEM SEGM       X
        SOURCE=((STSILOC,       IT IS THE INVENTORY LOCAT. SEGM X
                                UPWARD COMPAT WITH IMS/VS           X
        ,
        STDIDBP))               FOUND IN INVENTORY DATA BASE
DBDGEN                          REQUIRED TO MARK DBD END
FINISH                          FOR SOURCE COMPAT WITH IMS/VS
END
```

Figure 3-12. Phase 2 Logical DBD for the Inventory Data Base (Part 2 of 2)

# DBDGENS for Secondary Indexes

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

## *Coding an Index Target Data Base*

The control statements extended for the secondary index function are:

    FIELD
    LCHILD

A new control statement is added:

    XDFLD

The following control statements are unchanged:

    DBD
    DATASET
    SEGM
    DBDGEN
    FINISH
    END

## *Coding the Index Target Segment*

(See Figure 3-13.)

### SEGM Statement

SEGM
> is a standard SEGM statement for the root segment. It has no additional parameter for secondary indexes. It is recognized as an index target segment because of the following LCHILD and XDFLD statements. It cannot be a logical child or a dependent of a logical child segment.

### LCHILD Statement

| | LCHILD | NAME=(seg-name1,db-name),POINTER=INDX |
|---|---|---|

LCHILD
> This statement provides the link to the index data base.

NAME=(seg-name1,db-name)
> seg-name1 is the name of the index pointer segment as defined in the INDEX data base.
>
> db-name specifies the name of the HDAM or HIDAM data base that contains the index pointer segment.

PTR=INDX
> identifies the LCHILD statement as an index type.
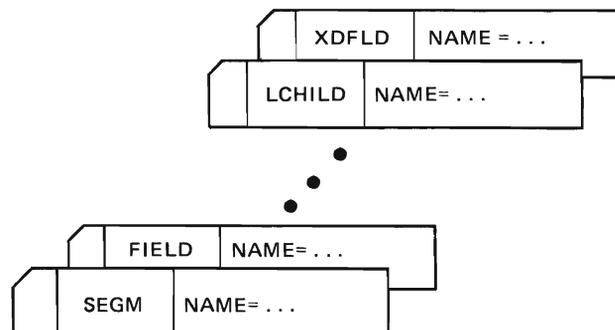


Figure 3-13. DBD Statements for Index Target Segment

Note: There are three types of LCHILD statements. One for the primary index of a HIDAM data base. One for the definition of a logical child under its logical parent, and one for the definition of the index target segment. All three types could occur below the root segment of a HIDAM data base. There could be multiple occurrences of LCHILD statements for both logical relationships and secondary indexes.

## XDFLD Statement

| | XDFLD | NAME=xdfld-name<br>[ ,SEGMENT=iss-name ]<br>,SRCH=list1<br>[ ,SUBSEQ=list2 ]<br>[DDATA=list3] |
|---|---|---|
| | | The following keywords can be used to suppress creation of an index entry.  A secondary index does not necessarily have to contain an entry for every index source segment occurrence.  These keywords are not used in the sample program documented in this manual and will not be discussed further.  See ''Suppress Creation of an Index Entry'' in the Utilities and Guide for the System Programmer if you need this function. |
| | | [ ,NULLVAL=value1 ]<br>[ ,EXTRTN=name1 ] |

XDFLD
>  This statement defines the index source fields; the fields used for the secondary index access. It defines the source data for the index search field in the INDEX data base.

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

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

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

SUBSEQ=list2 | DDATA=list3

> Either keyword must be coded if duplicate index pointer segments would occur. SUBSEQ and DDATA specify which, if any, fields of the index source segment are to be regarded as duplicated data in the index data base. Duplicated data consists of system-maintained fields of data copied from the indexed data base. The parameters list2 and list3 may each be a list of up to five field names defined by appropriate FIELD statements for the index source segment.

Names of system-related fields, as defined in the FIELD statement for the index source segment are allowed.

If two or more names are specified in one list, they must be separated by commas and enclosed in parentheses.

To the application program, SUBSEQ and DDATA have no significance because only the indexed field(s) specified in the SRCH operand is referenced in the segment search arguments. The duplicated data is accessable only if the index is processed as a data base itself.

The difference between the SUBSEQ and the DDATA keywords is as follows:

SUBSEQ

> Duplicated data described by the SUBSEQ operand is appended in the sequence specified by list2 as a subsequence to the field(s) specified in the SRCH operand. The implicit length of all specified SRCH plus SUBSEQ fields must be equal to the length of the key sequence field specified for the index pointer segment, which cannot exceed 236 bytes.

> The purpose of the SUBSEQ operand is to allow an internal expansion of the key of an index entry. This may eliminate the possibility of duplicate key values, since neither DL/I nor VSAM supports access to an indexed data base through nonunique keys.

> When SUBSEQ specifies a system-related field as /CKn, the defined portion of the concatenated key of the index source segment is appended to the SRCH field(s). If the system-related field is specified as /SXn, the 4-byte VSAM RBA of the index source segment is appended. The concatenation of SRCH and SUBSEQ fields define the key of the index pointer segment.

DDATA

> Duplicated data described by the DDATA operand is placed into the data portion (behind the specified key sequence field) of the index pointer segment in the sequence specified by list3. /CKn system-related field names can be specified in list3, but not /SXn names. /CKname must be the same as coded in the corresponding FIELD statement of the index source segment. (See next section: "Coding the Index Source Segment".)

## Coding the Index Source Segment
(See Figure 3-14.)

**SEGM Statement:**

SEGM

> This is a standard SEGM statement with no additional parameters. It is recognized as an index source segment because it is defined in a preceding XDFLD statement under the index target segment. It must not be a logical child.
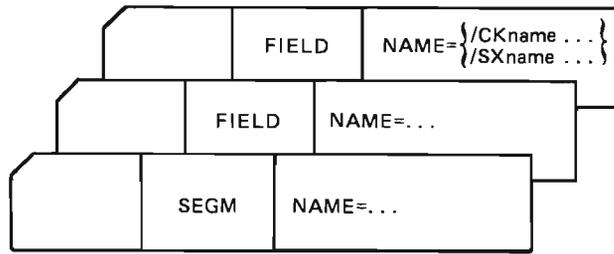
Figure 3-14. DBD Statements for Index Source Segment

## Field Statement

In addition to the normal FIELD statements for the segment the FIELD statement may be used, in the definition of an index source segment, to force uniqueness of the secondary index entries. In this case, the FIELD statement is used to define a system-related field, which can be referred to by the SUBSEQ operand of the corresponding XDFLD statement for the index target segment. This field is then appended to the key of the index entry, thus making it unique.

A system related field may be part of the source segment's concatenated key, or its VSAM RBA (relative byte address).

* For referring to the concatenated key, the format of the FIELD statement is:

|  | FIELD | NAME=/CKname |
|---|---|---|
|  |  | ,BYTES=bytes |
|  |  | ,START=pos |

NAME=/CKname
> specifies the name of the system-related field consisting of all or a portion of the concatenated key of the index source segment described by the preceding SEGM statement and name is to be replaced by 1 to 5 alphameric characters thus permitting unique identification of the field. More than one /CKname field can be specified for one index source segment. They may be noncontiguous or overlap each other.

BYTES=
> specifies the number of bytes of the concatenated key, or a portion of it, and must be a numeric value that does not exceed the length of the segment's concatenated key.

START=
> specifies the start position of this portion relative to the beginning of the concatenated key, the first byte of which is considered to have position 1. It must be a numeric term whose value does not exceed the length of the concatenated key plus 1, minus the value specified in the BYTES operand.

For example, consider the concatenated key shown in Figure 3-15.

If the uniqueness of the secondary index key has to be achieved by bytes 2-8 of the root key, byte 1 of the second key, and bytes 5-6 of the fourth key, the FIELD statements specifying this are as follows:

```
FIELD NAME = /CK1,BYTES=7,START=2
FIELD NAME = /CK2,BYTES=1,START=11
FIELD NAME = /CK3,BYTES=2,START=25
```

KEY 1        KEY 2    KEY 3        KEY 4

```
 1                          1    1             2                        3
                           1    4             1                        1
 |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|
   |      /CK1      |   |/CK2                       |/CK3 |
   L_ _ _ _ _ _ _ _J   L_J                         L_ _J
```
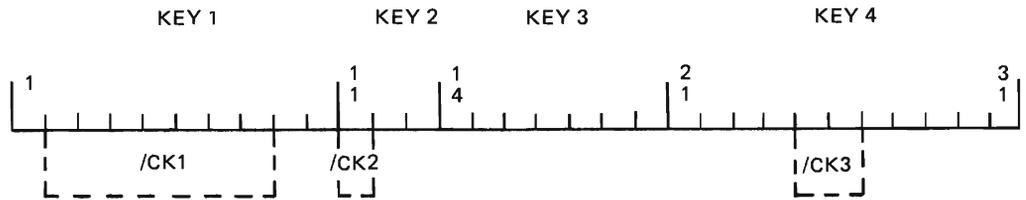
Figure 3-15. Concatenated Keys

/CK system related fields may also be used in the DDATA operand of the XDFLD statement.

- Referring to the index source segment's VSAM RBA, the FIELD statement has the following format:

| | FIELD | NAME=/SXname |
|---|---|---|

/SXname
    specifies the name of the system-related field consisting of the 4-byte VSAM RBA of the index source segment described by the preceding SEGM statement, and name is to be replaced by 1 to 5 alphameric characters. Only one /SXname field can be specified for one index source segment.

## Coding A Secondary Index DBD

The following statements are used in a secondary index DBD:

DBD
DATASET
SEGM
LCHILD
FIELD
DBDGEN
FINISH
END

### DBD Statement

| | DBD | NAME=dbname1 |
|---|---|---|
| | | ,ACCESS=INDEX |

NAME=dbname1
    specifies the name of the secondary index data base.

ACCESS=INDEX
    INDEX specifies this as an index data base.

## DATASET Statement

| | DATASET | DD1=ddname1 |
|---|---|---|
| | | ,DEVICE=device |

The values specified for the DD1 and DEVICE parameters are exactly the same as discussed under "BASIC DBDGEN".

## SEGM Statement

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

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

NAME=segname1
   specifies the name of the segment being defined. It should be unique among the segment names in your installation.

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

## LCHILD Statement

| | LCHILD | NAME=(segname1,dbname) |
|---|---|---|
| | | ,PTR=SNGL |
| | | ,INDEX=fldname |

NAME=(segname1,dbname)
   specifies the segment name of the index target segment and the name of the DBD in which it is defined.

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

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

**FIELD Statement**

| | FIELD | NAME=(fldname1,SEQ,U) |
| | | ,BYTES=bytes |
| | | ,START=1 |
| | | ,TYPE=n |

NAME=(fldname1,SEQ,U)

fldname1 is the name of this field. It should be specified following the rules of other fieldnames. SEQ,U defines this as the sequence field and must be specified.

BYTES=bytes

specifies the length of the field. This is the length of the search field as defined in the XDFLD statement, plus four if the /SX field is included. It also is the length of the key for the KSDS.

START=1

specifies the starting position of the field being defined in terms of bytes relative to the beginning of the segment.

TYPE=n

specifies alphameric data as explained in the description of the FIELD statement for the basic DBDGEN.

The DBDGEN, FINISH, and END statements should be coded as before. Figure 3-16 shows the physical DBDs and the associated secondary index DBDs for the phase 3 sample environment.

## Physical DBD - Inventory Data Base - Phase 3
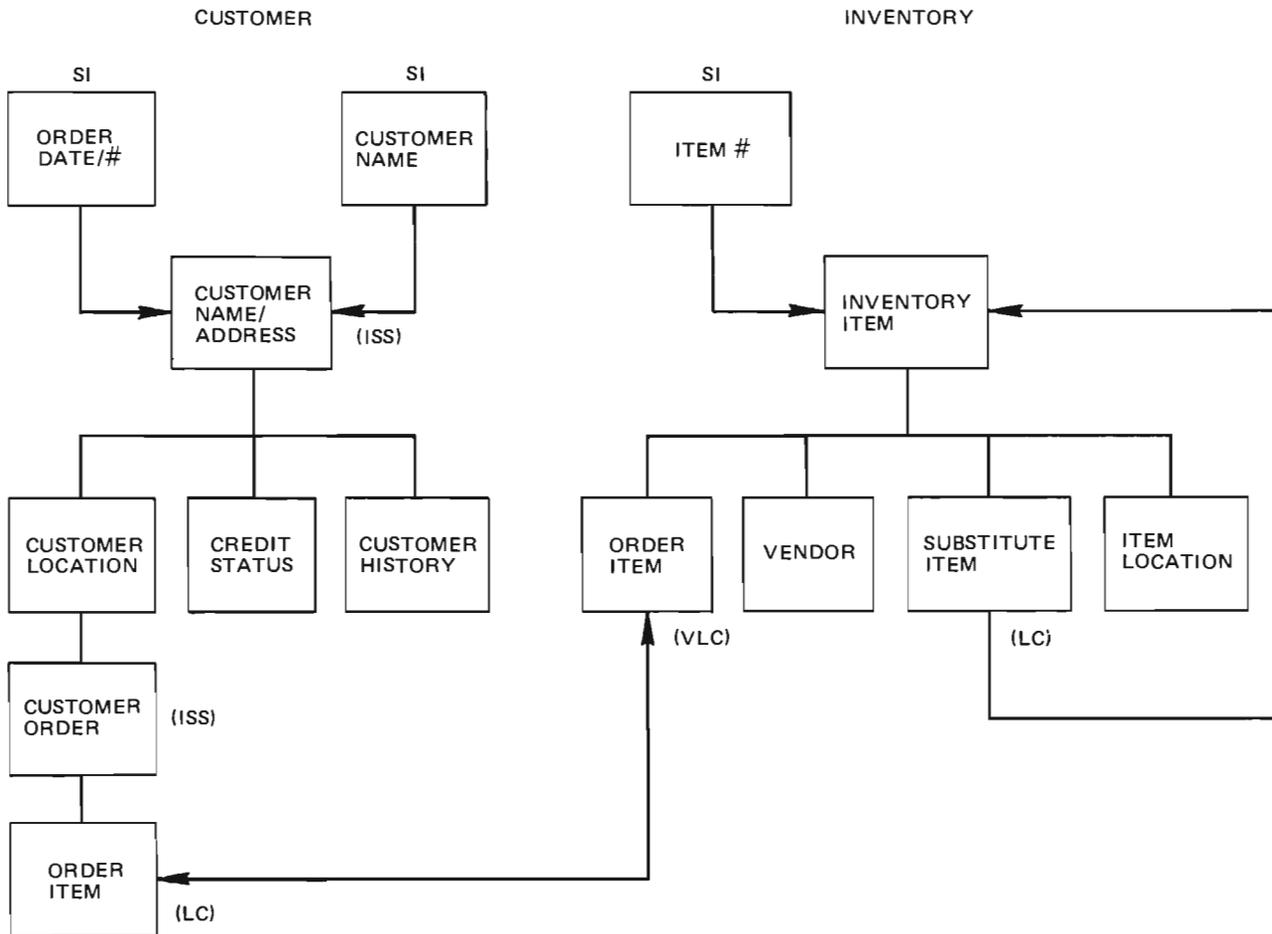


```
* IN THIS SECTION WE WILL REPEAT THE PHASE 2
* DBD STATEMENTS AS WELL AS INCLUDE THE REQUIRED
* SECONDARY INDEX ENTRIES (ITEM NUMBER WILL BE INDEX)
* NO COMMENTS WILL APPEAR EXCEPT FOR THE NEW ENTRIES
        PRINT NOGEN
        DBD   NAME=STDIDBP,ACCESS=HDAM,RMNAME=(DLZHDC30,3,100,400)
        DATASET DD1=STDIDBC,DEVICE=3340,BLOCK=(2048),SCAN=2
        SEGM  NAME=STPIITM,PARENT=0,BYTES=56,POINTER=TWIN,RULES=(PPV)
        FIELD NAME=(STQIINO,SEQ,U),BYTES=6,START=1,TYPE=C
* THE FOLLOWING LCHILD AND XDFLD MACROS ARE FOR THE SECONDARY INDEX
        LCHILD                                                     X
                POINTER=INDX,          SECONDARY INDEX             X
                NAME=(STIININ,         SEGMENT NAME IN INDEX DBD   X
                STDIX1P)               DBD NAME OF INDEX DBD
        XDFLD                                                      X
                NAME=STXININ,          INDEXED FIELD NAME          X
                SEGMENT=STPIITM,       SOURCE SEGMENT OF INDEX     X
                SRCH=STQIINO           INDEXED DATA WITHIN SOURCE SEGM
* THERE ARE NO FURTHER CHANGES IN THIS DBD FOR THE SECONDARY INDEX
        LCHILD POINTER=DBLE,NAME=(STCCITM,STDCDBP),PAIR=STVICOR,   X
                RULES=LAST
        LCHILD POINTER=NONE,NAME=(STCISUB,STDIDBP)
        FIELD NAME=STFIIDS,BYTES=25,TYPE=C    ITEM DESCRIPTION
        FIELD NAME=STFIIQH,BYTES=6,TYPE=C     QUANTITY ON HAND
        FIELD NAME=STFIIQO,BYTES=6,TYPE=C     QUANTITY ON ORDER
        FIELD NAME=STFIIQR,BYTES=6,TYPE=C     QUANTITY ON RESERVE
        FIELD NAME=STFIIPR,BYTES=6,TYPE=C     COST PER ITEM
        FIELD NAME=STFIIUN,BYTES=1,TYPE=C     UNIT OF ISSUE
```

Figure 3-16. Phase 3 Physical DBDs (Part 1 of 4)

```
        SEGM    NAME=STVICOR,PARENT=STPIITM,POINTER=PAIRED,              X
                SOURCE=((STCCITM,D,STDCDBP))
        SEGM    NAME=STSIVND,PARENT=STPIITM,BYTES=110,
                POINTER=TWIN
        SEGM    NAME=STCISUB,                                            X
                PARENT=((STPIITM,SNGL),(STPIITM,V,STDIDBP)),BYTES=6,     X
                POINTER=TWINBWD,RULES=(PPV,HERE)
        SEGM    NAME=STSILOC,PARENT=STPIITM,BYTES=12,POINTER=TWINBWD
        FIELD   NAME=(STQILNO,SEQ,U),BYTES=6,START=1,TYPE=C
        DBDGEN
        FINISH
        END
```

## Physical DBD - Customer Data Base - Phase 3

```
* IN THIS SECTION WE WILL REPEAT THE PHASE 1
* AND PHASE 2 DBD STATEMENTS AS WELL AS INCLUDE
* THE REQUIRED SECONDARY INDEX ENTRIES (DATE & ORDER# WILL BE INDEX)
* NO COMMENTS WILL APPEAR EXCEPT FOR THE NEW ENTRIES
        PRINT NOGEN
        DBD    NAME=STDCDBP,ACCESS=HDAM,RMNAME=(DLZHDC10,3,100,600)
        DATASET DD1=STDCDBD,DEVICE=3340,BLOCK=(2048),SCAN=2
        SEGM NAME=STSCCST,PARENT=0,BYTES=110,POINTER=TWIN
        FIELD NAME=(STQCCNO,SEQ,U),BYTES=6,START=1,TYPE=C
        FIELD                                                            X
                NAME=STUCCNM,             INDEX SOURCE SEG SEARCH FLD    X
                BYTES=25,                 FIELD LENGTH                   X
                START=7,                  WHERE IT STARTS IN SEGMENT     X
                TYPE=C                    ALPHAMERIC DATA
* THE FOLLOWING LCHILD AND XDFLD MACROS ARE FOR THE SECONDARY INDEXES
        LCHILD                                                           X
                POINTER=INDX,             SECONDARY INDEX                X
                NAME=(STICMNA,            SEGMENT NAME IN INDEX DBD      X
                STDCXIP)                  DBD NAME OF INDEX DBD
        XDFLD                                                            X
                NAME=STXCMNA,             INDEXED FIELD NAME             X
                SEGMENT=STSCCST,          SOURCE SEGMENT OF INDEX        X
                SRCH=STUCCNM,             INDEXED DATA WITHIN SOURCE SEGM X
                SUBSEQ=/CKSCCST           INDEX SUBSEQUENCE FIELD        X
        FIELD                                                            X
                NAME=/CKSCCST,            CONCATENATED KEY               X
                BYTES=6,                  FIELD LENGTH                   X
                START=1                   WHERE IT STARTS IN SEGMENT
        LCHILD                                                           X
                POINTER=INDX,             SECONDARY INDEX                X
                NAME=(STIRCRDN,           SEGMENT NAME IN INDEX DBD      X
                STDCX2P)                  DBD NAME OF INDEX DBD
        XDFLD                                                            X
                NAME=STXCRDN,             INDEXED FIELD NAME             X
                SEGMENT=STPCORD,          SOURCE SEGMENT OF INDEX        X
                SRCH=STQCODN,             INDEXED DATA WITHIN SOURCE SEGM X
                DDATA=/CKPCORD            FIELD NAME OF CONCATENATED KEY
        SEGM   NAME=STSCLOC,PARENT=STSCCST,BYTES=106,POINTER=TWINBWD
        FIELD NAME=(STQCLNO,SEQ,U),BYTES=6,START=1,TYPE=C
        SEGM   NAME=STPCORD,PARENT=STSCLOC,BYTES=51,POINTER=TWINBWD,     X
                RULES=(PPV)
        FIELD NAME=(STQCODN,SEQ,U),BYTES=12,START=1,TYPE=C
* THE FOLLOWING FIELD MACRO IS FOR THE DUPLICATE DATA
* FIELD THAT WILL BE CARRIED IN THE SECONDARY INDEX
* DATA BASE. IT DEFINES THE FIRST 12 BYTES OF THE ORDER
* SEGMENT FULLY CONCATENATED KEY. THE FIRST 6 BYTES ARE
* CUSTOMER NUMBER (STQCCNO) AND NEXT 6 ARE LOCATION
* NUMBER (STQCLNO). HAVING THIS DATA CARRIED IN SECONDARY
* INDEX WILL ALLOW US TO HAVE CUSTOMER AND LOCATION
* AVAILABLE TO US WHEN PROCESSING THE SECONDARY INDEX
* BY ITSELF. ALSO DL/I AUTOMATICALLY MAINTAINS THIS
* DATA WHEN PROCESSING THE CUSTOMER DATA BASE.
```

Figure 3-16. Phase 3 Physical DBDs (Part 2 of 4)

```
          FIELD                                                         X
                NAME=/CKPCORD,            MUST START WITH /CK            X
                BYTES=12,                 LENGTH DESIRED                 X
                START=1                   STARTING AT THIS POSITION
* THERE ARE NO FURTHER CHANGES IN THIS DBD FOR THE SECONDARY INDEX
          SEGM  NAME=STCCITM,PARENT=((STPCORD),(STPIITM,V,STDIDBP)),     X
                BYTES=38,POINTER=(TWINBWD),RULES=(PPV)
* THE FOLLOWING FIELDS ARE DEFINED TO SHOW AN EXAMPLE OF FIELD LEVEL
* SENSITIVITY.  NOTE THAT IT IS NOT REQUIRED FOR THE SEQUENCE FIELD
* TO BE DEFINED FIRST AND IF THE START PARAMETER IS NOT CODED THE FIELD
* IS ASSUMED CONTIGUOUS TO THE PRECEEDING FIELD.  SEE PSB'S STBCUSR
* AND STBCUSU FOR AN EXAMPLE OF HOW THE FIELDS ARE SELECTED BY THE
* APPLICATION PROGRAM.
          FIELD NAME=STKCIIN,             INVENTORY ITEM NUMBER         X
                BYTES=6,                  FIELD LENGTH                  X
                TYPE=C                    ALPHAMERIC DATA
          FIELD                                                         X
                NAME=(STQCILI,SEQ,U),     UNIQUE KEY FIELD (LINE #)     X
                BYTES=2,                  FIELD LENGTH                  X
                START=7,                  WHERE IT STARTS IN SEGMENT    X
                TYPE=C                    ALPHAMERIC DATA

          FIELD NAME=STFCIQO,BYTES=6,TYPE=C  QUANTITY ORDERED
          FIELD NAME=STFCIQS,BYTES=6,TYPE=C  QUANTITY SHIPPED
          FIELD NAME=STFCIQB,BYTES=6,TYPE=C  QUANTITY BACK ORDERED
          FIELD NAME=STFCIAM,BYTES=12,TYPE=C ITEM AMOUNT
          SEGM  NAME=STSCSTA,PARENT=STSCCST,BYTES=24,POINTER=TWIN        X
                RULES=(,FIRST)
          SEGM  NAME=STSCHIS,PARENT=STSCCST,BYTES=(130,53),             X
                POINTER=TWINBWD,COMPRTN=DLZSAMCP
          FIELD NAME=(STQCHDN,SEQ,U),BYTES=12,START=3,TYPE=C
          DBDGEN
          FINISH
          END
```

## Secondary Index DBD - Inventory Item #

```
          PRINT NOGEN                     NO MACRO EXPANSION PRINTING
          DBD                                                           X
                NAME=STDIX1P,             DATA BASE DESCRIPTION NAME    X
                ACCESS=INDEX              THIS IS AN INDEX
          DATASET                                                       X
                DD1=STDIX1C,              DLBL FILE NAME                X
                DEVICE=3340               DISK DEVICE
          SEGM                                                          X
                NAME=STIININ,             SEGMENT NAME OF THE INDEX     X
                PARENT=0,                 IT IS A ROOT SEGMENT          X
                BYTES=6                   LENGTH
          LCHILD                                                        X
                NAME=(STPIITM,            TARGET SEGMENT ITEM INFORMATION X
                STDIDBP),                 FOUND IN THE ITEM DATA BASE   X
                INDEX=STXININ,            INDEXED FIELD NAME            X
                POINTER=SNGL              SPECIFIES THIS IS INDEX PTR SEG
          FIELD                                                         X
                NAME=(STYIINO,SEQ,U),     UNIQUE KEY FIELD(ALSO THE INDX) X
                BYTES=6,                  FIELD LENGTH                  X
                START=1,                  WHERE IN SEGMENT IT STARTS    X
                TYPE=C                    ALPHAMERIC DATA
          DBDGEN                          TO MARK END OF DBD
          FINISH                          FOR SOURCE COMPAT WITH IMS/VS
          END
```

Figure 3-16. Phase 3 Physical DBDs (Part 3 of 4)

**Secondary Index DBD - Customer Order #**

```
PRINT NOGEN                          NO MACRO EXPANSION PRINTING
DBD                                                                X
      NAME=STDCX2P,                  DATA BASE DESCRIPTION NAME     X
      ACCESS=INDEX                   THIS IS AN INDEX
DATASET                                                            X
      DD1=STDCX2C,                   DLBL FILE NAME                 X
      DEVICE=3340                    DISK DEVICE
SEGM                                                               X
      NAME=STIRCRDN,                 SEGMENT NAME OF INDEX          X
      PARENT=0,                      IT IS A ROOT SEGMENT           X
      BYTES=24                       LENGTH INCL. INDEX & DUP DATA
LCHILD                                                             X
      NAME=(STSCCST,                 TARGET SEGMENT CUST. NAME/ADDR X
      STDCDBP),                      FOUND IN THE CUSTOMER DATA BASE X
      INDEX=STXCRDN,                 INDEXED FIELD NAME             X
      POINTER=SNGL                   SPECIFIES THIS IS INDEX PTR SEG
FIELD                                                              X
      NAME=(STYCRDS,SEQ,U),          UNIQUE KEY FIELD(ALSO THE INDX) X
      BYTES=12,                      FIELD LENGTH                   X
      START=1,                       WHERE IN SEGMENT IT STARTS     X
      TYPE=C                         ALPHAMERIC DATA
DBDGEN                               TO MARK END OF DBD
FINISH                               FOR SOURCE COMPAT WITH IMS/VS
END
```

**Secondary Index DBD - Customer Name**

```
PRINT NOGEN                          NO MACRO EXPANSION PRINTING
DBD                                                                X
      NAME=STDCX1P,                  DATA BASE DESCRIPTION NAME     X
      ACCESS=INDEX                   THIS IS AN INDEX
DATASET                                                            X
      DD1=STDCX1C,                   DLBL FILE NAME                 X
      DEVICE=3340                    DISK DEVICE
SEGM                                                               X
      NAME=STICMNA,                  SEGMENT NAME OF THE INDEX      X
      PARENT=0,                      IT IS A ROOT SEGMENT           X
      BYTES=31                       LENGTH INCL. INDEX & DUP DATA
LCHILD                                                             X
      NAME=(STSCCST,                 TARGET SEGMENT CUST. NAME/ADDR X
      STDCDBP),                      FOUND IN THE CUSTOMER DATA BASE X
      INDEX=STXCMNA,                 INDEXED FIELD NAME             X
      POINTER=SNGL                   SPECIFIES THIS IS INDEX PTR SEG
FIELD                                                              X
      NAME=(STYCMNS,SEQ,U),          UNIQUE KEY FIELD               X
                                     (ALSO THE INDX)
      BYTES=31,                      FIELD LENGTH                   X
      START=1,                       WHERE IN SEGMENT IT STARTS     X
      TYPE=C                         ALPHAMERIC DATA
DBDGEN                               TO MARK END OF DBD
FINISH                               FOR SOURCE COMPAT WITH IMS/VS
END
```

Figure 3-16. Phase 3 Physical DBDs (Part 4 of 4)

## Program Specification Block Generation (PSBGEN)

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

PSB generation is the execution of DL/I supplied macro instructions to define an application program's use of one or more data bases. The DL/I user creates control statements that are presented to the PSB generation procedure as a normal DOS/VS assembly job. The DL/I macro instructions used for PSB generation exist in a DOS/VS source statement library. The result of the PSB generation is the creation of

a PSB CSECT. The generated PSB is link-edited into a DOS/VS core image library (see Figure 3-17). The PSB is used as input to the application control blocks creation and maintenance utility to build other DL/I blocks for use at execution time.

Figure 3-18 shows the sequence of the macro statements in the PSBGEN input deck. The coding conventions for the PSB are exactly the same as for the DBD.
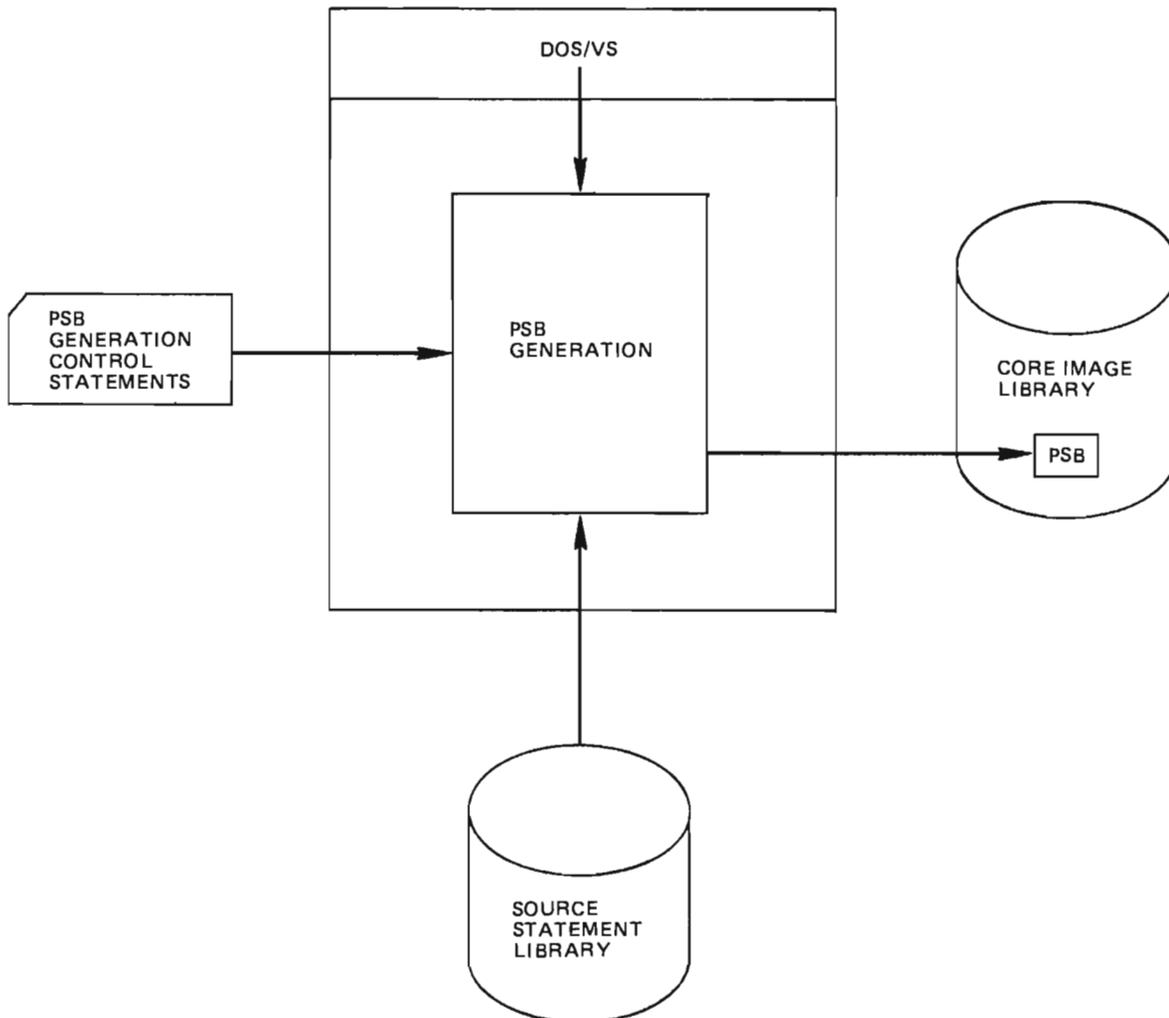


Figure 3-17. Program Specification Block Generation (PSBGEN)
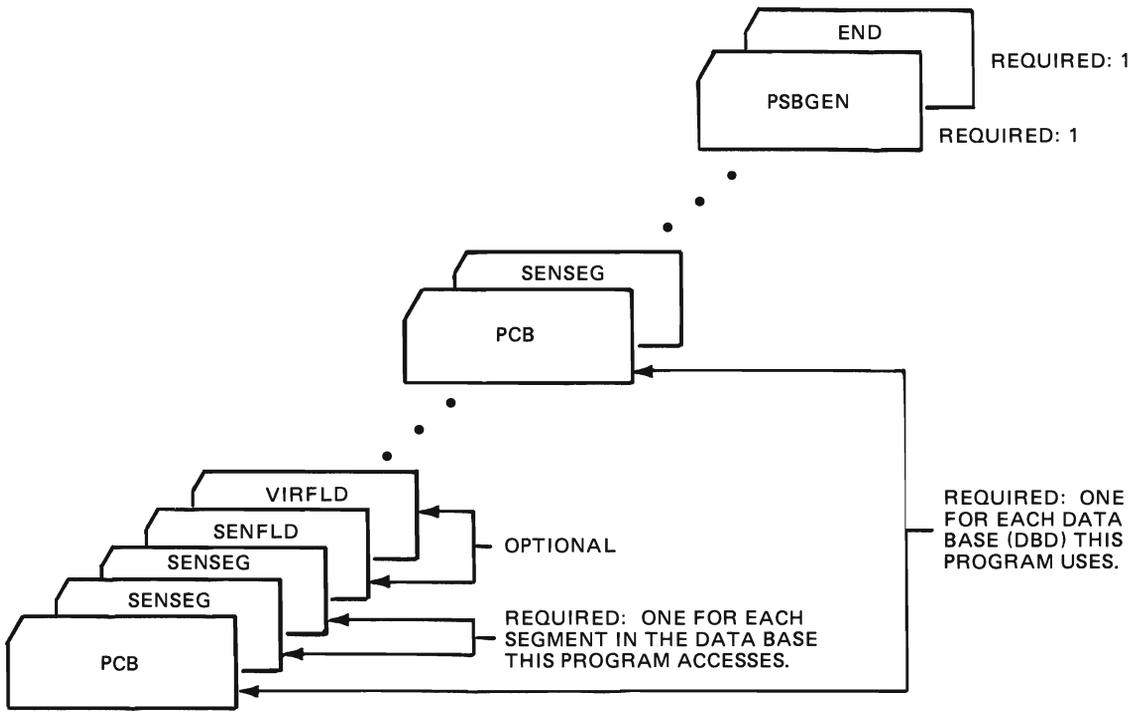
Figure 3-18. PSBGEN Input Deck Structure

## Basic PSB Coding

Following are the basic PSB control statement formats.

### PCB Statement

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

| | PCB | TYPE=DB |
|---|---|---|
| | | ,DBDNAME=ddname |
| | | ,PROCOPT={{[G][I][R][D]}           }<br>         {{A               }[P][E]}<br>         { L[S]                    }<br>         {{GO}[P]} |
| | | ,KEYLEN=value |
| | | [,POS={MULTIPLE}<br>       {SINGLE  }] |
| | | [,PROCSEQ=index-db-name] |

**TYPE=DB**
is a required keyword parameter for all data base PCBs.

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

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

G - Get function

I - Insert function

R - Replace function

D - Delete function

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

A - All, includes the above four functions.

P - Required if command code D (path call) is to be used on get type calls or insert calls. Determines the maximum length of the I/O area. To be used in conjunction with G, I, and A.

E - Exclusive use of the data base or segment. To be used in conjunction with G, I, D, R, and A. Use this option only when you wish to override program isolation in an MPS or online environment.

Note: PROCOPT=E is not propagated. Specifying it on the PCB statement does not force E onto the SENSEG statements if they also specify a PROCOPT. Therefore, PROCOPT=E must be coded on any senseg statement that requires it when any other PROCOPT is also coded.

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

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

O- Inhibits locking (enqueuing) by program isolation during retrievals of the same segment types in a data base. O can be used only in conjunction with G or GP.

See chapter 5 in this manual for a description of program isolation.

Note: Consider always coding PROCOPT=G on the PCB statement and using the SENSEG statement to override this specification as required.

KEYLEN=value

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

POS=

specifies whether single or multiple positioning is desired for this logical data structure. When SINGLE or S is specified, for a PCB, DL/I maintains only one position in that data base for that PCB. This is the position that will be used in attempting to satisfy all subsequent GN calls. If MULTIPLE or M is specified, DL/I maintains a unique position in each hierarchical path in the data base. For a detailed description of positioning, see the *Application Programming Reference Manual.*

Note: Use of single or multiple positioning affects application program logic. Therefore, ensure that the PSB and program logic match.

PROCSEQ

specifies the name of a secondary index that is used to process the data base named in the DBDNAME operand through a secondary processing sequence. This operand is optional. It is valid only if a secondary index exists for this data base. If this operand is used, subsequent SENSEG statements must reflect the secondary data structure of segment types in the indexed data base. For example, the first SENSEG segment must name the index target segment as the root segment. This operand is invalid if PROCOPT is L or LS.

**SENSEG Statement**

This statement is coded once for each segment the program is sensitive to in the DBD defined in the preceding PCB. The SENSEG statements should appear in the same hierarchical sequence as in the DBD. However, only those segments should be specified in the hierarchical path to any required segment. The basic format of the SENSEG statement is:

| | SENSEG | NAME=segname1 |
| --- | --- | --- |
| | | [ ,PARENT={ segname2 } ] |
| | |            { <u>0</u>       } |
| | | [ ,PROCOPT={ [G][I][R][D] }[P][E] |
| | |             { A         } |

NAME=

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

PARENT=

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

PROCOPT=

is the processing options available for use of this sensitive segment by an associated application program. This operand has the same meaning as the PROCOPT operand on the PCB statement. If this PROCOPT operand is not specified, the PCB PROCOPT operand is used as the default.

Note: PROCOPT=P does not propagate. Therefore PROCOPT=P must be coded on any SENSEG statement that requires it. This is in addition to coding it on the PCB statement.

If there is a difference in the processing options specified on the PCB and SENSEG statement, the SENSEG PROCOPT overrides the PCB PROCOPT. When loading a data base, you should specify a PROCOPT in the PCB statement.

## SENFLD Statement

This statement follows the SENSEG statement and is coded once for each field in the physical definition of this segment to which the application program is sensitive. This enables you to restrict an application program's access to only those fields in a segment that it needs. If the SENFLD statement is not specified, DL/I assumes that the application program has access to the entire segment identified in the previous SENSEG statement. This statement is part of the field level sensitivity feature. See "Field Level Sensitivity" in Chapter 2 for details. The format of the SENFLD statement is:

| | SENFLD | NAME=fldname<br>[ ,BYTES=n]<br>[ ,START=pos]<br>[ ,TYPE=t]<br>[ ,RTNAME=prog]<br>[ ,REPLACE= $\begin{Bmatrix} \underline{YES} \\ NO \end{Bmatrix}$ ] |
|---|---|---|

NAME=
> is the name of the related field defined in the physical DBD.

BYTES=
> specifies the length (in bytes) of this field. If specified, it must be a numeric value in the range of 1 through 256.

> *Rules and Restrictions*

> • Do not specify the BYTES parameter for field data types E, D, and L. These data types have implicit lengths of 4, 8, and 16, respectively.

> • The BYTES parameter is optional for field data types H and F. If omitted, DL/I assumes a field length of 2 for types H and 4 for type F.

> • The BYTES parameter is also optional for field data types X, P, C, and Z. If omitted, DL/I defaults to the same field length as is in the physical view.

START=
> specifies the starting position of this field. It can be the same starting position previously specified for the field in the FIELD statement during DBD generation, or it can be different.

> The starting position can be specified in terms of bytes relative to the beginning of your new view of the segment within which this field is defined. In this case, it must be a numeric value in the range of 1 through 32767. For the first byte of a segment it is one. Each field must not extend beyond the defined segment length (START position plus BYTE value).

> Subfields are permitted and can be defined on the START parameter in one of two ways. You can specify its starting position in bytes as previously described (START=position), or, if the subfield starts at the same location as another defined field you can simply specify the name of that field (START=name).

> The START parameter can be optionally omitted. If it is not specified, a DL/I feature called 'automatic definition sequencing' places this field adjacent to the end of the previous field, or if it is the first field in the segment, at the beginning of the segment (START=1).

**TYPE=**

specifies the type of data that is to be contained in the application program's view of this field. If you specify a data type that is different from that defined in the physical DBD for this field, DL/I will (in most cases) convert the data type to that needed by the program. (See "Field Level Sensitivity" in Chapter 2 for additional information.) If you do not specify this parameter, DL/I assumes the type to be the same as specified for this field in the physical DBD. The valid data types are:

'X' - hexadecimal data (binary)
'H' - half word binary
'F' - full word binary
'P' - packed decimal
'C' - character data
'Z' - numeric character data
'E' - floating point (short)
'X' - 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) |

Conversion of data types E, D, and L is not supported.

**Notes:**

- Restrictions on values
  - Binary (X, H, F)
  Packed or zoned decimal fields to be converted to binary fields are restricted to values within the range of 2147483647 to -2147483648. This is because numbers outside this range cannot be contained in a four byte binary word.

  - Packed and zoned decimal (P, Z)
  Hardware restrictions limit the size of decimal fields to 16 characters, so the values contained in fields to be converted must be within the range of ±9999999999999999.

- Length conversions
  Numeric data types (X, H, F, P, Z) are padded with zeros on the left to extend field lengths. Truncation also occurs from the left. Truncation of significant digits results in the field being set to the maximum (or minimum, if negative) value, and status code 'KA' is returned.

  Character fields are padded with blanks on the right to extend field lengths. Truncation also occurs from the right. Truncation of non-blank characters results in the return of status code 'KB'. Only character field length conversion is performed if both the physical and user's view data types are 'C'.

- Format checking
  To ensure valid formats, packed and zoned decimal fields are pre-scanned prior to conversion. An invalid format results in the setting of the converted field to the null value, and the return of status code 'KC'.

- Data type 'C'
  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 related field is specified as containing data of a different type. For instance, 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 any necessary length adjustments are made.

- Non-supported conversions
  Conversions that are not listed above as being 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 of 'KD' to be returned when encountered during an access of the field.

- Conversion Errors
  If not corrected (reset) by a user exit routine, an uncorrected status code results in an immediate termination of the request. No more fields or segments are processed. No provision is made for correction of errors by the application program. If required, conversion must be done via a user written field exit routine. See "User Field Exit Routine" under "Field Label Sensitivity" in Chapter 2.

**RTNAME=**
identifies the name of the user-written field exit routine in the DOS/VS core image library that is given control whenever this field is retrieved or stored. See "Field Level Sensitivity" in Chapter 2 for a description of this routine.

**REPLACE (or its abbreviation, REPL)**
indicates whether the program using this PSB may modify this field. If you specify NO, and an application program attempts to replace this field with a new value, DL/I returns a status code of 'KE'.

## VIRFLD Statement

This statement is used to define a field in the application program's view of a segment that does not exist in the physical segment. This statement also allows you to specify an initial value for the virtual field and/or the name of a user-written routine that is called to create the field as needed. See "Field Level Sensitivity" in Chapter 2 for a complete description of virtual fields. The format of the VIRFLD statement is:

| | VIRFLD | NAME=fldname<br>[,BYTES=n]<br>[,START=pos]<br>[,TYPE=t]<br>[,VALUE=value]<br>[,RTNAME=prog] |
| --- | --- | --- |

**NAME=**
specifies the name of the field.

**BYTES=**
specifies the length of this field in terms of bytes. BYTES is specified as a numeric whose value does not exceed 256. You must specify this parameter for field types X, C, Z, or P. Do not specify this parameter for field types E, D, or L. See the 'TYPE' parameter for field types.

**START=**
specifies the starting position of this field in terms of bytes relative to the beginning of the application program's view of the segment for which this field is defined. Start position for the first byte of the segment is 1; the maximum specification is 32767.

Subfields are permitted. If a subfield starts in the same position as another defined field, you may specify the name of that field, instead of a numeric value, to indicate the starting position.

If you do not specify this parameter, DL/I places this field adjacent to the end of the previous field, or if it is the first field in the segment, at the beginning of the segment (START=1).

**TYPE=**
specifies the type of data that is to be contained in the application program's view of this field. This parameter must be specified if the VALUE parameter is used. The valid data types are:

'X' - hexadecimal data (binary)
'H' - half word binary
'F' - full word binary
'P' - packed decimal
'C' - character data
'Z' - numeric character data
'E' - floating point (short)
'D' - floating point (long)
'L' - floating point (extended)

**VALUE=**

> specifies an initial value for this virtual field. If the RTNAME parameter is also used, this field is initialized before the user-written field exit routine is called.
>
> **Notes:**
>
> - The TYPE parameter must be specified if the VALUE parameter is specified.
> - If the VALUE parameter is specified for field type H, F, P, or Z, the initial value must be numeric.
> - If the number of characters supplied for VALUE is not sufficient to make up the length specified by the BYTES parameter, the initial value will be padded:
>   - With binary zeros on the left for types X, H, F, and P.
>   - With EBCDIC zeros on the left for type Z.
>   - With binary zeros on the right for types E, D, and L.
>   - With EBCDIC blanks on the right for type C.

**RTNAME=**

> specifies the name of the user-written field exit routine in the DOS/VS core image library that is given control whenever this field is retrieved or stored. See "Field Level Sensitivity" in Chapter 2 for a description of this routine.

## PSBGEN Statement

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

| | PSBGEN | {COBOL}<br>LANG={PL/I }<br>{ASSEM}<br>{RPG }<br>,PSBNAME=psbname |
|---|---|---|

**LANG=**

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

**PSBNAME=**

> is the parameter keyword for the alphameric name of this PSB. The name value for PSBNAME must be seven characters or less in length. However, the (@) must not be used. See notes.

**Notes:**

- The application control blocks creation and maintenance utility uses the output of the PSB generation to build a PSB containing other internal control blocks based on the related DBD characteristics. The name of this special PSB is generated by the utility program. Since this PSB is also cataloged and link-edited into a DOS/VS core image library, care must be taken to avoid duplicate names. The generated PSB name is eight characters in length and consists of the PSB generation CSECT name extended to seven characters by at-signs (@), if necessary, with P as the eighth character.
- There may be several PCB statements for data bases, but only one PSBGEN statement as input to a PSB generation. The PSBGEN statement must immediately precede the END statement.

## END Statement

This statement is required at the end of the PSB deck. It indicates the end of the assembly data.

| | END | |
|---|---|---|

## *Sample Basic PSBs*

Figure 3-19 shows two PSBs for the Phase 1 sample environment. The first one is the PSB for loading the Phase 1 Customer data base. The second one is an example of a PSB for an application program to process the phase 1 Customer data base.

Because the basic PSBs to load and process the Inventory data base for Phase 1 are similar to the above examples, they are not included here.

### Load PSB - Customer Data Base

```
       PRINT NOGEN                    NO MACRO EXPANSION PRINTING
       PCB                                                          X
             TYPE=DB,                 REQUIRED                      X
             DBDNAME=STDCDBP,         FROM DBD MACRO IN DBD ASSEMBLY X
             PROCOPT=L,               LOAD PSB                      X
             KEYLEN=50                LONGEST CONCATENATED KEY
*                                     26 IS THE LONGEST IN CUSTOMER
*                                     DATA BASE. 50 LEAVES EXPANSION
*                                     ROOM FOR FUTURE
       SENSEG                                                       X
             NAME=STSCCST,            USING THE SAME NAMES AS FOUND  X
             PARENT=0                 IN THE SEGM MACROS IN THE DBD
       SENSEG                                                       X
             NAME=STSCLOC,            ASSEMBLY FOR THE CUSTOMER DATA X
             PARENT=STSCCST           BASE AND PUTTING THE SENSEG
       SENSEG                                                       X
             NAME=STPCORD,            MACROS IN THE SAME ORDER AS    X
             PARENT=STSCLOC           THOSE SEGM MACROS IS REQUIRED
       SENSEG                                                       X
             NAME=STCCITM,                                          X
             PARENT=STPCORD
       SENSEG                                                       X
             NAME=STSCSTA,                                          X
             PARENT=STSCCST
       SENSEG                                                       X
             NAME=STSCHIS,                                          X
             PARENT=STSCCST
       PSBGEN                                                       X
             LANG=ASSEM,              OR PL/I OR COBOL               X
             PSBNAME=STBICLD          PROGRAM SPECIFICATION BLK NAME
       END
```

### PSB to Process Customer Data Base

```
       PRINT NOGEN                    NO MACRO EXPANSION PRINTING
       PCB                                                          X
             TYPE=DB,                 REQUIRED                       X
             DBDNAME=STDCDBP,         FROM DBD MACRO IN DBD ASSEMBLY X
             PROCOPT=AP,              A=ALL FUNCS.,P=PATH CALL POSSIB X
             KEYLEN=50                SEE LOAD PSB FOR EXPLANATION
*
* SENSEG MACROS WILL BE SAME AS LOAD PSB
*
       PSBGEN                                                       X
             LANG=ASSEM,              OR PL/I OR COBOL               X
             PSBNAME=STBCPHA          PROGRAM SPECIFICATION BLK NAME
       END
```

Figure 3-19. Sample PSBs for Phase 1

## Execution of PSBGEN - JCL

PSBGEN is run as a standard DOS/VS job and requires the following DOS/VS job control statements:

```
// JOB    PSBGEN
// OPTION CATAL
// EXEC ASSEMBLY
      PCB
      SENSEG      PSB GENERATION CONTROL STATEMENTS
      PSBGEN      FOR ONE PSB
      END
/*
// EXEC    LNKEDT
/&
```

## Description of PSBGEN Output

PSBGEN produces the following:

- Control statement listing
  This is a listing of the input statement images.

- Diagnostics
  Errors discovered during the processing of each control statement result in diagnostic messages, which are printed immediately following the image of the control statement. A message may reference either the control statement immediately preceding it or the preceding group of control statements. It is also possible for more than one message to be printed for each control statement. In this case, the messages follow each other on the output listing. After all control statements have been read, a further check is made of the reasonableness of the entire group. This may result in one or more additional diagnostic messages.

  If any error is discovered, all control statements are read, checked, and listed and the diagnostic message(s) are printed, but the other outputs are suppressed, before the PSB generation is terminated.

  The PSB error condition messages are contained in the *Messages and Codes* manual.

- Assembly listing
  A DOS/VS Assembler language listing of the PSB macro expansion created by PSB generation execution.

- Object Module
  After the PSB generation macro is assembled, the PSB is link-edited and cataloged as a load module in a DOS/VS core image library.

## Coding PSBs for Logical Data Bases

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

To use a logical data base, the program needs a separate PCB. This PCB is coded in the same manner as a PCB for a physical DBD. The only difference is that it refers to the DBD name and SEGMENT names of a logical DBD. You should code SENSEG statements only for the segments the program actually needs and the segments in the hierarchical path to those segments. All of this is based on the logical DBD, so the hierarchical path may well include physical and logical paths. Figure 3-20 shows sample PSBs for the phase 2 logical data bases.

## PSB Logical Inventory Data Base - Phase 2

```
PRINT NOGEN                      NO MACRO EXPANSION PRINTING
PCB                                                              X
      TYPE=DB,                   REQUIRED                        X
      DBDNAME=STDIDBL,           LOGICAL DBD NAME                X
      PROCOPT=AP,                A=ALL FUNCS.,P=PATH CALL POSSIB X
      KEYLEN=50                  SEE LOAD ITEM PSB FOR DISCUSSION
SENSEG                                                           X
      NAME=STPIITM,              USING THE SAME NAMES AS FOUND   X
      PARENT=0                   IN SEGM MACROS IN THE LOGICAL
SENSEG                                                           X
      NAME=STLICOR,              DBD FOR THE ITEM DATA BASE AND  X
      PARENT=STPIITM             PUTTING THE SENSEG MACROS IN
SENSEG                                                           X
      NAME=STSLOC,               THE SAME SEQUENCE AS THOSE SEGM X
      PARENT=STLICOR             MACROS IS REQUIRED
SENSEG                                                           X
      NAME=STSCCST,                                              X
      PARENT=STSCLOC
SENSEG                                                           X
      NAME=STSIVND,                                              X
      PARENT=STPIITM
SENSEG                                                           X
      NAME=STLISUB,                                              X
      PARENT=STPIITM
SENSEG                                                           X
      NAME=STSILOC,                                              X
      PARENT=STPIITM
PSBGEN                                                           X
      LANG=ASSEM,                OR PL/I OR COBOL                X
      PSBNAME=STBILGA            PROGRAM SPECIFICATION BLK NAME
END
```

## PSB Logical Customer Data Base - Phase 2

```
PRINT NOGEN                      NO MACRO EXPANSION PRINTING
PCB                                                              X
      TYPE=DB,                   REQUIRED                        X
      DBDNAME=STDCDBL,           LOGICAL DBD NAME                X
      PROCOPT=AP,                A=ALL FUNCS.,P=PATH CALL POSSIB X
      KEYLEN=50                  SEE LOAD CUST PSB FOR DISCUSSIN
SENSEG                                                           X
      NAME=STSCCST,              USING THE SAME NAMES AS FOUND   X
      PARENT=0                   IN SEGM MACROS IN THE LOGICAL
SENSEG                                                           X
      NAME=STSCLOC,              DBD FOR THE CUSTOMER DATA BASE  X
      PARENT=STSCCST             AND PUTTING THE SENSEG MACROS
SENSEG                                                           X
      NAME=STPCORD,              IN THE SAME SEQUENCE AS THOSE   X
      PARENT=STSCLOC             SEGM MACROS IS REQUIRED
```

Figure 3-20. Sample PSBs for Phase 2 (Part 1 of 2)

```
        SENSEG                                                  X
              NAME=STLCITM,                                     X
              PARENT=STPCORD
        SENSEG                                                  X
              NAME=STSILOC,                                     X
              PARENT=STLCITM
        SENSEG                                                  X
              NAME=STSCSTA,                                     X
              PARENT=STSCCST
        SENSEG                                                  X
              NAME=STSCHIS,                                     X
              PARENT=STSCCST
        PSBGEN                                                  X
              LANG=ASSEM,            OR PL/I OR COBOL           X
              PSBNAME=STBCLGA        PROGRAM SPECIFICATION BLK NAME
        END
```

Figure 3-20. Sample PSBs for Phase 2 (Part 2 of 2)

## *Coding PSBs for Secondary Indexes*

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

**PCB Statement**

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

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

Note: If non-unique fields are used, and subsequence is the /SX field, then the sequence of root segments with the same index field value will be unpredictable. This sequence will also vary during reorganization of the target data base.

Figure 3-21 shows the PSBs as used in the sample application for the phase 3 environment.

## Inventory and Customer Load PSBs - Phase 3

```
// JOB STJPSBGN GENERATE ALL PSBS
// OPTION CATAL,NODECK
// EXEC ASSEMBLY
        TITLE 'DL/I ONLINE PROGRAM - INVENTORY AND CUSTOMER LOAD PSBS
*
        PRINT NOGEN                  NO MACRO EXPANSION PRINTING     X
        PCB                                                          X
                TYPE=DB,             REQUIRED                        X
                DBDNAME=STDIDBP,     FROM DBD MACRO IN DBD ASSEMBLY  X
                PROCOPT=L,           LOAD PSB                        X
                KEYLEN=50,           LONGEST CONCATENATED KEY        X
*                                    26 IS THE LONGEST IN CUSTOMER
*                                    DATA BASE. 50 LEAVES ROOM FOR
*                                    FUTURE EXPANSION
                POS=S                SINGLE POSITIONING (DEFAULT)
        SENSEG                                                       X
                NAME=STPIITM,        USING THE SAME NAMES AS FOUND   X
                PARENT=0             IN THE SEGM MACRO IN THE DBD
        SENSEG                                                       X
                NAME=STSIVND,        ASSEMBLY FOR THE CUSTOMER DATA  X
                PARENT=STPIITM       BASE AND PUTTING THE SENSEG
        SENSEG                                                       X
                NAME=STCISUB,        MACROS IN THE SAME ORDER AS     X
                PARENT=STPIITM       THOSE SEGM MACROS IS REQUIRED
        SENSEG                                                       X
                NAME=STSILOC,                                        X
                PARENT=STPIITM
*
* THE FOLLOWING PCB IS FOR THE CUSTOMER DATA BASE.
* THE STATEMENTS ARE THE SAME FORMAT AS FOR THE INVENTORY DATA BASE,
* SO THE COMMENTS HAVE BEEN OMITTED.
*
        PCB    TYPE=DB,DBDNAME=STDCDBP,PROCOPT=L,KEYLEN=50
        SENSEG NAME=STSCCST,PARENT=0
        SENSEG NAME=STSCLOC,PARENT=STSCCST
        SENSEG NAME=STPCORD,PARENT=STSCLOC
        SENSEG NAME=STCCITM,PARENT=STPCORD
        SENSEG NAME=STSCSTA,PARENT=STSCCST
        SENSEG NAME=STSCHIS,PARENT=STSCCST
        PSBGEN                                                       X
                LANG=ASSEM,          APPLICATION PROG IS ASSEMBLER   X
                PSBNAME=STBICLD      PROGRAM SPECIFICATION BLK NAME
*
* DO NOT INCLUDE ANY SENSEG MACRO FOR VIRTUAL LOGICAL
* CHILDREN IN A LOAD PSB.
*
        END
/*
// EXEC LNKEDT
```

## Inventory and Customer Data Base PSBs - Logical

```
// OPTION CATAL,NODECK
// EXEC ASSEMBLY
        TITLE 'DL/I SAMPLE PROGRAM - INVENTORY AND CUSTOMER PSBS -LOGIX
            CAL'
        PRINT NOGEN                  NO MACRO EXPANSION PRINTING
*
```

Figure 3-21. PSBs Used for the Phase 3 Sample Application (Part 1 of 3)

```
* THIS PSB IS FOR THE LOGICAL DATA BASES USED BY THE PRINT PROGRAM
* DLZSAM50.
* THE FIRST PCB IS FOR THE INVENTORY LOGICAL DATA BASE.
* NOTE THE USE OF THE PROCSEQ PARAMETER FOR THE SECONDARY INDEX.
* THIS ALLOWS THE APPLICATION TO ACCESS THE INVENTORY ITEMS IN NUMERIC
* SEQUENCE.
*
* BECAUSE THE FORMAT OF THE PSB STATEMENTS IS THE SAME AS FOR THE LOAD
* PSB, NO FURTHER COMMENTS ARE INCLUDED.
*
         PCB     TYPE=DB,DBDNAME=STDIDBL,PROCOPT=G,KEYLEN=50,POS=S,     X
                 PROCSEQ=STDIX1P
         SENSEG  NAME=STPIITM,PARENT=0
         SENSEG  NAME=STLICOR,PARENT=STPIITM
         SENSEG  NAME=STSIVND,PARENT=STPIITM
         SENSEG  NAME=STLISUB,PARENT=STPIITM
         SENSEG  NAME=STSILOC,PARENT=STPIITM
         PCB     TYPE=DB,DBDNAME=STDCDBL,PROCOPT=G,KEYLEN=50,POS=S
         SENSEG  NAME=STSCCST,PARENT=0
         SENSEG  NAME=STSCLOC,PARENT=STSCCST
         SENSEG  NAME=STPCORD,PARENT=STSCLOC
         SENSEG  NAME=STLCITM,PARENT=STPCORD
         SENSEG  NAME=STSCSTA,PARENT=STSCCST
         SENSEG  NAME=STSCHIS,PARENT=STSCCST
         PSBGEN  LANG=ASSEM,PSBNAME=STBICLG
         END
/*
// EXEC LNKEDT,SIZE=100K
```

## Online Order Inquiry Application PSB - Read Only

```
// EXEC ASSEMBLY,SIZE=300K
         TITLE 'DL/I ONLINE SAMPLE PROGRAM - ORDER INQUIRY AND ENTRY PSX
               BS - READ ONLY'
         PRINT  NOGEN
*
* THIS PSB IS USED BY DLZSAM60 TO RETRIEVE SEGMENTS FROM THE CUSTOMER
* AND INVENTORY DATA BASES UNDER CICS/VS.
* THIS PSB CONTAINS PCB'S FOR TWO SECONDARY INDEX DATA BASES.
* (DBDNAME=STDCX1P AND DBDNAME=STDCX2P)
* THIS ALLOWS THE ONLINE PROGRAM TO USE THESE SECONDARY INDEXES
* AS DATA BASES.  THE INFORMATION MAINTAINED BY DL/I IN THE
* SECONDARY INDEXES IS USED TO ACCESS (BUILD SEARCH ARGUMENTS FOR)
* THE LOGICAL DATA BASE DEFINED BY THE OTHER PCB (DBDNAME=STDCDBL).
* THE PROCOPT FOR THIS PCB HAS A PROCOPT OF GP WHICH ALLOWS THE
* ONLINE PROGRAM TO ISSUE PATH CALLS (*D).
*
* THIS PSB SHOWS AN EXAMPLE OF FIELD LEVEL SENSITIVITY FOR SEGMENT
* STLCITM.  SINCE THE PHYSICAL LENGTH AND CHARACTER FORMAT ARE
* DESIRED THE TYPE AND BYTE PARAMETERS ARE NOT CODED.
*
```

Figure 3-21. PSBs Used for the Phase 3 Sample Application (Part 2 of 3)

```
            PCB    TYPE=DB,DBDNAME=STDCX1P,PROCOPT=G,KEYLEN=50,POS=S
            SENSEG NAME=STICMNA,PARENT=0
            PCB    TYPE=DB,DBDNAME=STDCDBL,PROCOPT=GP,KEYLEN=50,POS=S
            SENSEG NAME=STSCCST,PARENT=0
            SENSEG NAME=STSCLOC,PARENT=STSCCST
            SENSEG NAME=STPCORD,PARENT=STSCLOC
            SENSEG NAME=STLCITM,PARENT=STPCORD
              SENFLD NAME=STKCIIN
              SENFLD NAME=STQCILI
              SENFLD NAME=STFCIQO
              SENFLD NAME=STFCIQS
              SENFLD NAME=STFCIQB
              SENFLD NAME=STFCIAM
              SENFLD NAME=STQIINO
              SENFLD NAME=STFIIDS
              SENFLD NAME=STFIIQH
              SENFLD NAME=STFIIQO
              SENFLD NAME=STFIIPR
            PCB    TYPE=DB,DBDNAME=STDCX2P,PROCOPT=G,KEYLEN=50,POS=S
            SENSEG NAME=STIRCRDN,PARENT=0
            PSBGEN LANG=ASSEM,PSBNAME=STBCUSR
            END
/*
// EXEC LNKEDT
```

## PSB for the Online Application - Update

```
// OPTION CATAL,NODECK
// EXEC ASSEMBLY
            TITLE 'DL/I ONLINE SAMPLE PROGRAM - CUSTOMER AND INVENTORY PSBX
                  - UPDATE'
            PRINT  NOGEN
*
* THIS IS THE PSB WHICH ALLOWS DLZSAM60 TO UPDATE THE
* CUSTOMER AND INVENTORY DATA BASES.  UPDATE CAPABILITY VIA PATH
* CALL IS SPECIFIED BY PROCOPT=AP.
*
            PCB    TYPE=DB,DBDNAME=STDCDBL,PROCOPT=AP,KEYLEN=50,POS=S
            SENSEG NAME=STSCCST,PARENT=0
            SENSEG NAME=STSCLOC,PARENT=STSCCST
            SENSEG NAME=STPCORD,PARENT=STSCLOC
            SENSEG NAME=STLCITM,PARENT=STPCORD
              SENFLD NAME=STKCIIN
              SENFLD NAME=STQCILI
              SENFLD NAME=STFCIQO
              SENFLD NAME=STFCIQS
              SENFLD NAME=STFCIQB
              SENFLD NAME=STFCIAM
              SENFLD NAME=STQIINO
              SENFLD NAME=STFIIDS
              SENFLD NAME=STFIIQH
              SENFLD NAME=STFIIQO
              SENFLD NAME=STFIIPR
            PSBGEN LANG=ASSEM,PSBNAME=STBCUSU
            END
/*
// EXEC LNKEDT
/&
```

Figure 3-21. PSBs Used for the Phase 3 Sample Application (Part 3 of 3)

# Application Control Blocks Creation and Maintenance (DLZUACB0)

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. Thus, a third preparatory function, the creation of internal control blocks (DL/I application control blocks, or ACBs) is necessary prior to execution.

The application control blocks creation and maintenance utility is executed as a DOS/VS problem program and accepts control statements as input. The PSB for the application program and its related DBD(s) are loaded from a DOS/VS core image library. An expanded PSB is built from the PSB CSECT. A data management block (DMB) is created for each related DBD CSECT if the DMB does not already exist in a core image library.

The output of the application control blocks creation and maintenance utility must be link-edited and cataloged into a DOS/VS core image library (see Figure 3-22). 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.

## Control Statement Requirements

The control statement requirements for this program are free form. A statement is coded as a card image and is contained in columns 1-71. The control statement may contain a name starting in column 1. The operation field must be preceded by and followed by one or more blanks. The operand field is composed of one or more PSB names and optionally an output destination and/or a DMB generation control parameter. It must be preceded by and followed by one or more blanks. Commas, parentheses, and blanks can be used only as delimiting characters. Comments may be written following the last operand of a control statement, separated from the operand by one or more blanks.

A control statement or PSB operand may be contained on more than one line by inserting a comma after the last PSB name of the first line, inserting a character other than a blank in column 72, and continuing the statement in column 16 of the next line. Columns 1-15 of the continuation line must be blank.

CONTROL STATEMENT

DBDs

PSBs

DOS/VS CORE
IMAGE LIBRARY

APPLICATION CONTROL
BLOCKS CREATION
AND MAINTENANCE
UTILITY

MESSAGES

SYSPCH
OR
SYSLNK

PSBs
AND
DMBs

DOS/VS
LINKAGE
EDITOR

DMBs

PSBs

DOS/VS CORE
IMAGE LIBRARY

MESSAGES
AND STATISTICS
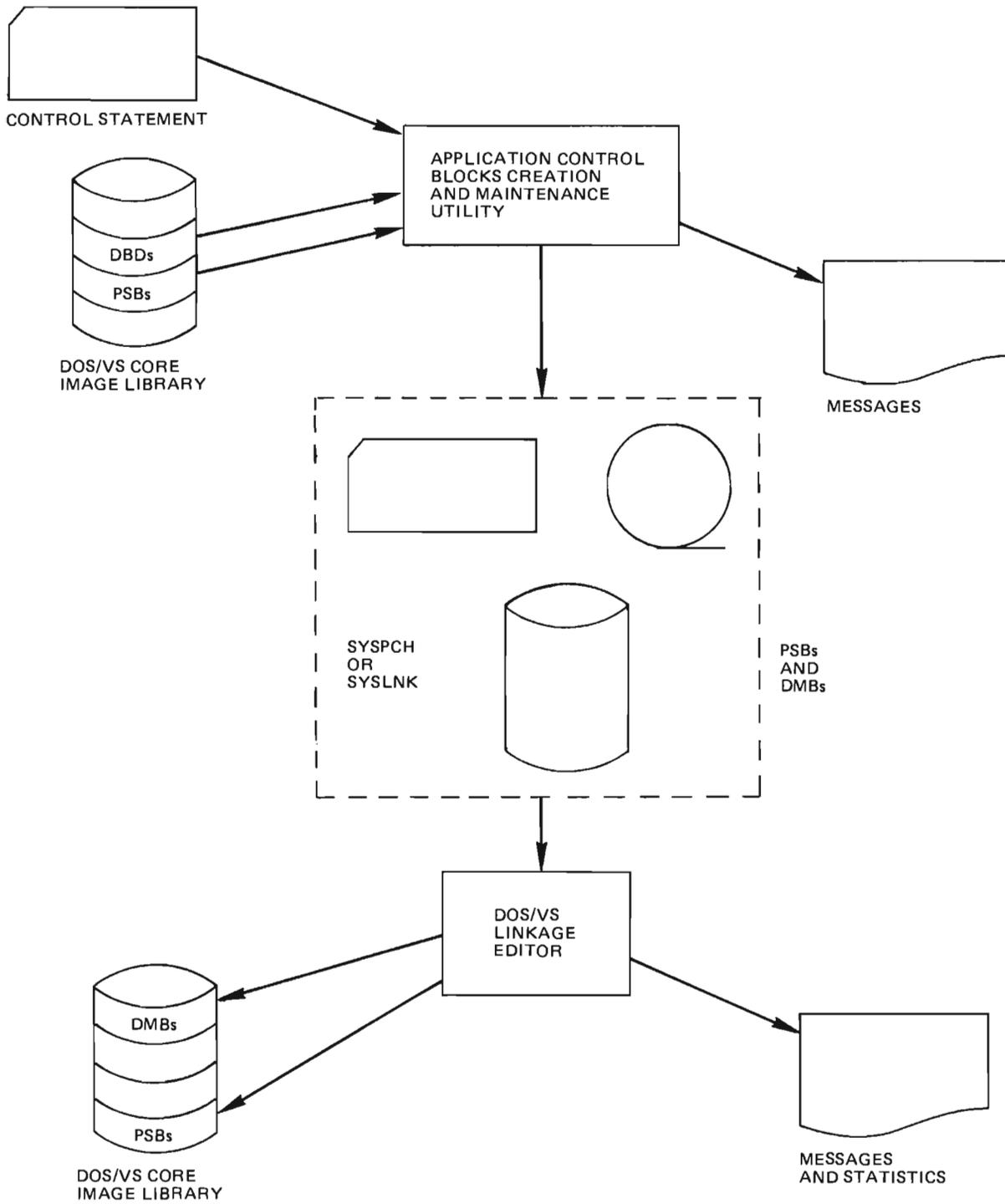
Figure 3-22. DL/I ACB Creation and Maintenance for Each PSB

The format of the control statement is:

| [label] | BUILD | PSB=(psbname,...) |
|---------|-------|-------------------|
|         |       | [,OUT=LINK] |
|         |       |         {COND}<br>[,DMB={YES }]<br>        {NO  } |

label
> is optional and is useful only for documentation purposes. If specified it must be a 1- to 8-character alphameric value.

BUILD
> indicates that blocks are to be built for the named PSBs.

PSB=(psbname,...)
> means blocks are to be built for all PSBs named. As many of this type of control card as required may be submitted.

OUT=LINK
> if specified in any BUILD statement the output destination of all the created control block(s) is SYSLNK. If the parameter is omitted, the output of all the control blocks is on SYSPCH.

DMB=

> {COND}
> {YES }
> {NO   }

> controls the generation of DMBs for data bases referenced by the named PSBs. The default, COND, indicates that only those DMBs not currently present in the DOS/VS core image library (or assigned private library) will be generated.

> If you specify DMB=YES, all DMBs will be generated. If DMB=NO is specified, no DMBs will be generated.

**Notes:**

1. This program creates PSB and DMB object modules that contain DOS/VS linkage editor control statements. This output must be cataloged and link-edited into a core image library (private or system) before control blocks may be accessed by DL/I. If output in on SYSPCH, the necessary DOS/VS JOB and EXEC LNKEDT control statements are also written.

2. A maximum of 255 DBDs may be referenced by one PSB. Included in this maximum are those DBDs that are indexes to, or are logically related to, those referred to by the PCBs in this PSB. Also included in this maximum are any other DBDs that have index or logical relationships with any of the above related DBDs, no matter how remote.

3. A maximum of 500 unique DBD names (for all PSBs) may be referenced in a single execution.

4. There is no maximum to the number of input control statements that may be submitted in a single job execution.

5. Control statements are read from the SYSIPT device.

6. DMBs are built for DBDs referenced directly in a PSB generation PCB statement (with the exception of a LOGICAL DBD) or referenced indirectly by a previously referenced DBD.

### JCL Requirements

The application control blocks creation and maintenance utility is executed as a standard DOS/VS application program. If you do not specify OUT=LINK in the BUILD statement, a job stream including DOS/VS JOB and EXEC LNKEDT statements as well as the requested object module is written onto SYSPCH. If you specify OUT=LINK on the BUILD statement, and object module is written to SYSLNK.

The following job stream is used to execute the application control blocks creation and maintenance utility and catalog and link-edit the object modules to a

DOS/VS core image library. This is the ACB generation for the phase 3 environment.

```
// JOB STJACBGN GENERATE ALL ACBS
// OPTION CATAL,NODECK,DUMP
// EXEC DLZUACB0,SIZE=200K
    BUILD PSB=(STBICLD,STBCUSR,STBCUSU),OUT=LINK,DMB=YES
    BUILD PSB=(STBICLG),OUT=LINK,DMB=YES
/*
// EXEC LNKEDT
/*
/&
```

# VSAM Requirements

Before your data bases can be loaded, they must first be defined to VSAM using the DOS/VS Access Method Services utility functions. The Access Method Services must be used to define a VSAM catalog, VSAM data space, and VSAM data sets.

- *VSAM Catalog:* A master catalog must be defined first and then, optionally, any number of VSAM user catalogs. A VSAM catalog is a central information point for all VSAM data sets and the direct-access storage volumes on which they are stored. The VSAM catalog provides VSAM with the information to allocate space for data sets, verify authorization to gain access to them, compile usage statistics on them and relate relative byte addresses (RBAs) to physical locations.

- *VSAM Data Spaces:* This is DASD space assigned to VSAM, from which VSAM allocates space for VSAM data sets. A record of this data space is maintained in a VSAM catalog. VSAM does its own DASD space management (for example, allocating space for VSAM data sets). Each VSAM data space can occupy part or all of a DASD volume.

- *VSAM Data Sets:* When a VSAM data set is defined, it is allocated space in a VSAM data space. A record of the data set and the space that it occupies is maintained in a VSAM catalog. All VSAM data sets must be cataloged.

The sample application supplied with Version 1.3 includes the Access Method Services job needed to define the VSAM data sets. It is assumed that you already have defined your VSAM catalog(s) and VSAM data spaces. That is, you have used the Access Method Services DEFINE command (DEFINE MASTERCATALOG, DEFINE USERCATALOG, DEFINE SPACE) to establish your VSAM system. This section covers the use of the Access Method Services DEFINE CLUSTER command. See *Using VSE/VSAM Commands and Macros,* SC24-5144, for additional information.

## *Data Set Definition*

All VSAM data sets are defined with the DEFINE CLUSTER command. At the time a data set is defined, its attributes and all volume serial numbers of the volumes for the data set are recorded in the catalog. A catalog record is set up for each component of the cluster and one for the cluster as a whole. This method of establishing a catalog record for each data set component and a catalog record for the cluster provides the structure to:

- store the information required to manage a data set

- allow access to each component of the data set as well as the whole data set.

As explained in Chapter 2, a VSAM KSDS consists of two components; the data component (the actual data to be processed) and the index component (used to address the data). A VSAM ESDS consists of one component -- the data component. Figure 3-23 shows the catalog entries made when the data set (cluster) is defined.
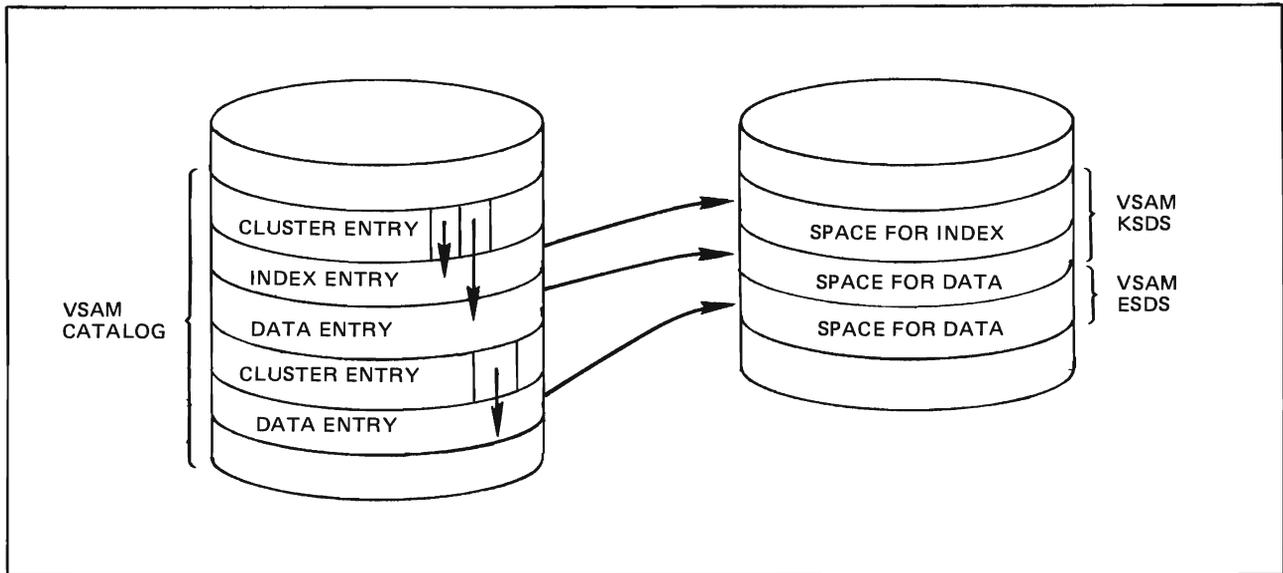
Figure 3-23. Defining VSAM Data Sets

An Access Method Services DEFINE command is used to define a VSAM data set. This means that space is allocated for the data set, the name is assigned, and other data set information is entered into the VSAM catalog. The DEFINE command does not put any data into the data set.

The following job is used to define the Inventory, Customer, and Index data bases to VSAM. Note the use of the DELETE CLUSTER command for each cluster at the beginning of the job. The DELETE command is necessary if you are redefining a cluster (to reload a data base) to remove the name of the file from the VSAM catalog and release the space allocated for it. The following DEFINE commands then cause the new data set definition to be recorded on the VSAM catalog. This is job STJDFINV in the sample jobstream.

```
// JOB STJDFINV DEFINE INVENTORY, CUSTOMER AND INDEX DATA BASES
// EXEC IDCAMS,SIZE=AUTO
         DELETE (SAMPLE.INVEN) CLUSTER NOERASE PURGE
         DELETE (SAMPLE.INVDX) CLUSTER NOERASE PURGE
         DELETE (SAMPLE.CUST) CLUSTER NOERASE PURGE
         DELETE (SAMPLE.CUSTDX1) CLUSTER NOERASE PURGE
         DELETE (SAMPLE.CUSTDX2) CLUSTER NOERASE PURGE
         DEFINE CLUSTER (                              -
                      NAME(SAMPLE.INVEN)               -
                      NONINDEXED   )                   -
                DATA  (                                -
                      NAME(INVENT)                     -
                      VOLUMES(111111)                  -
                      CYL(1 1)                         -
                      CNVSZ(2048)                      -
                      RECSZ(2038 2038))
         DEFINE CLUSTER (                              -
                      NAME(SAMPLE.INVDX)               -
                      INDEXED                          -
                      KEYS(06 10)  )                   -
                INDEX (                                -
                      VOLUMES(111111)                  -
                      NAME(SAMPLE.INVEN.INDEX))        -
                DATA  (                                -
                      NAME(SAMPLE.INDX1)               -
                      VOLUMES(111111)                  -
                      CYL(1 1)                         -
                      FREESPACE(10 10)                 -
                      CNVSZ(2048)                      -
                      RECSZ(18 18)  )
         DEFINE CLUSTER (                              -
                      NAME(SAMPLE.CUST)                -
                      NONINDEXED  )                    -
                DATA  (                                -
                      NAME(CUSTOMER)                   -
                      VOLUMES(111111)                  -
                      CYL(1 1)                         -
                      CNVSZ(2048)                      -
                      RECSZ(2038 2038) )
         DEFINE CLUSTER (                              -
                      NAME(SAMPLE.CUSTDX1)             -
                      INDEXED                          -
                      KEYS(31 10)  )                   -
                INDEX (                                -
                      VOLUMES(111111)                  -
                      NAME(SAMPLE.CUDX1.INDEX))        -
                DATA  (                                -
                      NAME(SAMPLE.CUDX1)               -
                      VOLUMES(111111)                  -
                      CYL(1 1)                         -
                      FREESPACE(10 10)                 -
                      CNVSZ(2048)                      -
                      RECSZ(42 42) )
         DEFINE CLUSTER (                              -
                      NAME(SAMPLE.CUSTDX2)             -
                      INDEXED                          -
                      KEYS(12 10)  )                   -
                INDEX (                                -
                      VOLUMES(111111)                  -
                      NAME(SAMPLE.CUDX2.INDEX))        -
                DATA  (                                -
                      NAME(SAMPLE.CUDX2)               -
                      VOLUMES(111111)                  -
                      CYL(1 1)                         -
                      CNVSZ(2048)                      -
                      RECSZ(36 36))
/*
/&
```

**Notes:**
- The file attribute information for the DEFINE commands is taken directly
  from the output listing of the DBDGEN for each data base. For example, the
  output of the physical DBDGEN for the Inventory data base is:

```
130+*,CONTROL INTERVAL SIZE FOR THIS DATA SET IS 2048
141+*,..NR BLKS IN TRK...3..IN CYL...60..
348+*,VSAM DATA SET DESCRIPTIONS
349+*,
350+*,DATA BASE NAME.....................  STDIDBP
351+*,DATA BASE ORGANIZATION.............  HDAM
352+*,DEVICE TYPE........................  3340
353+*,
354+*,ESDS DATA SET NAME.................  STDIDBC
355+*,CONTROL INTERVAL SIZE..............  2048
356+*,NUMBER OF RECORDS IN CI............  1
357+*,RECORD LENGTH......................  2038
358+*,
```

> The attributes CONTROL INTERVAL SIZE and RECORD LENGTH are used
> to specify the parameter values for CNVSZ and RECSZ in the DEFINE
> CLUSTER command for SAMPLE.INVEN.

- The values for the KEYS parameter of the DEFINE CLUSTER command for
  SAMPLE.INVDX are also in the DBDGEN listing for the Inventory Index - Item
  Number data base.

```
        •
        •
        •
162+*,KEY LENGTH.........................  6
163+*,RELATIVE KEY POSITION..............  10
        •
        •
```

> This is also the case for the Customer data base and its indexes. Using the
> output listing of the DBDGEN for parameter information will assure that you
> have defined your VSAM data sets correctly.

## Loading Data Bases

After the data set is defined, it can be loaded with the data intended for the data set
(in this case, the data base records). This entails moving of data records from a
source data set such as a sequential data set or an indexed-sequential data set to the
VSAM data set. DL/I data bases are loaded using a series of DL/I insert calls. This is
job STJLDCST in the sample jobstream (DLZSAM40). Because it is necessary to use
DL/I calls to load a data base, this program will be discussed in Chapter 4, follow-
ing the presentation of the DL/I call macros.

# Chapter 4: Processing Data Bases (Batch Considerations)

## Structure of This Chapter

This chapter is divided into four parts. The first part deals with a general introduction to DL/I data base processing. It defines the basic structure of a DL/I application program. The second part introduces basic DL/I calls against a single hierarchical data base structure. It therefore uses the phase 1 sample environment. It also gives guidelines for Assembler, COBOL, PL/I, and RPG II application programs. However, the visualization of each DL/I call in particular is done following the COBOL syntax. The third part covers the processing of logical data bases which are implemented with the DL/I logical relationships function. The fourth part deals with secondary indexes.

## Introduction to Data Base Processing

In general, data base processing is transaction oriented. See "Chapter 2. Data Base Design", for a more detailed discussion of transactions and data bases. Generally, an application program accesses one or more data base records for each transaction it processes. There are two basic types of DL/I application programs:

- The direct access program
- The sequential access program

A direct access program accesses, for every input transaction, some segments on one or more data base records. These accesses are based on data base record and segment identification. This identification is essentially derived from the transaction input. Normally it is the root-key value and additional (key) field values of dependent segments. For more complex transactions, segments could be accessed in several DL/I data bases concurrently.

A sequential application program accesses sequentially selected segments of all or a consecutive subset of a particular data base. The sequence of processing data base records is usually determined by the key of the root-segment. The most common class of sequential application programs are report programs, which list some part of the data base. For such programs, consider using PL/I, RPG II, or the report feature of COBOL.

A DL/I application program normally processes only particular segments of the DL/I data bases. The portion that a given program processes is called an *application data structure*. This application data structure is defined in the *program specification block* (PSB). There is one PSB defined for each application program. More than one application program may use the same PSB. An application data structure always consists of one or more hierarchical data structures, each of which is derived from a DL/I physical or logical data base.

## Program Structure and Interface to DL/I

### Language and Compilation

The application program is written in one of four languages: PL/I, COBOL, RPG II, or Assembler language. The program is compiled through the user-selected language compiler and placed in the appropriate program library, after it is link-edited with the DL/I language interface module. For RPG II, a translation step is required prior to compilation.

### Interface Components

A DL/I batch application program executes in a manner similar to any other DOS/VS job in a partition. It executes, however, under control of DL/I. To perform the data base accesses as required by the application program, DL/I uses its own processing modules which in turn invoke DOS/VS services. DL/I also relies on the defined DBD and PSB control blocks to determine the data base organization and the program's access characteristics. Figure 4-1 presents an overview of DL/I and the application program during execution.

Before you execute an application program, a *program specification block generation* (PSBGEN) must be performed to create the *program specification block* (PSB) for the program. The PSB contains at least one PCB for each DL/I data base (logical or physical) accessed by the application program. The PCBs specify which segments the program will use and the kind of access (retrieve, update, insert, delete) the program is allowed to do. The PSBs are maintained in the DOS/VS Core Image Library. The coding and generation of PSBs is described in Chapter 3 of this manual.

During initialization, both the application program and its associated PSB are loaded from the library by the DL/I DOS/VS system. The DL/I modules interpret and execute data base call requests issued by the program.

The application program interfaces with DL/I via the following program elements:

- An entry statement specifying the PCBs utilized by the program
- A PCB-mask that corresponds to the information maintained in the pre-constructed PCB and which receives return information from DL/I
- An I/O area for passing data segments to and from the data bases
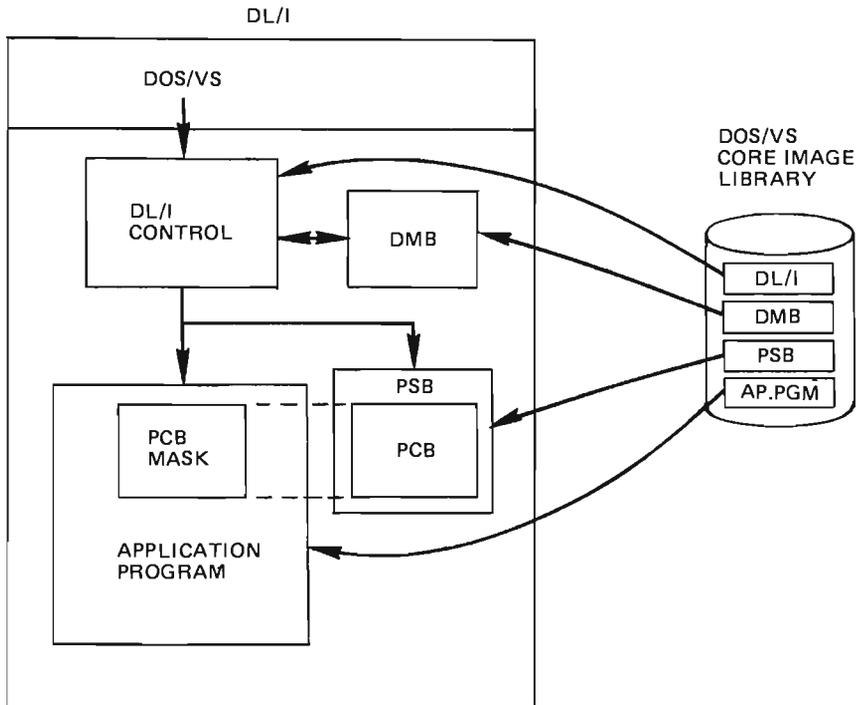
Figure 4-1. DL/I Interface with an Application Program

- Calls to DL/I specifying processing functions
- A termination statement

The PCB mask(s) and I/O areas are described in the program's data declaration portion. Program entry, calls to DL/I, processing, and normal termination are described in the program's procedural portion. Calls to DL/I, processing statements, and program termination may reference PCB mask(s) and/or I/O areas. In addition, DL/I may reference these data areas. Figure 4-2 illustrates how these elements are functionally structured in a program and how they relate to DL/I. The elements are discussed in the text that follows.

### Entry to an Application Program

Figure 4-2 shows that when DOS/VS gives control to the DL/I control facility, the DL/I control program in turn passes control to the application program (through the entry point as defined below). Register 1 contains an address of a list of pointers to PCBs used by the application program. At entry, all the PCBs used by the application program are specified. The order of the PCB-names in the entry statement must be the same as in the PSB for this application program. The sequence of the PCBs in the linkage section or declaration portion of the application program need not be the same as the sequence in the entry statement.

*COBOL:* The following statement must be the first in the procedure division.

ENTRY 'DLITCBL' USING pcb-name-1, ..., pcb-name-n.

*PLI:* The first statement of a PL/I program must be:

DLITPLI: PROCEDURE (pcb-pointer-1,...,pcb-pointer-n)
           OPTIONS(MAIN);

*RPG II:* In order to run an RPG II program using DL/I in batch mode, position 56 of the Header Specification must contain a "B". If "B" is not specified, the Translator does not perform any translate functions. For the DL/I control program to establish addressability to the PCBs and pass control to the application program, an *ENTRY PLIST must be the first entry in the Calculation Specifications.

The Translator will automatically generate the *ENTRY PLIST for a main program if the programmer does not explicitly specify it. However, the programmer must define all data bases as DB-files in the File Description Specifications with corresponding Continuation Lines (K-lines) specifying the PCBs. (For a detailed description of DB-files and PCB specification see "Data Base File Definition" in this chapter.) The entry parameter list will contain a PARM statement for each PCB, ordered according to the integers 'ij' as specified by PCBij in the K-line. If the programmer chooses to specify the *ENTRY PLIST himself, the PCB names in the PARM statements must be in the same sequence as in the PSB generation for the program. The Translator will not check the contents of the list.

*ASSEMBLER:* The entry point to an Assembler language program that utilizes DL/I may have any desired

name. However, when control is passed to the application program, register 1 contains the address of a variable-length fullword parameter list. Each word in this list contains a PCB control block address which must be saved by the application program. These addresses are in the same order as the PCB statements specified during PSB generation. The addresses in this list are subsequently used by the application program when executing DL/I calls.

Register 15 contains the address of the application program entry point. Additionally, registers 14 through 12 must be stored on entry to the application program in an 18 fullword save area which the application program must provide prior to the first DL/I call and which is pointed to by register 13. Generally, this is performed during program initialization.

APPLICATION PROGRAM COMPONENTS
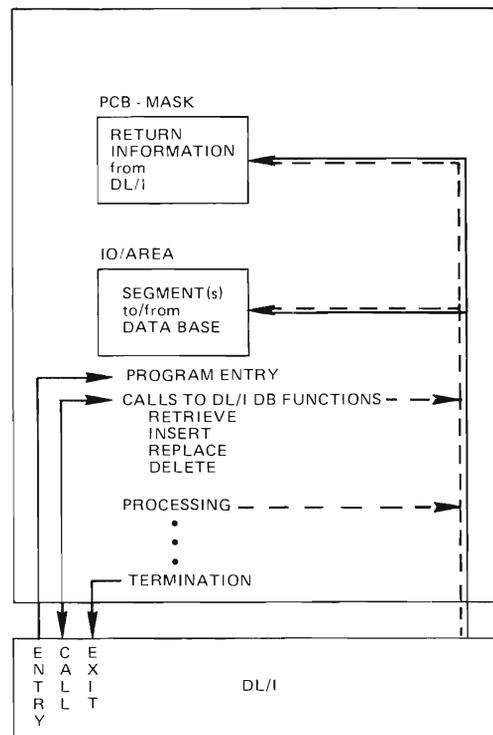


Figure 4-2. Structure of a Batch Application Program

The following is an example of an initialization performed by an application program; in this case it is the data base load program, DLZSAM40:

```
DLZSAM40  CSECT
          USING  DLZSAM40,R12
          SAVE   (14,12)
          LR     R12,R15          LOAD BASE REG
          ST     R13,SAVE+4       SAVE AREA CHAIN
          LA     R13,SAVE         SAVE AREA ADDR FOR THIS PROGRAM
          L      R9,0(R1)         ADDR OF INVENTORY PCB
          L      R10,4(R1)        ADDR OF CUSTOMER PCB
          .
          .
          .
DLZPARM   DC     A(COUNT)         START OF DL/I PARM LIST
DLZFUNC   DC     A(FUNCTION)      INSERT FUNCTION
DLZPCB    DC     A(0)             ADDRESS OF CURRENT PCB
DLZIOAR   DC     A(IOAREA)        ADDRESS OF SEGMENT TO INSERT
DLZSSA    DC     A(SSA)           ADDRESS OF SEGMENT NAME
COUNT     DC     F'4'             NUMBER OF PARMETERS IN LIST
SAVE      DS     18F              PROGRAM SAVE AREA
          .
          .
          .
          END
```

## PCB-Mask

A mask or skeleton data base PCB must be provided in the application program. The program views a hierarchical data structure via this mask. One PCB is required for each data structure. The details are shown in Figure 4-3.

Because the PCB does not actually reside in the application program, care must be taken to define the PCB-mask as an assembler DSECT, a COBOL Linkage Section entry, or a PL/I based variable.

For RPG II the PCB structure is defined in the input specifications. If you specify a K-line in the F-specs for a DB file, the Translator will generate this automatical-

ly. If you want to specify names of your choice, you may do so following the layout of the automatically generated PCB (Figure 4-3, part 2).

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

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

1. *Name of the PCB* - This is the name of the area which refers to the entire section of PCB fields. It

is used in program statements. This name is not a field in the PCB. It is the 01 level name in the COBOL mask in Figure 4-3. In RPG II, it is the data structure name of the PCB mask.

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

3. *Segment Hierarchy Level Indicator* - DL/I uses this area to identify the level number of the last segment encountered that satisfied a level of the call. When a retrieve is successfully completed, the level number of the retrieved segment is placed here. If the retrieve is unsuccessful, the level
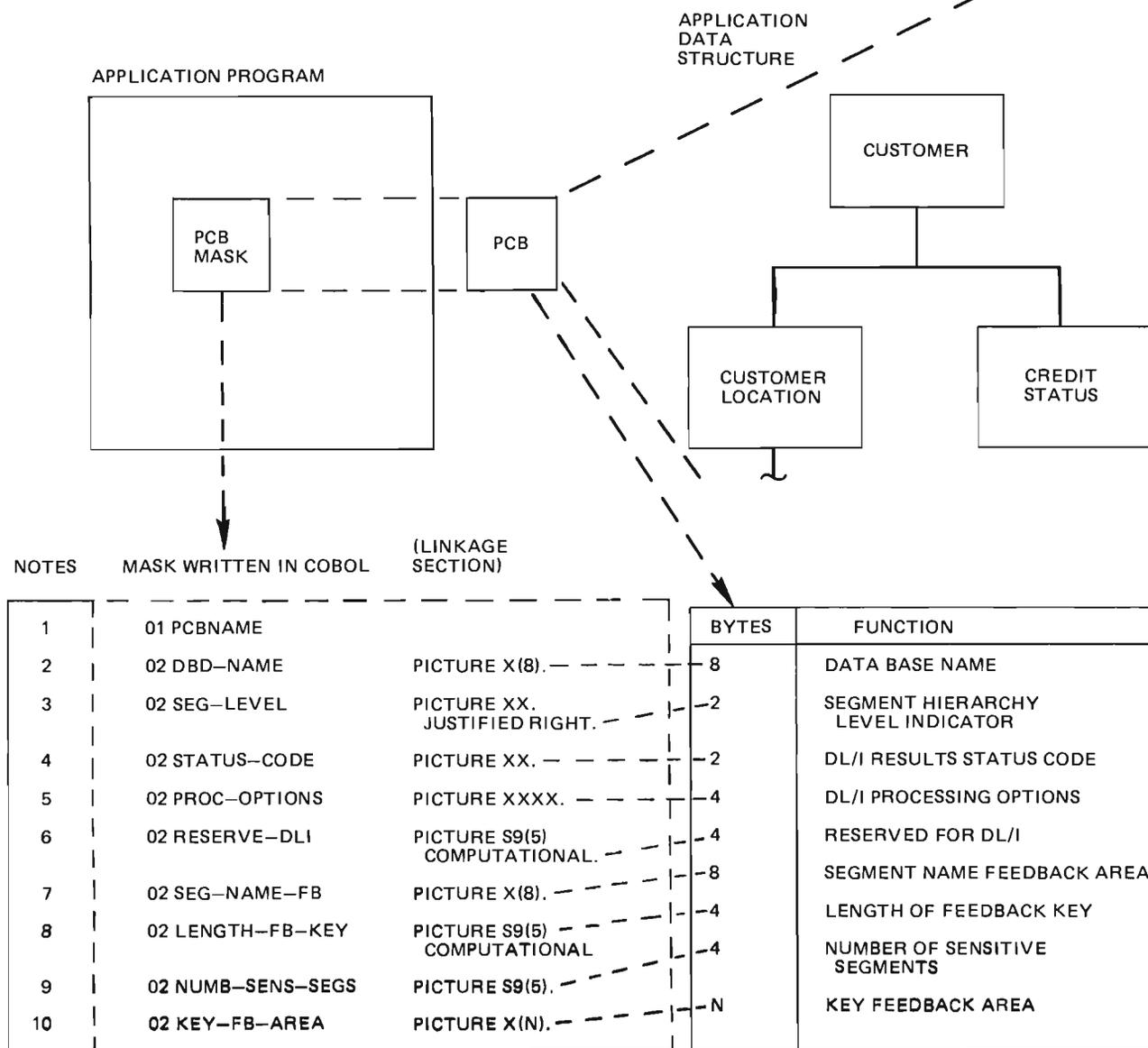


Figure 4-3. Application Program Data Base PCB Mask (Part 1 of 2)

GX21 9094-3 U/M050*
Printed in U.S.A

IBM    International Business Machines Corporation

| Program | | | Punching Instruction | Graphic | | | | | | Card Electro Number | | | Page | of | Program Identification | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Programmer | | Date | | Punch | | | | | | | | | | | | |

**I — Record Identification Codes / Field Location / Field Indicators**

(RPG Input Specifications coding form, filled-in entries)

| Line | Form Type | Filename | | From | To | Field Name |
| --- | --- | --- | --- | --- | --- | --- |
| 0 1 | I | PCBij | DS | | | |
| 0 2 | I | | | 1 | 8 | DBDNij |
| 0 3 | I | | | 9 | 10 | SEGLij |
| 0 4 | I | | | 11 | 12 | STCDij |
| 0 5 | I | | | 13 | 16 | PROCij |
| 0 6 | I | | B | 17 | 20 | RESRij |
| 0 7 | I | | | 17 | 20 | RESBij (RESBij overlays RESRij) |
| 0 8 | I | | | 21 | 28 | SEGNij |
| 0 9 | I | | B | 29 | 32 | KEYLij |
| 1 0 | I | | | 29 | 32 | KEYBij (KEYBij overlays KEYLij) |
| 1 1 | I | | B | 33 | 36 | SSGNij |
| 1 2 | I | | | 33 | 36 | SSGBij (SSGBij overlays SSGNij) |
| 1 3 | I | | | 37 | nnn | KEYAij |

nnn is the pcb-length + 36 (pcb-length is taken from pcb-keylength of the K-line of F-specs for DB-files).

Figure 4-3. Application Program Data Base PCB Mask (Part 2 of 2)

---

number returned is that of the last segment that satisfied the search criteria along the path from the root (the root segment level being '01') to the desired segment. If the call is completely unsatisfied, the level returned is '00'. This field contains character data; it is two bytes long and is right-justified numeric.

4. *DL/I Status Code* - A status code indicating the results of the DL/I call is placed in this field and remains here until another DL/I call uses this PCB. This field contains two bytes of character data. When a successful call is executed, this field is returned blank or with an informative status indi-cation. DL/I status codes are summarized for quick reference in Figure 4-4.

5. *DL/I Processing Options* - This area contains a character code that tells DL/I the 'processing intent' of the program against this data base, (for example, the kinds of calls that may be used by the program for processing data in this data base). This field is four bytes long. It is left-justified. It does not change from call to call. It gives the default value coded in the PCB PROCOPT parameter (see Chapter 3), although this value may be different for each segment. DL/I will not allow the application program to change this field, nor any other field in the PCB.

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

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

8. *Length of Key Feedback Area* - This entry specifies the current active length of the key feedback area described below. This field is a four byte binary number. For restrictions on the contents of binary fields in RPG II, see *DOS/VS RPG II Language.*

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

10. *Key Feedback Area* - DL/I places in this area the concatenated key of the last segment encountered that satisfied a level of the call. When a retrieve is successful, the key of the requested segment, and the key field of each segment along the path to the requested segment, are concatenated and placed in this area. The key fields are positioned from left to right, beginning with the root segment key and following the hierarchical path. When a retrieve is unsuccessful, the keys of all the segments along the path to the requested segment, for which the search was successful, are placed in this area. Segments without sequence fields are not represented in this area.

Note: This area is never cleared, so it should not be used after a completely unsuccessful call. See Chapter 2 for an explanation of concatenated keys.

| STATUS CODE | DATA BASE CALLS | | | | | | | | CALL COMPLETED | ERROR IN CALL or CONVERSION | I/O or SYSTEM ERROR | DESCRIPTION |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | GU GHU | GN GHN | GNP GHNP | DLET | REPL | ISRT (LOAD) | ISRT (ADD) | CHKP | | | | |
| AB | X | X | X | X | X | X | X | | | X | | SEGMENT I/O AREA REQUIRED, NONE SPECIFIED IN CALL |
| AC | X | X | X | | | X | X | | | X | | HIERARCHICAL ERROR IN SSAs |
| AD | | | | | | | | | | X | | INVALID FUNCTION PARAMETER |
| AH | | | | | | X | X | | | X | | CALL REQUIRES SSAs, NONE PROVIDED |
| AI | X | X | X | X | X | X | X | | | | X | DATA MANAGEMENT OPEN ERROR |
| AJ | X | X | X | X | X | X | X | | | X | | INVALID SSA QUALIFICATION FORMAT OR COMMAND CODE |
| AK | X | X | X | | | X | X | | | X | | INVALID FIELD NAME IN CALL |
| AM | X | X | X | X | X | X | X | | | X | | CALL FUNCTION NOT COMPATIBLE WITH PROCESSING OPTION OR SEGMENT OR PATH SENSITIVITY |
| AO | X | X | X | X | X | X | X | | | | X | I/O ERROR |
| DA | | | | X | | | | | | X | | SEGMENT KEY FIELD HAS BEEN CHANGED |
| DJ | | | | X | X | | | | | X | | NO PRECEDING SUCCESSFUL GET HOLD CALL |
| DX | | | | X | | | | | | X | | VIOLATED DELETE RULE |
| GA | | ★ | ★ | | | | | | X | | | CROSSED HIERARCHICAL BOUNDARY INTO HIGHER LEVEL (RETURNED ONLY ON CALLS WITH NO SSA SPECIFIED) |
| GB | | ★ | | | | | | | | | | END OF DATA SET, LAST SEGMENT REACHED |
| GE | ★ | ★ | ★ | | | | ★ | | | | | SEGMENT OR PARENT SEGMENT NOT FOUND |
| GK | | ★ | ★ | | | | | | X | | | DIFFERENT SEGMENT TYPE AT SAME LEVEL RETURNED (RETURNED ON UNQUALIFIED CALLS ONLY) |
| GP | | | X | | | | | | X | | | A GNP CALL AND NO PARENT ESTABLISHED, OR REQUESTED SEGMENT LEVEL NOT LOWER THAN PARENT LEVEL |
| II | | | | | | | ★ | | | | | SEGMENT TO INSERT ALREADY EXISTS IN DATA BASE OR IS NON-UNIQUE |
| IX | | | | | | | X | | | X | | VIOLATED INSERT RULE |
| KA | X | X | X | X | X | X | X | | | X | | NUMERIC TRUNCATION ERROR DURING CONVERSION |
| KB | X | X | X | X | X | X | X | | | X | | CHARACTER TRUNCATION ERROR DURING CONVERSION |
| KC | X | X | X | X | X | X | X | | | X | | INVALID PACKED/ZONED DECIMAL CHARACTER DURING CONVERSION |
| KD | X | X | X | X | X | X | X | | | X | | TYPE CONFLICT DURING CONVERSION |
| KE | | | | | X | | | | | X | | REPLACE VIOLATION |
| LB | | | | | | ★ | | | | | | SEGMENT TO INSERT ALREADY EXISTS IN DATA BASE OR IS NON-UNIQUE |
| LC | | | | | | ★ | | | | | | KEY FIELD OF SEGMENTS OUT OF SEQUENCE |
| LD | | | | | | ★ | | | | | | NO PARENT FOR THIS SEGMENT HAS BEEN LOADED |
| LE | | | | | | ★ | | | | | | SEQUENCE OF SIBLING SEGMENT NOT THE SAME AS DBD SEQUENCE |
| NA | | | | X | | | | | | X | | DATA IN SEARCH OR SUBSEQUENCE FIELD HAS BEEN CHANGED |
| NE | | | X | X | | | | | | X | X | INDEX MAINTENANCE CANNOT FIND SEGMENT |
| NI | | | X | X | X | X | | | | | X | INDEX MAINTENANCE UNABLE TO OPEN INDEX DATA BASE |
| NI | | | | | X | X | X | | | X | | DUPLICATE KEY FOUND FOR INDEX DATA BASE |
| NO | | | X | X | X | X | | | | | X | I/O ERROR |
| RX | | | | X | | | | | | X | | VIOLATED REPLACE RULE |
| V1 | | | | X | | X | X | | | X | | INVALID LENGTH FOR VARIABLE LENGTH SEGMENT |
| XD | | | | | | | | X | | | | ERROR DURING DATA BASE BUFFER WRITE OUT |
| XH | | | | | | | | X | | | | DATA BASE LOGGING NOT ACTIVE |
| ␣␣ | ★ | ★ | ★ | ★ | ★ | ★ | ★ | ★ | X | | | GOOD. NO STATUS CODE RETURNED, PROCEED! |

★ Indicates status code that could be expected as normal situation.

Figure 4-4. DL/I Status Codes

## Calls to DL/I

COBOL, PL/I and Assembler application programs communicate with DL/I using a program call. In RPG II, communication with DL/I is established by using an RQDLI (Request DL/I) command which is translated into a CALL statement by the Translator. Therefore, "call" in this manual implies "RQDLI command" for RPG II applications, unless RQDLI is specifically mentioned.

Note: Because the syntax of RPG II is significantly different, RPG II is discussed separately. See "DL/I Application Program for RPG II" later in this chapter.

A call request is composed of a call statement with an argument list. The argument list specifies the processing function to be performed, the hierarchical path to, and the segment occurrence of, the segment to be accessed. One segment or multiple segments along the hierarchical path of segments may be operated upon with a single DL/I call. However, a single call will never return more than one occurrence of one segment type.

The arguments contained within any DL/I call request include:

- For PL/I, a field (parm-count) containing the number of call arguments in the statement, excluding itself

- The input/output function to be performed

- The PCB name

- The segment input/output work area

- The identification of the data segment(s) to be operated upon.

Following is a sample of a basic call statement for COBOL:

```
CALL 'CBLTDLI' USING function,
     PCB-name,I/OArea,SSA1,...,SSAn.
```

function
> identifies the DL/I function to be performed. This argument is the name of a four-character field which describes the desired I/O operation. The DL/I functions are described briefly below, and in full detail later in this chapter.

PCB-name
> is the name of a data base program communication block (PCB). See "PCB-name Argument" below.

I/OArea
> is the name of an I/O work area. See the section "I/O Work Area Argument" below.

SSA1 through SSAn
> the names of segment search arguments (optional). A maximum of 1 SSA per level is allowed for the hierarchical path being accessed. See "Segment Search Arguments" below.

*Function Argument:* The I/O functions specified in the "function" argument of the call statement request data services of DL/I. The functions provide a full data processing capability of retrieving, updating, adding, and deleting data.

Following are the basic DL/I call functions to request DL/I data base services:

| Meaning | DL/I CAll Function |
|---|---|
| GET UNIQUE | 'GUbb' |
| GET NEXT | 'GNbb' |
| GET NEXT WITHIN PARENT | 'GNPb' |
| GET HOLD UNIQUE | 'GHUb' |
| GET HOLD NEXT | 'GHNb' |
| GET HOLD NEXT WITHIN PARENT | 'GHNP' |
| INSERT | 'ISRT' |
| DELETE | 'DLET' |
| REPLACE | 'REPL' |

Note: b stands for blank, each call function is always four characters.

The above calls constitute four categories of segment access:

- Retrieve a segment: GU, GN, GNP, GHU, GHN, GHNP

- Replace a segment: REPL

- Delete a segment: DLET

- Insert a segment: ISRT

In addition to the above data base calls, DL/I provides *system service* calls. These are used for requesting system services such as CHKP (checkpoint). All of the above calls are discussed in detail in the following sections. The CHKP call is discussed in detail in Chapter 7, "DL/I Data Base Recovery/Restart."

*PCB-name Argument:* "PCB-name" is the second (third in PL/I) argument in the call statement. It is the name of the PCB within the PSB that identifies for DL/I which specific hierarchical data structure the application program wishes to process.

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

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

Note: It is good practice to make the length of a general I/O area large enough to accomodate future segment extensions. An installation standard could be set for this.

*Segment Search Arguments:* One SSA can be provided for each segment accessed in a hierarchical path. The purpose of the SSA is to identify the segment to be accessed, by segment name and, optionally, by a field value.

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

- Narrow the field of search to a particular segment type, or to a particular segment-occurrence.

- Request that either one segment or a path of segments be processed.

- Alter DL/I's position in the data base for a subsequent call.

Segment search argument (SSA) names represent the fourth (fifth in PL/I) through last arguments (SSA1 through SSAn) in the call statement. There can be 0 or 1 SSA per level, and, since DL/I permits a maximum of 15 levels per data base, a call may contain from 0 to 15 SSA names. An SSA can consist of one, two, or three elements: the segment name, command code(s), and a qualification statement as shown in the following diagram.

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

**where:**

SEGMENT NAME

The segment name must be eight bytes long, left justified with trailing blanks as required. This is the name of the segment as defined in a physical and/or logical DBD referenced in the PCB for this application program.

COMMAND CODES

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

QUALIFICATION STATEMENT

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

Begin Qualification Character

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

Field Name

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

RO = Relational Operator

A set of two characters which express the manner in which the contents of the field, referred to by the field name, is to be tested against the comparative-value.

| Operator | Meaning |
|---|---|
| b= or =b | must be equal to |
| => | must be greater than or equal to |
| =< | must be less than or equal to |
| b> or >b | must be greater than |
| b< or <b | must be less than |
| ¬= | must be not equal to |

Note: b represents a blank character.

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

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

## *Qualification*

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

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

### EXAMPLE:
SEGNAMEbb last character blank to unqualify

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

### EXAMPLE: SEGNAMEb(FIELDXXX>=value)

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

## *General Characteristics of Segment Search Arguments*

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

- SSAs following the first SSA must proceed down a hierarchical path. Not all SSAs in the hierarchical path need to be specified. DL/I provides, internally, SSAs for missing levels according to the rules given later in this chapter. However, it is a good practice to always include SSAs for every segment level.

Examples of SSAs are given with the sample calls at each DL/I call discussion in the following section.

## *Termination*

At the end of processing of the application program, control must be returned to the DL/I control program.

| ANS COBOL | PL/I | Assembler | RPG II |
|---|---|---|---|
| GOBACK. | RETURN; | RETURN (14,12) | SETON LR |

The GOBACK or RETURN statement in a batch program returns control to DL/I. In RPG II control is returned to DL/I by setting on the Last Record (LR) indicator, specified in the calculation specifications. After DL/I resources are released and the data bases are closed, DL/I subsequently returns control to DOS/VS.

*Warning:* Since DL/I links to your application program, return to DL/I causes storage to be occupied by your program to be released. Therefore you should close all non-DL/I data sets for COBOL and Assembler before return to prevent abends during close by DOS/VS.

## *Status Code Handling*

After each DL/I call, a two-byte status code is returned in the PCB which is used for that call. The three categories of status codes are:

- The blank status code, indicating a successful call

- Exceptional conditions and warning status codes, for example, valid status codes from an application point of view

- Error status codes, specifying an error condition in the application program and/or DL/I.

The grouping of status codes in the above categories is somewhat installation dependent. The examples will, however, give a basic recommendation after each specific call function discussion.

You should also use a standard procedure for status code checking and the handling of error status codes. The first two categories should be handled by the application program after each single call; Figure 4-5 gives an example.

```
CALL 'CBLTDLI' USING ...
IF PCB-STATUS EQ 'GE' PERFORM
    PRINT-NOT FOUND.
IF PCB-STATUS NE 'bb' PERFORM
    STATUS-ERROR.
ELSE everything okay, proceed . . . . .
```

Figure 4-5. Testing Status Codes

Notice that it is more convenient to directly test the regular exceptions in-line instead of branching to a status code check routine. In this way, you clearly see the processing of conditions that you wish to handle from an application point of view, leaving other error situations to a central status code error routine. A detailed discussion of the error status codes and their handling is presented later in this chapter.

### *Sample Presentation of a Call*

The following sections introduce the DL/I calls. The discussion of each call includes a sample in the standard format as shown in Figure 4-6.

Although the sample application programs provided with DL/I are written in Assembler language, for ease of presentation the calls in the examples of this text are presented in ANS COBOL format. The coding of a call in PL/I, RPG II, or Assembler are presented later. Each call example contains three sections. The first section presents the essential elements of working storage as needed for the call. The second part, the processing section, contains the call itself. Note that the PCB-NAME parameter should refer to the selected PCB defined in the Linkage Section. Some examples include some processing function description before and/or after the call, in order to show the call in its right context. The third section contains the status codes and their interpretation, that can be expected after the call. The last category of status code, labelled "other: error situation", is normally handled by a user written status code error routine.

# DL/I Application Program for RPG II

Access to DL/I is provided in RPG II by means of RQDLI commands (Request DL/I) and, optionally, DB-files. The Translator tranlates the RQDLI commands into RPG II CALL statements and parameter lists and the DB-file specifications into File Description Specifications for SPECIAL files.

Note: The following syntax notation is used in the RPG II statement formats.

- | is used to separate alternatives, one of which has to be coded.
- (optional) is used to indicate that the construct is optional.
- uppercase letters are used to indicate system-defined information.
- lowercase letters are used to indicate user-defined information.

### *RQDLI Commands for DB Access*

The application program accesses a data base, which may be defined previously in the File Description Specifications, with the help of RQDLI commands, which have to be specified in the Calculation Specifications. An RQDLI command consists of an RQDLI statement followed by optional ELEM, USSA, and QSSA statements.

The format of the RQDLI statement is as follows:

| Position | Contents |
|---|---|
| 1-5 | see the publication, *DOS/VS RPG II Language* |
| 6 | C |
| 7-8 | blank \| Ln \| SR |
| 9-17 | see the publication, *DOS/VS RPG II Language* |
| 18-27 | func-name |
| 28-32 | RQDLI |
| 33-42 | file-name (optional) |
| 43-55 | blank |
| 56-57 | indicator |
| 58-59 | blank |
| 60-80 | see the publication, *DOS/VS RPG II Language* |

Note: No AN or OR lines are allowed with RQDLI commands.

```
77  GU-FUNC PICTURE XXXX VALUE 'GUbb'.
01  SSA001-GU-SE1PART.
  02  SSA001-BEGIN PICTURE ...
  02  ....
  02  ....
01  IOAREA    PICTURE X(256).
```
```
CALL 'CBLTDLI' USING GU-FUNC,PCB-NAME,IOAREA,SSA001-GU-STPIITM.
```
```
STATUS CODES:
   bb:  successful call
   --:  exceptional but correct condition
other:  error situation
```

Figure 4-6. Sample Call Presentation

**func-name:** The following function names may be used in an RQDLI statement:

- GU — Get unique
  GHU — Get hold unique
- GN — Get next
  GHN — Get hold next
- GNP — Get next within parent
  GHNP — Get hold next within parent
- DLET — Delete
- REPL — Replace
- ISRT — Insert:
  -load a new data base
  -add to an existing data base
- PCB — Schedule a PSB
- TERM — Release a PSB
- CHKP — Establish a checkpoint

The use and meaning is the same as explained under "Basic Data Base Processing" in this chapter.

**file-name:** The file-name specifies the data base to be accessed. If no FROM|INTO option is explicitly specified in the RQDLI command, standard RPG data transfer is used.

**standard RPG data transfer:** Extracting input fields from records, or building output records from fields. It is used if an RQDLI command requires a FROM or INTO option, which is not explicitly specified. In this case the I/O operation is executed in an RPG-like manner, namely using the record specification in the Input Specifications for input operations (that is, using the extract fields routine via READ statement instead of an explicit INTO option) or building the output record with the help of Output Specifications (that is, using the build lines routines via EXCPT instead of an explicit FROM option).

With an RQDLI command, only the first record is put out to the specified file; if more records are conditioned they are ignored. In addition, the RQDLI command causes all E-records with indicators on to be put out to the corresponding non-DB files. The user must ensure that files are conditioned in accordance with the RPG II rules for update files (read before write). A user-written EXCPT causes output to only non-DB files, but DB files also must be conditioned so that no output is attempted before a read. For standard data transfer, an EXCPT is automatically generated.

Note: Using the RPG II standard data transfer for an input operation on a DL/I data base, a READ will be issued even if the "record not found" condition is encountered. That means that in any case the contents of the fields within the record will be initiated with the information at which xREC is pointing.

**indicator:** An indicator must be reserved for use by the Translator. You may specify in the RQDLI command which indicator is to be used. If no indicator is

specified, the Translator will use indicator 13. The indicator should not be tested since, on return from DL/I, the status is undefined.

An RQDLI statement may be followed by one or more ELEM, USSA, or QSSA statements. The ELEM statements specify the FROM|INTO option, the PCB option, and the SSA option. The SSAs can also be specified by USSA and QSSA statements, which allow the definition of an SSA in RPG-like format. The statements specifying the SSA list must be in the proper hierarchical sequence.

The CHKP RQDLI statement may be followed by ELEM statements specifying the CHKPID option and the PCB option. No other ELEM statements are allowed.

An ELEM statement for the CHKPID option has the following format:

| Position | Contents |
|---|---|
| 1-5 | see the publication, *DOS/VS RPG II Language* |
| 6 | C |
| 7-8 | blank \| SR \| Ln |
| 9-17 | blank |
| 18-27 | CHKPID |
| 28-32 | ELEM |
| 33-42 | literal (see note) |
| 43-48 | var-name (see note) |
| 49-52 | optional entries (see 1. note) (2. the publication, *DOS/VS RPG II Language*) |
| 53-59 | blank |
| 60-80 | see the publication, *DOS/VS RPG II Language* |

Note: Entries in positions 33-42 and 43-52 are mutually exclusive.

The checkpoint identification can be specified either in positions 33-42 as an alphameric literal (maximum length eight bytes) or in positions 43-48 as a variable referring to an eight byte field. If no checkpoint identification is specified, the file-name, if any, specified in the CHKP RQDLI statement is used as a default checkpoint identification and for the PCB option if it is not explicitly specified and a K-line for a PCB has been defined for the DB-file.

**var-name:** denotes the name of a variable that describes an RPG II field, array, array-element, or data structure.

An ELEM statement for the FROM|INTO option has the following format:

| Position | Contents |
|---|---|
| 1-5 | see the publication, *DOS/VS RPG II Language* |
| 6 | C |
| 7-8 | blank \| SR \| Ln |
| 9-17 | blank |
| 18-27 | FROM\|INTO |
| 28-32 | ELEM |
| 33-42 | blank |
| 43-48 | var-name |
| 49-52 | optional entries (see the publication, *DOS/VS RPG II Language*) |
| 53-59 | blank |
| 60-80 | see the publication, *DOS/VS RPG II Language* |

If a FROM|INTO option is explicitly specified in an ELEM statement, the input/output request is executed using the specified area, ignoring any record definitions for the named DB-file in the Input or Output Specifications. If no FROM|INTO option is used with an RQDLI command, the record area optionally defined with the DB-file is loaded with the segment handled by the operation. The record area (corresponding to a data base segment) may be described in the Input or Output Specifications, depending on the requested function. The INTO option is used with input operations, and the FROM option is used with output operations.

An ELEM statement for the PCB option has the following format:

| Position | Contents |
| --- | --- |
| 1-5 | see the publication, *DOS/VS RPG II Language* |
| 6 | C |
| 7-8 | blank \| SR \| Ln |
| 9-17 | blank |
| 18-27 | PCB |
| 28-32 | ELEM |
| 33-42 | blank |
| 43-48 | var-name |
| 49-52 | optional-entries (see the publication, *DOS/VS RPG II Language*) |
| 53-59 | blank |
| 60-80 | see the publication, *DOS/VS RPG II Language* |

The PCB option may be used to specify the PCB-address to which the RQDLI request is directed. If not specified, the PCB-address is derived from the file-name specified with the RQDLI statement.

## *Statements for SSA Specification*

There are two kinds of statements used to describe an SSA, which may be used intermixed; either the SSA-option or the SSA specification in RPG-like format. In addition, an SSALIST option together with an ELIST-command are provided for ease of use. (The physical makeup of the SSA is fully described under "Calls to DL/I" earlier in this chapter.)

SSA-option

The SSA is a var-name. It is the user's responsibility to define the proper format and to put the correct values into it together with delimiters.

Note: The format of the area has to correspond exactly to the requirements as specified for the SSA in "Calls to DL/I".

ELEM statements of this kind are characterized by the keyword SSA in factor 1 of an ELEM statement and have the following format:

| Position | Contents |
| --- | --- |
| 1-5 | see the publication, *DOS/VS RPG II Language* |
| 6 | C |
| 7-8 | blank \| SR \| Ln |
| 9-17 | blank |
| 18-27 | SSA |
| 28-32 | ELEM |
| 33-42 | blank |
| 43-48 | var-name (see note) |
| 49-52 | optional entries (see the publication, *DOS/VS RPG II Language*) |
| 53-59 | blank |
| 60-80 | see the publication, *DOS/VS RPG II Language* |

The area referred to by var-name must describe the SSA with all required entries as defined under SSA in "Calls to DL/I" earlier in this chapter.

Note: For USSA and QSSA statements, var-name must not be an array name.

## *SSA Specification in RPG-Like Format: (USSA and QSSA Statement)*

The statement contains all the relevant fields of an SSA in RPG-like format. The Translator maps these fields into the proper DL/I format. For details see the following definitions.

### USSA Statement

For an *unqualified* SSA it is only necessary to specify either the segment-name in quotes or a field containing the segment name in factor1 of the Calculation Specifications in a USSA statement.

The proper area is provided by the Translator, and the segment will be moved into it with the required blanks.

USSA statements for an unqualified SSA have the following format in the Calculation Specifications:

| Position | Contents |
| --- | --- |
| 1-5 | see the publication, *DOS/VS RPG II Language* |
| 6 | C |
| 7-8 | blank \| SR \| Ln |
| 9-17 | blank |
| 18-27 | segment-name |
| 28-32 | USSA |
| 33-55 | blank |
| 56-57 | command code (optional) |
| 58-59 | blank |
| 60-80 | see the publication, *DOS/VS RPG II Language* |

**segment name:** Either var-name containing the name of a segment (up to 8 characters) or the name of a segment in apostrophes.

**command code:** One or two command codes may be specified. For a more detailed definition of command codes, see "Calls With Command Codes", later in this chapter.

### QSSA Statement

A QSSA statement for a *qualified* SSA has the following format:

| Position | Contents |
|---|---|
| 1-5 | see the publication, *DOS/VS RPG II Language* |
| 6 | C |
| 7-8 | blank \| SR \| Ln |
| 9-17 | blank |
| 18-27 | segment-name |
| 28-32 | QSSA |
| 33-42 | segment-field-name |
| 43-48 | comparative-value |
| 49-51 | blank |
| 52 | blank |
| 53 | blank |
| 54-55 | relational-operator |
| 56-57 | command-code (optional) |
| 58-59 | blank |
| 60-80 | see the publication, *DOS/VS RPG II Language* |

**segment name:** As above with unqualified SSA.

**segment-field-name:** Name of the segment-field in apostrophes or var-name containing name of the segment-field (up to 8 characters). The length of the field as defined in the DBD is specified by positions 49-51.

**comparative-value:** Var-name containing the value against which the contents of the field referred to by the segment-field-name are to be tested. The length of the contents of var-name should correspond to that defined in positions 49-51. This information is used to generate the proper area. The length as specified must correspond to the actual length of the field defined by the segment field name in the DBD.

**length:** Length of the segment-field (in bytes) in the DBD.

**position 52:** A blank entry indicates that the field is alphameric. MOVEL is used to put the comparative value into the generated SSA (possibly padded with blanks to the right).

**relational operator:** The following relational operators may be used:

| relational operator | meaning |
|---|---|
| EQ | equal to |
| GE | greater than or equal to |
| LE | less than or equal to |
| GT | greater than |
| LT | less than |
| NE | not equal to |

**command-code:** One or two command codes may be specified for each SSA. For a more detailed definition, see "Calls With Command Codes", later in this chapter.

### SSALIST-Option

It is possible to specify in an ELEM statement the name of an SSA-list. This ELEM statement has the following format:

| Position | Contents |
|---|---|
| 1-5 | see the publication, *DOS/VS RPG II Language* |
| 6 | C |
| 7-8 | blank \| SR \| Ln |
| 9-17 | blank |
| 18-27 | SSALIST |
| 28-32 | ELEM |
| 33-42 | name-of-SSA-list |
| 43-52 | blank |
| 53-59 | blank |
| 60-80 | see the publication, *DOS/VS RPG II Language* |

The keyword SSALIST indicates that this statement stands for a list of statements defined elsewhere in an ELIST. The Translator will expand the SSALIST-option by the list of SSAs defined in the ELIST. The indicator in position 7-8 of the SSALIST option is appended to each SSA. As default the indicator in position 7-8 of the RQDLI statement is used.

**name-of-SSA-list:** This name refers to the name of the ELIST defined in an ELIST statement.

### ELIST-Command

The ELIST command defines the SSA list. The ELIST command consists of an ELIST statement immediately followed by one or more statements specifying SSAs. The ELIST statement has the following format:

| Position | Contents |
|---|---|
| 1-5 | see the publication, *DOS/VS RPG II Language* |
| 6 | C |
| 7-8 | blank \| SR \| Ln |
| 9-17 | blank |
| 18-27 | name-of-SSA-list |
| 28-32 | ELIST |
| 33-59 | blank |
| 60-80 | see the publication, *DOS/VS RPG II Language* |

The statements specifying SSAs must be specified in the proper hierarchical sequence. The format of the statements is the same as that used to describe the SSA directly in the RQDLI commands.

**Restriction:** The SSALIST-option must not be used in an ELIST command. Optionally, a DB-file may be specified to access DL/I.

## DB (Data Base) File Definition

Each data base an application program wants to access may be defined in the File Description Sepcifications. The File Description Specifications for such a DB-file are only required if standard data transfer is intended for that DB-file and/or if use is made of the possibility of defining the PCB for a DB-file via a K-line in the File Description Specifications.

The File Description Specification for a DB-file has the following format:

| Position | Contents |
|---|---|
| 1-5 | see the publication, *DOS/VS RPG II Language* |
| 6 | F |
| 7-14 | file-name |
| 15 | I \| U \| O |
| 16 | D \| blank |
| 17-18 | blank |
| 19 | F \| blank |
| 20-23 | blank |
| 24-27 | maximum-segment-length |
| 28-39 | blank |
| 40-46 | DB |
| 47-74 | blank |
| 75-80 | see the publication, *DOS/VS RPG II Language* |

**file name:** The file-name can be freely chosen; it is the name by which the application refers to the data base.

**maximum-segment-length:** This length specifies the maximum length (in bytes) of the segments of the data base which the application is going to access. This length is used if no explicit FROM|INTO option is specified in an RQDLI command referencing the specific DB-file. In this case the segment has to be defined as a record in the Input or Output Specifications. If this length is omitted, a length of 80 is assumed.

**Notes:**

- If position 19 is blank, it will default to F.
- Output Specifications for DB-files must be of type E (position 15=E), exception records.

Additionally, for each DB File Description Specification, a continuation line may be specified which defines the corresponding PCB. The continuation line has the following format:

| Position | Contents |
|---|---|
| 1-5 | see the publication, *DOS/VS RPG II Language* |
| 6 | F |
| 7-23 | blank |
| 24-27 | pcb-key-length (optional) |
| 28-50 | blank |
| 51-52 | blank |
| 53 | K |
| 54-59 | PCB |
| 60-65 | PCBij |
| 66-74 | blank |
| 75-80 | see the publication, *DOS/VS RPG II Language* |

**PCBij:** This defines the program communication block (PCB) connected with the DB file. ij... establishes the relationship to the ordering of the PCBs in the PSB. ij defines this data base PCB as the element ij of the ordered list of PCBs. This ordering is used when the addressability of PCBs is established; ij may range between 01 and 99.

**pcb-key-length:** This integer specifies the length (less than or equal to 256) of the field in the data structure defining the PCB. If a K-line is specified, the Translator automatically generates the definition of the data structure for the PCB and puts it into the Input Specifica-

tions, with the names of the fields qualified by ij. The general format and the naming conventions can be seen in Figure 4-3 in "PCB Mask", in this chapter. If the K-lines for several DB Files define the same PCBij name, only the first causes the PCB data structure to be generated. The others are ignored and a warning message is issued. However, when these file names are specified in RQDLI statements, this PCBij name is used as the default value for the PCB option.

If no K-line is specified, it is the user's responsibility to define the proper PCB. For more detailed information, see "PCB Mask", in this chapter.

**Note:** With the automatic generation of the PCB data structure, name clashes with user-defined field names may occur.

The user should never write into PCB fields.

## Basic Data Base Processing

### DL/I Positioning

To satisfy a call, DL/I relies on two sources of segment identification:

- The established position in the data base as set by the previous call against the PCB

- The segment search arguments as provided with the call

The data base position is the knowledge by DL/I of the location of the last segment retrieved and all segments above it in the hierarchy. This position is maintained by DL/I as an extension of, and reflected in, the PCB. When an application program has multiple PCBs for a single data base, these positions are maintained independently. For each PCB, the position is represented by the concatenated key of the hierarchical path from the root segment down to the lowest level segment accessed. It also includes the positions of non-keyed segments.

If no current position exists in the data base, then the assumed current position is the start of the data base. This is the first physical data base record in the data base. With HDAM this is not necessarily the root-segment with the lowest key value.

### Sample Environment

The phase 1 sample environment is used to exemplify the basic DL/I calls presented in the following sections. The data base used is the Inventory data base as shown in Figure 4-7.
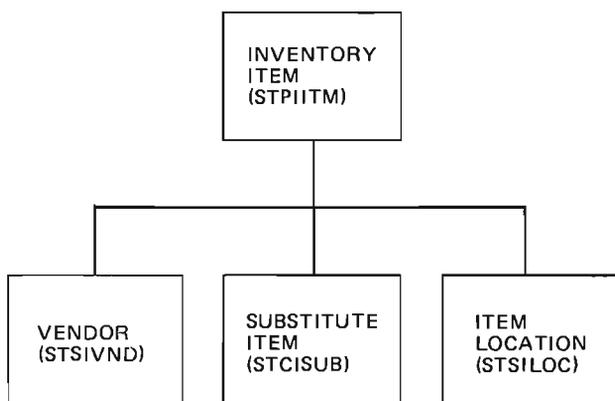
Figure 4-7. The Phase 1 Inventory Data Base

## Retrieving Segments

There are three basic functions in retrieving a segment:

- Retrieve a specific segment: GU

- Retrieve the next segment in the hierarchy: GN

- Retrieve the next segment within parent in the hierarchy: GNP

### *Get Unique Call (GU)*

The get unique call, function code 'GUbb', retrieves one segment in a hierarchical path. The segment retrieved is identified by an SSA for each level in the hierarchical path down to and including the requested segment. Each SSA should contain at least the segment name. The SSA for the root segment should provide the root-key value. Figure 4-8 shows an example of the get unique call.

The main use of the GU call is to position your program to a data base record to obtain (a path of) segment(s). Typically, the GU call is used only once for each data base record you wish to access. Additional segments within the data base record are then retrieved by means of get next or get next within parent calls (see following section). The GU call can also be used for retrieving a dependent segment, by adding SSAs to the call. For example, if you add a second SSA which specifies the item location, you would retrieve an ITEM LOCATION segment below the identified item. If the SSA did not provide an item location number, this would be the first occurrence of the ITEM LOCATION segment for this item.

```
77   GU-FUNC PICTURE XXXX VALUE 'GUbb'.
01   SSA001-GU-STPIITM.
  02   SSA001-BEGIN PICTURE X(19) VALUE 'STPIITMb(STQIINOb=b'.
  02   SSA001-STQIINO PICTURE X(8).
  02   SSA001-END PICTURE X VALUE ')'.
01   IOAREA PICTURE X(256).
```
```
MOVE ITEM-NUMBER TO SSA001-STQIINO.
CALL 'CBLTDLI' USING GU-FUNC,PCB-NAME,IOAREA,SSA001-GU-STPIITM.
```
```
STATUS CODES:
   bb:   requested INVENTORY ITEM segment has been moved to IOAREA
   GE:   segment not found; supplied item number not in data base
other:   error situation
```

Figure 4-8. Basic Get Unique Call

### *Get Next Call (GN)*

The get next call, function code 'GNbb', retrieves the next segment in the hierarchy as defined in the PCB. To determine the next segment, DL/I relies on the previously established position.

The unqualified get next call (Figure 4-9) using no SSAs will, if repeated, return the segments in the data base in hierarchical sequence. DL/I returns only those segments that are defined as sensitive in the PCB for the program issuing the call. If this call were issued after the get unique call of Figure 4-8, it would retrieve the first VENDOR segment for this INVENTORY ITEM (if one exists). Subsequent calls will retrieve all other VENDOR, SUBSTITUTE ITEM, and ITEM LOCATION segments for this INVENTORY ITEM. After this, the next

INVENTORY ITEM segment is retrieved and its dependent segments, etc., until the end of the data base is reached. DL/I returns special status codes whenever a different segment type at the same level or a higher level is returned. No special status code is returned when a different segment at a lower level is returned. You can check for reaching a lower level segment type using the segment level indicator in the PCB. Remember, only those segments to which your program is sensitive via its PCB are available to your application program.

Although the above unqualified GN call may be efficient, especially for report programs, you should use a qualified GN call whenever possible.

```
77  GN-FUNC PICTURE XXXX VALUE 'GNƀƀ'.
01  IOAREA   PICTURE X(256).
```
```
CALL 'CBLTDLI' USING GN-FUNC,PCB-NAME,IOAREA.
```
```
STATUS CODES:
  ƀƀ:   if a previous call retrieved an INVENTORY ITEM, then
        a VENDOR segment will be retrieved.
  GK:   a segment is returned in IOAREA, but it is a different
        type at the same level, for instance, a SUBSTITUTE
        ITEM segment after the last VENDOR segment.
  GA:   segment returned in IOAREA, but it is of a higher level
        than the last one, that is, a new INVENTORY ITEM segment
  GB:   end of data base reached, no segment retrieved
other:  error situation
```

Figure 4-9. Unqualified Get Next Call

*Qualified Get Next Call:* The qualified GN call should at least identify the segment you want to retrieve. In doing so, you will achieve a greater independence towards possible data base structure changes in the future. If you supply the segment name in the SSA, then you will retrieve all segments of that type from all data base records with subsequent get next calls (see figure 4-10).

Repetition of the qualified GN call (Figure 4-10) will retrieve all subsequent SUBSTITUTE ITEM segments of the data base until the end of the data base is reached. To limit this to a specific INVENTORY ITEM, you could add a fully qualified SSA for the INVENTORY ITEM segment. This would be the same as used in Figure 4-8.

Note: You could follow this call with the get next within parent call, function code 'GNPƀ', with a qualified SSA. See the *Application Programming Reference Manual* for specific details about coding this call.

An example of a get next call with a fully qualified SSA is shown in Figure 4-11. Because the fully qualified SSA always clearly identifies the hierarchical path and the segment you want to retrieve, it should be used whenever possible.

```
77  GN-FUNC PICTURE XXXX VALUE 'GNƀƀ'.
01  SSA002-GN-STCISUB PICTURE X(9) VALUE 'STCISUBƀƀ'.
01  IOAREA PICTURE X(256).
```
```
CALL 'CBLTDLI' USING GN-FUNC,PCB-NAME,IOAREA,SSA002-GN-STCISUB.
```
```
Note the use of the function code in the SSA name to help the
application programmer identify which SSA to use.  SSAs for each
type of call for each segment in each data base should be
constructed once by the data base administration function and
placed in the source statement library so all programs using
that data base will use common names.
```
```
STATUS CODES:
  ƀƀ:   next SUBSTITUTE ITEM segment has been moved to IOAREA
  GB:   end of data base reached, no more SUBSTITUTE ITEM segments
other:  error situation
```

Figure 4-10. Qualified Get Next Call

```
77   GN-FUNC PICTURE XXXX VALUE 'GNbb'.
01   SSA001-GU-STPIITM.
  02   SSA001-BEGIN PICTURE X(19) VALUE 'STPIITMb(STQIINOb=b.
  02   SSA001-STQIINO PICTURE X(6).
  02   SSA001-END PICTURE X VALUE ')'.
01   SSA002-GN-STCISUB PICTURE X(9) VALUE 'STCISUBbb'.
01   IOAREA PICTURE X(256).
```
```
CALL 'CBLTDLI' USING GN-FUNC,PCB-NAME,IOAREA,SSA001-GU-STPIITM,
               SSA002-GN-STCISUB.
```
```
STATUS CODES:
  bb:   next SUBSTITUTE ITEM segment is in IOAREA
  GE:   segment not found; no more substitute items for this item,
        or item number in SSA001 does not exist
other:   error situation
```

Figure 4-11. Get Next Call With Qualified SSA

## *Get Hold Calls*

To change the contents of a segment in a data base through a replace or delete call, the program must first obtain the segment. It then changes the segment's contents and requests DL/1 to replace the segment in the data base or to delete it from the data base.

This is done by using the get hold calls. These function codes are like the standard get function, except the letter 'H' immediately follows the letter 'G' in the code (for example, GHU, GHN, GHNP). The get hold calls function exactly as the corresponding get calls for the user. For DL/1, they indicate a possible subsequent replace or delete call.

After DL/1 has provided the requested segment to the user, one or more fields, but not the sequence field, in the segment may be changed.

After the user has changed or examined the segment contents, he can call DL/1 to return the segment to, or delete it from, the data base. If after issuing a get hold call, the program determines that it is not necessary to change or delete the retrieved segment, the program may proceed with other processing, and the 'hold' will be released by the next DL/1 call against the same PCB.

## Updating Segments

Segments can be updated by application programs and returned to DL/1 for restoring in the data base, with the replace call, function code 'REPL'. Two conditions must be met:

* The segment must first be retrieved with a get hold call (GHU, GHN, or GHNP); no intervening calls are allowed referencing the same PCB.

* The sequence field of the segment cannot be changed; this can only be done with combinations of delete and insert calls for the segment and all its dependents.

Figure 4-12 shows an example of a combination of a GHU and REPL call. Notice that the replace call must not specify an SSA for the segment to be replaced. If, after retrieving a segment with a get hold call, the program decides not to update the segment, it need not issue a replace call. Instead the program can proceed as if it were a normal call.

Note: Because there is very little performance difference between the get and the get hold call, you should use the get hold call whenever there is a reasonable chance that you will change the segment.

```
| 77  GHU-FUNC PICTURE XXXX VALUE 'GHUb'.
| 77  REPL-FUNC PICTURE XXXX VALUE 'REPL'.
01  SSA001-GU-STPIITM.
   02  SSA001-BEGIN PICTURE X(19) VALUE 'STPIITMb(STQIINOb=b'.
   02  SSA001-STQIINO PICTURE X(6).
   02  SSA001-END PICTURE X VALUE ')'.
01  SSA002-GN-STCISUB PICTURE X(9) VALUE 'STCISUBbb.

01  IOAREA PICTURE X(256).
```

```
MOVE  INVENTORY-ITEM-NO TO SSA001-STQIINO.
CALL 'CBLTDLI' USING GHU-FUNC,PCB-NAME,IOAREA,SSA001-GU-STPIITM,
                SSA002-GN-STCISUB.
   The retrieved SUBSTITUTE ITEM segment can now be changed
   in the IOAREA by the program.
CALL 'CBLTDLI' USING REPL-FUNC,PCB-NAME,IOAREA.
```

```
STATUS CODES (after REPL call):
   bb:   segment is replaced with contents in IOAREA
other:   error situation
```

Figure 4-12. Basic REPL Call

## Deleting Segments

To delete the occurrence of a segment from a data base, the segment must first be obtained by issuing a get hold call (GHU, GHN, or GHNP) through DL/I. Once the segment has been acquired, the DLET call may be issued.

No DL/I call that uses the same PCB must intervene between the get hold call and the DLET call, or the DLET call is rejected. Quite often a program may want to process a segment prior to deleting it. This is permitted as long as the processing does not involve a DL/I call that refers to the same data base PCB used for the get hold/delete calls. However, other PCBs may be referred to between the get hold and delete calls.

DL/I is advised that a segment is to be deleted when the user issues a call that has the function DLET. The deletion of a parent, in effect, deletes all the segment occurrences beneath that parent, whether or not the application program is sensitive to those segments. If the segment being deleted is a root segment, the whole data base record is deleted. The segment to be deleted must still be in the IOAREA of the delete call (with which no SSA is used), and its sequence field must not have been changed. Figure 4-13 gives an example of a DLET call.

```
77  GHU-FUNC PICTURE XXXX VALUE 'GHUb'.
77  DLET-FUNC PICTURE XXXX VALUE 'DLET'.
01  SSA001-GU-STPIITM.
   02  SSA001-BEGIN PICTURE X(19) VALUE 'STPIITMb(STQIINOb=b'.
   02  SSA001-STQIINO PICTURE X(6).
   02  SSA001-END PICTURE X VALUE ')'.
01  SSA002-GN-STCISUB PICTURE X(9) VALUE 'STCISUBbb'.

01  IOAREA PICTURE X(256).
```

```
CALL 'CBLTDLI' USING GHU-FUNC,PCB-NAME,IOAREA,SSA001-GU-STPIITM,
                SSA002-GN-STCISUB.
   The retrieved SUBSTITUTE ITEM segment can now be processed in
   the IOAREA by the program
CALL 'CBLTDLI' USING DLET-FUNC,PCB-NAME,IOAREA.
```

```
STATUS CODES (after DLET call):
   bb:   requested SUBSTITUTE ITEM segment is deleted from the
         data base; all its dependents, if any, are also
         deleted.
other:   error situation
```

Figure 4-13. Basic DLET Call

## Inserting Segments

Adding new segment occurrences to a data base is done with the insert call, function code 'ISRT'.

The DL/I insert call is used for two distinct purposes:

- Load the segments during creation of a data base, and

- Add new occurrences of an existing segment type into an established data base.

The processing options field in the PCB indicates whether the data base is being added to or loaded. The format of the insert call is identical for either use.

When loading or inserting, the last SSA must specify only the name of the segment being inserted. It should specify only the segment name, not the sequence field. Thus an unqualified SSA is always required.

Up to the level to be inserted, the SSA evaluation and positioning for an insert call is exactly the same as for a GU call. For the level to be inserted, the value of the sequence field in the segment in the user I/O area is used to establish the insert position. If no sequence field is identified, then the segment is inserted (assuming RULES=LAST) at the end of the physical twin chain. If multiple non-unique keys are allowed, then the segment is inserted after existing segments with the same key value.

Figure 4-14 shows an example of an ISRT call. The status codes in this example apply only to inserts after the data base has been loaded. The status codes at initial load time are discussed under the topic "Loading A Basic Data Base" later in this chapter.

Note: There is no need to check the existence of a segment in the data base with a preceding retrieve call. DL/I will do that at insert time, and will notify you with an II or GE status code. Checking previous existence is only relevant if the segment has no sequence field. However, if your application typically expects a segment to be present in the data base, then you should check for its existence first. If typically the segment does not exist, then insert first.

```
77  ISRT-FUNC PICTURE XXXX VALUE 'ISRT'.
01  SSA00-GU-STPIITM.
  02  SSA001-BEGIN PICTURE X(19) VALUE 'STPIITMb(STQIINOb=b'.
  02  SSA001-STQIINO PICTURE X(6).
  02  SSA001-END PICTURE X VALUE ')'.
01  SSA002-GN-STCISUB PICTURE X(9) VALUE 'STCISUBbb'.

01  IOAREA PICTURE X(256).
```

```
MOVE INVENTORY-ITEM-NO TO SSA001-STQIINO.

MOVE SUBSTITUTE-ITEM TO IOAREA.

CALL 'CBLTDLI' USING ISRT-FUNC,PCB-NAME,IOAREA,SSA001-GU-STPIITM,
                SSA002-GN-STCISUB.
```

```
STATUS CODES:
  bb:   new SUBSTITUTE ITEM segment is inserted in data base
  II:   segment to insert already exists in data base
  GE:   segment not found; the requested inventory item number
        (i.e., the parent of the segment to be inserted) is not
        in the data base.
other:  error condition
```

Figure 4-14. Basic ISRT Call

## Calls With Command Codes

Both unqualified and qualified SSAs may contain one or more optional command codes which specify functional variations applicable to either the call function or the segment qualification. Command codes in an SSA are always prefixed by an asterisk (*), which immediately follows the 8-byte segment name. Figure 4-15 illustrates this. Following are some important command codes.

### D Command Code

The 'D' command code is the one most widely used. It requests DL/I to issue *path calls*. A path call enables a hierarchical path of segments to be inserted or retrieved with one call. (A "path" was defined earlier as the hierarchical sequence of segments, one per level, leading from a segment at one level to a particular segment at a lower level). The meaning of the 'D' command code is as follows:

- For retrieval calls, multiple segments in a hierarchical path will be moved to the I/O area with a single call. This type of call will subsequently be referred to as a *path call*. The first through the last segment retrieved are concatenated in the user's I/O area. Intermediate SSAs may be present with or without the 'D' command code. If without, these segments are not moved to the user's I/O area. The segment named in the PCB *segment name feedback area* is the lowest-level segment retrieved, or the last level satisfied in the call in case of a not-found condition. Higher-level segments associated with SSAs having the 'D' command code will have been placed in the user's I/O area even in the not-found case. The 'D' is not necessary for the last SSA in the call, because the segment that satisfies the last level is always moved to the user's I/O area. A processing option of 'P' must be specified in the PSBGEN for any segment type for which a command code of 'D' is used.

- For insert calls, the 'D' command code designates the first segment type in the path to be inserted. The SSAs for lower-level segments in the path need not have the 'D' command code set, that is, the 'D' command code is propagated to all specified lower-level segments.

Figure 4-15 shows an example of a path call.

```
77   GU-FUNC PICTURE XXXX VALUE 'GUbb'.
01   SSA004-GUD-STPIITM.
  02   SSA004-BEGIN PICTURE X(21) VALUE 'STPIITMb*D(STQIINOb=b'.
  02   SSA004-STQIINO PICTURE X(6).
  02   SSA004-END PICTURE X VALUE ')'.
01   SSA005-GN-STSILOC PICTURE X(9) VALUE 'STSILOCbb'.
01   IOAREA PICTURE X(256).
```
```
CALL 'CBLTDLI' USING GU-FUNC,PCB-NAME,IOAREA,
               SSA004-GUD-STPIITM,SSA005-GN-STSILOC.
```
```
STATUS CODES:
    bb:   both segments (INVENTORY ITEM and ITEM LOCATION)
            have been placed in IOAREA
   GE:   segment not found; INVENTORY ITEM segment may be retrieved
            in IOAREA; check segment name and level indicator in PCB.
other:   error condition
```

Figure 4-15. Sample Path Retrieve Call

The correct use of the path call can provide a significant performance advantage. You should use it whenever possible, even if the chance of the existence or the need for the dependent segment(s) is relatively small. If, for instance, you would need, in 10% or more of the occurrences, the first dependent segment after you inspect the parent, then it is generally advantageous to use a path call to retrieve them both initially.

### N Command Code

When a replace call follows a path retrieve call, it is assumed that all segments previously retrieved with the path call are being replaced. If any of the segments have not been changed, and, therefore, need not be replaced, the 'N' command code may be set at those levels, telling DL/I not to replace the segment at this level of the path. The status codes returned are the same as for a regular replace call.

### F Command Code

This command code allows you to back up to the first occurrence of a segment under its parent. It has meaning only for a get next call. A get unique call always starts with the first occurrence. Command code 'F' is disregarded for the root segment.

### L Command Code

This command code allows you to retrieve the last occurrence of the segment type that satisfies the qualification statement; or, if unqualified, to retrieve the last occurrence of this segment type under its parent. If this command code is used at the root level, it is disregarded. When used with ISRT calls, the command code applies only to segments with a nonunique sequence field and with RULES=(,FIRST) or RULES=(,HERE), in which case the rule is overridden.

### Q Command Code

The 'Q' command code causes DL/1 to lock the segment(s) returned by the call to prevent modification by another task.

It provides a facility which permits segments to be enqueued (locked) when the application needs to examine a number of segments and at the same time, prevent any of them from being modified while the others are being examined. The application can obtain the segments using the 'Q' command code and then retrieve them again with the assurance that none of them can be modified until the application terminates or issues a checkpoint.

To provide IMS compatibility, the 'Q' command code must be followed by the character 'A'.

Note: By definition, the 'Q' command is always followed by a one-byte field. Therefore, the second byte after the 'Q' must contain another command code, a left paren, or a blank.

The 'Q' command code will be ignored by DL/1 unless the segment for which it was specified is actually returned to the user (that is, used with *D or with the lowest level SSA).

## Data Base Positioning After a DL/I Call

As stated before, the data base position is used by DL/I to satisfy the next call against the PCB. The segment level, segment name, and the key feedback areas of the PCB are used to present the data base position to the application program.

The following basic rules apply:

- If a get call is completely satisfied, current position in the data base is reflected in the PCB key feedback area.

- A replace call does not change current position in the data base.

- Data base position after a successful insert call is immediately after the inserted segment.

- Data base position after return of an II status code is immediately prior to the duplicate segment. This positioning allows the duplicate segment to be retrieved with a GN call.

- Data base position after a successful delete call is immediately after all dependents of the deleted segment. If no dependents existed, data base position is immediately after the deleted segment.

- Data base position is unchanged by an unsuccessful delete call.

- After a (partial) unsuccessful retrieve call, the PCB reflects the lowest level segment which satisfied the call. The segment name or the key feed back length should be used to determine the length of the relevant data in the key feedback area. Contents of the key feedback area beyond the length value must not be used, because the feedback area is never cleared after previous calls. If the level-one (root) SSA cannot be satisfied, the segment name is cleared to blank, and the level and key feedback length are set to 0.

In considering 'current position in the data base', remember that DL/1 must first establish a starting position to be used in satisfying the call. This starting position is the current position in the data base for get next calls, and is a unique position normally established by the root SSA for get unique calls.

The following are clarifications of 'current position in the data base' for special situations:

- If no current position exists in the data base, then the assumed current position is the start of the data base.

- If the end of the data base is encountered, then the assumed current position to be used by the next call is the start of the data base.

- If a get unique call is unsatisfied at the root level, then the current position is such that the next segment retrieved is the first root segment with a key value higher than the one specified for the unsuccessful call. Two exception are: 1. When the end of the data base is reached, and 2. For HDAM, where it is the next segment in physical sequence.

You can always reestablish your data base positioning with a GU call specifying all the segment key values in the hierarchical path. It is highly recommended that you use a get unique call after each not found condition.

## Using Multiple PCBs For One Data Base

Whenever there is a need to maintain two or more independent positions in one data base, you should use different PCBs. This avoids the reissue of get unique calls to switch forward and backward from one data base record or hierarchical path to another. There are no restrictions as to the call functions available in these multiple PCBs. However, to avoid *position confusion* in the application program, you should not apply changes via two PCBs to the same hierarchical path. For simplicity reasons, it is best to limit the updates to one PCB unless this would cause additional calls.

## COBOL Batch Program Structure

Figure 4-16 illustrates in outline form the fundamental parts in the structure of a COBOL batch program which, in this example is to retrieve data from a detail file to update a master data base. The following explanation relates to the reference numbers along the left side of the figure.

1. A 77 level or 01 level working storage entry defines each of the call functions used by the batch program. Each picture clause is defined as 4 alphameric characters and has a value assigned for each function (for example, 'GUbb'). If the optional count field were to be included in the call statement, count values could be initialized for each type of call. The COBOL copy function could be used to include these standard descriptions into the program.

2. A 9-byte area is set up to be used in the calls that require an unqualified SSA. Before the call is issued, a segment name is moved into this field. If a call requires 2 or more unqualified SSAs, additional areas may be required.

3. An 01 level working storage entry defines each SSA used by an application program.

   A separate SSA structure is required for each segment type accessed by the program because the key-value fields should be different. Once the fields other than key-value are initialized, they need not be altered.

4. A 01 level working storage entry defines the program segment I/O area. This area can be further defined with 02 entries. Separate I/O areas may be allocated for each segment type prescribed, or a single area can be used.

5. A 01 level Linkage Section entry describes the data base PCB entry for every input or output data base. It is through this linkage that a COBOL program may access the status codes after a DL/I call. The individual fields in the PCB are defined in the linkage section so that they may be referenced in the program.

6. This is the standard entry point in the procedure division of a batch program. After DL/I control has loaded the PSB for the program in the batch partition, it gives control to the application program. The PSB contains all the PCBs used by the program. The USING statement at the entry point to the batch program must contain the same number of names in the same sequence as there are PCBs in the PSB.

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

   Before issuing the call, the key value of the SSA must be initialized to specify the particular segment to be retrieved. Immediately following the call a test should be made of the status-code field of the PCB to determine if the call was successful.

8. This is a typical call to retrieve data from a data base using no SSA. This call is also a hold call for a subsequent delete or replace operation.

9. This statement replaces data in the data base with data from a COBOL batch program.

10. The GOBACK statement causes the batch program to return control to DL/I.

11. A language interface module (DLZLI000), which must be link-edited to the batch program after compilation, provides a common interface to DL/I. The call statement causes a V-type address constant (CBLTDLI) to be generated for the language interface module. When the application program is link-edited, the DOS/VS automatic library look-up (AUTOLINK) feature retrieves the language interface module from a DOS/VS relocatable library (system or private) and link-edits it with the application program. If AUTOLINK is suppressed, an INCLUDE statement must be present for the language interface module.

   Note: The user must include the following additional statements in the input to the linkage editor:

   ```
   INCLUDE   DLZBPJRA
   ENTRY     CBLCALLA
   ```

```
REF.
NO.        ENVIRONMENT DIVISION.
               .
               .
           DATA DIVISION.
           WORKING-STORAGE SECTION.
               77 FUNC-GU             PICTURE XXXX VALUE 'GU  '.
               77 FUNC-GHU            PICTURE XXXX VALUE 'GHU '.
               77 FUNC-GN             PICTURE XXXX VALUE 'GN  '.
1              77 FUNC-GHN            PICUTRE XXXX VALUE 'GHN '.
               77 FUNC-GNP            PICTURE XXXX VALUE 'GNP '.
               77 FUNC-GHNP           PICTURE XXXX VALUE 'GHNP'.
               77 FUNC-REPL           PICTURE XXXX VALUE 'REPL'.
               77 FUNC-ISRT           PICTURE XXXX VALUE 'ISRT'.
               77 FUNC-DLET           PICTURE XXXX VALUE 'DLET'.
               77 COUNT               PICTURE S9(5)VALUE +4 COMPUTATIONAL.
2              01 UNQUAL-SSA.
                  02 SEG-NAME         PICTURE X(08)VALUE '           '.
                  02 FILLER           PICTURE X    VALUE ' '.
3              01 QUAL-SSA-MAST.
                  02 SEG-NAME-M       PICTURE X(08)VALUE 'ROOT     '.
                  02 BEGIN-PAREW-M    PICTURE X    VALUE '('.
                  02 KEY-NAME-M       PICTURE X(08)VALUE 'KEY      '.
                  02 REL-OPER -M      PICTURE X(02)VALUE ' ='.
                  02 KEY-VALUE-M      PICTURE X(06)VALUE 'vvvvvv'.
                  02 END-PAREN-M      PICTURE X    VALUE ')'.
               01 QUAL-SSA-DET
                  02 SEG-NAME-D       PICTURE X(08)VALUE 'ROOT     '.
                  02 BEGIN-PAREN-D    PICTURE X    VALUE '('.
                  02 KEY-NAME-D       PICTURE X(08)VALUE 'KEY      '.
                  02 REL-OPER-D       PICTURE X(02)VALUE ' ='.
                  02 KEY-VALUE-D      PICTURE X(06)VALUE 'vvvvvv'.
                  02 END-PAREN-D      PICTURE X    VALUE ')'.
4              01 DET-SEG-IN.
                  02 --
                  02 --
               01 MAST-SEG-IN.
                  02 --
                  02 --
```

Figure 4-16. General COBOL Batch Program Structure (1 of 2)

```
5        LINKAGE SECTION.
              01 DB-PCB-MAST.
                 02 MAST-DBD-NAME   PICTURE X(8).
                 02 MAST-SEG-LEVEL  PICTURE XX.
                 02 MAST-STAT-CODE  PICTURE XX.
                 02 MAST-PROC-OPT   PICTURE XXXX.
                 02 FILLER          PICTURE S9(5) COMPUTATIONAL.
                 02 MAST-SEG-NAME   PICTURE X(8).
                 02 MAST-LEN-KFB    PICTURE S9(5) COMPUTATIONAL.
                 02 MAST-NU-SENSEG  PICTURE S9(5) COMPUTATIONAL.
                 02 MAST-KEY-FB     PICTURE X---X.
              01 DB-PCB-DETAIL.
                 02 DET-DBD-NAME    PICTURE X(8).
                 02 DET-SEG-LEVEL   PICTURE XX.
                 02 DET-STAT-CODE   PICTURE XX.
                 02 DET-PROC-OPT    PICTURE XXXX.
                 02 FILLER          PICTURE S9(5) COMPUTATIONAL.
                 02 DET-SEG-NAME    PICTURE X(8).
                 02 DET-LEN-KFB     PICTURE S9(5) COMPUTATIONAL.
                 02 DET-NU--SENSEG  PICTURE S9(5) COMPUTATIONAL.
                 02 DET-KEY-FB      PICTURE X---X.
         PROCEDURE DIVISION.
6             ENTRY 'DLITCBL' USING DB-PCB-MAST, DB-PCB-DETAIL.
                 .
                 .
7             CALL 'CBLTDLI' USING FUNC-GU, DB-PCB-DETAIL,
                    DET-SEG-IN, QUAL-SSA-DET.
                 .
                 .
              CALL 'CBLTDLI' USING COUNT, FUNC-GHU, DB-PCB-MAST,
                    MAST-SEG-IN, QUAL-SSA-MAST.
                 .
                 .
8             CALL 'CBLTDLI' USING FUNC-GHU, DB-PCB-MAST,
                    MAST-SEG-IN.
                 .
                 .
9             CALL 'CBLTDLI' USING FUNC-REPL, DB-PCB-MAST,
                    MAST-SEG-IN.
                 .
                 .
10            GOBACK.
11            COBOL LANGUAGE INTERFACE
```

Figure 4-16. General COBOL Batch Program Structure (2 of 2)

# PL/I Batch Program Structure

Figure 4-17 illustrates in outline form the fundamental parts in the structure of a PL/I batch program which, in this example, is to retrieve data from a detail file to update a master data base. The following explanation relates to the reference numbers along the left side of the figure.

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

2.  Each area defines one of the call functions used by the PL/I batch program. Each character string is defined as 4 alphameric characters, with a value assigned for each function (for example, 'GU '). Other constants may be defined in same the manner. Standard definitions could be stored in a source library and included using a %INCLUDE statement.

```
REF.
NO.        /*                                                    */
           /*                   ENTRY POINT                      */
           /*                                                    */
1              DLITPLI: PROCEDURE (DB_PTR_MAST_,DB_PTR_DETAIL
                   OPTIONS (MAIN);
           /*                                                    */
           /*                DESCRIPTIVE STATEMENTS              */
           /*                                                    */
           DCL  DB_PTR_MAST POINTER;
           DCL  DB_PTR_DETAIL POINTER;
           DCL  FUNC_GU      CHAR(4)     INIT('GU  ');
           DCL  FUNC_GN      CHAR(4)     INIT('GN  ');
           DCL  FUNC_GHU     CHAR(4)     INIT('GHU ');
           DCL  FUNC_GHN     CHAR(4)     INIT('GHN ');
           DCL  FUNC_GNP     CHAR(4)     INIT('GNP ');
           DCL  FUNC_GHNP    CHAR(4)     INIT('GHNP');
           DCL  FUNC_ISRT    CHAR(4)     INIT('ISRT');
           DCL  FUNC_REPL    CHAR(4)     INIT('REPL');
           DCL  FUNC_DLET    CHAR(4)     INIT('DLET');
           DCL 1    QUAL_SSA             STATIC UNALIGNED,
3                2    SEG_NAME           CHAR(8) INIT('ROOT    '),
                 2    SEG_QUAL           CHAR(1) INIT('('),
                 2    SEG_KEY_NAME       CHAR(8) INIT('KEY     '),
                 2    SEG_OPR            CHAR(2) INIT(' ='),
                 2    SEG_KEY_VALUE      CHAR(6) INIT('vvvvvv'),
                 2    SEG_END_CHAR       CHAR(1) INIT(')');
           DCL 1    UNQUAL_SSA           STATIC UNALIGNED,
                 2    SEG_NAME_U         CHAR(8) INIT('NAME    '),
                 2    BLANK              CHAR(1) INIT(' ');
           DCL 1    MAST_SEG_IO_AREA,
4                2 ---
                 2 ---
                 2 ---
           DCL 1    DET_SEG_IO_AREA,
                 2 ---
                 2 ---
                 2 ---
```

Figure 4-17. General PL/I Batch Program Structure (1 of 2)

3. A structure declaration defines each SSA used by the problem program. The unaligned attribute is required for SSA data interchange with DL/I. The SSA character string must reside contiguously in storage. Assignment of variables to key values, for example, could result in the construction of an invalid SSA if the key value has the aligned attribute.

A separate SSA structure is required for each segment type accessed by the program because the key-value fields should be different. Once the fields other than key-value are initialized, they should not have to be altered.

A 9-byte area should be reserved for use as an unqualified SSA. Before issuing an unqualified call, a segment name is moved into this field.

4. The segment I/O areas are defined as structures.

5. One level 1 declarative (similar to COBOL's linkage section) describes as a structure the data base PCB entry for each input or output data base. It is

through this description that a PL/I program may access the status codes after a DL/I call.

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

7. These are typical calls to retrieve data from a data base using a qualified SSA.

Prior to execution of the call the SEG_KEY_VALUE field of the SSA must be initialized if a fully qualified SSA is required. For a call using an unqualified SSA, the segment name field must be moved to one of the 9-byte UNQUAL_SSA areas.

Immediately following the call the status code field of the PCB must be checked to determine the results of the call.

8. This is a typical call to retrieve data from a data base using no SSA. This call is also a HOLD call for subsequent delete or replace operation.

9. This call is used to replace data in the data base with data from a PL/I batch program.

10. This RETURN statement causes the batch program to return control to DL/I.

11. A language interface module (DLZLI000), which must be link-edited to the batch program, provides a common interface to DL/I. The call statement causes a V-type address constant (PLITDLI) to be generated for the language interface mo-

dule. When the application program is link-edited, the DOS/VS automatic library look-up (AUTOLINK) feature retrieves the language interface module from a DOS/VS relocatable library (system or private) and link-edits it with the application program. If AUTOLINK is suppressed, an INCLUDE statement must be present for the language interface module.

Note: The user must include the following additional statements in the input to the linkage editor:

```
INCLUDE      IBMBPJRA
ENTRY        PLICALLB
```

```
         DCL  1    DB_PCB_MAST            BASED (DB_PTR_MAST),
                   2   MAST_DB_NAME       CHAR(8),
                   2   MAST_SEG_LEVEL     CHAR(2),
                   2   MAST_STAT_CODE     CHAR(2),
5                  2   MAST_PROC_OPT      CHAR(4),
                   2   FILLER             FIXED BINARY (31,0),
                   2   MAST_SEG_NAME      CHAR(8),
                   2   MAST_LEN_KFB       FIXED BINARY (31,0),
                   2   MAST_NO_SENSEG     FIXED BINARY (31,0),
                   2   MAST_KEY_FB        CHAR(X);
         DCL       DB_PCB_DETAIL          BASE (DB_PTR_DETAIL),
                   2   DET_DB_NAME        CHAR(8),
                   2   DET_SEG_LEVEL      CHAR(2),
                   2   DET_STAT_CODE      CHAR(2),
                   2   DET_PROC_OPT       CHAR(4),
                   2   FILLER             FIXED BINARY (31,0),
                   2   DET_SEG_NAME       CHAR(8),
                   2   DET_LEN_KFB        FIXED BINARY (31,0),
                   2   DET_NU_SENSEG      FIXED BINARY (31,0),
                   2   DET_KEY_FB         CHAR(X);
         DCL       THREE   FIXED BINARY      (31.0)   INITIAL(3);
         DCL       FOUR    FIXED BINARY      (31.0)   INITIAL(4);
         DCL       FIVE    FIXED BINARY      (31.0)   INITIAL(5);
6        DCL       SIX     FIXED BINARY      (31.0)   INITIAL(6);
         /*                                       `              */
         /*    MAIN PART OF PL/I BATCH PROGRAM                   */

7            CALL PLITDLI(FOUR,FUNC_GU,DB_PCB_DETAIL,
             DET_SEG_IO_AREA,QUAL_SSA);

             CALL PLITDLI(FOUR,FUNC_GHU,DB_PCB_MAST,
             MAST_SEG_IO_AREA,QUAL_SSA);

8            CALL PLITDLI(THREE,FUNC_GHN,DB_PCB_MAST,
             MAST_SEG_IO_AREA

9            CALL PLITDLI(THREE,FUNC_REPL,DB_PCB_MAST,
             MAST_SEG_IO_AREA);

10           RETURN;
             END DLITPLI;
11           PL/I LANGUAGE INTERFACE
```

Figure 4-17. General PL/I Batch Program Structure (2 of 2)

# RPG II Batch Program Structure

Figure 4-18 illustrates in outline form the fundamental
parts in the structure of an RPG II batch program
which, in this example, is to retrieve data from a detail
file to update a master data base. The following expla-
nation relates to the reference numbers along the right
side of the figure.



Figure 4-18. General RPG II Batch Program Structure (Part 1 of 6)

GX21-9094-3 U/M050*
Printed in U.S.A.

IBM International Business Machines Corporation

| Program | | Punching Instruction | Graphic | | | | | Card Electro Number | | Page |0|2| of __ | Program Identification | 75 76 77 78 79 80 |D|L|I|R|P|G| |
| Programmer | | Data | Punch | | | | | | | | | | | |

**I — Input Specifications**

| Line | Form Type | Filename | O A | Number (1/N) | Option (O), U | Record Identifying Indicator ** or DS | Position | Not (N) | C/Z/D | Character | Position | Not (N) | C/Z/D | Character | Position | Not (N) | C/Z/D | Character | P/B/L/R | From | To | Field Name | | Matching Fields or Chaining Fields | | Plus | Zero or Blank | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 1 | I | MASTDB | | | | | | | | | | | | | | | | | | | | | | | | | ④ |
| 0 2 | I | * RECORD LAYOT | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 3 | I | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 4 | I | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 5 | I | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 6 | I | DETAR | | | DS | | | | | | | | | | | | | | | | | | | | | | ⑤ |
| 0 7 | I | * IOAREA FOR DETAIL DB | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 8 | I | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 9 | I | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 0 | I | PCBDET | | | DS | | | | | | | | | | | | | | | | | | | | | | ⑥ |
| 1 1 | I | | | | | | | | | | | | | | | | | | 1 | 8 | DBNAM | | | | | | |
| 1 2 | I | | | | | | | | | | | | | | | | | | 9 | 10 | SEGLNG | | | | | | |
| 1 3 | I | | | | | | | | | | | | | | | | | | 11 | 12 | STATCD | | | | | | |
| 1 4 | I | | | | | | | | | | | | | | | | | | 13 | 16 | PROCOP | | | | | | |
| 1 5 | I | | | | | | | | | | | | | | | | | B | 17 | 200 | FILLER | | | | | | |
| 1 6 | I | | | | | | | | | | | | | | | | | | | | | | | | | | |

Figure 4-18. General RPG II Batch Program Structure (Part 2 of 6)

| Program | | Punching Instruction | Graphic | | | | | Card Electro Number | | Page |0|3| of __ | Program Identification | 75 76 77 78 79 80 |D|L|I|R|P|G| |
| Programmer | | Data | Punch | | | | | | | | | | | |

**I — Input Specifications**

| Line | Form Type | Filename | O A | Number (1/N) | Option (O), U | Record Identifying Indicator ** or DS | Position | Not (N) | C/Z/D | Character | Position | Not (N) | C/Z/D | Character | Position | Not (N) | C/Z/D | Character | P/B/L/R | From | To | Field Name | | Matching Fields or Chaining Fields | | Plus | Zero or Blank | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 1 | I | | | | | | | | | | | | | | | | | | 21 | 28 | SEGNAM | | | | | | |
| 0 2 | I | | | | | | | | | | | | | | | | | B | 29 | 32 | KFBLEN | | | | | | |
| 0 3 | I | | | | | | | | | | | | | | | | | B | 33 | 36 | SENSSN | | | | | | |
| 0 4 | I | | | | | | | | | | | | | | | | | | 37 | | KEYFB | | | | | | |
| 0 5 | I | QULSSA | | | DS | | | | | | | | | | | | | | | | | | | | | | ⑦ |
| 0 6 | I | * DEFINES QUALIFIED SSA | | | | | | | | | | | | | | | | | 1 | 8 | SGNAME | | | | | | |
| 0 7 | I | | | | | | | | | | | | | | | | | | 9 | 9 | SQUAL | | | | | | |
| 0 8 | I | | | | | | | | | | | | | | | | | | 10 | 17 | KEYNAM | | | | | | |
| 0 9 | I | | | | | | | | | | | | | | | | | | 18 | 19 | SOPR | | | | | | |
| 1 0 | I | | | | | | | | | | | | | | | | | | 20 | 25 | SKEYVA | | | | | | |
| 1 1 | I | | | | | | | | | | | | | | | | | | 26 | 26 | SENDCH | | | | | | |
| 1 2 | I | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 3 | I | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 4 | I | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 5 | I | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 6 | I | | | | | | | | | | | | | | | | | | | | | | | | | | |

Figure 4-18. General RPG II Batch Program Structure (Part 3 of 6)

GX21-9093-2 UM/050* Printed in U.S.A.
*No. of forms per pad may vary slightly

IBM International Business Machine Corporation

Program | Punching Instruction | Graphic | | Card Electro Number | Page 0 4 of __ | Program Identification 75 76 77 78 79 80 D L I R P G
Programmer | Date | Punch

| Line | Form Type | Control Level | Indicators | Factor 1 | Operation | Factor 2 | Result Field Name | Length | Decimal Positions | Resulting Indicators | Comments | |
|------|-----------|---------------|------------|----------|-----------|----------|-------------------|--------|-------------------|---------------------|----------|--|
| 0 1 | C | | | *ENTRY | PLIST | | | | | | | ⑧ |
| 0 2 | C | | | | PARM | | PCBD1 | | | | | |
| 0 3 | C | | | | PARM | | PCBDET | | | | | |
| 0 4 | C | | | | | | | | | | | |
| 0 5 | C | | | | | | | | | | | |
| 0 6 | C | | | | | | | | | | | |
| 0 7 | C | | | | MOVE | 'ROOT | SGNAME | | | | | ⑨ |
| 0 8 | C | | | | MOVE | '(' | SQUAL | | | | | |
| 0 9 | C | | | | MOVE | 'KEY | KEYNAM | | | | | |
| 1 0 | C | | | | MOVE | '-' | SOPR | | | | | |
| 1 1 | C | | | | MOVE | 'VVVVVV' | SKEYVA | | | | | |
| 1 2 | C | | | | MOVE | ')' | SENDCM | | | | | |
| 1 3 | C | | | GU | RQDLI | | | | | 1 3 | | ⑩ |
| 1 4 | C | | | PCB | ELEM | | PCBDET | | | | | |
| 1 5 | C | | | INTO | ELEM | | DETAR | | | | | |
| 1 6 | C | | | 3SA | ELEM | | QULSSA | | | | | |
| 1 7 | C | | | | | | | | | | | |
| 1 8 | C | | | | | | | | | | | |
| 1 9 | C | | | | | | | | | | | |
| 2 0 | C | | | | MOVE | 'XXXXXX' | KEYVAR | 8 | | | | ⑪ |

Figure 4-18. General RPG II Batch Program Structure (Part 4 of 6)

GX21-9093-2 UM/050* Printed in U.S.A.
*No. of forms per pad may vary slightly

IBM International Business Machine Corporation

Program | Punching Instruction | Graphic | | Card Electro Number | Page 0 5 of __ | Program Identification 75 76 77 78 79 80 D L I R P G
Programmer | Date | Punch

| Line | Form Type | Control Level | Indicators | Factor 1 | Operation | Factor 2 | Result Field Name | Length | Decimal Positions | Resulting Indicators | Comments | |
|------|-----------|---------------|------------|----------|-----------|----------|-------------------|--------|-------------------|---------------------|----------|--|
| 0 1 | C | | | GHU | RQDLI | MASTDB | | | | 1 3 | | ⑫ |
| 0 2 | C | | | ROOT | QSSA | KEY | KEYVAR | 8 | EQ | | | |
| 0 3 | C | | | | | | | | | | | |
| 0 4 | C | | | | | | | | | | | |
| 0 5 | C | | | GHN | RQDLI | MASTDB | | | | 1 3 | | ⑬ |
| 0 6 | C | | | | | | | | | | | |
| 0 7 | C | | | | | | | | | | | |
| 0 8 | C | | | REPL | RQDLI | MASTDB | | | | 1 3 | | ⑭ |
| 0 9 | C | | | | | | | | | | | |
| 1 0 | C | | | | | | | | | | | |
| 1 1 | C | | | | SETON | | | | | LR | | ⑮ |
| 1 2 | C | | | | | | | | | | | |

Figure 4-18. General RPG II Batch Program Structure (Part 5 of 6)

GX21-9090-3 UM/050*
Printed in U.S.A.

**IBM** International Business Machines Corporation

| Program | | | Punching Instruction | Graphic | | | | | Card Electro Number | | | Page 0 6 of __ | Program Identification | D L I R P G |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Programmer | | Date | | Punch | | | | | | | | | | |

| Commas | Zero Balances to Print | No Sign | CR | – | X – Remove Plus Sign |
|---|---|---|---|---|---|
| Yes | Yes | 1 | A | J | Y – Date Field Edit |
| Yes | No | 2 | B | K | |
| No | Yes | 3 | C | L | Z – Zero Suppress |
| No | No | 4 | D | M | |

| Line | Form Type | Filename | | Output Indicators | End Position in Output Record | Constant or Edit Word |
|---|---|---|---|---|---|---|
| 0 1 | O | MASTDB | | | | |
| 0 2 | O | *DEFINE INPUT RECORD FOR DB | | | | |
| 0 3 | O | | | | | |
| 0 4 | O | | | | | |
| 0 5 | O | | | | | |
| 0 6 | O | | | | | |
| 0 7 | O | | | | | |
| 0 8 | O | | | | | |
| 0 9 | O | | | | | |
| 1 0 | O | | | | | |
| 1 1 | O | | | | | |
| 1 2 | O | | | | | |
| 1 3 | O | | | | | |
| 1 4 | O | | | | | |
| 1 5 | O | | | | | |
| 1 6 | O | | | | | |
| 1 7 | O | | | | | |
| 1 8 | O | | | | | |
| 1 9 | O | | | | | |

(16)

Figure 4-18. General RPG II Batch Program Structure (Part 6 of 6)

0. The user may specify any program name.

1. Specify B in position 56 to tell the Translator that this is an RPG II program which will run under DL/I.

2. A data base may be defined as a DB-file. In this case the PCB can be specified in a continuation line (K in position 53) to the DB-file specification.

3. The continuation line defines the PCB associated with the DB-file. In this case the PCB will be automatically generated. The Translator will put the definition into the Input Specifications as a data structure.

4. This is the segment definition of the DB-file MASTDB.

5. This data structure provides an I/O area for use by RQDLI commands without referring to a File Description Specification.

6. This is an example of a user-defined PCB. It must have the same layout as the automatically generated PCB; however, the user may use names of his own choice.

7. This data structure can be used to build a qualified SSA.

8. Since not all data bases are defined as DB-files, the user must explicitly specify the *ENTRY PLIST. After the DL/I control program has loaded the PSB, it gives control to the RPG II program. The PSB contains all the PCBs used by the program. Therefore, the PARM statements of the *ENTRY PLIST must specify the PCBs in the same sequence they are specified in the PSB.

9. The following MOVE statements build a qualified SSA to be used in an SSA option of the RQDLI command. Reference numbers 10-12 show typical commands to retrieve data from a data base.

10. This RQDLI command explicitly uses the I/O area DETAR.

Immediately following the RQDLI command the status code field of the PCB must be checked by the user (by using compare operations, as illustrated in Figure 2-4, to determine the results of the RQDLI command.

11. This MOVE statement specifies that a new value for KEYVAR will be used in the following RQDLI command (reference number 12).

12. This RQDLI command does not use our I/O area. In this case the Translator takes the information and adds it to the records defined in the Input Specifications. This command is used to show

how it is possible to specify a qualified SSA in a QSSA statement.

13. This is a typical RQDLI command to retrieve data from a data base using no SSA. This RQDLI command is also a HOLD RQDLI command for subsequent delete or replace operations.

14. This RQDLI command is used to replace data in the data base with data from an RPG II batch program. It uses MASTDB as defined in the Output Specifications to define the I/O area.

15. SETON LR must be coded to return control to DL/I.

16. This is the segment definition for output of the MASTDB file.

17. A language interface module (DLZLI000) must be link-edited to the application program to provide a common interface to DL/I. When the application program is link-edited, the DOS/VS automatic library look-up (AUTOLINK) feature retrieves the language interface module from a DOS/VS relocatable library (system or private) and link-edits it with the application program. If AUTOLINK is suppressed, an INCLUDE statement must be present for the language interface module.

## Assembler Language Batch Program Structure

Figure 4-19 illustrates in outline form the fundamental parts in the structure of an Assembler Language batch program which, in this example, is to retrieve data from a detail file to update a master data base. The following explanation relates to the reference numbers along the left side of the figure.

```
        PGMSTART    CSECT
                    USING       *,R12
                    SAVE        (R14,R12)
1                   LR          R12,R15
                    ST          R13,SAVEAREA+4
                    LA          R13,SAVEAREA
                    .
2                   MVC         DBPCBMST(8),0(R1)
                    .
                    MVC         DLIFUNC,GU
                    MVC         PCB,DBPCBDET
                    LA          R1,DETSEGIO
                    ST          R1,IOAREA
                    LA          R1,SSANAME
                    ST          R1,SSA
                    LA          R1,PARMLIST
3                   CALL        ASMTDLI
                    .
                    MVC         DLIFUNC,GHU
                    MVC         PCB,DBPCBMST
                    LA          R1,MSTSEGIO
                    ST          R1,IOAREA
                    MVI         SSANAME+8,C'
                    LA          R1,PARMLIST
4                   CALL        ASMTDLI
                    .
                    MVC         DLIFUNC,GHN
                    MVI         PARMCT+3,3
                    LA          R1,PARMLIST
5                   CALL        ASMTDLI
                    .
                    MVC         DLIFUNC,REPL
                    LA          R1,PARMLIST
6                   CALL        ASMTDLI
                    .
                    L           R13,4(R13)
7                   RETURN      (R14,R12)
             *   CONSTANT AREA
                    .
8           PARMLIST    DC      A(PARMCT)
            FUNC        DC      A(DLIFUNC)
            PCB         DC      A(0)
            IOAREA      DC      A(0)
            SSA         DC      A(0)
                    .
```

Figure 4-19. General Assembler Language Batch Program Structure (1 of 2)

```
9      DBPCBMST    DC      F'0'
       DBPCBDET    DC      F'0'
                    .
       PARMCT      DC      F'4'
       DLIFUNC     DC      CL4'     '
       GU          DC      CL4'GU   '
10     GHU         DC      CL4'GHU  '
       GHN         DC      CL4'GHN  '
       REPL        DC      CL4'REPL'
                    .
       SSANAME     DS      0CL26
       ROOT        DC      CL8'ROOT      '
11                 DC      CL1'('
                   DC      CL8'KEY       '
                   DC      CL2' ='
       NAME        DC      CL6'vvvvvv'
                   DC      CL1')'
                    .
12     MSTSEGIO    DS      CL100
       DETSEGIO    DS      CL100
       SAVEAREA    DC      18F'0'
                    .
       PCBNAME     DSECT
       DBPCBDBD    DS      CL8     DBD NAME
       DBPCBLEV    DS      CL2     LEVEL FEEDBACK
       DBPCBSTC    DS      CL2     STATUS CODES
       DBPCBPRO    DS      CL4     PROC OPTIONS
       DBPCBRSV    DS      F       RESERVED
       DBPCBSFD    DS      CL8     SEGMENT NAME FEEDBACK
       DBPCBMKL    DS      F       CURRENT LENGTH OF KEY FEEDBACK
       *                          AREA
       DBPCBNSS    DS      F       NO OF SENSITIVE SEGMTS IN PCB
       DBPCBKFD    DS      CL6     KEY FEEDBACK AREA
                   END
13                 ASSEMBLER LANGUAGE INTERFACE
```

Figure 4-19. General Assembler Language Batch Program Structure (2 of 2)

1. The entry point to the Assembler Language Program. See the discussion under "Assembler" in the section "Entry to an Application Program" earlier in this chapter. The base register 12 is used in this example.

2. When control is passed to the application program, register 1 contains the address of a variable-length fullword parameter list. See the discussion under "Assembler" in the section "Entry to an Application Program" earlier in this chapter. As only explicit calls are issued in this example, there is no need to reset the 0 bit from 1 to 0.

3. This is a typical call to retrieve data from the detail data base using a qualified search argument. All DL/I calls should be executed with the CALL macro instruction. Prior to execution of the call statement register 1 must point to the variable-length fullword parameter list. This may be done through operands of the CALL macro instruction. If the user constructs his own parameter list, the leftmost bit of the last entry in the list

must be set to one unless a parameter count is specified as shown in this example. Immediately following the call, the status code of the PCB should be tested to determine the result of the call.

4. This call (and calls 5 and 6) are to the master data base and therefore the PCB address and I/O area must be reloaded. This call has an unqualified SSA by setting the left parentheses in position 9 to blank.

5. This is a typical call to retrieve data from a data base, which, by setting the parm-count to three, uses no SSA. This call is also a HOLD call for subsequent delete or replace.

6. This call is used to replace data in the detail data base.

7. The RETURN statement loads DL/I registers and causes the batch program to return control to DL/I.

8. In this illustration, one variable-length parameter list is defined to process all data base calls. The

list contains pointers to the parameter count, DL/I call function, PCB, I/O area, and SSA. The value in PARMCT determines the length of the parameter list. In this case, the count of four indicates that the following four addresses constitute the parameter list.

9. A fullword must be defined for every data base PCB. The Assembler language program may access the status codes after a DL/I call using the PCB base addresses.

10. The call functions are defined as 4-character constants.

11. The SSAs must be defined by the problem program.

12. An I/O area large enough to contain the largest segment of a data base must be provided. In Figure 4-19, it is assumed that the longest segment does not exceed 100 bytes. As previously mentioned, an 18-fullword register save area must be provided in the application program.

13. A language interface module (DLZLI000) must be link-edited to the batch program after assembly to provide a common interface to DL/I. The call statement causes a V-type address constant (CBLTDLI or ASMTDLI) to be generated for the language interface module. When the application program is link-edited, the DOS/VS automatic library look-up (AUTOLINK) feature retrieves the language interface module from a DOS/VS relocatable library (system or private) and link-edits it with the application program. If AUTOLINK is suppressed, an INCLUDE statement must be present for the language interface module.

## Restrictions

### *On COMREG Use*
Bytes 16 through 19 of the communication region are used by DL/I and therefore must not be used by the application program.

### *On Overlay Programs*
Overlay structures are not supported for application programs executed under DL/I. Although the COBOL SORT verb automatically produces an overlay structure the restriction does not apply if the job control statements used to compile and link-edit the program as shown below are used. (For COBOL programs that do not contain the SORT verb, the INCLUDE DLZBPJRA statement may, alternatively, be placed immediately before the entry statement.) The use of "PL/I-SORT" programs, using the sort program product, is not affect-

ed by this restriction provided that an overlay structure is not explicitly specified.

### *Set Exit Abnormal Linkage*
The DL/I user has the option, through the use of the user program switch indicator (UPSI), of permitting STXIT AB and STXIT PC linkage to pass control to DL/I prior to abnormal termination so that a controlled shutdown may occur. The DL/I system log and DL/I data base are closed and a storage dump is provided. However, non-DL/I files are not closed; this is a user responsibility.

If a COBOL application program is executing under DL/I control, any attempt by the application program to execute the COBOL debug function may cause unpredictable results. Therefore, no COBOL debug function (any COBOL option that makes use of a STXIT routine) should be used if DL/I STXIT is used. Refer to your COBOL publications for options that use STXIT linkages.

The High Level Language (HLL) Debugging facility makes the job of an application programmer writing in PL/I easier by allowing diagnostic information to be supplied by both PL/I and DL/I. When a program check is detected during application program execution, a STXIT PC routine will be given control if STXIT support has been requested of DL/I (UPSI bit 7 = 0 for batch, and always for MPS batch). This facility applies only to batch and MPS batch execution of DL/I.

## Job Control Statements
DL/I application programs cannot be processed in a compile-link-go environment. Programs must first be compiled and link-edited to a DOS/VS core image library and then executed as a separate job, as they run as a subprogram of the DL/I initialization program.

### *Compile and Link-Edit*

#### *COBOL*

```
// JOB COBSAMPL
// OPTION CATAL
   PHASE COBSAMPL,*
   INCLUDE DLZBPJRA
// EXEC FCOBOL
      •
      •
   SOURCE DECK
      •
      •
/*
   ENTRY CBLCALLA
// EXEC LNKEDT
/&
```

## PL/I

```
// JOB PLISAMPL
// OPTION CATAL
   PHASE PLISAMPL,*
// EXEC PLIOPT
     •
     •
   SOURCE DECK
     •
     •
/*
   INCLUDE IBMBPJRA
   ENTRY PLICALLB
// EXEC LNKEDT
/&
```

## Assembler

```
// JOB ASSSAMPL
// OPTION CATAL
   PHASE ASSSAMPL,*
// EXEC ASSEMBLY
     •
     •
   SOURCE DECK
     •
     •
/*
// EXEC LNKEDT
/&
```

## RPG II

The compilation of an RPG II program which is going
to run under DL/I is a batch operation with the follow-
ing steps:

**RPG II program**
(with B specified in Column 56
of Header Specifications)
↓
**Translator**
↓
**RPG II Compiler**
(Auto report optional)
↓
**Appl. prog.**

The RPG II program may optionally use Auto Re-
port, to include RPG II program pieces from libraries,
etc.

The function of the Translator is to accept as input a
source program, written in RPG II, in which DL/I re-
quests have been coded via RQDLI commands. The
Translator produces as output an equivalent source
program in which the DL/I requests have been translat-
ed into CALL statements together with READ and EXCPT
statements. At execution time the CALL statements
invoke DL/I, passing appropriate arguments.

The Translator is executed in a separate job step.
The job step sequence for compiling an application
program is thus translate-compile-link edit. The Trans-
lator requires a minimum of 96K bytes of virtual stor-
age. Its phase name is RPGIXLTR.

The Translator reads its input from SYSIPT, pro-
duces its output (the translated source program) on
SYSPCH, or optionally on SYS002 or SYS003, and writes
the source listing, error messages, etc. on SYSLIST.

The RPG II Translator provides a number of options.
They may be specified in the // OPTION Job Control
Statement.

The Translator options for RPG II in a DL/I batch
environment are:

```
LIST  |  NOLIST
DUMP  |  NODUMP
```

Defaults are according to SYSGEN.

LIST: A listing of the source program is printed on
SYSLST.
DUMP: When unrecoverable errors occur, a dump is
produced.
NODUMP: When unrecoverable errors occur, no dump
is produced.

## Translator Output

The DB-file descriptions are translated into File De-
scription Specifications for SPECIAL files.

An RQDLI command not specifying a file-name or
with an explicit FROM or INTO option is translated into
a CALL statement followed by PARM statements.

For standard data transfer (existing Input and/or
Output Specifications and no FROM or INTO option
explicitly specified) the RQDLI command is translated
into a CALL statement followed by a READ statement
for input, or an EXCPT statement for output.

Additionally, the Translator generates data struc-
tures and fields together with MOVE or Z-ADD state-
ments to build the proper SSAs from the USSA and QSSA
statements, which are then used in the PARM state-
ments of the generated CALL statement.

Note: When link-editing an RPG II batch program using standard
data transfer, an unresolved external reference message for DFHE11
will appear in the linkage editor output listing unless the entry exists
in the user's core image library. The reason for this is that, in the
case of standard data transfer, the RPG module contains entry points
DFHE11 and RPGDLI for both CICS/VS and the batch environ-
ment respectively.

If the leftmost two bits of the UPSI byte are 00 (either
the standard setting, or set by // UPSI 00), the Transla-
tor directs its output to SYSPCH.

**Example of JCL for output on SYSPCH:**

```
// JOB T
// EXEC RPGIXLTR (Translator)
     •
     •   Source to be translated
     •
/*
/&
```

If the leftmost two bits of the UPSI byte are 01, the
Translator directs its output to SYS002. This method

should be used if an RPG II compilation is to immediately follow the Translator run. The RPG II compiler will then read its input from SYS002 instead of SYSIPT.

*Example of JCL output on SYS002:*

```
// JOB TRPG
// UPSI 01
// EXEC RPGIXLTR
     •
     •   Source to be translated
     •
/*
// EXEC RPG II
/&
```

If the leftmost two bits of the UPSI byte are 10, the Translator directs its output to SYS003. This method should be used if an Auto Report compilation is to immediately follow the Translator run. Auto Report will then read its input from SYS003 instead of SYSIPT.

*Example of JCL for output on SYS003:*

```
// JOB TAR
// UPSI 10
// EXEC RPGIXLTR
     •
     •   Source to be translated
     •
/*
// EXEC RPGAUTO
/&
```

## Batch Application Program Execution

When a DL/I application program is executed, it actually runs under control of DL/I. The EXEC statement in the job stream names the DL/I initialization module rather than the application program name.

### Parameter Statement

The application program to be executed and the PSB it uses are identified in a parameter statement that follows the EXEC statement.

The format of the parameter statement, beginning in column 1, is as follows:

```
DLI,progname,psbname[,{buf}]
                     {1}
    [,HDBFR=({bufno}[,dbdname1,dbdname2,...]) ][,...]
            {32 }
    [,HSBFR=({indno},{ksdsbuf},[{esdsbuf}],[dbdname3) ][,...]
            {3 } {2 }   {2    }
    [,TRACE=modname][,ASLOG=YES][,LOG=({TAPE },{PAUSE   }))]
                                       {DISK1 }  {NOPAUSE }
                                       {DISK2 }
```

The parameters HDBFR, HSBFR, TRACE, ASLOG, and LOG as well as continuation statements, can be used if input is on SYSIPT. Continuation, if required, is indicated by a non-blank character in position 72 of the statement being continued. The keyword parameters, once started, can be stopped anywhere up to column 71, and then continued in the next statement. The continuation may start in any position from 1 to 71. The parameters can also be entered from SYSLOG, but with a limitation of 71 characters. Continuation is not permitted with SYSLOG entry.

progname
    specifies a one to eight alphameric character name of the application program or utility to be executed.

psbname
    specifies a one to seven alphameric character name of the PSB as indicated in the PSB generation and referenced by the application program.

buf
    specifies the number of data base subpools required for this execution which can be a numeric value 1 to 255; if omitted, 1 is assumed. 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 may be 512 or any multiple of 512 bytes up to 4096. A data base is assigned a subpool that contains buffers equal to or greater in size than the size of the data base control interval. More information on the DL/I data base buffer pool is contained in Chapter 7 of the *Utilities and Guide for the System Programmer.*

HDBFR
    describes one DL/I subpool (See Chapter 7 of the *Utilities and Guide for the System Programmer.*

    • bufno specifies the number of buffers to be allocated for this subpool and is a numeric value from 2 to 32. If omitted for a specific subpool, 32 is assumed. A specification ex-

ceeding 2 digits will cause an abnormal termination.

- dbdname1,dbdname2,... specifies the names of DBDs that are to be allocated to this subpool. If no dbdnames are specified, this subpool is used for DMBs not explicitly assigned; the parentheses around the number of buffers are still required. The DBD name used should be the physical DBD even though a logical DBD is being used. However, since a logical DBD has 2 or more physical DBDs, all physical DBDs should be specified that are to be allocated to a specific subpool.

HSBFR

defines VSAM buffer allocation for HISAM, SHISAM, and INDEX data bases. See Chapter 7 of the *Utilities and Guide for the System Programmer.*

- indno specifies the number of index buffers for a KSDS; if omitted, 3 is assumed. A specification of 1 or 2 digits is permitted. A specification exceeding 2 digits will cause an abnormal termination.

- ksdsbuf specifies the number of data buffers for a KSDS; if omitted, 2 is assumed. A specification of 1 or 2 digits is permitted. A specification exceeding 2 digits will cause an abnormal termination.

- esdsbuf specifies the number of data buffers for the ESDS (applies to HISAM only); if omitted, 2 is assumed. A specification of 1 or 2 digits is permitted. A specification exceeding 2 digits will cause an abnormal termination.

- dbdname3 is the name of the HISAM or INDEX DBD referenced by the application program.

TRACE

indicates that tracing is to be active during execution. See the *DL/I DOS/VS Diagnostic Guide* for details on tracing.

ASLOG=YES

specifies that asynchronous logging is to be used. See Chapter 5 of the *Utilities and Guide for the System Programmer.*

LOG

specifies the type of logging to be used.

- *TAPE* indicates the log records are to be written to a tape device. It is the default if the LOG parameter is omitted.

- *DISK1* indicates the log records are to be written on one disk extent with the filename DSKLOG1.

- *DISK2* indicates that the log records are to be written on two disk extents. If one disk extent becomes full, the extent is closed and the other extent is used. DSKLOG1 is used first; then DSKLOG2. If DKSLOG2 becomes full, logging will switch back to DSKLOG1 and continue to repeat the sequence.

- *PAUSE* indicates that before reusing the only disk extent (DISK1) or before switching to the next extent (DISK2), the operator is notified and the partition waits for the operator's reply. PAUSE is the default if the second option in the LOG parameter is omitted.

- *NOPAUSE* indicates that reusing a log extent or switching log extents is done without notifying the operator.

Note: The UPSI byte (bit 6=0) must be set to indicate that DL/I logging is required.

If anything other than the above parameters are specified, an error message is issued and the job is canceled.

In the online and/or MPS environments, DL/I disk logging is not supported. If program isolation is active, the user must select CICS/VS journalling services for writing log information.

## UPSI Byte Settings for Batch DL/I

Several execution-time functions can be controlled by the UPSI byte setup. The format of the UPSI statement is as follows:

```
// UPSI     x0000xxx
```

The meanings of the bit settings are as follows:

| Bit 0 | = 0 | Read parameter information via SYSIPT |
| | = 1 | Read parameter information via SYSLOG |
| Bits 1-4 | | Available for use by the application programmer |
| Bit 5 | = 0 | Storage dump on set exit (STXIT) abnormal task termination if STXIT active (that is, bit 7 = 0). |
| | = 1 | No storage dump in STXIT abnormal task termination if STXIT active (that is, bit 7 = 0). |
| Bit 6 | = 0 | All data base modifications written on to the DL/I system log tape. |
| | = 1 | DL/I system log function inactive |
| Bit 7 | = 0 | Set exit (STXIT) linkage to DL/I for abnormal task termination. |
| | = 1 | STXIT inactive |

Note that UPSI byte settings in the online environment have different meanings than those for batch. See Chapter 5 of this manual, and Chapter 10, "DL/I Online System" in the *Utilities and Guide for the System Programmer* for more information. If you are unsure of the significance of these functions consult your system programmer or data base administrator.

### Job Control Statements

If data base changes are to be logged, either disk (batch only) or tape logging must be specified on the DL/I parameter statement with the LOG parameter. If the LOG parameter is omitted and UPSI byte bit 6=0, the default is tape logging.

If tape logging is used, ASSGN and TLBL statements as shown below are required. The log tape must have a standard label.

```
// ASSGN   SYS011,X'cuu'
// TLBL    LOGOUT
```

If disk logging is used, ASSGN, DLBL, and EXTENT statements as shown below are required. The log file must have been previously defined with a DEFINE command because it is a VSAM file.

```
// ASSGN SYSxxx,X'cuu'
// DLBL {DSKLOG1},'cluster-name',,VSAM
      {DSKLOG2}
// EXTENT extent information
```

The execution job stream must contain ASSGN, DLBL, EXTENT, or TLBL statements that define the data base(s) that are to be processed. When initially loading a data base, additional DLBL and EXTENT statements may also be required for system work files.

The EXEC statement specifies the DL/I initialization and the SIZE parameter. Typically, you will require a 512K virtual partition for execution with a size parameter of 250K. See *DOS/VSE System Control Statements,* GC33-5376, for details.

```
// EXEC    DLZRRC00,SIZE=xxxK
```

## Data Base Load Processing

### Loading A Basic Data Base

After generating the physical DBD, you can load your data base using a load program. Basically the load program reads a sequential file with the data base record contents; it builds the segments and inserts them in the data base in hierarchical order. Quite often the data to be stored in the data base already exists in one or more files, but merge and sort operations may be required to present the data in the correct sequence. Sometimes even clean-up and correction activities are required, especially when multiple files with redundant data are merged into one data base (see Figure 4-20).
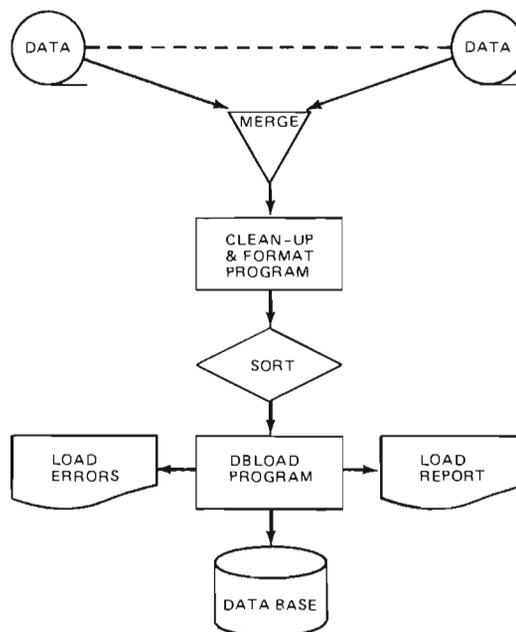


Figure 4-20. Basic Data Base Load Process

### Loading Data Bases With Logical Relationships

To establish the logical relationships during initial load of data bases with logical relationships, DL/I provides a set of utility programs. These are necessary because the sequence in which the logical parent is loaded is normally not the same as the sequence in which the logical child is loaded. To cope with this, DL/I will automatically create a workfile whenever you load a data base that contains a logical child and/or logical parent. This workfile contains the necessary information to update the pointers in the prefixes of the logically related segments. Before doing so, the workfile is sorted in physical data base sequence with the *prefix resolution utility* (DLZURG10). This utility also checks for missing logical parents. Next, the segment prefixes are updated with the *prefix update utility* (DLZURGP0). After this, the data base(s) are ready to use. The above data base load, prefix resolution and update should be preceded by the *prereorganization utility* (DLZURPR0). This utility generates a control data set to be used by data base load. A detailed discussion of this data base load process and the associated utilities can be found in Chapter 6, "Data Base Reorganization/Load Processing".

### Sample Data Base Load Program

The sample data base load program, DLZSAM40, is used to load the sample data bases. This sample is provided for use as a guide in writing your own data base load program(s).

### Loading a HIDAM Data Base

When loading a HIDAM data base initially, you must specify PROCOPT=LS in the PCB. Also, the data base records must be inserted in ascending root key sequence, and the segments must be inserted in their hierarchical sequence.

*Sorting segments in their hierarchical sequence:* If there is a need to sort on a segment level, you must provide the following sort control fields with each segment (Figure 4-21).

When loading a HIDAM data base, DL/I will also load the primary index data base.

Note: When loading a HIDAM data base, DL/I will automatically insert a high key (X'FF...') at the end of the data base. This is for its chain maintenance, it is completely transparent to your program. You should not use this key in your application.

### Loading a HDAM Data Base

When initially loading a HDAM data base, you should specify PROCOPT=L in the PCB. There is no need for DL/I to insert the data base records in root key order, but you must still insert the segments in their hierarchical order. For performance reasons it is advantageous to sort the data base records into physical sequence. The physical sequence should be the ascending sequence of the block and root anchor point values as generated by the randomizing algorithms. This can be achieved by an E61 type sort exit routine, which gives each root key to the randomizing module for address conversion, and then directs SORT to sort on the generated address + root key value.

### Status Codes for Loading Data Bases

The following status codes can be expected after the ISRT call when loading data bases:

| | |
|---|---|
| bb | OK, segment is inserted in data base |
| LB: | the segment you tried to insert already exists in the data base |
| LC: | key field of segment is out of sequence |

| | |
|---|---|
| LD: | no parent has been inserted for this segment in the data base. |
| other: | error situation |

## Status Code Error Routines

There are essentially two categories of error status codes: those caused by application program errors and those caused by system errors. Sometimes, however, a clear split cannot be made immediately. Figure 4-4 contains a listing of the status codes in both of these categories.

To handle all other status codes, it is recommended that a standard error routine be made available by the data base administrator that will print as much information as possible prior to termination of the program. This would include all fields in the PCB, I/O area, etc. Most of the status codes in this category are usually encountered during the debugging of the application program. The standard error routine could be included in each program using COBOL COPY, PL/I %INCLUDE, or Assembler COPY statements.

## DL/I DOS/VS Buffer Pool Characteristics Report

DL/I will print a report on SYSLST of the characteristics of its buffer pool at initialization time. This report contains information on number of subpools, subpool size, DMB assignment. etc.

Note: When coding DL/I application programs, you should avoid printing preprinted forms such as checks, etc. on SYSLST and use a programmer logical unit instead. This way the printing of the buffer pool characteristics on the preprinted forms can be supressed by assigning SYSLST to IGN.

## Processing With Logical Relationships

Generally, there is no difference between the processing of physical data bases and logical data bases; all call functions are available for both. Some considerations do apply when accessing a logical child or a con-

| ROOT KEY | LEVEL 2 SEGMENT CODE | LEVEL 2 KEY | LEVEL 3 SEGMENT CODE | LEVEL 3 KEY | etc. |
|---|---|---|---|---|---|

Notes:

1. For every level, the key field length should be equal to the largest segment key field on that level. Shorter keys should be left adjusted and padded with low value characters.

2. Segments on the lowest level need not have a key field if no sequence field is defined, however, their sequence below their parent might be different after the sort. If no sequence field is available in the segment itself, you should provide one. This could be a simple dependent segment counter provided by a user written "clean up and format" program as shown in Figure 4-19.

Figure 4-21. Control Field for Sorting Segments into Hierarchical Sequence

catenated segment. For a definition of these terms see "DL/I Logical Relationships" in Chapter 2.

### Accessing A Logical Child In A Physical DBD

When accessing a logical child in a physical DBD, you should remember the layout of the logical child. It always consists of the logical parent concatenated key (i.e., all the consecutive keys from the root segment down to and including the logical parent) plus the logical child itself; the intersection data (see Figure 2-15). This is especially important when inserting a logical child. You will also get an IX status code when you try to insert a logical child and its logical parent does not exist (except at initial load time). This will typically happen when you forget the LPCK in front of the LCHILD.

Note: In general, physical data bases should not be used when processing logical relationships.

### Accessing Segments in a Logical DBD

The considerations that apply for each call function when accessing segments in logical DBDs is directly related to the rules for logical relationships as discussed in Chapters 2 and 3. Review these sections before attempting to insert, delete, or replace a concatenated segment. Additional information can be found in the *System Application Design Guide*.

## Processing With Secondary Indexes

For a review of the terminology and functions of secondary indexes see "DL/I Secondary Indexes" in Chapter 2.

As discussed before, DL/I will always maintain the secondary index, whether or not the program making the change is using the index. As a consequence, DL/I must always have access to the index data bases when processing the main data base. So, the DD statements for the index data bases must be supplied in the JCL of every job which could change the secondary index.

### Accessing Segments Via a Secondary Index

#### Retrieving Segments

The same calls are used as before. However, the index search field, defined by an XDFLD statement in the DBD will be used in the SSA for the get unique of the root segment. It defines the secondary processing sequence.

Figure 4-22 shows an example of a get unique call for an INVENTORY ITEM using the secondary processing sequence. After the successful completion of this call, the PCB and IOAREA look the same as after the basic GU of Figure 4-8, except that the key feedback area now starts with the INVENTORY ITEM number defined by this secondary processing sequence.

```
77   GU-FUNC PICTURE XXXX VALUE 'GU  '.
01  SSA005-GU-STPIITM.
   02   SSA005-BEGIN PICTURE X(19) VALUE   'STPIITM (STXININ  ='.
   02   SSA005-STXININ PICTURE X(8).
   02   SSA005-END PICTURE X VALUE ')'.
01  IOAREA    PICTURE X(256).
```
```
MOVE INVENTORY-ITEM-NO TO SSA005-STXININ.
CALL 'CBLTDLI' USING GU-FUNC,PCB-NAME,IOAREA,SSA005-GU-STPIITM.
```
```
STATUS CODES:
   bb:   requested INVENTORY ITEM segment has been moved to IOAREA
   GE:   segment not found, requested purchase order number not in
         data base
other:   error situation
```

Figure 4-22. GU Call Using a Secondary Index

When using the secondary processing sequence, consecutive get next calls for the INVENTORY ITEM segment will present the INVENTORY ITEM segments in item number sequence. This is done in our sample application for printing the Inventory data base, because the randomizing module does not store the INVENTORY ITEM segments in Item Number sequence.

If both the primary and the secondary processing sequence are needed in one program, you should use

two PCBs. Should your application, however, require more complex search strategies, then consult the other DL/I DOS/VS publications.

#### Replacing Segments

To replace segments in the indexed data base a combination of get hold and replace calls can be used as before. Again, no sequence fields may be changed. The index search fields, however, can be changed. If an

index search field is changed, DL/I will automatically update the index data base via a delete old and insert new pointer segment.

Note: When using a secondary processing sequence, this could result in the later reaccessing of a data base record.

**Deleting Segments**
When using a secondary processing sequence, you cannot delete the index target segment (i.e., the root segment). If you have a need to do so, you should use a separate PCB with a primary processing sequence.

**Inserting Segments**
Again, when using a secondary processing sequence, you cannot insert the index target segment. In all other cases, the ISRT call will function as before.

## Secondary Index Creation
A secondary index can be created during initial load of the indexed data base or later. The secondary index data base is created with the DL/I reorganization utilities. No application program is required for this creation. Chapter 6 will cover this in detail.

This section contains the DL/I considerations for the design of online programs. DL/I is used in conjunction with CICS/VS to form a complementary system to aid users in online application development and implementation. All CICS/VS program design guidelines and recommendations apply equally in a CICS/VS-DL/I environment and in a pure CICS/VS environment. To understand how to use DL/I in an online environment, you must have a good understanding of both DL/I and CICS/VS. See *Customer Information Control System/Virtual Storage (CICS/VS) General Information,* GC33-0066 for additional information.

Because of the dependencies of MPS on CICS/VS, the programming considerations for MPS are also included in this chapter.

Figure 5-1 illustrates DL/I operation under CICS/VS, where the application calls are issued by the online application program. DL/I resolves any conflict that might be created when different transactions wish to update the same data base simultaneously. Execution of the call request is identical to the batch environment.

### *DL/I Online System Execution*

Initialization of CICS/VS includes loading the DL/I facility and, optionally, opening the DL/I log. It does not necessarily include opening DL/I data bases that can be accessed online, because CICS/VS allows the operator to add or remove data bases in the online system.

The arrows 1 and 2 in Figure 5-1 indicate CICS/VS transaction input and output between the terminal and the application program. CICS/VS creates a task based upon 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.

DL/I verifies that the program can request DL/I services before it is given control.

All application programs that use DL/I have a language interface link-edited with the application program. The language interface accepts data base calls from the application program and passes control to the DL/I data base modules (arrow 3).

Once the online program begins execution, the following functions are performed:

- The program issues a DL/I scheduling call to acquire a PSB prior to issuing calls involving data. This is necessary because unlike the batch environment, many CICS/VS transactions accessing the same data base may be active at one time. Con-

current updating and deletion of data base records or segments as well as deadlock conditions make it necessary for DL/I to schedule resources selectively. In scheduling a PSB for a program, DL/I considers processing options and segment sensitivity in relation to other DL/I application programs currently executing. The address of all PCB pointers for a PSB is placed in the TCA (line A) for use by the application program, if the scheduling call is successful.

- The application program makes CICS/VS requests. These may include transient data input or output, storage requests, and non-DL/I file activity (arrows 4).

- CICS/VS controls all wait situations that may occur either in the application program itself or in any DL/I module. Therefore, control is passed back and forth between CICS/VS and the application program (or DL/I) during processing of the application program (arrows 4).

- When the program makes a data base call, the same operations take place in the online system as in the batch environment (arrows 5 through 8).

- When the program no longer needs the services of DL/I, it issues a termination call, or DL/I will do it automatically on task termination. This call is the opposite of a DL/I scheduling call. It causes the program to relinquish all DL/I resources. This includes the PSB and the ability to access DL/I data bases until the next scheduling call is issued.

## MPS (Multiple Partition Support)

MPS allows several application programs running in different partitions to access the same data bases concurrently with full data base integrity.

MPS uses the CICS/VS-DL/I DOS/VS interface and thus requires CICS/VS to operate. Batch DL/I programs communicate their DL/I requests to the CICS/VS partition via the cross partition event control facilities of DOS/VS. Special transactions in the CICS/VS partition receive these requests, issue the appropriate DL/I calls and pass back the results to the batch partitions. All data base I/O is performed by the DL/I facility within the CICS/VS partition.

MPS is transparent to the application program. Existing batch application programs need not be changed when used in this environment.

MPS is especially applicable to users who must keep data bases online for the major part of a day and need to be able to run batch reports or perform minor file
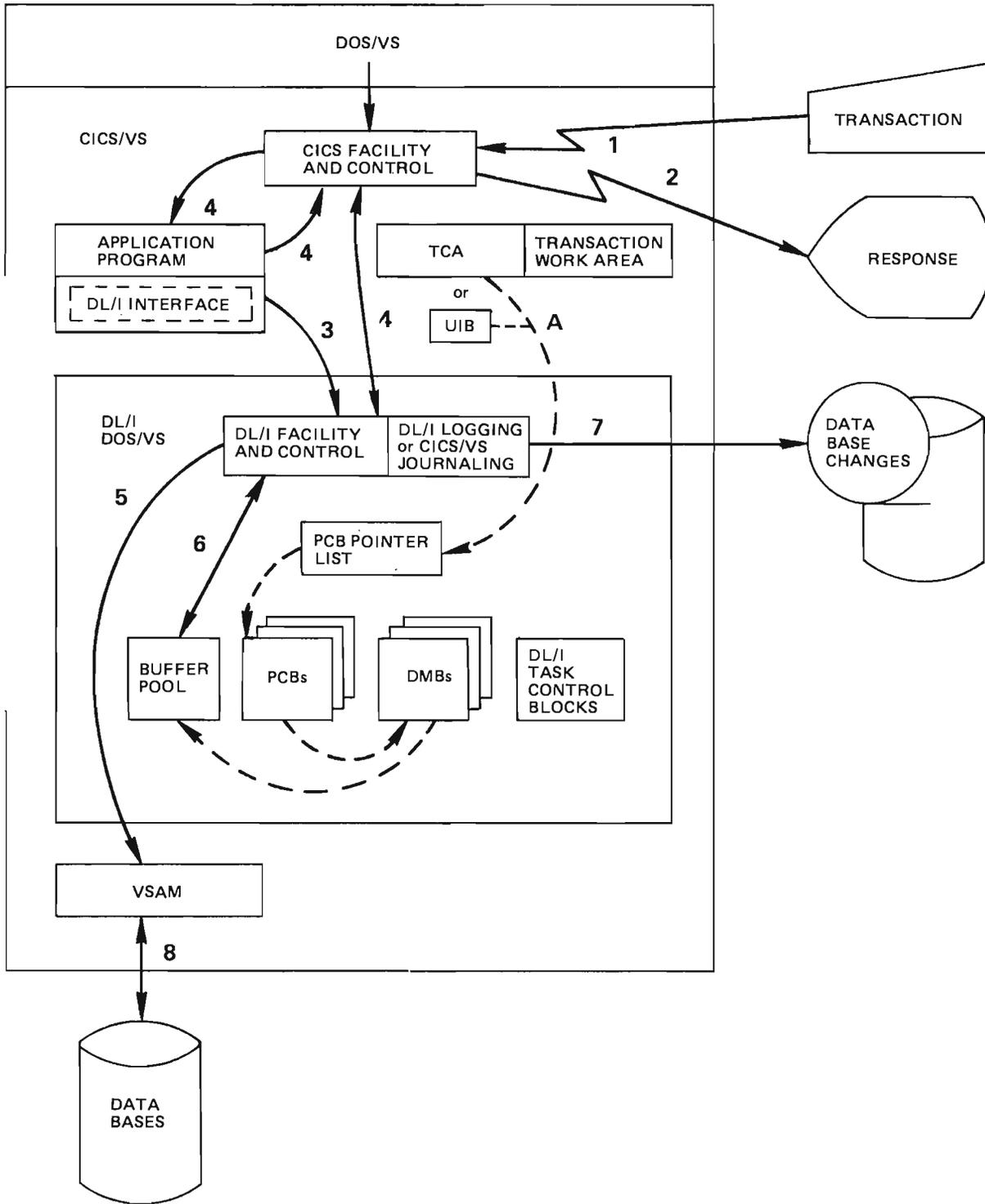
Figure 5-1. Online Data Base System Flow

updating during this time. Without MPS, the data bases would have to be closed in the online system while the batch programs are run. DL/I DOS/VS is the only DOS/VS data management facility that provides full multiple partition support with complete data base integrity.

Figure 5-2 describes the control flow for an MPS batch program when it has been given control.

- A parameter address list is provided by the DL/I system control facility when the application program is entered. The addresses in this list establish a connection from the application program to each data base PCB in the PSB (arrow 1). These data base PCB addresses are subsequently used by the application program when issuing data base call requests.

  Arrows 2 and 3 indicate the transaction input and response output functions in the application program. These functions are the same as in any batch application programs.

  All application programs which operate under DL/I have a "language interface" automatically link-edited with the application program. The language interface accepts a data base call from the application program and passes control to the DL/I facility (arrow 4). The parameter addresses in a call list, except for the PCB address, are verified by the MPS facility.

- The MPS facility communicates with the online task created by DL/I for this MPS batch job. This task is the BPC (batch partition controller), (arrows 5).

- The BPC issues the DL/I call, (arrow 6). DL/I performs normal verification, as in the case of an ordinary online task, except for parameter addresses which are verified by the MPS facility in the batch partition.

- The same operations occur then as in regular online batch processing (arrows 7 - 10).

## Differences Between Batch, MPS, and Online DL/I

The differences between batch, MPS, and online programming with DL/I are minor, as shown in the following table. Otherwise, all batch functions are similar, using the same languages, call formats, and status codes.

| Batch | Online |
|---|---|
| The application program is a subprogram to DL/I. | The application programs are CICS/VS tasks, utilizing the common DL/I code. |
| Because the PSB for the application is defined to DL/I at initialization, the application program does not have to explicitly request a PSB. | The online program must explicitly issue a scheduling call for a PSB. |
| Code need not be reentrant. | Code must be quasi-reentrant (unless RELOAD=YES is specified in the CICS/VS PPT). |
| Assembler call statement is CALL ASMTDLI. | Assembler call statement is CALLDLI ASMTDLI. |

**MPS**

The application program is a subprogram to DL/I. It also is connected with a CICS/VS task that uses the common DL/I code.

Because the PSB for the application is defined to DL/I at initialization, the application program does not have to explicitly request a PSB.

Code need not be reentrant.

Assembler call statement is CALL ASMTDLI.

## Security

Special CICS/VS transactions are provided by DL/I to allow the user to dynamically enable and disable the multiple partition support facility.

Additional security is provided via the DL/I ACT (application control table). The ACT serves as a master list of all authorized programs and PSB combinations that may be used to access a data base. This facility, in conjunction with VSAM's share option 1, will prevent batch programs from accessing online data bases except under control of MPS.

The DL/I PSB provides an additional level of security on a program-by-program basis. The PSB allows the user to control the access rights of individual programs to any segment within a data base record.

## Integrity

Because all data base requests from both batch and online programs are channeled to a single facility, DL/I is able to prevent programs from attempting to update the same information simultaneously. In a similar way, DL/I is able to detect deadlocks between programs and resolve them.

The key to DL/I's data base integrity mechanism is its program isolation and segment intent checking facilities.
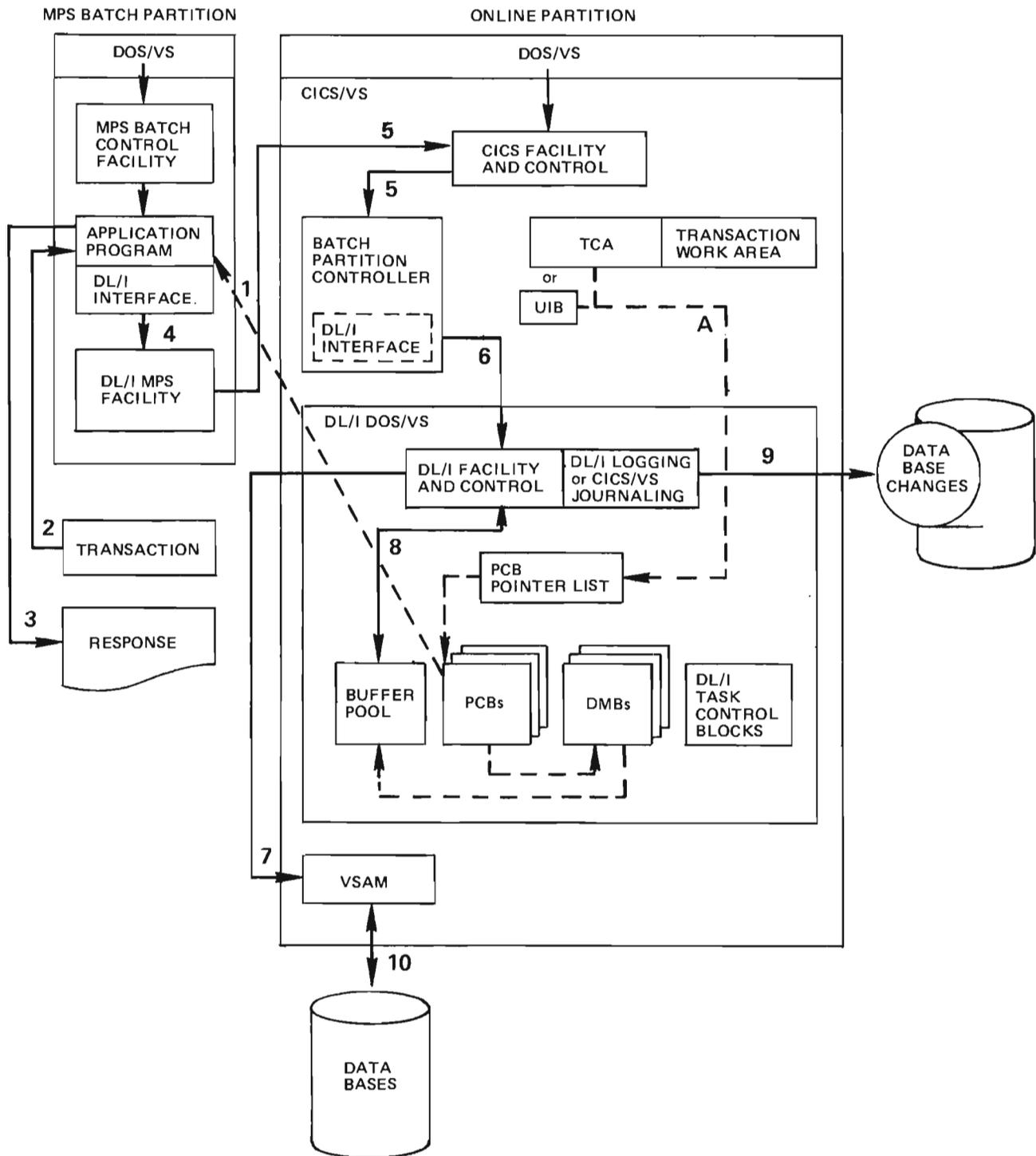
Figure 5-2. MPS Batch Data Base System Flow

DL/I provides logging facilities and a comprehensive set of utilities to allow for recovery of the data base in the event of a software or hardware failure. With MPS, a single log is written for all DL/I partitions (from the CICS/VS partition). This log may be assigned to the CICS/VS system journal, or created as an independent (of CICS/VS) DL/I log. If desired, with CICS/VS journaling, CICS/VS Emergency Restart will invoke the DL/I backout utility during emergency restart processing to restore the DL/I data bases back to a known point where processing may be restarted, backing out any updates resulting from inflight tasks.

### *Performance*

DL/I provides excellent data base performance when:

- Sufficient real storage is available.

- Segment intent conflict is minimal.

- Good data base and application program design principles are followed.

Experience has shown that where DL/I does not meet performance expectations, one or more of the above conditions was not met. With use of MPS, minimizing segment intent conflicts is of utmost importance. Potential MPS users should carefully evaluate potential intent conflict situations between batch and online programs.

### *Restrictions*

Certain DL/I programs are restricted from running in the MPS environment, i.e. the data bases they access may not be shared across several partitions while these programs are executing. The programs in this category are:

- All DL/I utilities

- All programs that access SHSAM or HSAM data bases

- All programs that load data bases

In addition any batch DL/I programs that modify the contents of the DL/I control blocks cannot run under MPS since the DL/I control blocks no longer exist in the batch partition.

All data bases accessed by a program running under MPS control must be defined in the CICS/VS partition and accessed through MPS. A program running under MPS control cannot access any data bases not known to MPS, i.e. not defined in the CICS/VS partition.

## VSAM Data Set Share Options

Share option 1 is the only VSAM share option that should be used with any VSAM data set being accessed by DL/I. DL/I operating in an MPS environment does not require any different VSAM share options than DL/I operating in a non-MPS environment. Since any DL/I-MPS programs residing in different partitions are really accessing the data bases through a common partition, the CICS/VS partition, VSAM share option 1 is sufficient.

Because DL/I always opens its VSAM data sets for update, VSAM share option 2 provides no functional benefits to DL/I.

VSAM share option 3 should never be used in conjunction with DL/I. It provides no protection against inadvertently scheduling non-MPS DL/I programs that update the same data bases in two different partitions simultaneously. Damage to data bases used in this manner may go undetected for some time, after which recovery is very difficult.

VSAM share option 4 does not apply to ESDSs processed at the control interval level. As this is the way DL/I processes its HD ESDSs, share option 4 provides no functional benefits to DL/I. Share option 4 causes VSAM to override updated contents of records in HISAM, simple HISAM, or INDEX data bases which results in the loss of delete or replace calls.

## CICS/VS System Generation

There are no parameters for CICS/VS system generation specifically for DL/I MPS. The parameter DL1=YES must be specified in the DFHSG TYPE=INITIAL macro to generate support for DL/I in a CICS/VS system, whether or not MPS will be used.

If program isolation is used your CICS/VS system must have dynamic transaction backout support and CICS/VS journalling generated.

### *CICS/VS System Table Preparation*

For DL/I MPS support within CICS/VS the following CICS/VS tables require specific entries.

| | |
|---|---|
| FCT | File Control Table |
| JCT | Journal Control Table (if logging to CICS/VS journal) |
| PCT | Program Control Table |
| PPT | Program Processing Table |
| SIT | System Initialization Table |
| PLT | Program List Table |

#### FCT (File Control Table)

An entry in the FCT is required for each DL/I data base referenced either by an online DL/I transaction or by a batch DL/I MPS program. The only parameters that can be specified are dataset name (use the dbdname from

the DBD statement), access method (DL/I), and open option (initial or deferred).

Because the entries are referenced only during CICS/VS initialization, for best performance they should be grouped with other low activity entries in your FCT.

### JCT (Journal Control Table)

The JCT is used to describe your journal data sets (optional) to CICS/VS. There must be one entry in the table to define the CICS/VS system journal.

The minimum journal buffer size is 554 bytes (512 bytes plus the size of the CICS/VS label record). The maximum journal buffer size handled by DL/I is 32,767 bytes.

### PCT (Program Control Table)

A PCT is used to define all transactions that may be processed by the system. Each transaction is described in a table entry that includes:

- Transaction identifier (up to four characters)

- Transaction priority value

- Security key

- Name of program that (initially) processes the transaction

Several PCTs can be assembled and identified by suffix characters.

There are four DL/I MPS transactions that must be defined in the PCT. The entries are coded as follows:

```
DFHPCT  TYPE=ENTRY,PROGRAM=DLZMSTR0,     X
        TRANSID=CSDA
DFHPCT  TYPE=ENTRY,PROGRAM=DLZMPC00,     X
        TRANSID=CSDB,TWASIZE=268
DFHPCT  TYPE=ENTRY,PROGRAM=DLZBPC00,     X
        TRANSID=CSDC,TWASIZE=256
DFHPCT  TYPE=ENTRY,PROGRAM=DLZMSTP0,     X
        TRANSID=CSDD
```

These transactions are used as follows:

CSDA  Start MPS operation
CSDB  Master Partition Controller
CSDC  Batch Partition Controller
CSDD  Stop MPS Operation

Because these transactions are probably of low activity compared to most of the transactions in your system, their entries should be placed with other low activity entries in your PCT. Use the CICS/VS statistics to aid you in determining their placement in the PCT. These transactions should be specified as CLASS=LONG.

Only the CSDA and CSDD transactions are ever entered from a terminal. The CSDB and CSDC transactions are internally attached by DL/I MPS. If you inadvertently enter the CSDB or CSDC transaction-id from a terminal, the Master Partition Controller or Batch Par-

tition Controller program ignores the request, however, no message is returned to the terminal.

Since the CSDA and CSDD are special transactions you will probably want to put a transaction security on these so that they can only be executed by the master terminal operator or other authorized users.

Note: Because these transaction-ids start with the letter "C" you cannot use the CICS/VS TCLASS/CMXT facility with these transactions.

If PI is used, then all DL/I transactions must be marked for dynamic transaction backout (DTB=YES). Failure to do so can cause loss of data integrity.

### PPT (Program Processing Table)

Use the PPT to define all application programs that are valid for processing under CICS/VS. Each application program is described in a table entry that includes:

- Program name

- Source coding language (ANS COBOL, PL/I or Assembler)

Several PPTs can be assembled and identified by suffix characters.

The programs referenced by the four DL/I MPS transactions must be defined in the PPT. They are coded as follows:

```
DFHPPT  TYPE=ENTRY,PROGRAM=DLZMSTR0
DFHPPT  TYPE=ENTRY,PROGRAM=DLZMPC00
DFHPPT  TYPE=ENTRY,PROGRAM=DLZBPC00
DFHPPT  TYPE=ENTRY,PROGRAM=DLZMSTP0
```

Because these programs are probably of low activity relative to the regular online programs, they should be placed with other low activity entries in your PPT. Since DLZMSTR0 and DLZMSTP0 are low usage (normally used only once per CICS/VS session), they should be specified as RES=PGOUT. DLZMPC00 and DLZBPC00 should be specified as RES=YES for best performance. These specifications can also be made via an ALT (see the example later in this chapter).

If a DL/I formatted dump is printed in the event of a CICS/VS ABEND, the DL/I Formatted System Dump Program (DLZFSDP0) must be identified in the PPT. As this is used only in the event of a CICS/VS ABEND, it should be specified as RES=PGOUT for best performance.

### SIT (System Initialization Table)

The SIT contains user-specified data that controls the system initialization process; in particular, a SIT can identify (by suffix characters) the user-specified versions of CICS/VS system control programs and CICS/VS tables that are to be loaded.

The system programmer can assemble several SITs -- each SIT being identified by suffix characters. Then, whenever the system is initialized, the required SIT can be specified by its suffix. Parameters in the chosen SIT can also be overridden at system initialization time for greater flexibility.

The SIT must have DL1=YES or DL1=xx specified. In addition, DL/I requires entries in a shutdown PLT and optionally in an initialization PLT. Specify the suffix of the appropriate initialization and shutdown PLTs in the PLTPI and PLTSD parameters respectively.

Each active batch MPS program requires a CICS/VS task to support it. Therefore, you may want to increase your current AMXT and MXT specifications. The DL/I MPS Master Partition Controller task (CSDB) is not counted by CICS/VS when calculating the number of active tasks for AMXT purposes, just as the CICS/VS terminal control, task control and journal control tasks are not counted. The performance implications of the AMXT and MXT parameters are discussed later in this chapter.

### PLT (Program List Table)

The PLT contains program identifications that perform various functions within CICS/VS. Each program identified must also appear in the PPT, making the PLT a subset of the PPT. The PLT provides the following:

- List of programs to be executed during the post-initialization phase of system initialization.

- List of programs to be executed during the first quiesce stage of system termination.

- List of programs to be executed during the second quiesce stage of system termination.

By providing suffixes for PLTs, you can use many versions of these tables.

DL/I does not process any MPS batch programs until the Master Partition Controller program is initiated in the CICS/VS partition. This program can be initiated in the following ways:

- by a terminal operator entering the transaction "CSDA".

- by letting CICS/VS execute the MPS start program (DLZMSTR0) during its system initialization processing.

If the second option is taken, a start-up PLT must be coded with the DL/I MPS Start program as one of the entries. The suffix of the start-up PLT must be specified in the "PLTPI" parameter of the SIT.

If you want DL/I MPS to stop automatically when CICS/VS starts its shut-down processing, you should make an entry in the shut-down PLT for the program DLZMSTP0. This entry must appear before the DFHDELIM entry.

DL/I requires an entry in a shut-down PLT for the program DLZSTP00 following the DFHDELIM entry. This is required whether or not MPS is used. The suffix of this shut-down PLT must be specified in the PLTSD parameter of the SIT.

Remember to code the PLT names in the PPT.

*Caution:* An operator should never do an Immediate Shut-down of CICS/VS. CICS/VS Emergency Restart, DL/I Backout, or Forward Recovery may be required following an Immediate Shut-down of CICS/VS to recover the data bases.

## DL/I Application Control Table

The online DL/I Application Control Table (ACT) has two functions. One function is to provide DL/I with environmental information such as buffer pool size, DMB assignments, maximum number of concurrent DL/I tasks that may execute, etc. If you have an existing CICS/VS - DL/I system, you should review your ACT environmental parameters (MAXTASK, etc.) to see if they require changing for MPS. Each active batch DL/I MPS program requires a Batch Partition Controller (BPC) transaction in the online system to support it. Therefore, you may need to increase the values of MAXTASK and CMAXTSK. If you are introducing more data bases into the online system (those that were formerly used only in batch and now are to be accessed via MPS), you should examine the size of your buffer pool and DMB assignments.

The second function of the ACT is to define the valid application program and PSB combinations for the online system. When an online DL/I transaction attempts to schedule a PSB, DL/I checks to insure that the program issuing the scheduling call is authorized to use that PSB. DL/I accomplishes this by scanning its ACT for the name of the requesting program. Once DL/I locates the program name in the ACT, it then checks to see if the requested PSB is authorized for that program.

After assembly and link-edit, the ACT becomes the online DL/I nucleus (DLZNUCxx).

## Establishing the Control Section for the DL/I Application Control Table

The control section into which the ACT is assembled is established by means of the DLZACT macro:

```
DLZACT      TYPE=INITIAL[,SUFFIX=xx]
```

This macro must be coded as the first statement in the source deck used to assemble the DL/I ACT.

## SUFFIX

specifies a 2-character alphameric suffix for the DL/I ACT being assembled. The suffix, if specified, is appended to the standard module name (DLZNUC) and is used to name the module on the linkage editor output library. If this operand is omitted, a suffix is not provided.

## *Defining the Online Environment for DL/I*

The DLZACT TYPE=CONFIG macro instruction defines the online environment for the CICS/VS DL/I DOS/VS user. Information from this statement is used to determine the size of PST prefix table and to initialize fields in the SCD. There may be only one DLZACT TYPE=CONFIG statement for each DL/I ACT generation.

The DLZACT TYPE=CONFIG macro instruction can include the following operands:

```
DLZACT TYPE=CONFIG
    [,MAXTASK={nnn}]
              {10 }
    [,CMAXTSK={nnn         }]
              {maxtaskvalue}
    [,BFRPOOL=nnn]
    [,PASS={password}]
          {DLZPASS1}
    [,SLC=phname][,PI={YES}][,REMOTE={YES}]
                      {NO }          {NO }
```

## MAXTASK

specifies the maximum number of DL/I tasks that may be processed concurrently, where nnn is a numeric value from 1 to 255. If this operand is omitted, the value 10 is assumed for MAXTASK. See "Controlling the Number of CICS/VS and DL/I Tasks" later in this chapter.

## CMAXTSK

specifies the maximum number of concurrent DL/I DOS/VS tasks allowed, where nnn is a number from 1 to 255. However, it must not exceed the value specified for MAXTASK. If this operand is omitted, the value specified for MAXTASK is used. See "Controlling the Number of CICS/VS and DL/I Tasks" later in this chapter.

## BFRPOOL

specifies the number of buffer subpools to be acquired and formatted during the initialization of the online DL/I system, where nnn is a numeric value from 0 to 255. If the value is omitted or zero, a request for the value is made at 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 is some multiple of 512 bytes. The buffer size value is deter-

mined at DL/I system initialization and is based on the value specified in BFRPOOL, the number of data bases, and the size of the VSAM control intervals. A data base is assigned a subpool that contains buffers equal to or greater in size than the size of the data base control interval.

## PASS

specifies the password (1 to 8 alphameric characters) associated with the special functions of the system control calls. See "DL/I System Call Format and Returns" in the *Utilities and Guide for the System Programmer*. If this parameter is omitted the value "DLZPASS1" is used as default.

## SLC

specifies the phase name of the storage layout control table to be used. See "Storage Layout Control Option" in the *Utilities and Guide for the System Programmer*.

## PI

specifies the program isolation option (default is YES). See "Program Isolation" later in this chapter.

## REMOTE

REMOTE=YES simplifies DL/I online nucleus generation for processing requests from other systems. This optional parameter enables all PSBs defined in this (local) system's DL/I nucleus to be accessed from other (remote) systems through the CICS/VS mirror program DFHMIR.

The REMOTE=YES parameter accomplishes this by generating a DLZACT TYPE=PROGRAM,PGMNAME=DFHMIR statement for the CICS/VS mirror program which includes the PSB names of all PSBs that are 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 these PSBs. If, in addition, REMOTE=YES is specified, all PSBs that can be accessed by the local system are added to the list of PSBs specified in the DLZACT TYPE=PROGRAM,PGMNAME=DFHMIR statement. See the *Utilities and Guide for the System Programmer* for details.

## *Describing the Application Program Relationship to DL/I Data Bases*

A logical connection between a CICS/VS application program and a DL/I data base is achieved through the DLZACT TYPE=PROGRAM macro instruction.

A maximum of 4095 DLZACT TYPE=PROGRAM state-

ments and a maximum of 4095 unique entries (an entry consisting of program name and one PSBNAME) may occur in one ACT generation. Therefore, a maximum of 4095 unique program names and 4095 PSB names is possible.

**Note:** During the DL/I scheduling call, if the PSBNAME is omitted from the call parameter list, the first PSBNAME defined in order of appearance in the macro generation associated with the program becomes the default PSB for that program.

```
DLZACT      TYPE=PROGRAM,
            PGMNAME=name,
            PSBNAME=(name,name,...)
```

PGMNAME

> specifies the application program name (may be 1 to 8 alphameric characters) defined for the TYPE=PROGRAM statement.

PSBNAME

> specifies the PSBNAME(s) associated with the program's entries. Valid PSB names are from 1 to 7 alphameric characters. The first PSB name encountered in the PSBNAME list becomes the default PSB for the application program named in this entry. Because all batch MPS programs actually communicate to DL/I through the BPC (batch partition controller) running in the online partition, you must make an entry in the ACT for this program. You should list with this program the names of all PSBs that are used with MPS batch application programs. Note that the DL/I utility programs cannot run under MPS. Nor can you execute any batch DL/I programs under MPS that require a PCB PROCOPT of L or use SHSAM or HSAM access methods. The form of this ACT entry is as follows:

```
DLZACT TYPE=PROGRAM,PGMNAME=DLZBPC00,    X
       PSBNAME=(name,name...)
```

Potentially you could have quite a few PSBs to be associated with the BPC. The DLZACT macro accepts up to 4095 PSB names (and program names). In general, this quantity is sufficient for any user. The macro language of the DOS/VS assembler, however, accepts no more than 255 characters (including parentheses) in a sublist as the operand in a macro. A sublist is one or more entries separated by commas and enclosed in parentheses, such as the list of PSB names following the PSB name key word parameter in the DLZACT macro. Because this probably won't be sufficient for the BPC entry in the ACT, DL/I provides a continuation facility. To continue the PSBNAME sublist, code the parameter "CONT=YES" on each line that is to be continued as shown in the following example.

```
DLZACT  TYPE=PROGRAM,PGMNAME=DLZBPC00,   X
        PSBNAME=(PSBA,PSBB,PSBC),         X
        CONT=YES
DLZACT  PSBNAME=(PSBD,PSBE,PSBF),         X
        CONT=YES
DLZACT  PSBNAME=(PSBG,PSBH)
```

In this example, PSBs A through H are associated with the BPC. The use of the continuation facility is not limited to the BPC entry in the ACT.

### *Specifying a Data Base Resident on Another System*

Using CICS/VS intersystem communication support, DL/I application programs can access a data base that is resident on another CPU. The application program can be unaware of where the data base is located. Intersystem communication support is invoked when the DL/I program request handler detects a remote PSB during a scheduling call.

The system programmer, when generating a DL/I online nucleus, can optionally specify the location of remote PSBs through the DLZACT TYPE=RPSB macro instruction. See the *Utilities and Guide for the System Programmer* for details.

### *Specifying Buffer Pool Control Options*

The size of DL/I buffer subpool and their assignments to DMBs as well as VSAM buffer usage can optionally be specified through the DLZACT TYPE=BUFFER macro instruction. For each buffer subpool or HISAM or INDEX data base, one TYPE=BUFFER macro instruction can be specified.

```
DLZACT  TYPE=BUFFER,{HDBFR=(bufno
         [,dbdname1,dbdname2,...])
         [,HSBFR=...]   }
         {HSBFR=(indno,ksdsbuf,
         [esdsbuf],dbdname)[,HDBFR=...]}
```

HDBFR describes one DL/I subpool where:

- bufno
  specifies the number of buffers to be allocated for this subpool and is a numeric value from 2 to 32. If omitted for a specific subpool, 32 is assumed, and

- dbname1,dbname2,...
  specifies the name of DBDs that are to be allocated for this subpool.

HSBFR defines VSAM buffer allocation for HISAM and INDEX data bases.

- indno
  specifies the number of index buffers for a KSDS.

- ksdsbuf
  specifies the number of data buffers for a KSDS.

- esdsbuf
  specifies the number of data buffers for the ESDS (applies to HISAM only).

- dbdname
  the name of the HISAM or INDEX DBD referenced by the application program.

### Specifying the End of the DL/I Application Control Table

The end of the ACT generation is indicated by the following macro instruction:

DLZACT TYPE=FINAL

### JCL for Creating the Online Nucleus

Online nucleus generation is run as a standard DOS/VS job and requires the following job control statements:

```
//JOB      NUCGEN
//OPTION   CATAL
//EXEC     ASSEMBLY,SIZE=272K
          DLZACT  TYPE=INITIAL
          DLZACT  TYPE=CONFIG,...     ONLINE NUCLEUS GENERATION
          DLZACT  TYPE=PROGRAM,...    CONTROL STATEMENTS
          DLZACT  TYPE=RPSB,...
          DLZACT  TYPE=BUFFER,...
          DLZACT  TYPE=FINAL
          END
/*
  ENTRY   DLZNUC
//EXEC     LNKEDT
/&
```

**Note:** When assembling an ACT a size parameter of at least 272K is required (// EXEC ASSEMBLY,SIZE=272K), otherwise the assembler will terminate with an IPK100 error.

### Description of Online Nucleus Generation Output

Online nucleus generation produces three types of printed output and one load module. Each of these items of output is described in the following paragraphs.

### Control Statement Listing

This is a listing of the input.

### Diagnostics

Errors discovered during the processing of each control statement result in diagnostic messages. These messages are printed immediately following the image of the control statement to which they apply. The message may reference either the control statement immediately preceding it or the preceding group of control statements. It is also possible that more than one message could be printed for each control statement; in this case, the messages follow each other on the output listing. After all control statements have been read, a check of the entire deck is made to determine reasonability. This may result in one or more additional diagnostic messages.

Discovery of any errors results in the diagnostic message(s) being printed, the control statements being listed, and the other output being suppressed. However, all control statements are read and checked before the online nucleus generation execution is terminated.

### Assembly Listing

A DOS/VS Assembler language listing of the assembled online nucleus is provided.

### Load Module

After the online nucleus generation is assembled, the online nucleus must be link-edited and cataloged into a DOS/VS core image library. During the link-edit step, the relocatable library modules for both CICS/VS and DL/I must be available to the linkage editor as modules from both products are required in the DL/I online nucleus.

# CICS/VS-DL/I Table Example

The following is an example of the parameters required in the various CICS/VS and DL/I tables to support DL/I (and MPS). The numbers on each line refer to the comments that follow.

```
 1. DFHSIT TYPE=CSECT,DL1=01,FCT=01,PCT=01,PPT=01,JCT=01,ALT=01,   X
           PLTPI=SU,PLTSD=SD,DBP=01,DBUFSZ=...,...

 2. DFHFCT TYPE=INITIAL,SUFFIX=01
                 •
                 •
 3. DFHFCT TYPE=DATASET,DATASET=DB1,ACCMETH=DL/I,OPEN=...
    DFHFCT TYPE=FINAL
 4. DFHPCT TYPE=INITIAL,SUFFIX=01
 5. DFHPCT TYPE=ENTRY,PROGRAM=DL1PROG,DTB=YES,...
 6. DFHPCT TYPE=ENTRY,PROGRAM=DLZMSTR0,TRANSID=CSDA,CLASS=LONG
 7. DFHPCT TYPE=ENTRY,PROGRAM=DLZMPC00,TRANSID=CSDB,TWASIZE=268,    X
           CLASS=LONG
 8. DFHPCT TYPE=ENTRY,PROGRAM=DLZBPC00,TRANSID=CSDC,TWASIZE=256,    X
           CLASS=LONG,DTB=YES
 9. DFHPCT TYPE=ENTRY,PROGRAM=DLZMSTP0,TRANSID=CSDD,               X
           CLASS=LONG
                 •
                 •
    DFHPCT TYPE=FINAL
10. DFHPPT TYPE=INITIAL,SUFFIX=01
                 •
                 •
11. DFHPPT TYPE=ENTRY,PROGRAM=DLZMSTR0
12. DFHPPT TYPE=ENTRY,PROGRAM=DLZMPC00
13. DFHPPT TYPE=ENTRY,PROGRAM=DLZBPC00
14. DFHPPT TYPE=ENTRY,PROGRAM=DLZMSTP0
15. DFHPPT TYPE=ENTRY,PROGRAM=DLZSTP00
16. DFHPPT TYPE=ENTRY,PROGRAM=DL1PROG
17. DFHPPT TYPE=ENTRY,PROGRAM=DFHPLTSU
18. DFHPPT TYPE=ENTRY,PROGRAM=DFHPLTSD
19. DFHPPT TYPE=ENTRY,PROGRAM=DLZFSDP0,RES=PGOUT
20. DFHPPT TYPE=ENTRY,PROGRAM=DFHDBP01
                 •
                 •
    DFHPPT TYPE=FINAL
21. DFHJCT TYPE=INITIAL,SUFFIX=01
22. DFHJCT TYPE=ENTRY,JFILEID=SYSTEM,BUFSIZE=1024,...
                 •
                 •
    DFHJCT TYPE=FINAL
23. DFHPLT TYPE=INITIAL,SUFFIX=SU
24. DFHPLT TYPE=ENTRY,PROGRAM=DLZMSTR0
                 •
                 •
    DFHPLT TYPE=FINAL
25. DFHPLT TYPE=INITIAL,SUFFIX=SD
26. DFHPLT TYPE=ENTRY,PROGRAM=DLZMSTP00
                 •
                 •
27. DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
28. DFHPLT TYPE=ENTRY,PROGRAM=DLZSTP00
                 •
                 •
    DFHPLT TYPE=FINAL
29. DFHALT TYPE=INITIAL,SUFFIX=01
                 •
                 •
30. DFHALT TYPE=ENTRY,PROGRAM=DLZMPC00,ADRSPCE=LOW,PAGEOUT=NO,     X
           ALIGN=...
31. DFHALT TYPE=ENTRY,PROGRAM=DL1PROG,ADRSPCE=LOW,...
                 •
                 •
```

```
32.  DFHALT TYPE=ENTRY,ALIGN=YES,ADRSPCE=HIGH,                    X
          PROGRAM=(DLZMSTRO,DLZMSTPO,DLZSTP00,DLZFSDP0),          X
          PAGEOUT=YES
                •
                •
     DFHALT TYPE=FINAL
33.  DLZACT TYPE=INITIAL,SUFFIX=01
34.  DLZACT TYPE=CONFIG,PI=YES,...
35.  DLZACT TYPE=PROGRAM,PGMNAME=DL1PROG,PSBNAME=(...)
36.  DLZACT TYPE=PROGRAM,PGMNAME=DLZBPC00,PSBNAME=(...)
                •
                •
37.  DLZACT TYPE=BUFFER,...
     DLZACT TYPE=FINAL
```

1. This defines the SIT with a suffix of "01" for the CICS/VS FCT, PCT, PPT, JCT, and ALT; a suffix of "SU" for the start-up PLT; "SD" for the shut-down PLT; and a suffix of "01" for the DL/I ACT. DBP= and DBUFSZ= must be specified to schedule the DTB facility in this execution ofCICS/VS. Note that there must be an entry in the PPT for the version of the dynamic transaction backout program specified here.

2. This defines the FCT with a suffix of "01" to match the SIT specification.

3. This is an example of a DL/I data base FCT entry. Since these entries are only referenced during open processing they should be placed at the end of the FCT.

4. This defines the PCT with a suffix of "01" to match the SIT specification.

5. Defines a DL/I transaction (non-batch MPS) with dynamic transaction backout specified as required for DL/I PI support.

6. DL/I MPS Start transaction.

7. DL/I MPS Master Partition Controller transaction.

8. DL/I MPS Batch Partition Controller transaction. Note that dynamic transaction backout support is specified for the MPS Batch Partition Controller transaction. This will cause the DL/I updates made by a batch MPS program to be backed out if the batch program abnormally terminates.

9. DL/I MPS Stop transaction.

10. This defines the PPT with a suffix of "01" to match the SIT specification.

11. DL/I MPS Start program referred to by (6).

12. DL/I MPS Master Partition Controller program referred to by (7).

13. DL/I MPS Batch Partition Controller program referred to by (8). Note that this program is also defined in the DL/I ACT (36).

14. DL/I MPS Stop program referred to by (9).

15. DL/I System Termination Program.

16. An example of a DL/I application program PPT entry. Note that this program is also defined in the CICS/VS ALT (31) and the DL/I ACT (35).

17. Start-up PLT entry referenced in (1) and defined in (23).

18. Shut-down PLT entry referenced in (1) and defined in (25).

19. DL/I's formatted system dump program used during CICS/VS ABEND processing.

20. This defines the dynamic transaction backout program specified in (1).

21. This defines the JCT with a suffix of "01" to match the SIT specification.

22. Although a buffer size of 1024 bytes (default value) is defined here, the maximum size handled by DL/I logging is 32,767 bytes.

23. This defines the start-up PLT with a suffix of "SU" to match the SIT specification.

24. The MPS Start program is listed in the start-up PLT so that MPS operation will be initiated automatically during CICS/VS initialization.

25. This defines the shut-down PLT with a suffix of "SD" to match the SIT specification.

26. This defines the MPS stop-transaction in the shut-down PLT so that MPS operation will be stopped during the first stage of CICS/VS shut-down processing.

27. This is the special CICS/VS delimiter entry for shut-down PLTs. Programs listed after this point are executed during the second shut-down processing stage.

28. This is the required DL/I System Termination Program entry.

29. This defines the ALT with a suffix of "01" to match the SIT specification.

30. This marks the DLZMPC00 program resident in low address space. Its alignment (YES or NO) and its position in the ALT must be determined based on its usage relative to other CICS/VS application programs. The CICS/VS statistics can be used to better determine this for your system.

31. This makes this DL/I application resident in low address space. Its alignment (YES or NO) and its position in the ALT must be determined based on its usage relative to other CICS/VS application programs. The CICS/VS statistics can be used to better determine this for your system.

32. This makes the DL/I MPS Start and Stop programs, the DL/I System Termination program, and the DL/I Formatted System Dump Program resident in high address space. Because they have very low usage CICS/VS has been requested to page them out (PAGEOUT=YES). They are aligned to allow a clean page-out, i.e. the page-out will not carry code for other programs with it.

33. This defines the DL/I ACT with a suffix of "01" to match the SIT specification.

34. This specifies the DL/I environmental factors. Note that PI=YES is specified. This is also the default action.

35. This is an example of a DL/I online application program entry. Note that this program is also defined in the CICS/VS PPT (16).

36. This defines all PSBs that are valid for use by the MPS Batch Partition Controller program. Normally this would be all previous batch application program PSBs. Because this program is probably infrequently accessed relative to online DL/I application programs, it should be placed towards the end of the ACT. This program is also defined in the CICS/VS PPT (13). This is the only one of the four DL/I MPS programs that should be defined in the DL/IACT.

37. This specifies the DL/I buffer pool characteristics.

**CICS/VS-DL/I Tables for the Sample Program**
The following tables are specified for use by the online sample program, DLZSAM60:

```
DFHFCT TYPE=INITIAL,SUFFIX=HV,        X
       •
       •
       •
       •
DFHFCT TYPE=DATASET,DATASET=STDCDBP, X
       ACCMETH=DL/I
DFHFCT TYPE=DATASET,DATASET=STDIDBP, X
       ACCMETH=DL/I
DFHFCT TYPE=DATASET,DATASET=STDIX1P, X
       ACCMETH=DL/I
DFHFCT TYPE=DATASET,DATASET=STDCX1P, X
       ACCMETH=DL/I
DFHFCT TYPE=DATASET,DATASET=STDCX2P, X
       ACCMETH=DL/I
       •
       •
       •
       •
DFHFCT TYPE=FINAL
END    DFHFCTBA
DFHPCT TYPE=ENTRY,TRANSID=DLZZ,       X
       PROGRAM=DLZSAM60,TWASIZE=2048, X
       INBFMH=EODS,LOGREC=NO
DFHPPT TYPE=ENTRY,PROGRAM=DLZMAPS     X
DFHPPT TYPE=ENTRY,PROGRAM=DLZSAM60,   X
       RELOAD=YES
DLZACT TYPE=PROGRAM,                  X
       PGMNAME=DLZSAM60,              X
       PSBNAME=(STBCUSR,STBCUSU)
```

Note: DLZSAM60 is the online sample program. DLZMAPS is the mapping module for DLZSAM60.

# Initialization of the DL/I Online System

DL/I online initialization consists of a CICS/VS initialization overlay phase called by the CICS/VS system initialization program. This phase of initialization depends on information contained in the DL/I online nucleus and use of the DOS/VS UPSI byte information.

The DL/I initialization job control requirements are essentially the same as the batch DL/I requirements, however, you may elect to make use of the combined CICS/VS-DL/I journal facility. See Chapter 7 "DL/I Data Base Recovery Restart."

When using this feature the DL/I log records are written directly to the CICS/VS system journal file using CICS/VS journal requests. If the DL/I logger is used, SYS011 must be assigned to the output log file device. The DL/I logger may not be used with PI support active.

## DOS/VS UPSI Byte Settings (Online)

| | |
|---|---|
| Bits 0 - 2 | Reserved for CICS/VS. |
| Bits 3 - 5 | Available subject to Note. |
| Bit 6 = 0 | DL/I system log function active. |
| = 1 | DL/I system log function inactive. |
| Bit 7 = 0 | DL/I system log function on CICS/VS system log. |

=1  DL/I system log function on DL/I system log device (SYS011-LOGOUT).

**Note:** For further information, see the *CICS/VS System Programming Guide* (DOS/VS).

## Programming Considerations

Before attempting to write a CICS/VS-DL/I program, you should be familiar with DL/I DOS/VS batch programming concepts and Customer Information Control System/Virtual Storage (CICS/VS) programming fundamentals. References to the prerequisite publications are contained in the preface to this manual.

A programmer in a CICS/VS-DL/I environment accesses data bases in much the same manner as in the batch environment. Because the CICS/VS-DL/I user may share access to DL/I data bases with other applications programs, the user has additional responsibilities when writing a CICS/VS-DL/I program. This chapter discusses these additional programming requirements and considerations. Batch and online requirements are the same, except where differences are noted.

The CICS/VS application programmer requests DL/I services by issuing a DL/I call just as would be done in a batch environment (except in Assembler language, where CALLDLI must be used instead of CALL). All call function codes described for batch use are valid in the online environment.

In CICS/VS, if several transactions requiring the use of one message processing program are being serviced, one copy of the program is executed in a reentrant manner by several CICS/VS subtasks. Therefore, DL/I areas that can be modified, such as PCB pointers, segment I/O areas, and SSAs may not be placed in either static storage or working storage. In order to maintain quasi-reentrancy, storage for PCB pointers, SSAs, and segment I/O areas must either be obtained from CICS/VS dynamic storage or be defined in the transaction work area. The transaction work area is preferred for several reasons, the prime one being its ease of use.

The steps to request DL/I services are:

1. Obtain addresses of PCBs for use by the transaction by issuing a DL/I call with 'PCB' as the function code. This is described in the following discussion, "Obtaining the Address of the PCB: The Scheduling Call."

2. Acquire working storage for Assembler language programs: for COBOL and PL/I, this is automatically handled. For Assembler, there are several ways to reserve storage for the count field, function code field, I/O area, SSAs, and parameter list.

    a. The recommended approach is to reserve these in the transaction work area.

    b. You may issue one CICS/VS GETMAIN macro to hold all these areas.

    c. The least efficient method is to issue one CICS/VS GETMAIN macro for each area.

    As with all CICS/VS GETMAIN operations, storage account areas must be considered when techniques "b" and "c" are used.

3. Set the parameter count and function code as in the batch environment.

4. Furnish the PCB address provided by the 'PCB' call previously issued.

5. Issue the call.

## Obtaining the Address of the PCB: The Scheduling Call

Before accessing DL/I data bases, a CICS/VS program must issue a special DL/I call to initiate scheduling of a PSB. It is the responsibility of the programmer to determine the results of this call, details of which are given under "Checking the Response to a Scheduling or Termination Call".

The format of the scheduling call is:

**For COBOL:**

```
Call 'CBLTDLI' USING [parm-count,]
    call-function[,psbname[,uibparm]].
```

**For PL/I:**

```
CALL PLITDLI (parm-count,
    call-function[,psbname[,uibparm]]);
```

**For Assembler:**

```
CALLDLI ASMTDLI,([parm-count,]
    call-function[,psbname[,UIBPARM]])
```

parm-count
    is the name of a binary fullword containing the parameter count. This count equals 1, 2, or 3 depending on whether or not psbname and uibparm are specified. This parm-count parameter is optional in Assembler and COBOL.

call-function
    is the name of the field containing the 4-character function 'PCBɓ'

psbname
    is the name of the 8-byte field containing the 1- to 7-character PSB generation name (right padded with blanks) that the application program accesses. If uibparm is not used this parameter is optional, and, if omitted, the default is the first PSB name associated with the application program

name in the DL/I application control table generation.

uibparm
> is the name of a fullword to which DL/I returns the address of the User Interface Block. Use of this parameter is optional, but desirable in COBOL, PL/I, and Assembler calls. RPG II requires use of the User Interface Block. The User Interface Block (UIB) is a control block used to pass to the user the address of the PCB list and the scheduling and termination response and error codes. If this parameter is omitted, the PCB list address and response and error codes will be returned in fields in the CICS/VS task communication area (TCA).

**For RPG II:**

```
PCB        RQDLI            in
[PSBNAME   ELEM   psb-name]
SET        ELEM   BUIB
```

in
> indicator required in positions 56-57.

psb-name
> is the name of the PSB to be scheduled. It can be specified either as a var-name (a variable psb-name containing the psbname as a character string, as shown in [---] above) or as an alphameric literal constant

```
        PSBNAME    ELEM    'PSBNAM1'
```

> where PSBNAM1 is the name of the PSB.

BUIB
> refers to a field in DFHDUM that is the base of DLIUIB. User must specify such a field in DFHDUM to establish the addressability of PCBs after a scheduling call. (See Figure 3-7 for examples.)

If no PSB name is explicitly specified, the Translator will generate a PSB name of '*', which will cause the first PSB to be scheduled by default.

After a successful scheduling call, the field UIBPCBAL or TCADLPCB if the UIB is not used, or UAPCBL in RPG II, contains the address of a PCB list. The PCB list consists of a series of 4-byte addresses that point to the PCBs within the PSB that has been scheduled. The last address in the list is indicated by the high order bit being 1.

# Releasing a PSB in a CICS/VS Application Program: The Termination Call

To reduce intent contention, the CICS/VS application program should release the PSB when no more DL/I services are required by the program.

Conversational programs should release the PSB before writing output onto a terminal so that other transactions can use the PSB while the conversational program is waiting for an operator response. Before issuing any other DL/I CALLs requesting DL/I access to a data base, however, the application program must again schedule the PSB using a scheduling call. (This is not necessary if PI is used, however, it is probably a good practice to do so anyway.) If necessary, position in the data base must also be reestablished.

To release a PSB for use by other transactions, the program issues a call of the following format:

**For COBOL:**

```
CALL 'CBLTDLI' USING [parm-count,]
    call-function.
```

**For PL/I**

```
CALL PLITDLI (parm-count,
    call-function);
```

**For  Assembler:**

```
CALLDLI {ASMTDLI},([parm-count,]
        [CBLTDLI]
        call-function)
```

Note: The contents of registers 1, 14, and 15 are altered.

parm-count
> is the name of a fullword containing a binary value of 1.

call-function
> is the name of a 4-byte field containing the value 'TERM' or 'T' and three blanks.

**For RPG II:**

```
TERM RQDLI
```

Note: A 'T', or 'TERM' call causes a CICS/VS synchronization point. Also, a CICS/VS synchronization point causes a 'T' call if a PSB is still scheduled by the transaction. Refer to the section on Recovery Services in the *CICS/VS System Application Design Guide*, SC33-0068. However, in the latter case, storage used by DL/I is held by the application program until a "T" call is issued or the CICS/VS transaction ends. Therefore, the application programmer *should* issue a "TERM" call instead of a DFHSP macro.

# Checking the Response to a DL/I Call

When an application programmer issues a PCB, TERM, or data base call, he should check the response to his request using either of the two ways described below. In addition the application programmer should check the PCB status code field after all data base calls, as in batch application programs. However, the response field in the TCA or UIB must always be checked first to insure that the call was accepted.

1. Include code immediately following the call to examine the field UIBFCTR, or TCAFCTR (TCAFCRC in ANS COBOL) if the UIB is not used or UAPCBL in RPG II, and, based on its contents, transfer control if necessary to an exception-handling routine. The possible response codes in these fields are:

*Scheduling Call*

| Condition | Response Code | | | |
|---|---|---|---|---|
| | Assem. | ANS COBOL | PL/I | RPG II |
| NORESP Normal Response | X'00' | 12-0-1-8-9 | 00000000 | 00000000 |
| INVREQ Invalid Request | X'08' | 12-8-9 | 00001000 | 00001000 |
| NOTOPEN Not Open | X'0C' | 12-4-8-9 | 00001100 | 00001100 |

NORESP
: indicates that the requested function was completed normally and that the field UIBPCBAL, or TCADLPCB if the UIB is not used, or UAPCBL in RPG II contains the address of the PCB list.

INVREQ
: indicates that field UIBDLTR or TCADLTR if the UIB is not used, or UDLTR in RPG II, contains one of the following error codes:

X'01'
: PSB name, as provided in the scheduling call, is not in the PSB dictionary.

X'02'
: The calling program name is not defined in the DL/I application control table.

X'03'
: The calling program has already successfully issued a scheduling (PCB) call that has not been followed by a termination (TERM) call.

X'04'
: Either the calling program is written in PL/I and the language specified in the PSB is not PL/I; or the calling program is not written in PL/I, but the PSB specifies PL/I.

X'05'
: The PSB could not be initialized by DL/I online initialization.

X'06'
: The PSB in the scheduling call is not defined in the program's application control table entry.

X'09'
: An MPS batch program attempted to issue a PCB call for a read-only PSB or for a non-exclusive PSB if program isolation is active.

X'FF'
: The DL/I interface has been terminated or DL/I initialization failed.

NOTOPEN
: indicates that one or more DBD entries associated with this PSB are stopped or that a scheduling conflict with an MPS-scheduled task has occurred. Stopped means that the data base is not available for use because of an initialization error or an I/O error, or because it is closed. Conflict with an MPS-scheduled task means that a task running under MPS in a batch partition that has not done its own scheduling has update sensitivity to a segment for which update sensitivity has been requested.

Field UIBDLTR or TCADLTR if the UIB is not used, or UDTLR in RPG II, contains one of the following error codes:

X'01'
: One or more DBD entries associated with the PSB are stopped.

X'02'
: A scheduling conflict with a currently active MPS batch partition occurred.

*Data Base or Termination Call*

| Condition | Response Code | | | |
|---|---|---|---|---|
| | Assem. | ANS COBOL | PL/I | RPG II |
| NORESP Normal Response | X'00' | 12-0-1-8-9 | 00000000 | 00000000 |
| INVREQ Invalid Request | X'08' | 12-8-9 | 00001000 | 00001000 |

**NORESP**

indicates that the DL/I resources have been re-leased.

**INVREQ**

indicates that the field UIBDLTR, or TCADLTR if the UIB is not used, or UDLTR in RPG II, contains one of the error codes that are listed together with their explanations below:

X'07' "TERM" requested but task not scheduled.
X'08' A DL/I call was made but the task has not scheduled a PSB.
X'FF' The DL/I interface has been terminated or DL/I initialization failed.

If a DL/I task abnormal termination occurs during online processing, control is not returned to the application program and the transaction is terminated with a CICS/VS message. In that message, the numeric part of the code that follows the word ABEND corresponds to the numeric portion of the applicable DL/I message number as listed in Chapter 3 of *DL/I DOS/VS Messages and Codes*. The code normally begins with D but it begins with E if the termination cannot be noted on the transient data destination CSMT.

2. Include the DFHFC TYPE=CHECK macro immediately following the call. The operands that are appropriate for checking the CICS/VS-DL/I interface response and their meanings are summarized in the following table:

| | DFHFC | TYPE=CHECK |
|---|---|---|
| | | [,NORESP=symbolic address] |
| | | [,INVREQ=symbolic address] |
| | | [,NOTOPEN=symbolic address] |

**TYPE=CHECK**

is always coded.

**NORESP**

specifies the entry label of a user-written routine to which control is passed upon normal execution of the request.

**INVREQ**

specifies the entry label of the user-written routine to which control is passed if the application program has not scheduled a PSB and obtained PCB addresses.

**NOTOPEN**

specifies the entry label of the user-written routine to which control is passed if the data base specified in the PCB used in this request is stopped. Stopped means that the data base is not available for use because of an initialization error or an I/O error, or because it is closed. The PCB does not contain an AI status code.

If a DL/I task abnormal termination occurs during online processing, control is not returned to the application program and the transaction is terminated with a CICS/VS message. In that message, the numeric part of the code that follows the word ABEND corresponds to the numeric portion of the applicable DL/I message number as listed in the DL/I System Messages chapter of the *Messages and Codes*. The code normally begins with D but it begins with E if the termination cannot be noted on the transient data destination CSMT.

### MPS (Multiple Partition Support) Considerations

An online program receives a return code from a PCB call if it conflicts with an MPS batch job, instead of waiting. In this case, UIBFCTR, or TCAFCTR (TCAFCRC in ANS COBOL), or UFCTR in RPG II contains a X'0C' and UIBDLTR or TCADLTR or UDLTR in RPG II contains a X'02'. When the data base is not open, the fields will contain X'0C' and X'01' respectively.

If any online tasks must wait for a resource owned by a batch MPS task, the MPS task will be informed on the next and all subsequent calls until a DL/I checkpoint is issued. (Note that making the resource available through some other action makes no difference. The passback indicates that a wait was required, not necessarily that a task is currently waiting.) This condition is indicated by setting the high-order bit of the first byte of the "JCB Address" field in the PCB to a one (X'80'). The application program must test for this condition by examining the field in the PCB mask that is reserved for DL/I. In COBOL, it is labeled "RESERVE-DL/I"; in PL/I, "RESERVE DL/I; in RPG II, "RESRij"; and in Assembler, "DBPCBSRV".

MPS batch jobs not using program isolation are permitted to issue PCB and TERM calls. This allows tasks conflicting with the batch job to run before the batch job completes. However, this practice is not recommended because of the following restrictions:

1. The first PCB call is issued automatically by DL/I so before an MPS batch job issues its first PCB call, it must issue a TERM call.

2. The PSB name used by an MPS batch job in a PCB call must always be the one specified in the DL/I parameter statement. This PSB must not be a read-only PSB. The PCB addresses are the same as at the start of the application program. These addresses should be used after a PCB call.

3. The format of the PCB and TERM calls are the same as in online execution except the CALL macro is used instead of CALLDLI.

4. The user must not issue PCB calls and TERM calls for a read-only PSB.

5. There is no feedback information passed to the program. The MPS batch program request handler intercepts the return code, and if it is non-zero, it will ABEND the batch job.

6. After an MPS batch job has successfully issued its own PCB call, it will be considered to be an online task from a scheduling viewpoint.

**Notes:**

- If PCB and TERM calls are used by MPS batch jobs, the jobs must not be run in a non-MPS batch environment or in an MPS batch environment using program isolation. The use of these calls is also not upward compatible with IMS/VS, that is these calls are not permitted in batch IMS/VS.

- Using CICS/VS Intersystem Communication Support, DL/I application programs can access a data base that is resident on another CICS/VS system. The application program, except in the following situation, need not be aware of where the data base is located: If your MPS batch application program is to run on a system where Intersystem Communication support is active, it must not issue 'PCB' calls. If issued, such calls would receive an abnormal return code of X'08' in UIBFCTR, or TCAFCTR (TCAFCRC in ANS COBOL) if the UIB is not used, or UFCTR in RPG II; and X'09' in UIBDLTR, or TCADLTR if the UIB is not used, or UDLTR in RPG II.

## *Issuing the DL/I Call*

DL/I data base services are available to CICS/VS application programs through call statements. The call statement formats for ANS COBOL and PL/I are similar. For assembler language application programs, a CALLDLI macro instruction is used. The general formats of the DL/I calls are as follows:

**For COBOL:**

```
CALL 'CBLTDLI' USING [parm-count,]
                     call function,
                     db-pcb-name,
                     i/o-area[,
                     ssa...].
```

**For PL/I:**

```
CALL PLITDLI (parm-count,call-function,
             db-pcb-name,i/o-area[,
             ssa...]);
```

**For RPG II:**
The format of the RQDLI commands in RPG II is the same as in the batch environment, see Chapter 4, "RQDLI Commands for DB Access".

**For Assembler:**
```
CALLDLI {ASMTDLI,}([parm-count,]
        {CBLTDLI}
        call-function,db-pcb-name,
        i/o-area[,ssa...])
```

parm-count
is the name of a binary fullword containing the parameter count. For COBOL and Assembler it is optional.

call-function
is the name of the field containing the 4-character DL/I call function desired.

db-pcb-name
is the name of the PCB (or DSECT if Assembler).

i/o-area
is the name of the I/O area.

ssa...
are the names of the SSAs; these parameters are optional.

**Notes:**

1. If no parameters are specified in an Assembler language CALLDLI macro instruction, register 1 is assumed to contain the address of a parameter list.
2. In Assembler language, the following format may be used as an alternative.
   ```
   CALLDLI {ASMTDLI},MF=(E,{(register)})
           {CBLTDLI}        {address }
   ```
   Register contains the address of the parameter list. Address is the address of the parameter list.
   Register 13 must contain the address of a 72-byte user-provided save area. The CALLDLI macro alters the contents of registers 1, 14, and 15.
3. If the application program makes a DL/I data base call without previously making a successful scheduling call, a 1-byte response code (X'08') is placed in the field UIBFCTR or TCAFCTR (TCAFCRC in ANS COBOL) or UFCTR in RPG II indicating an invalid request. If the call is accepted, the field is set to binary zeros, however, the user must still check the DL/I PCB status code.

## *Online Application Coding Examples*

The following examples assume the application programmer has a thorough understanding of CICS/VS coding requirements and techniques. The examples, therefore, only illustrate the use of the DL/I portions of the application programs.

## *DL/I Requests in an ANS COBOL Program*

The PCB addresses must be obtained upon entry by issuing a scheduling call. After CICS/VS returns the control to the application program, the programmer moves the contents of UIBPCBAL, or TCADLPCB if the UIB is not used, to the BLL pointer which is the base for the layout of the PCB pointers in the linkage section. He then moves the addresses of the PCBs to their BLL pointers to provide the base addresses for the PCBs. When this has been done, the program is in the same state as a DL/I DOS/VS batch application program in which the following statement has been executed.

```
ENTRY 'DLITCBL' USING PCB1,PCB2.
```

The following examples show how to code DL/I. The first example shows how to write DL/I requests in an ANS COBOL program when the UIB is used. The second example is for when the UIB is not used. Only some of the possible combinations of operands are shown, but other combinations are acceptable.

```
COBOL  Example  1  (UIB  used)
IDENTIFICATION DIVISION.
PROGRAM-ID. DLIEIB.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
77   PSBNAME PICTURE X(8) VALUE 'PSBNAME'.
77   PCB-FUNCTION PICTURE X(4) VALUE 'PCB '.
77   FUNCTION-1 PICTURE X(4) VALUE 'DLET'.
01   WORKA1  PICTURE X(40).
01   SSAREA.
     02 SSA1 PICTURE X(40).
     02 SSA2 PICTURE X(60).
LINKAGE SECTION.
01   BLLCELLS.
     02 FILLER       PICTURE 9(8) COMP.
     02 UIP-PTR      PICTURE 9(8) COMP.
     02 B-PCB-PTRS PICTURE 9(8) COMP.
     02 B-PCB1       PICTURE 9(8) COMP.
     02 B-PCB2       PICTURE 9(8) COMP.
*    COPY UIB DEFINITION - HERE IT IS EXPANDED BY HAND
01   UIB.
     02 UIBPCBAL     PICTURE 9(8) COMP.
     02 UIBFCTR      PICTURE X(1).
     02 UIBDLTR      PICTURE X(1).
01   PCB-PTRS.
     02 PCB1-PTR     PICTURE 9(8) COMP.
     02 PCB2-PTR     PICTURE 9(8) COMP.
01   PCB1.
     •
     •
     •
01   PCB2.
     •
     •
     •
 PROCEDURE DIVISION.
*    GET PCB ADDRESSES
     CALL 'CBLTDLI' USING PCB-FUNCTION, PSBNAME, UIB-PTR.
*    MOVE ADDRESS OF PCB ADDRESS LIST INTO BLL SLOTS
*    SO PCB ADDRESS LIST CAN BE ADDRESSED
     MOVE UIBPCBAL TO B-PCB-PTRS.
*    MOVE PCB ADDRESSES INTO BLL SLOTS SO PCB'S CAN BE ADDRESSED
     MOVE PCB1-PTR TO B-PCB1.
     MOVE PCB2-PTR TO B-PCB2.
     •
     •
     •
*    ISSUE DL/I REQUEST
     CALL 'CBLTDLI' USING FUNCTION-1, PCB1, WORKA1, SSA1, SSA2.
*    TEST CICS/VS-DL/I INTERFACE RESPONSE
     IF UIBFCTR IS NOT EQUAL LOW-VALUES THEN GO TO ERROR1.
*    TEST DL/I RESPONSE
     IF STATUS-CODE IS NOT EQUAL SPACES THEN GO TO ERROR2.
     GOBACK.
```

```
COBOL Example 2 (UIB not used)
WORKING STORAGE SECTION
77 PCB-FUNCTION PICTURE X(4) VALUE 'PCBb'.
77 PSBNAME PICTURE X(8) VALUE 'COBOLPSB'.
77 FUNCTION-1 PICTURE X(4) VALUE 'DLET'.
77 SSA-COUNT PICTURE S9(8) COMPUTATIONAL VALUE + 2.
LINKAGE SECTION.
01 DFHBLLDS COPY DFHBLLDS.
   02 ...                         NOTE POINTERS TO OTHER CICS/VS
*                                 AREAS NEEDED
   02 B-PCB-PTRS PICTURE S9(8) COMPUTATIONAL.
   02 B-PCB1 PICTURE S9(8) COMPUTATIONAL.
   02 B-PCB2 PICTURE S9(8) COMPUTATIONAL.
   02 B-WORKAREA PICTURE S9(8) COMPUTATIONAL.
   02 B-SSAs PICTURE S9(8) COMPUTATIONAL.
01 DHFCSADS COPY DFHCSADS.
01 DHFTCADS COPY DFHTCADS.
   •                              NOTE TWO DEFINITIONS.
   •                              NOTE OTHER AREA DEFINITIONS.
01 PCB-PTRS.
   02 PCB1-PTR PICTURE S9(8) COMPUTATIONAL.
   02 PCB2-PTR PICTURE S9(8) COMPUTATIONAL.
01 PCB1.
   •
   •
   •
01 PCB2.
   •
   •
   •
01 WORKAREA.
   02 FILLER PICTURE X(8).    STORAGE PREFIX
   02 WORKA1 PICTURE X(40).
   •
   •
   •
01 SSAREA.
   02 FILLER PICTURE X(8).
   02 SSA1 PICTURE X(40).
   02 SSA2 PICTURE X(60).
   •
   •
   •
PROCEDURE DIVISION.
* GET PCB ADDRESSES
CALL 'CBLTDLI' USING PCB-FUNCTION
    DFHFC TYPE=CHECK,NOTOPEN=OPENERR,INVREQ=OTHERERR
* SAVE PCB ADDRESSES IN BLL TABLE SO PCB'S CAN BE ADDRESSED
    MOVE TCADLPCB TO B-PCB-PTRS
    MOVE PCB1-PTR TO B-PCB1
    MOVE PCB2-PTR TO B-PCB2.
* OPTIONALLY, ACQUIRE STORAGE FOR WORK AREA
    DFHSC TYPE=GETMAIN,...
    MOVE TCASCSA TO B-WORKAREA.
* OPTIONALLY, ACQUIRE STORAGE FOR SEGMENT SEARCH ARGUMENTS
    DFHSC TYPE=GETMAIN,...
    MOVE TCASCSA TO B-SSAS.
* CALL DL/I VIA CALL
    CALL 'CBLTDLI' USING FUNCTION-1, PCB1, WORKA1, SSA1, SSA2.
```

## DL/I Requests in a PL/I Program

The PCB addresses must be obtained upon program entry by issuing a scheduling call. When CICS/VS returns control to the application program, the base address of a structure of PCB pointers is in UIBPCBAL, or TCADLPCB if the UIB is not used. The PL/I programmer must move the value from TCADLPCB to the based variable for his declared structure of PCB pointers. He then loads the pointers to all PCBs from this structure. When this has been done, the program is in the same state as a DL/I DOS/VS batch application program in which the following statement has been executed.

```
DLITPLI: PROCEDURE (pcbname1,...)
         OPTIONS (MAIN);
```

The PL/I programmer may then make DL/I requests, as the following examples show.

```
PL/I  Example  1  (UIB  used)
TEST:   PROC OPTIONS (MAIN,RE-ENTRANT);
DCL 1   DLIO,
        2               DESCRIBE DL/I I/O AREA
        2
DCL 1   SSA,
        2               DESCRIBE SSA
        2
DCL     COUNT FIXED BIT (31);
DCL     FUNCTION CHAR (4);
DCL     UIBPTR PTR;
% INCLUDE DLIUIB;
/*  PRODUCES THE FOLLOWING         */
/*                                 */
/*  DLC 1 UIB BASED (UIBPTR),      */
/*         2 UIBPCBAL PTR,         */
/*         2 UIBRCODE CHAR (2),    */
/*           3 UIBFCTR CHAR (1),   */
/*           3 UIBDLTR CHAR (1);   */
/*                                 */
DCL 1 PCB_ADDR BASED (UIBPCBAL),
      2 PCB1_PTR,
      2 PCB2_PTR;
DCL 1 PCB_MAST     BASED  (PCB1_PTR),
    2 MAST_DB_NAME  CHAR (8),
    2 MAST_SEG_LEV  CHAR (2),

      _
      _
      _
    2 MAST_KEY_FB   CHAR(*);
PCB_CALL:
   COUNT=3;
   FUNCTION='PCB ';
   CALL PLITDLI (COUNT,FUNCTION,,UIBPTR);
   IF UIBFCTR¬= '00000000'B THEN GOTO
       DATA_BASE_ERROR_RTN;
SET_UP_CALL:
   COUNT=4;
   FUNCTION='GU  ';
   CALL PLITDLI (COUNT,FUNCTION,PCB1_PTR,DLIO,SSA);
```

**PL/I  Example  2  (UIB  not  used)**

```
%INCLUDE DFHCSADS;                              /*CSA DEFINITION */
%INCLUDE DFHTSADS;                              /*CSA DEFINITION-INCLUDES*/
                                                /*DL/I FIELDS*/
     DECLARE 1 PCB POINTERS BASED (B_PCB_PTRS),
             2 PCB1_PTR POINTER,
             2 PCB2_PTR POINTER;
     DECLARE 1 PCB1 BASED (BPCB1),
             2 ...,
             2 ...;

       _
       _
       _
     DECLARE 1 DLI_IOAREA BASED (BDLIIC) , /* DL/I I/O AREA */
             2 STORAGE_PREFIX CHAR (8),
             2 IOKEY CHAR (6),
             2 ... ;
     DECLARE 1 DLI_SSADS BASED (BSSADS), /* DL/I SSA LIST */
             2 STORAGE_PREFIX CHAR (8),
             2 SSA1,
               3 SSA1SEG CHAR (8),
               3 ...,
               3 ...;
     DECLARE   SSADEF CHAR (20) DEFINED SSA1;
     DECLARE   DLI_FUNCTION CHAR (4) INIT('GU  ');
     DECLARE   PARM_CT        BIN FIXED(31)  INIT(5);
/* OBTAIN PCB POINTERS */
   CALL PLITDLI (parmcount,'PCB ');
       DFHC TYPE=CHECK,NOTOPEN=OPENERR,INVREQ=OTHERERR
/* SAVE POINTERS IN PCB BASES */
   B_PCB_PTRS=TCADLPCB;
   BPCB1=PCB1_PTR;
   BPCB2=PCB2_PTR;
/* ACQUIRE STORAGE FOR DL/I I/O AREA */
       DFHSC TYPE=GETMAIN,CLASS=USER,...
       BDLIIO=TCASCSA;
/* OPTIONALLY ACQUIRE STORAGE IN WHICH TO BUILD SSA'S */
       DFHSC TYPE=GETMAIN,CLASS=USER,...
       BSSADS=TCASCSA;
/* OPTIONALLY BUILD SEGMENT SEARCH ARGUMENTS */
       SSA1SEG=SEGNAME
           •
           •
           •
/* CALL DL/I */
       CALL PLITDLI (PARM-CT,DLI-FUNCTION,PCB1,IOKEY,SSADEF,
           SSA2);
```

## Requests in an Assembler Language Program

The application programmer must first obtain the PCB addresses. The following examples show the options available to the application programmer in a few of the acceptable combinations. Note that the application program must be quasi-reentrant.

```
Assembler   Example  1  (UIB  used)
R4              EQU    4
R7              EQU    7
PCBPTRS         DSECT
PCB1PTR         DS     A
PCB2PTR         DS     A
DFHEISTG        DSECT
UIBPTR          DS     A
IOAREA          DS     CL100
                DLIUIB
* PRODUCES THE FOLLOWING:                                        *
*****************************************************************
* DLIUIB          DSECT                                          *
* UIB             DS     0F          EXTENDED CALL USER INTFC BLK *
* UIBPCBAL        DS     A           PCB ADDRESS LIST            *
* UIBRCODE        DS     0XL2        DL/I RETURN CODES           *
* UIBFCTR         DS     X           RETURN CODE                 *
* UIBDLTR         DS     X           ADDITIONAL INFORMATION      *
*                 DS     2X          RESERVED FOR DL/I           *
*                 DS     0F          LENGTH OF FULLWORD MULTIPLE  *
* UIBLEN          EQU    *-UIB       LENGTH OF UIB               *
*****************************************************************
* USER SHOULD ADD CSECT INSTRUCTION IF NO FURTHER DSECTS FOLLOW
PCBFUN          DC     CL4'PCB '
GNFUN           DC     CL4'GN  '
                USING UIB,R7
                •
                •
                •
                CALLDLI ASMTDLI,(PCBFUN,,UIBPTR)

                L      R7,UIBPTR

                CLI    UIBFCTR,0            IF BAD RETURN CODE,
                BNE    SCHEDERR             .. BRANCH TO ERROR ROUTINE
                USING PCBPTRS,R4

                L      R4,UIBPCBAL          GET ADDR OF PCB LIST

                CALLDLI ASMTDLI,(GNFUN,PCB1PTR,IOARA)
```

## Assembler Example 2 (UIB not used)

```
// JOB ASSEMBL
// OPTION CATAL
    PHASE PITRAN,*
// EXEC ASSEMBLY
EXAMPLE  CSECT
R0        EQU    0
R1        EQU    1
R3        EQU    3
R4        EQU    4
R9        EQU    9
R12       EQU    12
R13       EQU    13
          BALR   R3,R0                 LOAD BASE REGISTER (R3)
          USING  *,R3                  ...AND TELL ASSEMBLER
          USING  DFHTCADS,R12          TELL ASSEMBLER ABOUT TCA
          USING  DFHCSADS,R13          ...AND CSA ADDRESSABILITY
*           •
*         ESTABLISH ADDRESSABILITY TO OTHER CICS/VS AREAS
*         AS REQUIRED BY THE APPLICATION PROGRAM
*           •
*         -----         SET UP AND ISSUE SCHEDULING (PCB) CALL
*                       (MF=E FORM OF CALLDLI MACRO DEMONSTRATED)
          LA     R1,COUNT              SET DL/I COUNT PARAMETER
          ST     R1,COUNTADR           ...ADR IN CALL PARM LIST
          LA     R1,PCB                GET ADR OF PCB FUNCTION CODE
          ST     R1,FUNADR             ...AND STORE IT IN PARM LIST
*         -----         OPTIONALLY SPECIFY NAME OF PSB TO BE SCHEDULED
          LA     R1,PSBNAME            GET ADR OF NAME OF PSB TO SCHED
          ST     R1,PCBADR             ...AND STORE IT IN PARM LIST
          MVC    COUNT,=F'2'           SET PARM COUNT = 2
*                                      IF PSB NAME WAS NOT SPECIFIED
*                                      ...COUNT SHOULD BE SET TO ONE
          LA     R1,DLIPARMS           POINT R1 AT PARM LIST
          ST     R13,CSASAVE           SAVE CSA ADR PRIOR TO MF=E
*                                      ...CALLDLI MACRO FORMAT USAGE
          LA     R13,CALLSAVE          PUT ADR OF SAVE AREA IN R13
*                                      ...PRIOR TO USING MF=E CALLDLI
*                                      ...MACRO FORMAT (CSA ADR LOST)
          CALLDLI ASMTDLI,MF=(E,(1))   ISSUE PCB CALL (MF=E FORMAT)
          L      R13,CSASAVE           RECOVER CSA ADR AFTER MF=E
*                                      ...CALLDLI MACRO FORMAT USAGE
*         -----         CHECK SUCCESS OF SCHEDULING CALL - METHOD 1
          CLI    TCAFCTR,X'00'         CALL SUCCESSFUL?
          BNE    SCHERROR              ...NO, GO DETERMINE PROBLEM
*         -----         CHECK SUCCESS OF SCHEDULING CALL - METHOD 2
          DFHFC  TYPE=CHECK,INVREQ=SCHERROR,NOTOPEN=SCHERROR
*         -----         SCHEDULE CALL OK, ESTABLISH ADDRESSABILITY TO PCBS
          L      R9,TCADLPCB           GET ADR OF PCB ADDRESSES
          USING  PCBADRS,R9            ...AND TELL ASSEMBLER
          L      R4,PCB1ADR            GET ADR OF 1ST PCB IN PSB
          USING  PCB1,R4               ...AND TELL ASSEMBLER
*           •
*         ESTABLISH ADDRESSABILITY TO THE REMAINING PCBS IN THE PSB
*         AND CONTINUE WITH APPLICATION PROGRAM LOGIC
*           •
*         -----         INITIALIZE SSAS
          MVC    COMCODE1,=C'*--'      SET NULL COMMAND IN 1ST SSA
          MVI    RP1,C')'              ...AND ENDING RIGHT PAREN
*           •
*         -----         SET UP TO RETRIEVE A SEGMENT
          MVC    SEGNAME1,=CL8'ROOT'   PUT SEG NAME IN SSA
          MVI    LP1,C'('              MAKE CALL QUALIFIED
          MVC    KEYNAME1,=CL8'SEQ'       ...PUT KEY FIELD NAME
          MVC    RO1,=C'= '
          MVC    KEY1,=CL5'00001'      ...AND KEY FIELD VALUE IN SSA
          MVC    COUNT,=F'4'           INDICATE 4 PARMS USED IN CALL
          CALLDLI ASMTDLI,(COUNT,GU,PCB1,SEGIO,SSA1)
*         -----         CHECK FOR CALL ACCEPTANCE - METHOD 1
          CLI    TCAFCTR,X'00'         WAS CALL ACCEPTED?
          BNE    CALLERR               ...NO, GO DETERMINE REASON
*         -----         CHECK FOR CALL ACCEPTANCE - METHOD 2
          DFHFC  TYPE=CHECK,INVREQ=CALLERR,NOTOPEN=CALLERR
```

```
*          •
*          CALL WAS ACCEPTED.  CHECK DL/I PCB STATUS CODE
*          AND CONTINUE APPLICATION PROGRAM LOGIC
*          •
           DFHPC    TYPE=RETURN
*          -----    DL/I CALL ERROR ROUTINES
CALLERR  DS         0H
SCHERROR DS         0H
*          AT THIS POINT THE PROGRAM CAN DETERMINE THE REASON FOR
*          THE ERROR BY EXAMINING THE FIELD 'TCADLTR'.  IN MOST
*          CASES A CICS/VS ABEND SHOULD BE ISSUED.
           DFHPC    TYPE=RETURN
*          •
*          •
*          -----    DL/I ONLINE FUNCTION CODE CONSTANTS
*                   (COULD BE A COPY BOOK)
PCB      DC         CL4'PCB'
GU       DC         CL4'GU'
GHU      DC         CL4'GHU'
GN       DC         CL4'GN'
GHN      DC         CL4'GHN'
GNP      DC         CL4'GNP'
GHNP     DC         CL4'GHNP'
REPL     DC         CL4'REPL'
ISRT     DC         CL4'ISRT'
DLET     DC         CL4'DLET'
TERM     DC         CL4'TERM'
*          -----    MISCELLANEOUS PROGRAM CONSTANTS
PSBNAME  DC         CL8'PIPSBA1'          NAME OF PSB TO BE SCHEDULED
TWALEN   DC         A(TWASTOP-TWASTART) LENGTH OF TWA REQUIRED
*          •
*          OTHER PROGRAM CONSTANTS
*          •
*          -----    CICS/VS DSECTS
           COPY DFHCSADS
           COPY DFHTCADS
*          -----    TWA STARTS HERE
TWASTART EQU        *
CSASAVE  DS         F                     CSA ADR SAVE AREA FOR MF=E CALLS
CALLSAVE DS         18F                   REG SAVE AREA FOR MF=E CALLS
DLIPARMS DS         0F                    DL/I CALL PARM LIST
*                                         FOR USER CREATED CALL PARM LISTS
COUNTADR DS         A                     ADR OF PARM COUNT VALUE
FUNADR   DS         A                     ADR OF FUNCTION CODE
PCBADR   DS         A                     ADR OF PCB USED WITH CALL
IOADR    DS         A                     ADR OF SEGMENT I/O AREA USED
SSA1ADR  DS         A                     ADR OF 1ST SSA USED IN CALL
SSA2ADR  DS         A                     ADR OF 2ND SSA USED IN CALL
*          •
*          •
*          -----    SSAS (COULD BE COPY BOOKS)
SSA1     DS         0CL29
SEGNAME1 DS         CL8                   SEGMENT NAME
COMCODE1 DS         CL3                   COMMAND CODE AREA OF SSA
LP1      DS         CL1                   LEFT PAREN '('
KEYNAME1 DS         CL8                   SEGMENT KEY FIELD NAME
RO1      DS         CL2                   RELATIONAL OPERATOR
KEY1     DS         CL5                   KEY FIELD VALUE
RP1      DS         CL1                   SSA ENDING RIGHT PAREN
*          -----    MISCELLANEOUS WORKING STORAGE AREAS
COUNT    DS         F                     NUMBER OF PARMS IN LIST
SEGIO    DS         0CL40                 A SEGMENT I/O AREA (COPY BOOK?)
ROOTKEY  DS         CL6                   ROOT KEY FIELD IN ROOT SEGMENT
*          •
*          DEFINITIONS OF OTHER FIELDS IN SEGMENT
*          •
*          •
*          DEFINITIONS OF OTHER WORKING STORAGE AREAS
*          REQUIRED BY PROGRAM
*          •
TWASTOP  EQU        *                     END OF TWA
*          -----    DSECT USED TO ESTABLISH ADDRESSABILITY TO PCBS
*                   (COULD BE A COPY BOOK FOR EACH PSB IN INSTALLATION)
PCBADRS  DSECT
```

```
PCB1ADR   DS        A                       ADR OF 1ST PCB IN PSB
PCB2ADR   DS        A                       ADR OF 2ND PCB IN PSB
*         •
*         CONTINUE FOR AS MANY PCBS IN PSB
*         •
PCB1      DSECT     (COULD BE CONTINUATION OF PSB COPY BOOK)
PCB1DBDN DS         CL8                      DBD NAME
PCB1LEV  DS         CL2                      LEVEL FEEDBACK
PCB1STC  DS         CL2                      STATUS CODE
PCB1PRO  DS         CL4                      PROCESSING OPTIONS
         DS         F                        RESERVED
PCB1SFD  DS         CL8                      SEGMENT NAME FEEDBACK
PCB1KFDL DS         F                        CURRENT LENGTH OF KEY FEEDBACK
PCB1NSS  DS         F                        NUMBER OF SENSITIVE SEGMENTS
PCB1KFD  DS         CL255                    KEY FEEDBACK AREA
*         •
*         CONTINUE FOR AS MANY PCBS IN PSB
*         •
*         COPY OTHER CICS/VS DSECTS AS REQUIRED BY PROGRAM
*         •
          END
/*
// EXEC LNKEDT
/&
```

## RQDLI Commands in an RPG II Program

The following lists all the peculiarities for RPG II applications using DL/I under CICS/VS.

1. File Description Specifications for DB-files may be specified and have the same format as in a batch environment. Their implication on the RQDLI command for standard data transfer is the same.

   The RQDLI commands have the same format as those specified for the batch environment in Chapter 1.

**Exception:** For a scheduling call (func-name=PCB), additionally a SET option and a PSB-name option are supported.

2. *ENTRY PLIST for DL/I under CICS/VS.
   In addition to the PARMs required by CICS/VS, additional parameters for DLIUIB, the PSB, and PCBs have to be specified by the user. The bases for those parameters must also be specified in DFHDUM, in the same order as in the *ENTRY PLIST.

   An example of an online RPG II application program follows:

```
I*THE LAYOUT IN DFHDUM MUST EXACTLY CORRESPOND TO THE
I*LAYOUT OF THE *ENTRY PLIST STARTING WITH DFHDUM
IDFHDUM        DS
I                                       1    4 SELPTR
I                                       5    8 BUIB
I                                       9   12 BPSB
I                                      13   16 BPCB01
I                                      17   20 BPCB02
I
IPCB01         DS
I ............
IPCB02         DS
I ............
I* USER MUST SPECIFY THE PROPER LAYOUT OF THE PCBS
I* PSB DEFINES THE ADDRESSLIST OF THE PCBADDRESSES
IPSB           DS
I                                       1    4 PCB01P
I                                       5    8 PCB02P
I/INSERT   .DLIUIB
I* See ''/INSERT Statement in RPG II'' following this example.
I* THE FOLLOWING DATA STRUCTURE FOR THE UIB CONTROL BLOCK
I* WILL BE INSERTED FROM THE LIBRARY BY THE TRANSLATOR
IDLIUIB        DS
I                                       1    4 UPCBAL
I                                       5    5 UFCTR
I                                       6    6 UDLTR
I                                           ..............
I* END OF THE INSERTED UIB CONTROL BLOCK
```

```
C*   USER HAS TO SPECIFY AT LEAST THE PARAMETERS AFTER DFHDUM
C              *ENTRY     PLIST
C                         PARM              DFHEIB
C                         PARM              DFHCOM
C                         PARM              DFHDUM
C                         PARM              DLIUIB
C                         PARM              PSB
C                         PARM              PCB01
C                         PARM              PCB02
C*  BEFORE ACCESSING THE DATA BASE THE FOLLOWING
C*  STATEMENTS MUST BE CODED BY THE USER
C              PCB        RQDLI                     13
C              SET        ELEM              BUIB
C              PSBNAME    ELEM   'PSBNAM1'
C*     PSBNAME OPTION MUST BE SPECIFIED IN
C*     AN ELEM STATEMENT
C* THIS ESTABLISHES THE ADDRESSABILITY OF THE DATA STRUCTURE
C* DLIUIB AND ITS FIELDS
C* WITH THE HELP OF A MOVE STATEMENT THE ADDRESS OF THE PCBADDRESS
C* LIST IS PUT INTO THE BASE OF THE DS DEFINING THE ADDRESSLIST
C                         MOVE UPCBAL       BPSB
C                         CALL 'ILNSAD'
C* CHECK CICS/VS INTERFACE RESPONSE
C                         TESTB'01234567'UFCTR       10
C    10                   GOTO NORESP
C                         TESTB'4'          UFCTR          10
C                         TESTB'5'          UFCTR          11
C    10N11                GOTO INVREQ
C    10 11                GOTO NOTOPEN
C* DATA STRUCTURE CONTAINING THE ADDRESS LIST OF THE PCBS IS
C* NOW ADDRESSABLE IN THIS CASE PSBNAM1 CONTAINS ONLY
C* PCB01
C* ESTABLISH THE ADDRESSABILITY FOR PCB01 BY MOVE STATEMENT
C* FOLLOWED BY CALL TO ILNSAD
C                         MOVE PCB01P       BPCB01
C                         CALL 'ILNSAD'
C* NOW THE USER CAN ACCESS THE DATA BASE PCB01
C              GU         RQDLI                     13
C              PCB        ELEM              PCB01
C              INTO       ELEM              IOAREA200
C*     CONTINUE WITH PROGRAM
C              TERM       RQDLI
C* NOW PCB01 CAN NO LONGER BE ADDRESSED
C...........
C* BEFORE ACCESSING A NEW DATA BASE THE FOLLOWING
C* STATEMENTS MUST BE CODED BY THE USER
C              PCB        RQDLI                     13
C              SET        ELEM              BUIB
C              PSBNAME    ELEM   'PSBNAM2'
C*     PSBNAME OPTION MUST BE SPECIFIED IN
C*     AN ELEM STATEMENT
C* THIS ESTABLISHES THE ADDRESSABILITY OF THE DATA STRUCTURE
C* DLIUIB AND ITS FIELDS
C                         MOVE UPCBAL       BPSB
C                         CALL 'ILNSAD'
C* DATA STRUCTURE CONTAINING THE ADDRESS LIST OF THE PCBS IS
C* NOW ADDRESSABLE IN THIS CASE PSBNAM2 CONTAINS ONLY
C* PCB02
C* ESTABLISH THE ADDRESSABILITY FOR PCB02 BY MOVE STATEMENT
C* FOLLOWED BY CALL TO ILNSAD
C                         MOVE PCB01P       BPCB02
C                         CALL 'ILNSAD'
C* NOW THE USER CAN ACCESS THE DATA BASE PCB02
C              GU         RQDLI                     13
C              PCB        ELEM              PCB02
C    ..............
C              TERM       RQDLI
C* NOW USER CAN NO LONGER ACCESS PCB02
C* BEFORE ACCESSING A NEW DATABASE THE FOLLOWING
C* STATEMENTS MUST BE CODED BY THE USER
C              PCB        RQDLI                     13
C              SET        ELEM              BUIB
C              PSBNAME    ELEM   'PSBNAM3'
C* PSBNAME OPTION MUST BE SPECIFIED IN
```

```
C* AN ELEM STATEMENT (PSBNAM3 SCHEDULES PCB01 AND PCB02)
C* THIS ESTABLISHES THE ADDRESSABILITY OF THE DATA STRUCTURE
C* DLIUIB AND ITS FIELDS
C                        MOVE UPCBAL      BPSB
C                        CALL 'ILNSAD'
C* DATA STRUCTURE CONTAINING THE ADDRESS LIST OF THE PCBS IS
C* NOW ADDRESSABLE IN THIS CASE PSBNAM3 CONTAINS
C* PCB01 AND PCB02
C* ESTABLISH THE ADDRESSABILITY FOR BOTH BY MOVE STATEMENTS
C* FOLLOWED BY CALL TO ILNSAD
C                        MOVE PCB01P      BPCB01
C                        MOVE PCB02P      BPCB02
C                        CALL 'ILNSAD'
C* NOW THE USER CAN ACCESS THE DATA BASES PCB01 AND PCB02
C        GU         RQDLI                      13
C        PCB        ELEM            PCB02
C        ...........
C* CONTINUE WITH PROGRAM
```

## /INSERT Statement in RPG II

/INSERT may be used to include data structures, pro-
gram pieces, etc. from source statement libraries. The
inserted text must be *untranslated* source and must not
itself contain /INSERT statements.

Note: For RPG II the /INSERT statement is a facility of the Trans-
lator and, therefore, the inserted text is untranslated source. For
COBOL, PL/I, and Assembler, COPY or INCLUDE is a language
facility and, therefore, the inserted text is translated source.

Format of the /INSERT Statement:

| Position | Contents |
|---|---|
| 1-5 | see the publication, *DOS/VS RPG II Language* |
| 6 | H\|F\|E\|L\|I\|C\|O |
| 7-13 | /INSERT |
| 14 | blank |
| 15 | sublibrary-name\|blank |
| 16 | • |
| 17-24 | book-name |
| 25-49 | blank |
| 50-74 | comment |
| 75-80 | see the publication, *DOS/VS RPG II Language* |

**sublibrary-name**
Name of the sublibrary from which the insertion
should be made. If no sublibrary is specified the name
is defaulted to R.

**book-name**
Name of sublibrary member to be inserted.

**Function of the /INSERT Statement**
Inserts the contents of the book specified by book-
name, from the sublibrary specified by sublibrary-
name, in place of the /INSERT statement.

## *Executing CICS/VS With DL/I MPS*

There are several additional operational considerations
when executing DL/I MPS under CICS/VS as compared
with executing CICS/VS with a non-MPS version of DL/I.

With MPS you may have defined additional data
bases in the ACT that were formerly used in batch parti-
tions only. If this is so, then any ASSGN, DLBL, and
EXTENT statements necessary for these data bases have

to be placed in the JCL stream used to execute CICS/VS.
In addition, the presence of additional data bases in the
CICS/VS partition increases the GETVIS requirements of
the partition. This is due to the additional VSAM con-
trol blocks and buffers needed to support the new data
bases. Therefore, it may be necessary to change the
SIZE parameter on the EXEC statement or increase the
virtual partition size of the CICS/VS partition.

With the addition of MPS to your CICS/VS system,
you should expect the volume of log records written by
CICS/VS-DL/I to increase. Therefore, if you are logging
to disk, you should consider allocating larger extents to
the log data sets.

## *Executing Batch MPS Programs*

In order to execute a batch MPS application program
CICS/VS must be running in another partition and MPS
operation must have been started (CSDA transaction).

The batch UPSI byte settings for MPS are the same as
for non-MPS batch execution with the exception that
bits 6-7 are not used. Data base logging, normally con-
trolled by UPSI bit 6, is controlled in the CICS/VS parti-
tion under MPS operation. STXIT linkage to DL/I for
abnormal task termination, normally controlled by
UPSI bit 7, is always active under MPS operation. It may
not be turned off as in the case of non-MPS batch exe-
cution.

No ASSGN, DLBL, EXTENT, or TLBL statements are
required to describe the data bases or DL/I log in the
batch MPS job stream. This information is contained in
the JCL for the CICS/VS partition.

The DL/I parameter information for batch MPS oper-
ation need only specify the program name and the PSB
name. Any other parameter information such as buffer
pool options, etc. are ignored as this is controlled in the
CICS/VS partition.

The phase name on the EXEC statement must be
"DLZMPI00". A size parameter is not required unless
your application program invokes some DOS/VS func-

tion that requires a partition GETVIS area; however the use of the SIZE parameter is recommended as an operational standard. DL/I MPS does not require a partition GETVIS area in the batch MPS partition.

# DL/I Data Base Integrity Online

DL/I provides two facilities to insure integrity of data bases concurrently accessed by multiple tasks in an online or MPS environment. These facilities are program isolation and intent scheduling. Program isolation generally allows a greater degree of concurrency but requires additional processing and real storage. Intent scheduling requires less processing and real storage but generally permits fewer DL/I tasks to execute concurrently. Program isolation is the default choice and should normally be used. However, 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. Each of these facilities is discussed in the following sections.

## Intent Scheduling

During PSB generation, you specify processing options on each sensitive segment in each data base PCB within the PSB. This is done using the PROCOPT parameter in either the PCB statement or in the SENSEG statements for each PCB.

Scheduling of a task is by segment intent, that is, according to the manner in which you intend to access segments in your logical data structure. The intent derived from the processing options specified 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.

## Intent Conflict

Intent conflict is a consequence of the intent scheduling data integrity mechanism within DL/I. In order to insure against losing updates and preventing deadlock situations with intent scheduling, DL/I does not allow two tasks to access the same data base concurrently if there is a possibility that each could update a segment being processed by the other. "Update" in this context means insert, delete, or replace.

DL/I determines a task's update intent by examining the PSB the task is using. If two tasks are using PSBs which permit the capability to update the same segment type, DL/I makes the second task wait until the first task is finished, i.e. until a TERM call has been issued by the first task. The intent checking is performed during the scheduling of the PSB, i.e. when the PCB call is issued. Batch MPS programs normally don't

issue PCB or TERM calls; these calls are issued on their behalf during initialization and termination of the program by the MPS Batch Partition Controller program DLZBPC00.

Note that this intent checking is performed not only against running tasks, but also against tasks that are waiting as a result of intent conflict. Because of this, tasks can be waiting due to intent conflict even though they do not conflict with running tasks. Consider the following example:

```
           UPDATE INTENT
           ----SEGMENT----
            1    2    3    4
Task A      X    X
Task B           X    X
Task C                X    X
```

Assume Task A is running. If Task B attempts to schedule, DL/I forces it to wait due to intent conflict with Task A. Now, if Task C attempts to schedule, it will also waits because it has an intent conflict with Task B. Note that programs A and C do not have any intent conflicts. In this example the response time of Task C is the sum of the times for programs A, B, and C since they are single threading due to intent conflict.

One of the tasks in this example could be a batch MPS program. Batch programs typically have run times measured in minutes and hours where online transactions have run times measured in seconds or less. If Task B were a batch MPS program and Task C were an online transaction, then Task C might wish to inform the terminal operator that the data base may be unavailable for quite a while. For this reason, transactions that have intent conflict with batch MPS programs are not forced to wait. Instead of making the transactions wait, DL/I returns a code indicating that they have intent conflict with a batch MPS program. The transaction can then notify the terminal operator or take other alternate action. Batch programs that have intent conflict with other batch programs are abended during initialization.

To provide compatibility for previously existing online application programs, DL/I returns a "not open" condition, X'0C', in TCAFCTR (TCAFCRC in ANS COBOL) when a program has intent conflict with a batch MPS program. Thus for existing online application programs, intent conflict with a batch MPS program results in the online program executing its logic for "data base not open" conditions. To provide the ability for new online application programs to discriminate between true not open conditions and intent conflict with batch MPS programs, the field TCADLTR contains a X'02'. For true not open conditions TCADLTR contains a X'01'. Figure 5-3 summarizes the consequences of intent conflict under the various possible conditions.

| Condition | Consequence |
|---|---|
| Online vs. Online | 2nd task waits |
| Online vs. Batch | Batch program waits |
| Batch vs. Online | Online is returned "data base not open" (TCAFCTR=X'0C') plus TCADLTR=X'02' |
| Batch vs. Batch | 2nd batch program ABENDs |

Figure 5-3. Intent Conflict Consequences

## Determining the Intent

Because of the consequences of intent conflict, you need to be able to determine which segments in a data base have update intent for each program (PSB) used in your system. Segments in a data base to which a PSB is not sensitive, or specified with a read-only processing option (PROCOPT=G), may still have update intent propagated to them by DL/I due to processing options chosen for other segments. For example, if delete sensitivity is specified at the root, then update intent is propagated down to all segments in the data base, even if they are not defined in the PSB. This is because deleting the parent implies the deletion of all its children. Basically, the intent propagation rules used by DL/I are as shown in Figure 5-4.

| PROCOPT | Intent and Propagation |
|---|---|
| G | No update intent, no propagation |
| R | Update intent, no propagation |
| I | Update intent, propagated down one level |
| D | Update intent, propagated down all levels and up one level |
| E | Propagates down to all non-sensitive segments |

Figure 5-4. DL/I Intent Propagation Rules

The presence of logical relationships may cause update intent to be propagated to segments in related data bases. How the intent is propagated in these circumstances depends on what insert, delete, and replace rules were used in the generation of the DBDs involved. More details on intent propagation can be found in the *System/Application Design Guide.*

One of the functions of the Application Control Blocks creation and maintenance utility program (DLZUACB0) is to determine and propagate intent for each segment of each data base referenced in a PSB. This information is stored in the expanded PSB created by the ACB utility in an area called the PSB segment intent list (PSIL). Therefore, if PSBs for the application programs already exist, a CSERV display of the expanded PSB can be used to determine the actual intent of each segment, rather than attempting to apply the intent propagation rules manually.

The PSIL is a variable length list found near the front of the expanded PSB. The PSIL contains an entry for each DMB used by that PSB. Each entry varies in length depending on the number of segment types defined in the DBD.

The format of each PSIL entry is as follows:

| Displacement | Description |
|---|---|
| 0-7 | DMB name for this list entry |
| 8 | Segment intent description byte. X'00' in core image library. Filled in during CICS/VS-DL/I initialization. |
| 9 | Length of this entry in list. |
| 10-?? | Segment intent bits. Two bits are used for each segment in the DMB and represent the PSB's sensitivity to each segment. Their meanings are: |

| Bit | Meaning |
|---|---|
| 00 | PSB is not sensitive to the segment. |
| 01 | PSB is read-only sensitive to the segment. |
| 10 | PSB is update sensitive to the segment. |
| 11 | PSB requests exclusive control of the segment. |

The bits are allocated to the segments in the following manner:

| | Byte 1 | | | | | | | | Byte 2 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Segment | 4 | | 3 | | 2 | | 1 | | 8 | | 7 | | 6 | | 5 | |

Note that the root segment sensitivity is found in bits 6-7 of byte 1, etc. Refer to the section "PSB Intent List - DLZPSIL" in the *Logic Manual* for more information on the PSIL.

## *Potential Intent Conflict Matrix*

With many programs potentially capable of executing against several data bases in a variety of ways, determination of potential intent conflicts becomes an important part of both application and data base design. In addition, determination of when to run batch MPS so as to minimize intent conflict situations with online and other batch MPS programs becomes an important factor in computer operations and scheduling.

One good technique to aid in this environment is to develop a "Potential Intent Conflict Matrix". The development and maintenance of this matrix is a system wide function. Much of it should be the responsibility of the system programmer, or data base administrator.

To develop the matrix, list along the top of a large sheet of paper the names of all the segments in all data bases. Along one side list all batch MPS programs and online transactions that can access any of the data bases with an update PSB. At the intersection of the pro-

gram and the segment that can be updated, enter the product of the number of online transactions per day (or number of times the batch program is run per day) times the number of calls that the transaction makes against that segment. As new data bases and programs come into existence, this matrix should be updated.

By updating this chart, you can determine which programs and transactions can potentially conflict with one another. This matrix can also show relative activity between data bases and online transactions and batch MPS programs. This can assist you in deciding how to initially assign the data bases to the DL/I buffer subpools. These assignments can then be modified based on the buffer pool statistics.

### Potential Intent Conflict Matrix Example

Figure 5-6 is an example of a potential intent conflict matrix. This example is based upon a simple order entry application using the data bases as shown in Figure 5-5.

There are six online transactions and two batch programs in this example whose functions are as follows:

ORDE

> Order entry online transaction. This transaction inserts a record in the Open Order data base. About 800 new orders are entered each day.

ORDM

> Open order maintenance transaction. This online transaction allows changes to be made to any part of an existing open order. About 10% of the new open orders require some maintenance each day.

NEWC

> New customer online transaction. This transaction inserts a record in the Customer data base. Normally two new customers are entered into the Customer data base each day.

CUSM

> Customer maintenance transaction. This online transaction allows changes to be made to an existing customer master record in the Customer data base. It is used only about once a day.

SHIP

> This is an online transaction to mark an order as ready to ship complete. About 80% of the orders are shipped complete each day.

PART

> This is an online transaction to mark an order to be shipped partially complete, flagging those items that cannot be shipped. About 20% of the orders are partially shipped each day.

INVP

> This is a batch program that runs twice a day. This program scans the Open Order data base and prints invoices for those orders that have been marked as ready to ship.

CASH

> This is a batch program that runs once a day. This program applies cash receipts from customers and deletes the corresponding open orders. About 100 customers remit daily, closing about 800 open orders.

## Minimizing Intent Conflicts

Because of the performance implications of intent conflicts, every attempt should be made to minimize potential intent conflict situations during the design and coding of DL/I applications. There are several ways in which applications and programs can be designed and written to minimize intent conflicts. They involve:

* selecting the proper PSB PROCOPT

* using multiple PSBs within one program

* scheduling a PSB for a short duration of time
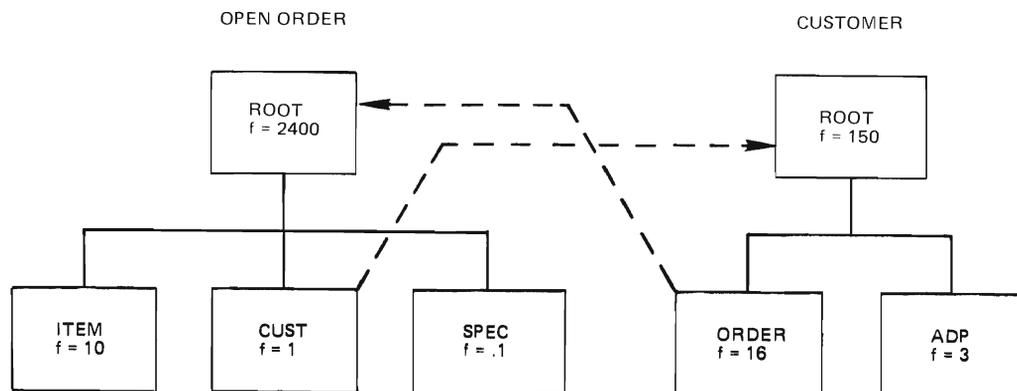


Figure 5-5. Sample Order Entry Application Data Bases

| PGM/TRAN | UPDATED SEGMENTS | | | | | | | NO. TRAN |
|---|---|---|---|---|---|---|---|---|
| | OPEN ORDER DB | | | | CUSTOMER DB | | | |
| | ROOT | ITEM | CUST | SPEC | ROOT | ORDER | ADR | |
| ORDE | 800 | 4000 | 800 | 80 | 800 | 800 | | 800 |
| ORDM | 80 | 160 | 10 | | | | | 80 |
| NEWC | | | | | 2 | | 6 | 2 |
| CUSM | | | | | 1 | | 3 | 1 |
| SHIP | 640 | | | | | | | 640 |
| PART | 160 | 480 | | | | | | 160 |
| INVP* | 4800 | | | | | | | 2 |
| CASH* | 800 | (4000) | (800) | (80) | 100 | 800 | | 1 |
| | 7280 | 8640 | 1610 | 160 | 903 | 1600 | 9 | 1686 |

\* indicates batch program

( ) indicates update intent via intent propagation

Figure 5-6. Potential Intent Conflict Matrix

Each of these areas will be examined using the sample order entry application previously described.

## PSB PROCOPT Selection

Proper PROCOPT selection prevents unnecessary propagation of update intent. By specifying PROCOPT=G on the PCB statement, and then overriding it on the SENSEG statement as required, you can insure that intent is not propagated unnecessarily.

When specifying PROCOPT, use the following 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. To minimize the number of PSBs with a PROCOPT of D, consider establishing a "delete flag" in the segment and using replace logic to "delete" the segment. The "deleted" segments could be actually deleted on some periodic basis, perhaps in a non-MPS batch maintenance run weekly or at some other appropriate interval.

- The use of PROCOPT=A is justified only when all processing options, i.e. 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.

Figure 5-6 shows that the CASH program is the only one with update intent propagated to non-sensitive segments. This is due to the delete sensitivity against the root segment of the Open Order data base. The CASH program's logic could be changed to use a "delete flag" in the root segment, or to otherwise indicate the open order record was deleted through replace logic. This eliminates the intent propagation to dependent segments. However, the potential intent conflict matrix shows that the CASH program still has potential intent conflict with almost all other batch and online transactions through the root segment. Therefore, in this case, the use of the "delete flag" is not justified.

## Using Multiple PSBs Within One Program

The shorter the duration that a task has an update PSB scheduled, the less time other tasks with intent conflict have to wait. One way the time a task has an update PSB scheduled can be minimized is to use different PSBs at different points during the execution of the task according to the needs of the task at that point in time. Consider the following example CICS/VS-DL/1 transaction.

```
PCB call (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 acknowledgement message to │
    terminal and return to CICS/VS ─┘
```
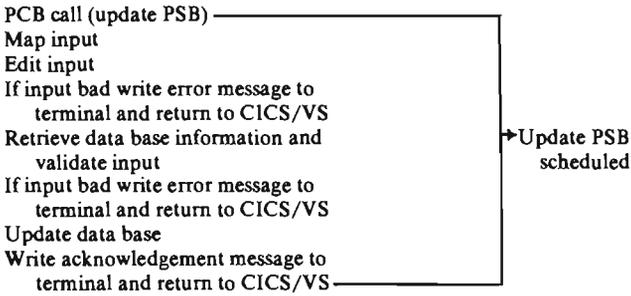
Note that this transaction has an update PSB scheduled for the entire life of the transaction, even though it is needed only after the decision to update the data base has been made. The transaction could have been written as follows:

```
Map input
Edit input
If input bad, write error message to
    terminal and return to CICS/VS
PCB call (read-only PSB)
Retrieve data base information and
    validate input
If input bad write error message to
    terminal and return to CICS/VS
TERM call (read-only PSB)
PCB call (update PSB) ──────────┐
Reposition in data base         │ ─→Update PSB
Update data base                │    scheduled
TERM call (update PSB) ─────────┘
Write acknowledgement message to
    terminal and return to CICS/VS
```

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

Transactions should not normally schedule an update PSB more than once during one execution. To schedule more than one may logically invalidate the backing out of the effects of a task should the task or system fail while an update PSB is scheduled. This is because DL/I backout only backs out changes made to the data bases that were not followed by a TERM call. For example, if a transaction scheduled an update PSB twice, or two different PSBs one after the other, and the system failed during processing with the second PSB, DL/I would only back out the changes made by the task while the second PSB was scheduled. The TERM call made with the first PSB indicates to DL/I that any processing up to that point is complete and should not be backed out in the event of a subsequent failure. If the logic of the application was such that the processing

using the first PSB was related to the processing using the second PSB, then backing out the effects of processing using the second PSB (which was not completed), but not backing out the effects of processing using the first PSB (which was completed), may result in information in the data base which is logically incorrect (from the application point of view). Note that the data bases are not damaged in this case, but the data within the data bases may not be logically consistent from an application viewpoint.

Batch MPS programs should not schedule a different PSB from that which is specified in the DL/I statement because there is no mechanism to reestablish addressability to a different PSB once the batch program has been initialized. However, batch MPS programs may terminate and reschedule the same PSB several times during execution if desired. The considerations and consequences of this are discussed in the following section.

### Scheduling a PSB for a Short Duration At a Time

When an online DL/I transaction or a batch DL/I MPS program has a considerable amount of processing to do using an update PSB, it may be desirable to occassionally release the PSB (via a TERM call). This allows other tasks with intent conflict to get a chance at the data base. This can be thought of as "time-sharing" the data base among several update PSBs with intent conflict.

Because this technique involves scheduling an update PSB multiple times within a single task execution, you must insure this does not affect the logical consistency of the data base should a failure and subsequent backout occur while processing in this manner. Terminal applications where the operator fills the screen with multiple items of input that are not related by application logic are well suited to this technique. If a failure occurs, the terminal operator could inquire into the data base to determine what input from the last screen was not processed and reenter it.

## Program Isolation

Program isolation provides the capability for online and/or MPS users to perform concurrent updates on the same segment type in a data base. Use of this feature results in less resource contention in a DL/I-CICS/VS environment (including MPS). It is an optional replacement for intent scheduling and, as does intent scheduling, applies only to the online environment. It is inactive for normal batch (non-MPS) mode processing. PI is the recommended choice for DL/I online and MPS applications.

The disadvantage of intent scheduling is that no two tasks that state conflicting intents for a particular seg-

ment type are allowed to execute concurrently. This may be an undesirable limitation. In the first place, two tasks cannot have a conflict for a segment type; the conflict exists for a particular segment occurrence. In the second place, the conflict occurs only if the segment is referenced, and a particular task may have to state intent for more segment types than it actually refers to during any particular execution.

Program isolation attempts to reduce the amount of unnecessary contention by managing it, for most cases, only when segments are actually referenced. Thus program isolation operates at the segment occurrence level where intent scheduling operates at the segment type level. It does this with two functional areas, 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 and then rescheduling them. This is functionally equivalent to intent scheduling processing. There are, however, some differences, which are explained in the following comparisons of the results of a task 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: | |
| Is not scheduled until all other tasks with any specified intent for this segment type have issued a TERM call. | Identical to intent scheduling. |
| Restrictions on other tasks: | |
| Other task is not scheduled if it specified any intent for this segment type until this task has issued a TERM call. | 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: | |
| Is not scheduled until any other task with either exclusive or update intent specified for this segment has issued a TERM call. | a. Is not scheduled until any other task with exclusive intent specified for this segment type has issued a TERM call. |
| | b. Waits to read any segment updated by another task with update intent for that segment type until that task has issued a TERM call. |
| | c. Waits to update any segment read by another task with a Q command code until that task has issued a TERM call. |
| Restrictions on other tasks: | |
| Other task is not scheduled if it specified update or exclusive intent for this segment type until this task has issued a TERM call. | a. Any other task with exclusive intent for this segment type waits to be scheduled until this task issues a TERM call. |
| | b. Any other task with update intent for this segment type waits to update a segment read by this task with the Q command code until this task has issued a TERM call. |
| | c. Any other task with either read-only or update intent for this segment type waits to read segments updated by this task until this task has issued a TERM call. |

*Comparison:* Specification of update intention under program isolation provides the same protection from data usage conflicts as does intent scheduling. Additionally, 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: | |
| Is not scheduled if any other scheduled task has specified exclusive intent for this segment until that task has issued a TERM call. | a. Identical to intent scheduling. |
| | b. Waits to read any segment updated by another scheduled task until that task has issued a TERM call. |
| Restrictions on other tasks: | |
| Any other task with exclusive intent for this segment type waits to be scheduled until this task has issued a TERM call. | 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 failure of another task. This cannot occur under program isolation.

**Deadlock Avoidance:** Deadlock avoidance recognizes and remedies the case where two or more tasks are interlocked on resources for which they are waiting. Program isolation detects deadlocks on requests for segments. 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 is terminated (i.e. MPS batch vs. MPS batch or online vs. online).

CICS/VS dynamic transaction backout is called for the terminated task.

**Specification in DLZACT Macro:** Program isolation is automatically included in the application control table generation (DLZACT macro) unless you specify otherwise in the TYPE=CONFIG statement. The optional keyword parameter used is

```
[ ,PI={YES}]
      {NO }
```

with YES as the default. If you wish to avoid additional CPU load inherent to program isolation, you may do so by specifying PI=NO. In that case, you cause intent scheduling to be in effect.

**Operational Considerations:** Three informational error messages are issued by program isolation. The one indicating task termination due to invalid data usage conflict (DLZ106I) indicates a resource contention problem that is very dependent on the current mix of tasks. The task may execute successfully if rerun. If not, some evaluation of the mix of other tasks and the resources they reference is required.

Message DLZ108I, indicating insufficient space, identifies a problem in getting more space from the CICS/VS-DL/I partition for queueing 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 SYNCH point, thereby using up a lot of storage.

Message DLZ031I indicates that the user did not enable CICS/VS dynamic transaction backout. DL/I needs this facility to ensure data base integrity when running with program isolation. DL/I initialization is terminated. The CICS/VS system stays active.

## *Programming Considerations*

1. MPS batch programs with update intent for segments should either issue frequent CHKP calls or, if possible without increasing contention, use PROCOPT=E on all segments to avoid excessive queue lengths.

   Note: A code is passed back in 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.

   Existing MPS batch programs doing TERM/PCB call sequences with an update PSB must be modified to use the CHKP call instead. Under PI, addressability to the update PSB is not correct after a TERM/PCB call sequence.

2. If data is read that will be used in data base update, and the PCB used references a different data base record before the update occurs, the Q command code should be specified to protect the data from modification by another concurrently executing task.

3. CICS/VS transactions that are not restartable (i.e. are not specified with the DTB operand of the DFHPCT macro) must use PROCOPT=E on all segments to bypass PI and thus avoid the possibility of termination due to deadlock conditions.

4. Program isolation may cause an increased level of transaction backouts. Additionally, these can now occur when there is no task error. Where users may have been willing to accept unrestartable task failures, they may not be willing to accept failures due to deadlock conditions.

5. Users having transaction processing that requires special backout code must either rewrite the code or schedule tasks with 'exclusive' intent for all segments for which sensitivity has been specified.

6. For program isolation, logging is required and the CICS/VS journal must be used for such logging.

7. Any task that relies on replace, delete, or insert intent to protect a segment being used from later modification by another task must specify the Q command code on the read to accomplish the same purpose.

8. Specifying processing option 'O' in the PCB will avoid all PI locking. This is useful for a task with read only processing when the integrity of the data received is not critical.

# Controlling the Number of CICS/VS and DL/I Tasks

There are a variety of parameters present in the CICS/VS and DL/I system that affect performance by controlling the number of tasks that can be active at

one time. The parameters include:

- CICS/VS MXT
- CICS/VS AMXT
- CICS/VS CMXT and TCLASS
- DL/I MAXTASK
- DL/I CMAXTSK

By choosing the proper values for these parameters you can prevent your CICS/VS-DL/I system from overcommitting the resources available. The following discussion should help you in understanding the effect of each of these parameters on system performance.

Tasks in a CICS/VS system are either "active" or "suspended". The CICS/VS Task Control Program maintains the status of each task through the use of two chains: the active chain and the suspend chain. The active task chain is maintained in task priority sequence by CICS/VS. When a new task is created or a suspended task is resumed, it is placed in the active chain according to its priority relative to other tasks on the chain. Tasks of equal priority are placed in the active chain in FIFO sequence. The active chain is searched by the task dispatcher when attempting to locate a task that can be dispatched. The suspend chain is only used in response to specific requests by other CICS/VS management modules. A task on the suspend chain must be moved to the active chain before it can be dispatched. However, a task may be on the active chain and still not be dispatchable.

## CICS/VS MXT Parameter

The total number of tasks in the CICS/VS system, active and suspended, is controlled by the CICS/VS MXT parameter. A request to CICS/VS to create a new task (attach) is honored provided the MXT value is not exceeded. If a request to attach a new task would cause MXT to be exceeded, then the request is generally queued. In addition, CICS/VS does not initiate any new polling of the terminals because there is no point in inviting more tasks when the limit has already been reached. Note, however, that if a user program requests an attach, the request is honored even if MXT is exceeded. Therefore, it is possible to have more tasks in the system than specified in the MXT parameter. In particular, the DL/I MPS Master Partition Controller and Batch Partition Controller tasks are attached with a "user" request. Therefore, if your CICS/VS system is running at or near MXT, executing batch DL/I MPS programs can cause your CICS/VS MXT limit to be exceeded, thus suspending polling operations. This situation is likely to occur only in a system with MXT set too low and primarily conversational terminal tasks.

Every task requires some amount of CICS/VS dynamic storage. Therefore, the primary resource usage that you can control with the CICS/VS MXT parameter is dynamic storage requirements. Since the cost of dynamic storage is small (as it is primarily virtual storage), and the consequences of reaching MXT are severe (CICS/VS BTAM systems stop polling and begin to discard input messages causing increased network traffic), you should provide sufficient dynamic storage to support a MXT value large enough to avoid reaching it in normal circumstances. Setting the MXT value equal to the sum of the number of terminals in your CICS/VS system, plus the number of journals, plus the number of DL/I MPS tasks to be run concurrently should be adequate.

## CICS/VS AMXT Parameter

Tasks on the active chain are either "dispatchable" or "nondispatchable". When a new task is created or a suspended task is resumed, it is placed on the active chain in a nondispatchable state. Note that the terms "dispatchable" and "nondispatchable" as used here do not exactly match the terms used within CICS/VS internal program documentation (PLMs and listings).

Tasks on the active chain are also either "ready" or "waiting". CICS/VS dispatches the highest priority ready task that is dispatchable. If a dispatchable task is waiting, it remains dispatchable. CICS/VS assumes that dispatchable tasks that are waiting will become "ready" in a short time (a few milliseconds). Therefore, the task dispatcher makes no attempt to swap the dispatchable status of a waiting task with one that is ready but nondispatchable.

Once a task becomes dispatchable it remains dispatchable until it terminates or is suspended. A task can be suspended for one of the following reasons:

- The task issued a terminal control wait. This type of task is commonly called "conversational".
- The task issued an interval control wait.
- The task issued an enqueue and could not be granted the resource. Often this enqueue request is not explicitly coded in the application program, but results from a file control request (CICS/VS enqueues on logical record for update requests), a transient data PUT to an intra-partition data set with logical recovery (CICS/VS enqueues on the data set id), or a temporary storage PUTQ, RELEASE or PURGE against recoverable temporary storage data sets (CICS/VS enqueues on the data-id name).
- The task issued an unconditional GETMAIN that could not be satisfied (short on storage condition).

- The task issued a DL/I PCB call and DL/I CMAXTSK had already been reached.

- The task issued a DL/I PCB call and DL/I determined it had intent conflict with another DL/I task.

- The task issued a DL/I data base call and the segment or record requested is currently enqueued by the program isolation facility for another DL/I task.

A suspended task is resumed when the condition that caused it to be suspended disappears. However, when it is resumed it reenters the active chain in a non-dispatchable state.

A task can be nondispatchable for one of several reasons:

- AMXT limit reached

- CMXT limit reached

- Short-on-storage

When a task is placed on the active chain, either because it is a new task, or a task being resumed, it is normally marked nondispatchable for AMXT reasons. The task is changed to dispatchable status during a task dispatcher scan of the active chain if, in fact, AMXT has not been reached. The remaining three nondispatchable conditions can only occur as a result of an unconditional user program attach request during abnormal CICS/VS conditions (short on storage or CMXT limit reached).

During a scan of the active chain, the task dispatcher changes a task from nondispatchable to dispatchable state provided the total number of currently dispatchable tasks is less than the AMXT parameter value and no other condition exists (CMXT or short on storage) to prevent the task from becoming dispatchable. When the task dispatcher changes a task from nondispatchable to dispatchable state, the count of dispatchable tasks is increased by one. When a task is terminated or suspended, the count of dispatchable tasks is decreased by one. The only exceptions to this are "special" tasks. These "special" tasks are the CICS/VS terminal control, task control, and journal tasks and the DL/I MPS Master Partition Controller task. Although they may be dispatchable, they are not counted as part of the dispatchable tasks for AMXT purposes. The total number of tasks that can be dispatchable at any point in time is limited by the AMXT parameter value (not counting "special" tasks).

The task dispatcher scans the active chain looking for a task to dispatch until either it finds a dispatchable task ready to run (i.e. not waiting), or it passes as many dispatchable tasks (which are waiting) as specified in AMXT. Because of this, the DL/I MPS Master Partition

Controller task, which is not counted when determining if AMXT has been reached, should be given a high priority to insure it is always examined by the task dispatcher during its scan of the active chain. The other "special" CICS/VS tasks already have a priority higher than user tasks so they are always examined by the CICS/VS task dispatcher during its scan of the active chain.

Because AMXT limits the number of tasks that can be dispatched by CICS/VS, AMXT controls the number of tasks that can compete for resources within the CICS/VS system. The key resource that AMXT controls is real storage, because only dispatchable tasks are allowed to execute, and only executing programs use real storage. Therefore, you should set AMXT no higher than the amount of real storage available for the user tasks, divided by the average storage required for a user task.

## CICS/VS CMXT and TCLASS Parameters

The CMXT and TCLASS parameters allow you to separate transactions into classes and control the number of tasks of a class that can be dispatchable at any point in time. Note that this facility can be used only with transactions whose transaction-id does not begin with the letter "C". Thus it cannot be used for any CICS/VS provided transactions or the DL/I MPS transactions CSDA, CSDB, CSDC, and CSDD. If a task that is being attached has been specified as belonging to a class (via the TCLASS parameter in the PCT), CICS/VS checks the count of the number of tasks currently in existence of that class. CICS/VS does not attach the task if doing so causes the CMXT limit for that class to be exceeded. If the attach request is unconditional, and attaching the task causes CMXT to be exceeded for that class, the task is entered on the active chain in a nondispatchable state (for CMXT reasons). If the attach request is conditional, and attaching the task would result in exceeding CMXT for that class, the attach is not performed and the requesting task is returned a code indicating the attach was unsuccessful. All CICS/VS attach requests, e.g. from terminal control, are conditional requests.

## DL/I MAXTASK Parameter

The DL/I MAXTASK parameter in the ACT specifies an upper limit on the number of DL/I tasks that may exist concurrently under CICS/VS. This parameter cannot 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.

### DL/I CMAXTSK Parameter

The intent of the DL/I CMAXTSK parameter is to allow the upper limit of DL/I tasks within the system to be lowered dynamically. A special DL/I system call is available to change the CMAXTSK value, (see "Controlling the DL/I Online System Environment" in the *Utilitites and Guide for the System Programmer*). Note that CMAXTSK can never exceed MAXTASK.

If a DL/I task issues a PCB call and DL/I CMAXTSK has already been reached, then DL/I suspends the scheduling task. Suspending the task reduces the CICS/VS active task count (AMXT) and allows other non-DL/I tasks to run.

The DL/I CMAXTSK parameter should be used to set an upper limit on the number of DL/I tasks that you wish to have exist concurrently. Since only dispatchable tasks can issue PCB calls, and the number of dispatchable tasks is limited by AMXT, there is no benefit in setting the DL/I CMAXTSK parameter higher than the CICS/VS AMXT parameter. The CICS/VS TCLASS and CMXT parameters can be used to classify and control DL/I tasks, according to individual characteristics.

For best performance, the combined effect of the CICS/VS AMXT and the DL/I MAXTASK and CMAXTSK parameters must be considered. Three key points to keep in mind when choosing values for these parameters are:

- The major controlling factor in the CICS/VS environment is the value of AMXT, i.e. the number of active tasks that are allowed to execute concurrently.

- A DL/I task that has requested DL/I services and is waiting for VSAM I/O is considered an active task by CICS/VS.

- A DL/I task that has been suspended due to CMAXTSK being reached is not considered active by CICS/VS.

# Chapter 6: Data Base Reorganization/Load Processing

## Introduction

This chapter introduces the function of data base reorganization in a DL/I environment. It is a first-time introduction into the requirements for, and the process of, data base reorganization and load processing. As in the previous sections of this manual, the primary emphasis is on HDAM and HIDAM data bases.

The content of this chapter is intended to be a general overview; for specific details on how to code the utility programs, see the *Utilities and Guide for the System Programmer.* Additional information is also contained in the *System Application Design Guide.*

DL/I provides eight utility programs of two distinct types to assist you: those dealing with data base reorganization (physical reorganization utilities), and those dealing with load processing (logical relationship resolution utilities).

The utility programs supplied for load processing are in no way concerned with the actual loading of the data base. This is your responsibility. However, certain pointer relationships cannot be resolved during initial loading, and it is for the resolving of these relationships that the utility programs are provided.

## What is Reorganization

Reorganization is the process of changing the physical storage and/or structure of a data base to better achieve the application's performance requirements. The two types of reorganization are:

- Physical reorganization, to optimize the physical storage of the data base.

- Logical reorganization, to optimize the data base structure.

### When to Reorganize

A general guideline is to reorganize a data base when the benefits you expect to receive from reorganization more than offset the time required to reorganize. The amount of time needed to reorganize depends on the data base access method used and the reorganization facility used.

The DL/I reorganization programs provide statistical data that can help data base administration personnel determine whether a data base should be reorganized. Determining when a data base should be scheduled for reorganization depends, to a large extent, on knowing the overall activity on the data base (that is, the number of segment additions and deletions), the physical organization of the data base, the relationship of the data base to other data bases, and the installation's planned use of the data base in the future.

Most data base reorganizations are done to recover space occupied by deleted segments and/or to resequence segments physically in their logical order. The number of segment insertions and deletions can be determined from data provided by the application accounting report, and the distribution of transaction response times. When segment chains become long, and when they involve segments that are in different areas of a storage device, response times tend to increase. Growing response times may indicate a need for data base reorganization.

Frequency of reorganization should be considerably less for HDAM and HIDAM than for HISAM data bases, because both HDAM and HIDAM reuse space freed by deleted segments, and because both HDAM and HIDAM attempt to place inserted segments physically near their logically related segments (that is, near segments to which they are chained by physical child/physical twin forward pointers).

## Overview of the Reorganization/Load Utilities

The DL/I reorganization utilities provide three basic functions:

- The reorganization of DL/I data bases

- Establishing logical relationship connections when initially loading data bases having logical relationships

- Creation of secondary index data base(s) when loading your data bases or when you reorganize them

The eight basic utility programs involved in data base reorganization/load processing are:

- HISAM Reorganization Unload (DLZURUL0)

- HISAM Reorganization Reload (DLZURRL0)

- HD Reorganization Unload (DLZURGU0)

- HD Reorganization Reload (DLZURGL0)

- Data Base Prereorganization (DLZURPR0)

- Data Base Prefix Resolution (DLZURG10)

- Data Base Prefix Update (DLZURGP0)

- Data Base Scan (DLZURGS0)

# Reorganization of HDAM and HIDAM Data Bases

Two methods can be used to accomplish reorganization of HDAM and HIDAM data bases. The first method involves use of GET NEXT and INSERT calls with a user-written application program.

The second method is to use the DL/I HD reorganization unload and reload utilities. These utilities use unqualified GET NEXT calls to sequentially unload segments from the data base to be reorganized.

These utilities can also be used to reorganize HISAM data bases. The HISAM unload/reload utility offers better performance, but if structural changes are required, you must use the HD reorganization utilities.

# Logical Relationship Resolution

A set of four utility programs is used in conjunction with initially loading and/or reorganizing data bases that are involved in logical relationships.

The logical relationship resolution utility program set includes:

- The data base prereorganization utility.

  This program controls the execution of the other utility programs that are concerned with the resolution of logical relationships.

- The data base scan utility

  This program searches one or more data bases for all segments that are involved in logical relationships. For each such segment, the program generates one or more output records. This output serves as input to the prefix resolution utility.

- The data base prefix resolution utility

  The general operation of this program consists of combining and sorting all work files that are defined as input to this program. These input files may be files generated by the prereorganization utility, the scan utility, the HD reload utility, or a user load program.

- The data base prefix update utility

  This program applies the necessary changes to the prefix of segments involved in logical relationships after an initial load or a reload. It uses as input the file generated by the prefix resolution utility.

# Reorganization/Load Flowchart

Figure 6-1 shows the necessary programs required for physical reorganization and/or logical relationship resolution processing during initial load or reorganization of an HDAM or HIDAM data base.
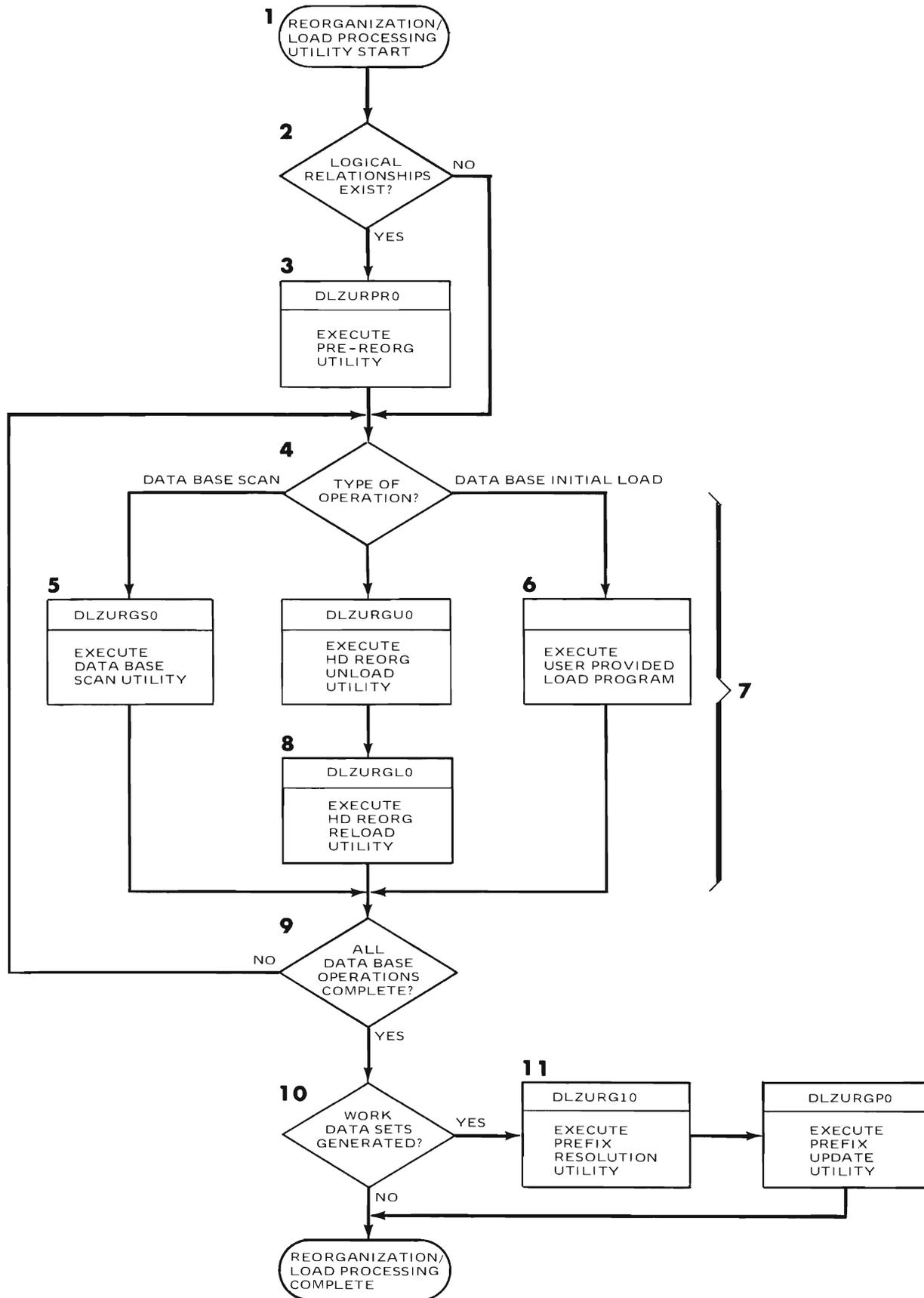
Figure 6-1. Reorganization/Load Flowchart

**Notes:**

1. The physical reorganization/logical relationship resolution utilities may be used to operate on one or more data bases concurrently. For example, one or more data bases may already exist, or any number of existing data bases may be reorganized while other data bases are being initially loaded. Any or all of the data bases being operated upon may be logically interrelated. A data base operation is defined to be an initial data base load, a data base unload/reload (reorganization), or a data base scan.

   Invalid combinations can occur when a mixed operation of initial loading and reorganization is performed on logically related data bases. See the section "Restrictions" of "Data Base Logical Relationship Resolution Utilities" in the *Utilities and Guide for the System Programmer* for details.

2. The YES branch must be taken if any segment in any data base being operated upon is involved in a logical relationship, or if a data base involved in secondary index relationships is being reorganized. Taking the YES branch is also recommended when loading data bases with secondary index relationships; see the section "With Secondary Indexes" of "Data Base Initial Load/Reload" later in this chapter. In other circumstances the NO branch should normally be taken (this includes the case when primary index relationships exist) but it need not be, as you may wish to let the prereorganization utility determine which data base operations are to be performed.

3. If virtual logical child segments are present in the data base being reorganized, then the data base containing the physical segment of the physical-virtual pair must also be scanned or reorganized to retain the proper logical relationships through prefix resolution and prefix updates.

4. Based upon the information presented to it in control statements, the data base prereorganization utility provides a list of data bases that must be initially loaded, reorganized, or scanned.

5. This program should be executed for each data base listed in the output of the prereorganization utility. A work file may be generated for each data base that this utility scans. Data bases to be scanned are listed after the characters "DBS=" in one or more output messages of the prereorganization utility.

6. The user-provided initial data base load program may automatically cause the generation of a work file to be later used by the prefix resolution utility. You need not add code to your initial load program to generate the work file. This is done automatically by internal routines. Data bases to be initially loaded are listed after the characters "DBIL=" in one or more output messages of the prereorganization utility.

7. This area of the flowchart must be followed once for each data base to be operated upon, whether the operation consists of an initial load, a reorganization, or a scan. The operations may be done for all data bases concurrently, or for one data base at a time. However, all scans and unloads for logically related data bases must be done before any reload.

8. The HD reorganization reload utility may cause the generation of a work file to be later used by the prefix resolution utility. Data bases to be reorganized using the HD unload/reload utilities are listed after the characters "DBR=" in one or more output messages of the prereorganization utility.

9. Be sure that all operations indicated by the prereorganization utility (if it was executed) are completed prior to taking the YES branch.

10. If any work files were generated during any of the data base operations that were executed, the YES branch must be taken. Note that the presence of a logical relationship in a data base does not guarantee that work files will be generated during a data base operation. The physical reorganization/logical relationship resolution utilities determine the need for work files dynamically, based upon the actual segments presented during a data base operation. If any segments that participate in a logical relationship are loaded, work files are generated and the YES branch must be taken.

    If for any specific data base operation no work file is generated for the data base, processing of that data base is complete, and it is ready for use.

    When a HIDAM data base is initially loaded, its index is automatically generated. This may also apply to secondary indexes. See the section "With Secondary Indexes" of "Data Base Initial Load/Reload" later in this chapter.

11. The data base prefix resolution utility combines the output from the data base scan utility, the HD reorganization reload utility, and the user initial data base load program to create an output file to be used by the prefix update utility. The prefix update utility then completes all logical relationships defined for the data bases that were operated upon.

# Data Base Initial Load/Reload

It is your responsibility to provide a program for initially loading a data base. This program must use a PCB with a PROCOPT of either L or LS. If LS is specified, or if the organization is HIDAM or HISAM, or HSAM (simple HSAM) with keys, the data base must be loaded sequentially. The PCB must reference a physical DBD, not a logical DBD. The load program must use the DL/I ISRT call function to load segments into a data base. Each segment must be placed in the I/O area with a length and field placement as described in the physical DBD for the data base. The segments of a data base record must be inserted in hierarchical order. See "Data Base Load Processing" in Chapter 4 for additional details.

## *With Logical Relationships*

If a data base to be loaded or reloaded contains segments involved in logical relationships, one or more of the logical relationship resolution utilities may have to be executed as shown in Figure 6-1.

If a segment is a logical child, both the logical parent's fully concatenated key and the logical child intersection data, if any, must be placed in the user I/O area. The data for the logical parent must be loaded in a separate DL/I insert call. Be sure that the logical parent for each logical child loaded during initial data base load is loaded before the prefix resolution utility (DLZURG10) is executed.

All work files produced when data bases that participate in a logical relationship are initially loaded must be supplied as input to one execution of the prefix resolution utility.

If the data base being initially loaded or reloaded contains logical relationships, job control statements must be provided for the load or reload program for one input and one output file as follows:

- The input file contains control information and is created by the data base prereorganization utility. Filename must be specified as CONTROL. The

logical unit assignment must be SYS012. The file can be only DASD.

- The output file contains logical relationship information created by the loading of the data base. Filename must be specified as WORKFIL. The logical unit assignment must be SYS013. The file can be tape or DASD.

In the case of loading only logical parent segments and no logical child segments, the execution of the logical relationship resolution utilities can be bypassed by:

- Specifying // ASSGN SYS013,IGN in the job loading the data base, so that no work file is generated. Message DLZ0071 with a return code of 04 will be printed as a warning but processing continues.

- Loading the logical child segments subsequently in an update type job, (that is, with a PCB that has a PROCOPT of A or I). See "Loading Data Bases with Logical Relationships" in Chapter 4 for additional details.

### With Secondary Indexes
If the data base initially loaded contains secondary index relationships, the secondary indexes are built automatically during initial loading, provided all ASSGN, DLBL, and EXTENT statements for the second-

ary index data bases are included in the job stream for initial load. However, because the index records are not normally in ascending key sequence, this usually leads to a significant performance degradation. Therefore, you may choose to treat secondary index relationships as normal logical relationships. ASSGN and TLBL (or DLBL and EXTENT) statements must then be provided for the work file WORKFIL, as mentioned above, and the logical relationship resolution utilities must be used to build the secondary index data base(s). In this case the DLBL and EXTENT statements for the secondary index data bases must be omitted in order to prevent the indexes from being created automatically.

Note: When using this method, 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.

This discussion also applies to reloading a data base with the HD reload program.

## Resolution Utilities Overview
Figure 6-2 provides an overview of how to use the logical relationship resolution utilities when loading data bases with logical relationships and/or secondary indexes.

| INPUT | PROCESSING | OUTPUT |
|---|---|---|
| CONTROL CARDS | PREREORGANIZATION UTILITY DLZURPR0 | MESSAGES<br>CONTROL DATA SET<br>[CONTROL]<br>CONTROL CARDS, optional |
| DATA BASE(S)<br>CONTROL CARDS<br>or<br>CONTROL DATA SET<br>[CONTROL] | DATA BASE SCAN UTILITY DLZURGS0 | WORK FILE 1<br>[WORKFIL] |
| CONTROL DATA SET }<br>[CONTROL } LR only | EACH USER LOAD PROGRAM<br>physical DBD indicates presence of<br>logical relationships or secondary<br>indexes | LOADED DATA BASE(S)<br>physical pointers established<br>WORK FILE 1 }<br>[WORKFIL] } LR only<br>SECONDARY INDEXES, optional |
| CONTROL DATA SET<br>[CONTROL] | HD RELOAD UTILITY<br>DLZURGL0<br>physical DBD indicates presence of<br>logical relationships or secondary<br>indexes | RE-LOAD DATA BASE<br>physical pointers established<br>WORK FILE 1<br>[WORKFIL]<br>SECONDARY INDEXES, optional |
| CONTROL DATA SET<br>[CONTROL]<br>1 TO n WORK FILE 1's<br>[WRKIN01-WRKIN0n] | PREFIX RESOLUTION UTILITY DLZURG10 | MESSAGES<br>SORT WORK FILES<br>[SORT WK n]<br>WORK FILE 2<br>[INTRMED]<br>WORK FI' E 3<br>[WORKFIL]<br>control information and sequenced<br>pointer values<br>SECONDARY INDEX<br>WORK FILE, optional<br>[INDXWRK] |
| WORK FILE 3<br>[WORKFIL]<br>SECONDARY INDEX<br>WORK FILE, optional<br>[INDXWRK] | PREFIX UPDATE UTILITY DLZURGP0 | MESSAGES<br>USER DATA BASE(S)<br>logical pointers established<br>SECONDARY INDEXES, optional |

Figure 6-2. Loading Data Bases with Logical Relationships and/or Secondary Indexes

Figure 6-3 is an example of loading two data bases with logical relationships and secondary indexes
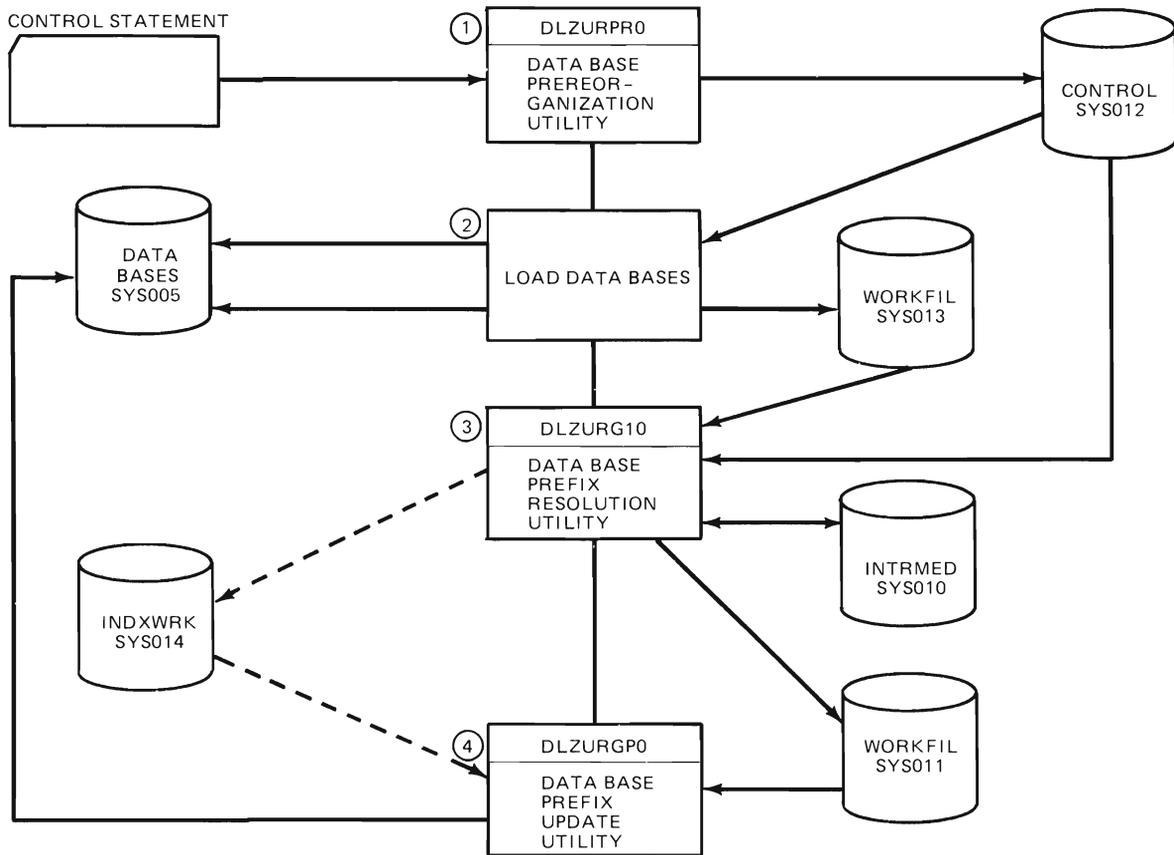


Figure 6-3. Loading of Two Data Bases with Logical Relationships and Secondary Indexes

The following JCL examples relate to Figure 6-3 for the Inventory and
Customer data bases used in the sample application.

```
①// JOB STJPREOR PRE-REORGANIZATION UTILITY
  // OPTION PARTDUMP
  // DLBL CONTROL,'CONTROL FILE',0,SD
  // EXTENT SYS012,111111,1,0,1680,10
  // ASSGN SYS012,X'230'
  // EXEC DLZRRC00,SIZE=300K
  ULU,DLZURPR0
  DBIL=STDIDBP ,STDCDBP
  OPTIONS=(NOPUNCH,STAT,SUMM)
  /*
  /&
②// JOB STJLDCST LOAD INVENTORY AND CUSTOMER DATA BASES
  // OPTION PARTDUMP
  // ASSGN SYS005,X'230'
  // DLBL STDIDBC,'SAMPLE.INVEN',,VSAM
  // EXTENT SYS005,111111
  // DLBL STDCDBC,'SAMPLE.CUST',,VSAM
  // EXTENT SYS005,111111
  *  NOTE: NO DLBL EXTENTS FOR SECONDARY INDEXES (BUILT BY PREFIX UPDATE)  - Note 1
  // DLBL CONTROL,'CONTROL FILE',0,SD
  // EXTENT SYS012,111111,1,0,1680,10
  // ASSGN SYS012,X'230'
  // DLBL WORKFIL,'WORKFILE J',0                                          - Note 2
  // EXTENT SYS013,111111,1,0,1690,5
  // ASSGN SYS013,X'230'
  // UPSI 00000010       NO LOG                                          - Note 3
  // EXEC DLZRRC00,SIZE=300K
  DLI,DLZSAM40,STBICLD,1,HDBFR=(6)
       •
       •  data cards
       •
  /*
  /&
```

**Notes:**

1. No DLBL and EXTENT statements are included in this job for secondary indexes.
   Because of this an OPEN error message is printed, but the appropriate work file records
   are written on the file WORKFIL for later processing by the prefix resolution and prefix
   update utilities to create the secondary indexes.

2. This job loads two data bases, Inventory and Customer, so only one work file is produced.
   This affects the prefix resolution utility control statement.

3. The UPSI byte setting is described in the *Utilities and Guide for the System Programmer*
   (Chapter 8).

```
③ // JOB STJPRRES PREFIX RESOLUTION UTILITY
  // OPTION PARTDUMP
  // ASSGN SYS001,X'230'
  // DLBL SORTWK1,'SORTWK FILE NR1',0,SD                              - Note 1
  // EXTENT SYS001,111111,1,0,3600,100
  // ASSGN SYS002,X'230'
  // DLBL SORTWK2,'SORTWK FILE NR2',0,SD
  // EXTENT SYS002,111111,1,0,3700,100
  // ASSGN SYS003,X'230'
  // DLBL SORTWK3,'SORTWK FILE NR3',0,SD
  // EXTENT SYS003,111111,1,0,3800,100
  // DLBL WRKIN01,'WORKFILE J',0,SD
  // EXTENT SYS013,111111,1,0,1690,5
  // ASSGN SYS013,X'230'
  // DLBL CONTROL,'CONTROL FILE',0,SD
  // EXTENT SYS012,111111,1,0,1680,10
  // ASSGN SYS012,X'230'
  // ASSGN SYS010,X'230'
  // DLBL INTRMED,'INTERMEDIATE WORK FILE',0,SD
  // EXTENT SYS010,111111,1,0,1620,20
  // ASSGN SYS011,X'230'
  // DLBL WORKFIL,'PREFIX WORK FILE LR',0,SD
  // EXTENT SYS011,111111,1,0,1640,10
  // ASSGN SYS014,X'230'
  // DLBL INDXWRK,'PREFIX WORK FILE SI',0,SD                          - Note 2
  // EXTENT SYS014,111111,1,0,1660,20
  // EXEC DLZURG10,SIZE=300K
  R              3
  /*
  /&
```

**Notes:**

1. The number of sort work files is specified in the utility control statement and may be from one to eight for DASD and from three to nine for tape.

2. These DLBL and EXTENT statements are required because the secondary index is not created at initial load time.

```
④ // JOB STJPRUPD PREFIX UPDATE UTILITY
  // OPTION PARTDUMP
  // ASSGN SYS005,X'230'
  // DLBL STDIDBC,'SAMPLE.INVEN',,VSAM
  // EXTENT SYS005,111111
  // DLBL STDCDBC,'SAMPLE.CUST',,VSAM
  // EXTENT SYS005,111111
  // ASSGN SYS011,X'230'
  // DLBL WORKFIL,'PREFIX WORK FILE LR',0,SD
  // EXTENT SYS011,111111,1,0,1640,10
  // ASSGN SYS014,X'230'
  // DLBL INDXWRK,'PREFIX WORK FILE SI',0,SD                    - Note 1
  // EXTENT SYS014,111111,1,0,1660,20
  // DLBL STDCX2C,'SAMPLE.CUSTDX2',,VSAM
  // EXTENT SYS005,111111
  // DLBL STDCX1C,'SAMPLE.CUSTDX1',,VSAM                        - Note 2
  // EXTENT SYS005,111111
  // DLBL STDIX1C,'SAMPLE.INVDX',,VSAM                          - Note 2
  // EXTENT SYS005,111111
  // UPSI 00000010        NO LOG                                - Note 3
  // EXEC DLZRRC00,SIZE=300K
  ULU,DLZURGP0
  U
  /*
  /&
```

**Notes:**

1. These DLBL and EXTENT statements are required because the secondary indexes are to be created by this job.

2. These DLBL and EXTENT statements are always required for secondary indexes.

3. The UPSI byte setting is described in the *Utilities and Guide for the System Programmer* (Chapter 8).

# Chapter 7: DL/I Data Base Recovery/Restart

## Introduction

In any data processing environment, inevitably some type of failures or errors occur that damage the data being maintained. Successful recovery and restart from these failures and errors can be characterized by a five step process:

1. Detection of the error or failure
2. Determination of the cause of the error or failure
3. Recovery of the data from the error or failure
4. Correction of the cause of failure
5. Restart of processing

In traditional batch file processing, normally not much thought or effort is spent 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 7-1.

In a data base environment where timeliness of information is important, this simple procedure often is 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 inherent in the data base. 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 replace the interrelationships between the data bases. Nor can the jobs executed since the last dump be rerun because the related data bases already reflect the relationships created by the original execution of those jobs. To recover in the traditional manner, all related data bases have to be restored from earlier dumps. Unless the 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, tying up personnel in many user departments. In extreme cases the entire enterprise will be affected while the damage is corrected.

Recovery in an online environment is further complicated because other input to the online programs may not be available for reprocessing. For example, in a customer service environment where the terminal input may come from direct interaction with the customer over a telephone, the input data is often unreproducible.

DL/I provides several special mechanisms to overcome these problems and facilitate recovery in an online data base environment. Of course, these facilities can be used to advantage in developing a recovery system for a batch-only DL/I environment as well. These recovery and restart facilities include:

- A logging facility to capture all changes made to the data bases
- An abnormal termination routine to minimize data base damage on a program check or other condition that prevents a normal end of job
- A set of utility programs to correct errors or reconstruct data bases after damage
- A checkpoint facility to minimize the time required to restart processing

In addition to these facilities provided by DL/I, you will have to establish standards and procedures for detection of errors and failures and for subsequent recovery and restart. The facilities provided by DL/I are not complete by themselves. You have to combine their use with other facilities provided by VSAM, DOS/VS, and your own user written programs.

You are not required to use the recovery facilities provided by DL/I. However, an understanding of exactly what these facilities can provide will help you in planning for recovery in your application.

Recovery and restart procedures must be planned and designed as part of the design of your data base applications. As you define and develop a data base application, you should also define and develop recovery procedures for all possible failure conditions for the application. 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. This list would include:

- Power failure, "hard wait" in DOS/VS and other failures where the DL/I abnormal termination routine is not executed
- Physical damage to the data base that renders the data unreadable, for example, disk head crash, dropped disk pack, etc.
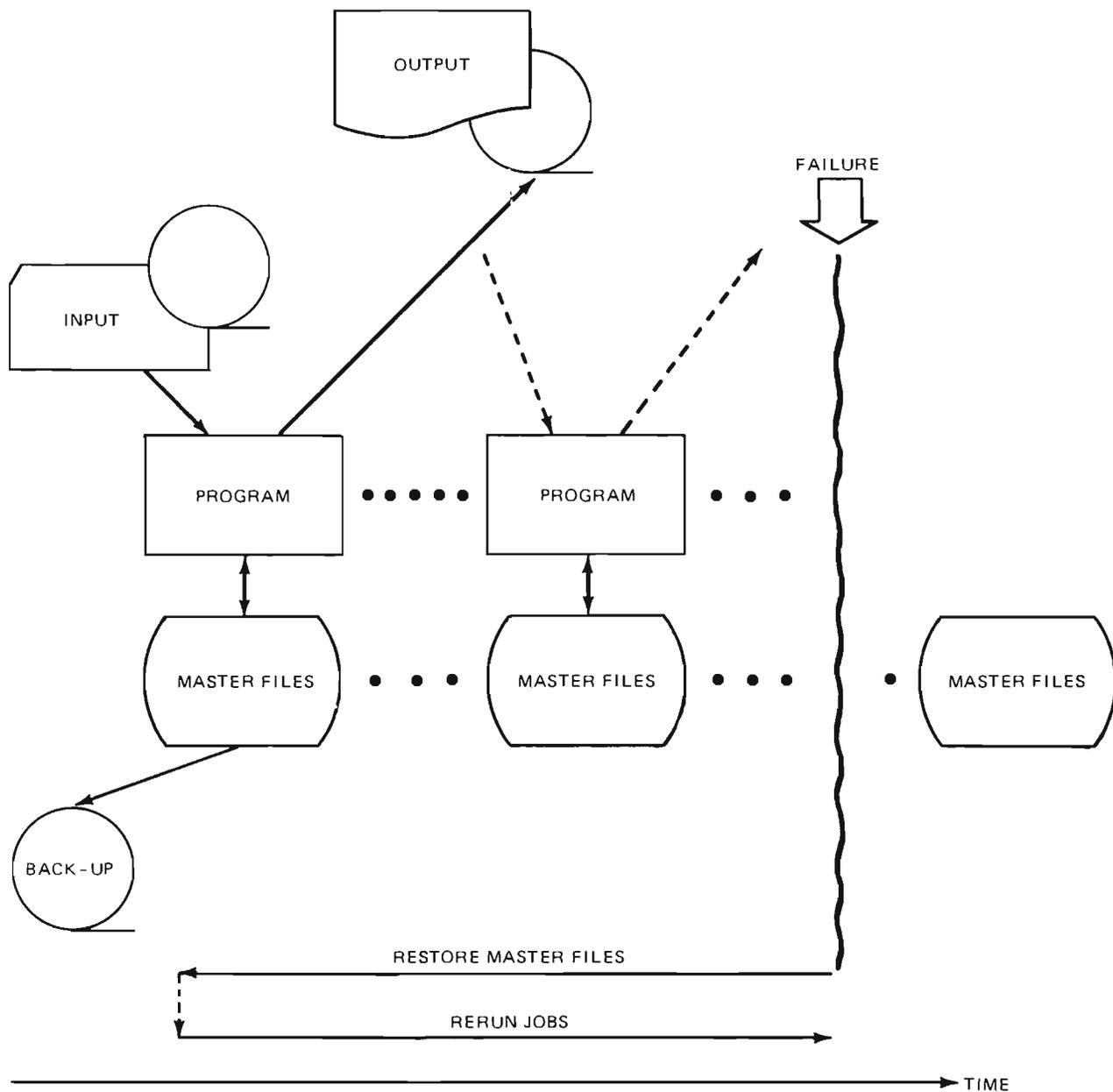
Figure 7-1. Traditional Recovery Approach

- Batch application program abend where the DL/I abnormal termination routine is executed

- Online transaction abends

- Logic error in an application program such that the program terminates normally but updates the data base incorrectly, for example, the application adds to a field in a segment when it should subtract

- 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", no room, etc.

By reviewing the various DL/I messages found in the DL/I Messages and Codes manual, 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 both? 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 dump

- Log record volumes, if logging is used, and consequently 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. The time to test your recovery procedures is before the first real failure. 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 5 minutes, and the planned data base will be ten times larger, then you can approximate the DL/I image copy time for the full data base as 50 minutes. This aids you in determining if your planned frequency of image copy is reasonable. A 50 minute image copy once a day may be reasonable where a 5 hour image copy once a day may not. Similar times should also be developed for such operational steps as DOS/VS 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. For example, a recovery flow-chart for a non-MPS batch program abend might look something like Figure 7-2.

Notice that this recovery procedure includes the use of facilities in addition to those provided by DL/I, for example, VSAM VERIFY. Often user written programs are required as part of a recovery procedure. For example, 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 flow-chart refer to additional narrative documentation. This documentation 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, for example, save for later analysis or destroy. This is especially important if sensitive or confidential data is involved.

- Instructions on how to execute the particular recovery step, including JCL, 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.

# 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 application program. For example, if an application program deletes a segment, all the segment's children are also deleted, even if the application program is not sensitive to them. This same delete request may 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 (that is, when the processing option "L" or "LS" is selected in the PCB).

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 medium. 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 act on the request.

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.
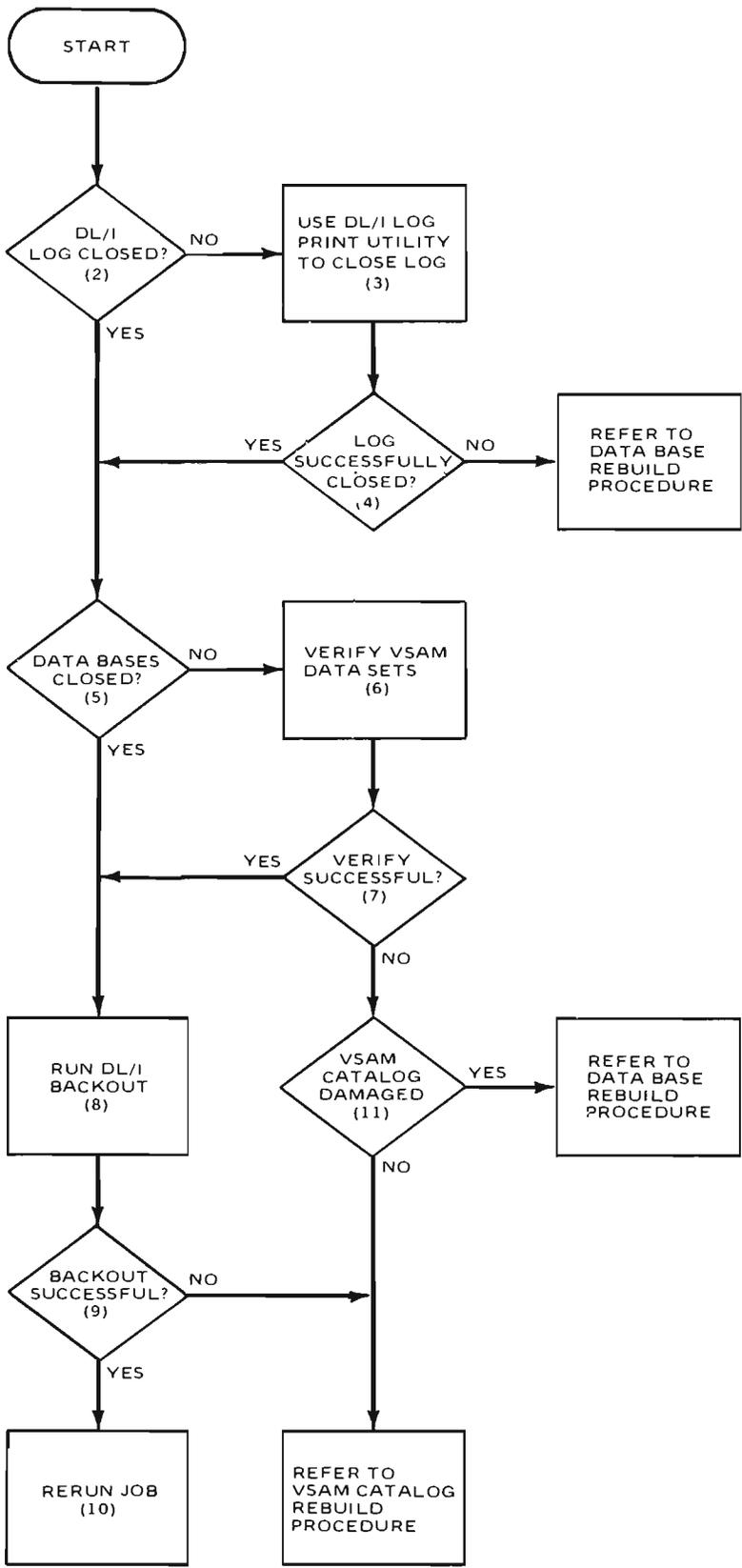
Figure 7-2. Sample Recovery Procedure Flowchart

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

There is one case where the data base can be physically updated before the corresponding log record is physically written to the log medium. This can occur if you elect to use the "asynchronous" logging option in a batch DL/I execution and a failure occurs of the type that prevents DL/I from executing its abnormal termination routine successfully, for example, a power failure. Asynchronous logging is provided as a performance option by DL/I. The potential disadvantage of this option 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.

The use of the asynchronous logging option is restricted by DL/I to batch jobs only. It may not be used in a CICS/VS or MPS environment. Generally, the performance improvements to be gained by using this option are small and not worth the risk of not being able to use the DL/I backout utility in the event of a failure.

### Logging and Performance

The performance of your application programs can be very dependent on the efficiency of the DL/I logging mechanism. You can effect 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 the efficiency of the logging mechanism. Physical writes to the log are required whenever:

- The log buffer fills up

- 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. Note that the effect of using the asynchronous logging

option is to skip the force writing of the log as described 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 will occur. Therefore, DL/I must assume 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 has complete knowledge of when physical writes to the data base are actually occurring. 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, either 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 may 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.

### Choosing the DL/I Log Medium

DL/I has the capability to write its log records on several different media. However, only one can be chosen for any DL/I program execution. Your choices are:

- VSAM ESDS on disk

- SAM data set on standard labeled tapes

- The CICS/VS system journal

In a batch non-MPS environment, you may use either of the first two choices.

In a batch-MPS environment, the logging is performed from the CICS/VS partition, and no log device is assigned in the batch-MPS partition.

In a CICS/VS environment, the last two choices are available. However, if either of the CICS/VS recovery facilities (Emergency Restart or Dynamic Transaction Backout) are to be used with DL/I tasks, the DL/I log must be assigned to the CICS/VS system journal. This choice usually performs better in a CICS/VS environment as well. The CICS/VS system journal is a SAM data set and can reside on either disk or tape. Note that, unlike batch DL/I logging, CICS/VS does not use standard labels on its tape journals. However, the DL/I utilities, with the exception of backout, accept all forms of log input. The DL/I backout utility will not accept a CICS/VS disk journal as input. Any CICS/VS journal records on the log input are ignored by the DL/I utilities.

If the DL/I log is assigned to the CICS/VS system journal, CICS/VS assumes all responsibility for opening and closing the logging device and performing I/O as required. DL/I still maintains its write ahead logging logic correctly when its log is assigned to the CICS/VS journal.

You should attempt to use the device that will give you the best performance for your log. Figure 7-3 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.

As can be seen in Figure 7-3, device start-stop time (rotational delay for disk devices), is generally the major factor in the log write time. This is more pronounced if the log records are shorter than 1K, which is often the case in an online environment.

| IBM Device type | Data transfer time for 1K bytes (ms) | Start-Stop 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 |

Figure 7-3. Example Log Write Times

# 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, 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 (non-MPS) 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 this message, the abnormal termination routine was entered because of a program check or some other condition that causes DOS/VS to invoke a STXIT AB exit. Possible causes of a STXIT AB wait can be found in *DOS/VS Supervisor and I/O Macros,* GC33-5373, 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 DOS/VS STXIT AB and STXIT PC macros in a batch environment to establish its abnormal termination routine. Consequently, your batch application programs should not use the STXIT AB and STXIT PC functions as well. To do so would disrupt the proper operation of DL/I's abnormal termination routine. While the STXIT AB and STXIT PC facilities are not directly usable in COBOL programs, any use of 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, via the UPSI byte settings, to bypass the use of DL/I's abnormal termination routine and thus DL/I's use of STXIT AB and STXIT PC. This can be done only in a non-MPS batch environment. This ability 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 however, as DL/I always resets PL/I's STXIT PC, even if the DL/I abnormal termination routine is bypassed. 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 may cause additional program errors due only to the incorrect status of the data bases and not because of an application program bug.

### Abnormal Termination in MPS

The DL/I abnormal termination routine in a DL/I batch-MPS program cannot be bypassed. In a batch-MPS 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 DOS/VS to invoke a STXIT AB exit. Possible causes of a STXIT AB can be found in *DOS/VS Supervisor and I/O Macros*, GC33-5373, 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 batch-MPS job is abending. If the condition that caused the abend originated in the batch-MPS 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 jobsteps in the job are not executed.

Because DL/I always uses STXIT AB and STXIT PC in a batch-MPS environment, your application programs should not use any facilities that require STXIT AB or STXIT PC. 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

There are three types of DL/I abnormal termination possible in CICS/VS:

1. An application task is abnormally terminating due to some reason that does not affect subsequent DL/I operation.

2. DL/I has detected an error internally and is terminating its operation.

3. CICS/VS has detected an error internally and is terminating its operation.

For a DL/I application task abend, including the abend of a MPS batch partition controller task, DL/I's abnormal termination routine performs the following functions:

- Writes a dump of the DL/I control blocks and related information that are unique to that task on the CICS/VS dump data set.

- If the abending task is a MPS batch partition controller task, the XECBs defined by that task are deleted.

- Issues a TERM call on behalf of the abending task to cause any data base records altered by that task that are still in main storage 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 this task are released.

- Writes a DL/I termination record on the log and a CICS/VS synch-point record on the CICS/VS system journal.

- Releases any DL/I resources that belong to this task, such as, PST, PPST, PSB, etc.

- 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. If CICS/VS dynamic transaction backout is specified for this task, CICS/VS performs that processing before the DL/I abnormal termination routine gains control.

For the condition where DL/I has detected an internal error that does not permit it to continue processing,

the DL/I abnormal termination routine performs the following functions:

- Abends all DL/I tasks for which it is currently processing a call with a CICS/VS abend code of "D062"

- Disables the CICS/VS-DL/I interface. An error code is returned in the TCA to any new DL/I tasks attempting to issue a call after this condition occurs.

- Writes message "DLZ062I" to the system console

- Returns control to CICS/VS

Note that no attempt is made to close DL/I data bases on this type of error. No DL/I task termination records are written on the log either. However, CICS/VS writes its own task termination record on the CICS/VS system journal.

Because of the presence of the CICS/VS end-of-task record on the CICS/VS system journal, CICS/VS emergency restart processing will not consider these tasks to be "in-flight" and thus will not attempt to back them out. However, the DL/I backout utility, if executed in batch, will back out the data base updates made by these tasks since there is no DL/I termination record on the log.

DL/I does not terminate CICS/VS on this condition. CICS/VS remains operational so that non-DL/I tasks may continue to be processed. When a CICS/VS shutdown 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.

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 DOS/VS 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 DOS/VS 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

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.

## Data Base Backout

When the status of a data base is uncertain because the program that was updating the data base terminated abnormally, the data base backout utility may be used to eliminate (back out) the effects of the program. This utility reads backwards the log created during the processing of the erroneous program. Using the data base log records thus read, 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 HISAM reorganization unload/reload utilities are executed immediately after a successful data base backout utility execution.

The data base backout utility removes (backs out) the effects of any user program that accessed data bases

using DL/I calls. If the program was operating in a DL/I batch partition, 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 calls, 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 to reflect the backout history. The log must be included in any data base recovery attempted for the data bases involved in the backout.

Input to the data base backout utility as illustrated in Figure 7-4 consists of:

1. One PSB and one or more DMBs loaded from a DOS/VS core image library during DL/I initialization.

2. The input data base(s) with which the data base backout utility is to execute.

3. The DL/I log for the DL/I job execution which abnormally terminated.

Output from the data base backout utility as illustrated in Figure 7-4 consists of:

1. The data bases reflecting the status prior to the DL/I job execution which abnormally terminated.

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 SYSLIST and SYSLOG devices.

For online operation, DL/I backout can be invoked automatically by CICS/VS emergency restart, or dynamic transaction backout. See the *CICS/VS System Application Design Guide*, SC33-0068, for more information.
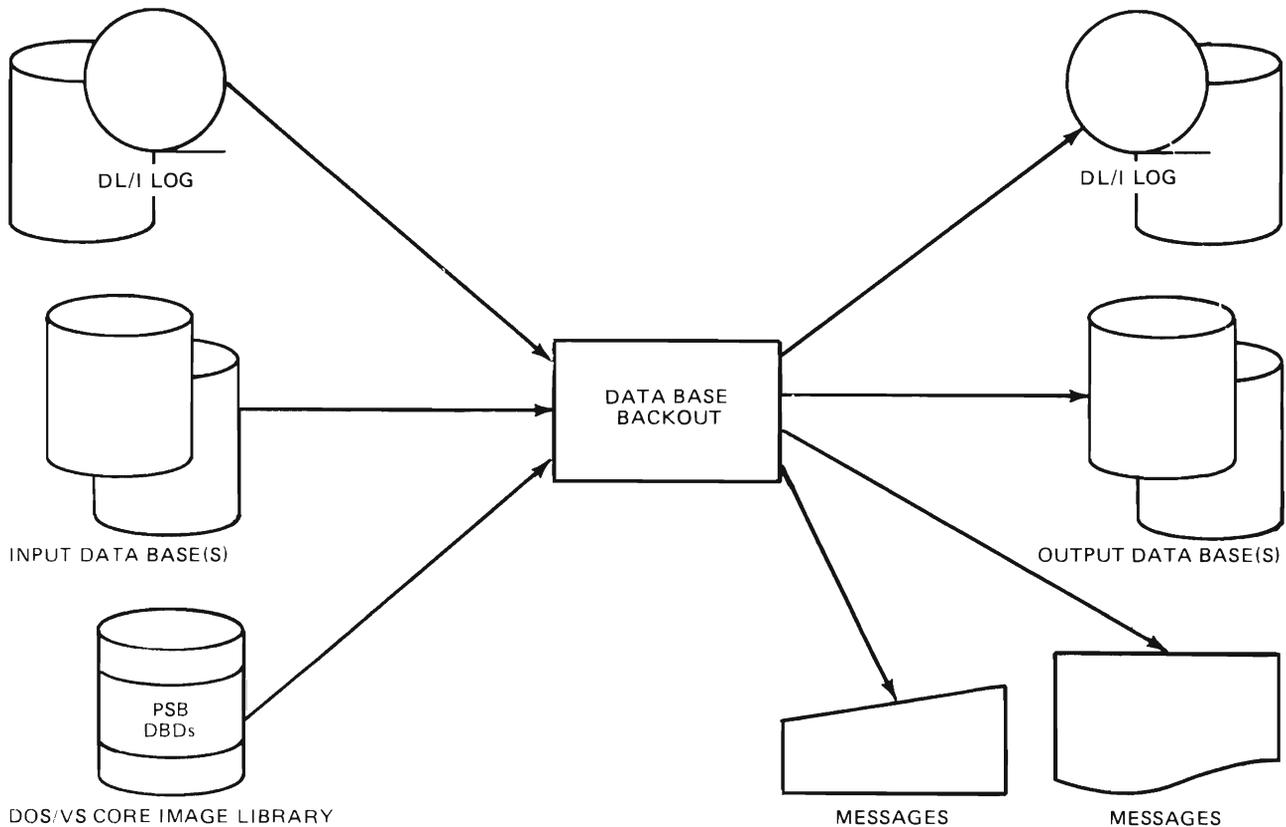


Figure 7-4. Data Base Backout Utility

## 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 logs necessary for data base recovery depends upon 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 logs 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. Thus the information necessary for data base recovery is contained within 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 synonomous with a data base. This is true with HDAM, INDEX, HIDAM, 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 7-5, are required to acccomplish 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 may 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.
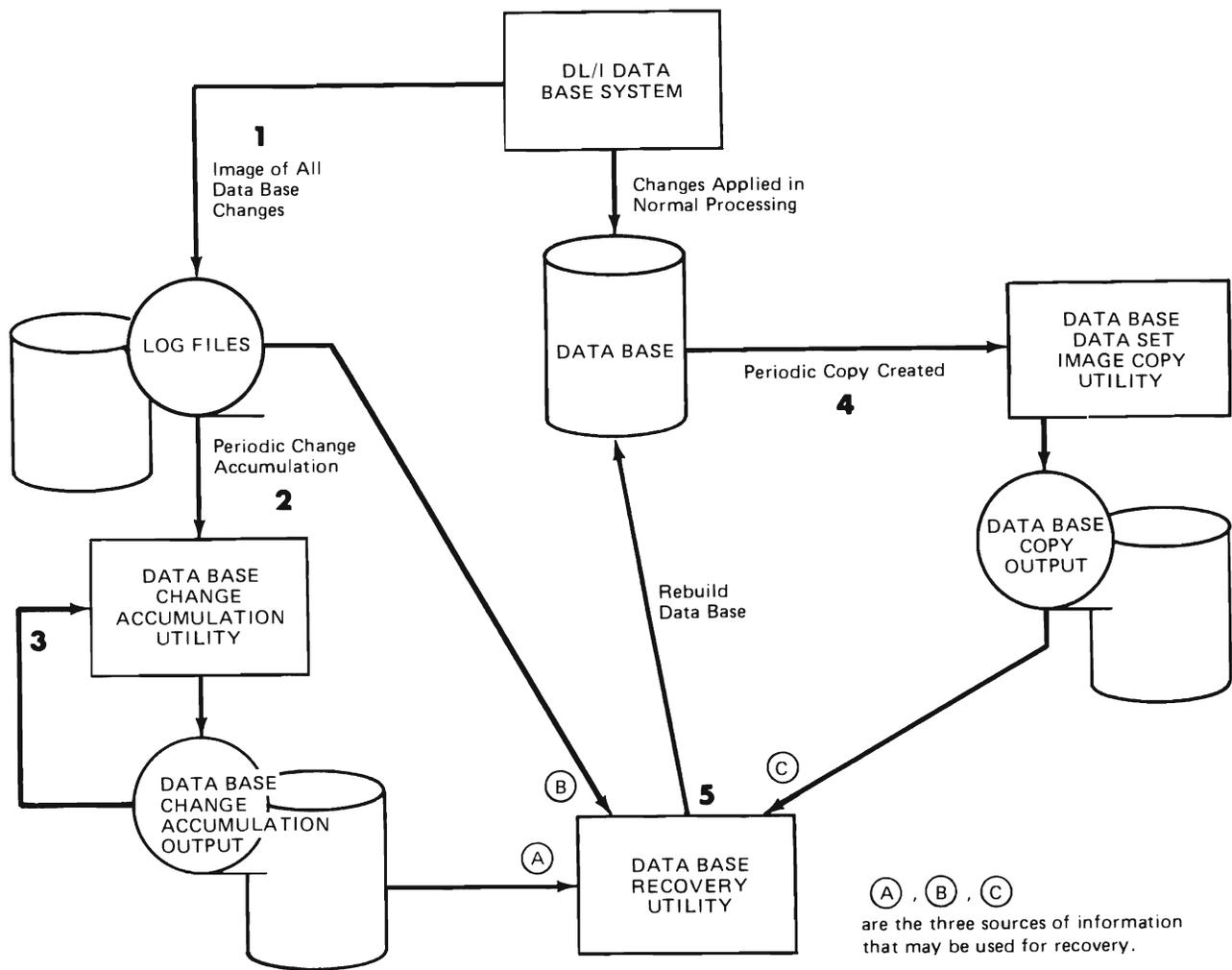
Figure 7-5. Data Base Recovery

# 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 DOS/VS 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 the DOS/VS facility.

## *CHKP (Checkpoint) Call*

The CHKP call 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 CHKP call 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 CHKP is issued, a PSB scheduling call is not required after the CHKP call. In fact, a scheduling call issued under these circumstances causes a scheduling error. (See the *Application Programming Reference Manual,* "Checking the Response to a Scheduling or Termination Call".)

You should issue periodic CHKP calls 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 CHKP call 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 the CICS/VS journalling be

active. See the *Application Programming Reference Manual* for additional details about the CHKP call.

## *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 call, "CHKP", at intervals during your program's execution, you can shorten the time required for this recovery step. When a batch non-MPS program issues a checkpoint call, 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 call has been sucessfully executed.

4. Returns control to the application program.

Thus after a checkpoint call, all data base changes are reflected on the data bases. However, the data bases are not closed on a checkpoint call. Note that after a checkpoint call your program must reestablish position in the data bases before continuing.

If your program were to fail immediately after the checkpoint call, 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 consequently reduces the time required to back out changes after a failure. The effect of a checkpoint call 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 7-6 illustrates the relationship between the checkpoint records and the DL/I backout utility.

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 call 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 call 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 8-byte identification field that your program can specify in the checkpoint call. This information can be used as input to your program when it is restarted to assist it in reestablishing 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 DOS/VS CHKPT macro and similar facilities in COBOL and PL/I cannot be used in your non-MPS batch DL/I programs. The DOS/VS 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 call, thus preventing the use of the DOS/VS CHKPT facility.

## *DL/I Checkpoint in Batch MPS Programs*

DL/I checkpoint should only be used in a batch-MPS 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 PPT entry.

When a DL/I checkpoint call is issued in a DL/I batch-MPS 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 synch-point record on the CICS/VS system journal.

3. Dequeues any records or segments enqueued for this program if PI is being used.
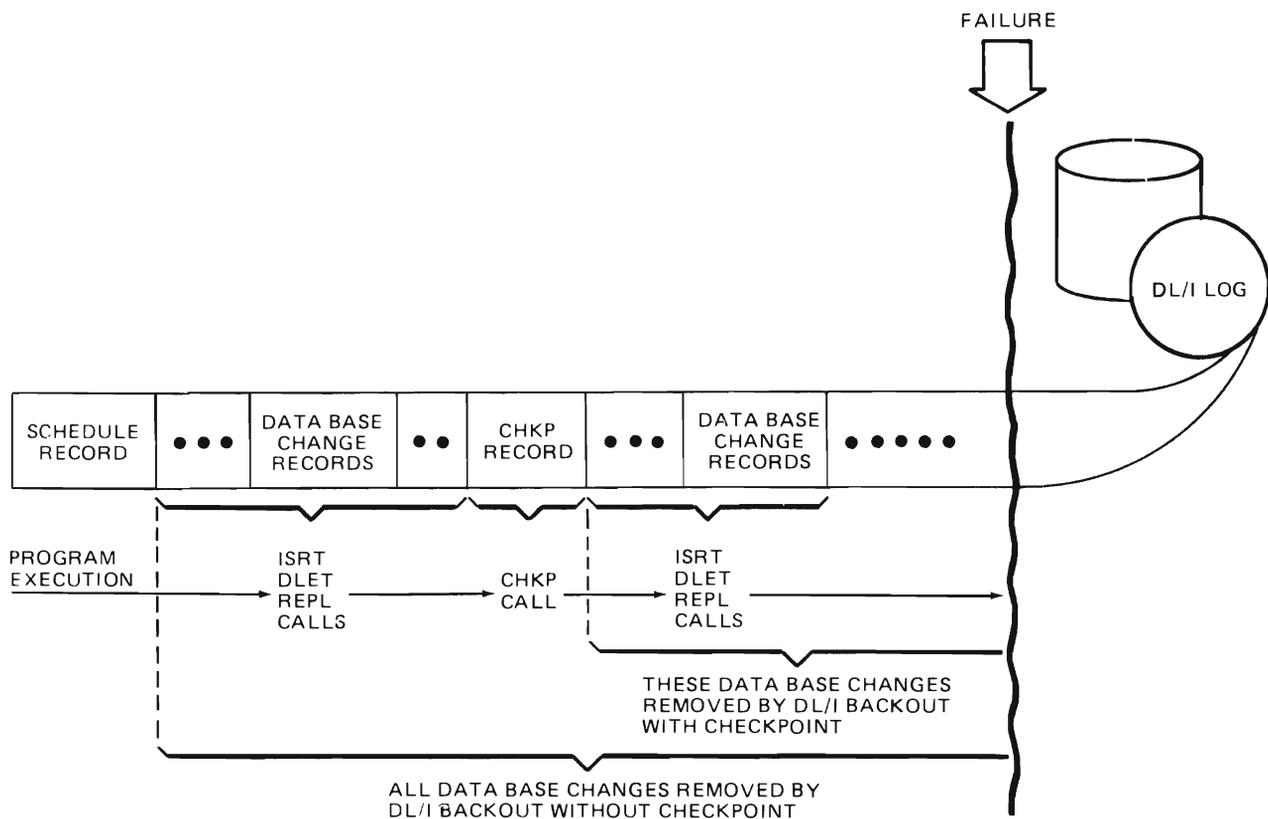
Figure 7-6. Checkpoint Records and DL/I Backout

4. Writes message "DLZ105I" to the system console indicating that a DL/I checkpoint call has been successfully executed.

5. Returns control to the application program.

If a failure subsequently occurs, CICS/VS invokes its dynamic transaction backout facility and backs out all data base changes made since the last checkpoint call was issued.

As is the case with non-MPS batch programs, your batch-MPS program must reestablish the environment at the point of the last checkpoint call when restarting after a failure. Unlike the non-MPS batch environment, however, there are no open DL/I VSAM data sets in the batch-MPS partition. Therefore, your programs may use the DOS/VS CHKPT facilities as long as they obey the other restrictions imposed by DOS/VS. However, there is no restart facility in DL/I CHKP, so do not use it with DOS/VS CHKPT.

## 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. This will prevent more than one partition from updating the data set at a time. Note that DL/I always opens its data base data sets for update, regardless of the intent of the PSB. Therefore, share option 2 provides no functional benefit for DL/I's VSAM data sets. 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 DOS/VS 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, DOS/VS 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, e.g., due to a power failure or VSAM control block damage, 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 insure that the high RBA value in the catalog is correct.

# Chapter 8: DL/I Sample Application

The DL/I sample application utilizes the Phase 3 customer and inventory data bases, and demonstrates an online order processing application. The sample application along with this manual can be very useful to the data base administrator, system programmer, and application programmer as an aid in understanding and implementing a DL/I data base system.

The sample application represents a fictitious wholesale distribution firm that offers a wide variety of electronic components. The components are purchased from various vendors and sold to customers. Most customer orders are processed by telephone, so an application program was developed to implement an interactive online order inquiry - order entry system.

The application programs, written in Assembler language, consist of a load program (DLZSAM40), a batch print program (DLZSAM50), and the online sample application program (DLZSAM60). These programs have been assembled, link edited and placed in the core image library. The mapping module (DLZMAPS) needed for DLZSAM60 is also included in the core image library. In addition, a sample compression/expansion routine (DLZSAMCP) is included to demonstrate the segment edit/compression facility of DL/I for variable length segments.

The source code for equivalent programs written in COBOL, PL/I, and RPG II is also included in File 3 of the PID distribution tape. They are located in the DL/I Optional Source Statement Library as A. Books. The program names are:

| COBOL | PL/I | RPG II |
|---|---|---|
| DLZCBL40 | DLZPLI40 | DLZRPG40 |
| DLZCBL50 | DLZPLI50 | DLZRPG50 |
| DLZCBL60 | DLZPLI60 | DLZRPG60 |
| DLZCBMAP | DLZPLMAP | DLZRGMAP |

These source code files differ from the Assembler language programs only in that they do not include an example of field level sensitivity.

This sample application does not include a job to create the VSAM master catalog. It is assumed that your installation has already defined the VSAM master catalog and any optional user catalogs for VSAM. If your installation has not done this, see "Defining the VSAM Master Catalog" in this chapter.

## Sample Application Job Stream

File 7 of the PID distribution tape contains the DOS/VS job control and input data statements necessary to define the logical and physical data structures to DL/I and VSAM and to execute the online sample application programs. The online sample application utilizes the phase 3 data bases as described earlier in this manual. The sample program source and object modules are included with the DL/I source and object modules.

The DL/I online sample application assumes a 3340 direct access storage device with the volume label '111111', residing on unit 230. If the unit you assigned is other than 230, you will want to punch a new card to assign the proper unit. Volume '111111' is chosen because this is normally used as a work pack by most users, or a work pack can easily be renamed to this.

You should also check the EXTENT statements in the JCL for the utility programs to insure that the relative track specification and number of tracks specified meet the requirements of your installation. The JCL for the reorganization and logical relationship resolution utilities used for the online sample application is included in Chapter 6 of this manual.

File five on the PID tape contains the following jobstream:

- Assemble DBDs (STJDBDGN)
- Assemble PSBs (STJPSBGN)
- ACB generation (STJACBGN)
- Prereorganization Utility (STJPREOR)
- Access Method Services DEFINE (STJDFINV)
- Data Base Load (STJLDCST, executes DLZSAM40)
- Prefix Resolution Utility (STJPRRES)
- Prefix Update Utility (STJPRUPD)
- List Data Bases (STJPLIST, executes DLZSAM50)

Because the DBD generation, PSB generation, ACB generation, access method services DEFINE job, and utilities for the online sample application are discussed in earlier chapters, this chapter will cover only DLZSAM40, DLZSAM50, and DLZSAM60.

# Defining the VSAM Master Catalog

These DOS/VS job control and input job statements maybe used to define the master catalog in volume '111111'. This job may be skipped if you wish to use your own catalog. If this job is executed, be sure you have previously named the device address for the master catalog during IPL so that it is consistent with the assignment in this job.

```
// JOB DEFINE MASTERCATALOG
// ASSGN SYS000,X'336'
// DLBL IJSYSCT,'AMASTCAT'
// EXTENT SYSCAT,111111,1,0,20,20
*
* ***** IF NO MASTER CATALOG CREATION DESIRED CANCEL - ELSE EOB
*
* ***** USER MAY ENTER NEW ASSIGNMENT FOR SYS000
* ***** IF OTHER THAN ABOVE
*
// PAUSE
//   EXEC   IDCAMS,SIZE=25K
  DEFINE                -
  MASTERCATALOG (       -
     NAME(AMASTCAT)     -
   FILE(IJSYSCT)        -
     VOLUMES(111111)    -
     CYLINDERS(1 0))
/*
/&
```

# DLZSAMCP - Sample Program Compression/Expansion Routine

DLZSAMCP is a sample routine that demonstrates one way to use the segment edit/compression facility of DL/I to optimize storage required for individual occurrences of variable length segments.

The function of this routine is to remove all trailing blanks from the data portion of the variable length segment, STSCHIS, in the customer data base before that segment is stored in the data base.

When the segment is called by an application program, the expansion routine replaces the blanks removed by the compression routine to restore the segment to its maximum length for processing by the application program.

This routine is invoked by DL/I whenever the variable length segment, STSCHIS, is processed by the load program, or by the batch or online application programs.

```
              TITLE 'DLZSAMCP - ONLINE SAMPLE PROBLEM COMPRESSION/EXPANSION X
                    ROUTINE'
       PUNCH '           CATALR DLZSAMCP,1.4'
       PUNCH '           PHASE DLZSAMCP,*'
*************************************************************************
*                                                                     *
*  DLZSAMCP - SAMPLE PROBLEM COMPRESSION/EXPANSION ROUTINE.    aD14D399 *
*    THIS ROUTINE PERFORMS THE COMPRESSION/EXPANSION FUNCTIONS FOR THE  *
*    VARIABLE LENGTH SEGMENT STSCHIS IN THE CUSTOMER DATA BASE.         *
*         THE COMPRESSION ROUTINE REMOVES ALL TRAILING BLANKS BEFORE    *
*         THE SEGMENT IS STORED IN THE DATA BASE.                       *
*                                                                     *
*         THE EXPANSION ROUTINE PADS THE SEGMENT FROM THE END OF THE    *
*         STORED DATA OUT TO THE MAXIMUM SEGMENT LENGTH.                *
*                                                                     *
*    THE INPUT TO THIS PROGRAM IS AS FOLLOWS:                          *
*              REGISTER 1  - PST ADDRESS                               *
*              REGISTER 2  - ADDRESS OF FIRST BYTE OF SEGMENT TO BE     *
*                            PROCESSED (SOURCE ADDRESS)                *
*              REGISTER 3  - ADDRESS OF WORK AREA WHERE SEGMENT IS TO   *
*                            BE MOVED (DESTINATION ADDRESS)            *
*              REGISTER 4  - PSDB ADDRESS                              *
*              REGISTER 5  - ADDRESS OF SEGMENT COMPRESSION TABLE (SEE  *
*                            DMBCPAC DSECT)                            *
*              REGISTER 6  - ENTRY FUNCTION CODE                       *
*                            0 - COMPRESS SEGMENT                      *
*                            4 - EXPAND SEGMENT                        *
*              REGISTER 13 - SAVE AREA ADDRESS                         *
*              REGISTER 14 - DL/I RETURN ADDRESS                       *
*              REGISTER 15 - ENTRY ADDRESS OF DLZSAMCP                 *
*************************************************************************
       EJECT
DLZSAMCP START
       USING *,R15
       ST    R13,SAVE+4
       STM   R14,R12,12(R13)    SAVE DL/I'S REGS FOR RETURN
       LA    R13,SAVE           SAVE AREA FOR THIS ROUTINE
*************************************************************************
*  CHECK REGISTER 6 TO DETERMINE IF COMPRESS OR EXPAND.               *
*************************************************************************
       LTR   R6,R6              IF ZERO, THEN COMPRESS FUNCTION
       BZ    COMPRESS
       SPACE 3
*************************************************************************
* THIS IS THE EXPANSION ROUTINE.  THE SEGMENT IS EXPANDED AND MOVED    *
* INTO THE DESTINATION ADDRESS SPECIFIED IN REGISTER 3.  THE           *
* SEGMENT WILL ALWAYS APPEAR AS THE MAXIMUM LENGTH TO THE APPLI-       *
* CATION PROGRAM.                                                      *
*************************************************************************
       SPACE 1
       LA    R8,2(R2)           ADDRESS OF DATA FOR SOUCE
       LA    R10,2(R3)          ADDRESS OF DATA FOR DESTINATION
       USING DMBCPAC,R5
       LH    R11,DMBCPSGL       PICK UP MAXIMUM SEGMENT LENGTH
       STH   R11,0(R3)          NEW LENGTH TO SEGMENT
       LH    R9,0(R2)           PICK UP COMPRESSED SEGMENT LENGTH
       SH    R9,=H'2'           MINUS 2 BYTE LENGTH FIELD
       SH    R11,=H'2'          MINUS 2 BYTE LENGTH FIELD
       L     R7,=X'40000000'    LOAD PADDING CHARACTER FOR MVCL
       OR    R9,R7              PUT PAD CHARACTER IN RIGHT REG
       MVCL  R10,R8             MOVE SEGMENT AND EXPAND TO MAX
       SPACE 3
EXIT   EQU   *
       L     R13,SAVE+4         ADDRESS OF DL/I'S SAVE AREA
       LM    R14,R12,12(R13)    RESTORE DL/I REGISTERS
       BR    R14                RETURN TO DL/I
       SPACE 3
```

```
**************************************************************************
* THIS IS THE COMPRESSION ROUTINE.  THE TRAILING BLANKS ARE           *
* REMOVED FROM THE SEGMENT AND THE LENGTH FIELD IS UPDATED TO         *
* REFLECT THE NEW LENGTH OF THE SEGMENT.                              *
**************************************************************************
COMPRESS EQU    *
         LH     R10,DMBCPSGL        PICK UP MAX SEGMENT LENGTH
         LA     R9,0(R2,R10)        STEP TO SEGMENT END
         SH     R9,=H'1'            BACK UP TO LAST CHARACTER
NEXT     EQU    *
         CLI    0(R9),C' '          IS THIS POSITION BLANK?
         BNE    DONE                NO, THEN ALL BLANKS ARE OUT
         BCTR   R9,0                BACK UP ONE POSITION
         BCT    R10,NEXT            REDUCE SEGMENT LENGTH AND LOOP
DONE     EQU    *
         STH    R10,0(R3)           COMPRESSED LENGTH TO SEGMENT
         SH     R10,=H'2'           COMPRESSED DATA LENGTH
         LR     R9,R10              CORRECT REG FOR MVCL
         LR     R11,R9              OTHER LENGTH IS SAME FOR MVCL
         LA     R8,2(R2)            DATA ADDRESS FOR SOURCE
         LA     R10,2(R3)           DATA ADDRESS FOR DESTINATION
         MVCL   R10,R8              MOVE ONLY DATA TO DESTINATION
         B      EXIT
SAVE     DS     18F                 SAVE AREA FOR DLZSAMCP
         EJECT
         DLZQUATE                   REGISTER EQUATES
DMBCPAC  DSECT                      SEGMENT COMPRESSION TABLE
DMBCPCNM DS     CL8                 SEGMENT NAME
DMBCPCSG DS     CL8                 COMPRESSION ROUTINE NAME
DMBCPEP  DS     A                   COMPRESSION ROUTINE ENTRY ADDR
DMBCPFLG DS     XL1                 FLAG BYTE
DMBCPSQF DS     XL1                 EXECUTABLE LENGTH OF SEQ. FIELD
DMBCPSQL DS     H                   SEQUENCE FIELD OFFSET IN SEGMENT
DMBCPSGL DS     H                   MAXIMUM SEGMENT LENGTH
DMBCPLNG DS     H                   LENGTH OF CSECT
         END
```

# DLZSAM40 - DL/I Online Sample Load Program

DLZSAM40 is the load program for the data bases used by the online sample application program, DLZSAM60. All input to the program is from SYSRDR. The program reads the card images from SYSRDR, constructs the data base segments from the card images, and issues DL/I insert calls to load the segments. The segment input for the inventory data base must be in the reader first. The segment input for the customer data base must be preceded by a card with the word 'CUSTOMER' in the first 8 columns. Each segment name must be in columns 1-7 of the input cards. The segment data starts in column 8 following the segment name. If continuation cards are needed for segment data, use a non-blank continuation punch in column 72. The segment data on continuation cards starts in column 1.

The PSB name STBICLD is referenced in the parameter information statement. This name is converted by the DL/I initialization module (DLZRRC00) to STBICLDP so that the internal DMB and PSB control blocks can be loaded.

The DOS/VS job control and DL/I input statements to execute DLZSAM40 are as follows:

```
// JOB STJLDCST LOAD INVENTORY AND CUSTOMER DATA BASES
// OPTION PARTDUMP
// ASSGN SYS005,X'230'
// DLBL STDIDBC,'SAMPLE.INVEN',,VSAM
// EXTENT SYS005,111111
// DLBL STDCDBC,'SAMPLE.CUST',,VSAM
// EXTENT SYS005,111111
* NOTE: NO DLBL EXTENTS FOR SECONDARY INDEXES (BUILT BY PREFIX UPDATE)
// DLBL CONTROL,'CONTROL FILE',0,SD
// EXTENT SYS012,111111,1,0,1680,10
// ASSGN SYS012,X'230'
// DLBL WORKFIL,'WORKFILE J',0
// EXTENT SYS013,111111,1,0,1690,5
// ASSGN SYS013,X'230'
// UPSI 00000010       NO LOG
// EXEC DLZRRC00,SIZE=500K
DLI,DLZSAM40,STBICLD,1,HDBFR=(6)
STPIITM000100INTEGRATED CIRCUIT        00250000200000000000000010
STSIVND000010COMPANY A INC.            207 FREY AVENUE         ENDICOTTX
,NY 13760        MR. JOHN DOE
STCISUB000300
STSILOC000001   12-2
                •
                •
                •
CUSTOMER
STSCCST000001COMPANY X INC.            10 MAIN STREET          NEW YORKX
, NY 10010       MR. JOHN SMITH
STSCLOC000010EASTERN REGION            69 BROAD STREET         PHILADELX
PHIA, PA 11020   MR. JOHN DOE
STPCORD770129100500FIRST 1977 ORDER          000040000000000400
STCCITM000100010000400000400000000000000000400
STSCLOC000020WESTERN REGION            5296 BATTLESHIP BLVD.   SAN DIEGX
O, CA 93210      MR. JOHN SMITH
STPCORD770510102050SECOND 1977 ORDER         000035000000000088
STCCITM000200010000180000080000100000000000054
STCCITM000300020000170000170000000000000000034
STSCSTA0000002500000000000001000
STSCHISSL761205928654LAST 1976 ORDER          10000000015000ORDER SHIPPX
ED COMPLETE AND ON TIME
                •
                •
                •
/*
/&
```

# DLZSAM50 - DL/I Online Sample Print Program

DLZSAM50 is the print program. This program prints the customer and inventory data bases as loaded by DLZSAM40. This program uses the logical DBDs for the sample application and is dependent upon the order of the segments as defined to format the output listing. Both data bases are printed with the customer list appearing first. This program issues a series of unqualified DL/I get next calls to access all segments in both the customer and inventory data bases through logical relationships from the customer data base.

The DOS/VS job control and DL/I input statements to execute DLZSAM50 are as follows:

```
// JOB STJPLIST LIST INVENTORY AND CUSTOMER DATA BASES
// OPTION PARTDUMP
// ASSGN SYS005,X'230'
// DLBL STDIDBC,'SAMPLE.INVEN',,VSAM
// EXTENT SYS005,111111
// DLBL STDCDBC,'SAMPLE.CUST',,VSAM
// EXTENT SYS005,111111
// DLBL STDCX2C,'SAMPLE.CUSTDX2',,VSAM
// EXTENT SYS005,111111
// DLBL STDCX1C,'SAMPLE.CUSTDX1',,VSAM
// EXTENT SYS005,111111
// DLBL STDIX1C,'SAMPLE.INVDX',,VSAM
// EXTENT SYS005,111111
// UPSI 00000010       NO LOG
// EXEC DLZRRC00,SIZE=500K
DLI,DLZSAM50,STBICLG,1,HDBFR=(6)
/*
/&
```

The output listing for each customer has the following format:

```
                         LIST OF CUSTOMER DATA BASE
NAME: COMPANY X INC.             NUMBER: 000001  CONTACT: MR. JOHN SMITH
   STREET: 10 MAIN STREET               CITY: NEW YORK, NY 10010
      REGION: EASTERN REGION            CONTACT: MR. JOHN DOE
         STREET: 69 BROAD STREET        CITY: PHILADELPHIA, PA 11020
         01/29/77   ORDER NUMBER: 100500   DESCRIPTION: FIRST 1977 ORDER        DOLLAR AMOUNT: $      400.00
            ITEM NUMBER       DESCRIPTION           ORDERED   SHIPPED  BACK ORDERED
            000100         INTEGRATED CIRCUIT       000040    000040     000000
      REGION: WESTERN REGION            CONTACT: MR. JOHN SMITH
         STREET: 5296 BATTLESHIP BLVD.  CITY: SAN DIEGO, CA 93210
         05/10/77   ORDER NUMBER: 102050   DESCRIPTION: SECOND 1977 ORDER       DOLLAR AMOUNT: $       88.00
            ITEM NUMBER       DESCRIPTION           ORDERED   SHIPPED  BACK ORDERED
            000200         TRANSISTOR               000018    000008     000010
            000300         RESISTORS                000017    000017     000000
      CREDIT BALANCE: $    1000.00   AMOUNT LAST ORDER: $      15000.00
      LAST ORDER DATA: ORDER SHIPPED COMPLETE AND ON TIME
```

The output listing for each inventory item has this format:

```
                         LIST OF INVENTORY DATA BASE
ITEM NUMBER: 000100   DESCRIPTION: INTEGRATED CIRCUIT        UNIT COST: $    10.00
   QUANTITY ON HAND: 002500     QUANTITY ON ORDER: 002000
      OPEN ORDERS:    DATE     ORDER NUMBER   ORDERED   SHIPPED
                    01/29/77      100500      000040    000040
      VENDOR NAME: COMPANY A INC.              CONTACT: MR. JOHN DOE
            STREET: 207 FREY AVENUE               CITY: ENDICOTT, NY 13760
   WAREHOUSE LOCATION:    12-2    SUBSTITUTE ITEM NUMBER: 000300
```

# DLZSAM60 - DL/I Online Sample Application Program

DLZSAM60 is an assembler written application program that runs as a transaction under DOS/VS CICS/VS and which accesses and updates several business oriented data bases using DL/I. DLZSAM60 makes use of the basic mapping support feature of CICS/VS to request directions and data from terminal users. This program works only with IBM 3277 terminals and makes no use of the optional 3277 terminal program function keys.

In order to use DLZSAM60, you must first run the jobstream supplied on the DL/I PID distribution tape to define and load the sample customer and inventory data bases. This jobstream also includes DL/I utility jobs which create the secondary index data bases used with the customer and inventory data bases.

DLZSAM60 also requires several additions to your CICS/VS control tables. Entries must be placed in the DFHFCT table for the following DBD names:

- STDCDBP
- STDIDBP
- STDIXIP
- STDCX1P
- STDCX2P

An entry must also be placed in the DFHPCT table for the transaction id (in our example, we used DLZZ). In addition entries are needed in the DFHPPT for the program name, DLZSAM60, and for DLZMAPS, the mapping module. Besides these CICS/VS required entries, you must add an entry to the DLZACT DL/I online nucleus generation job that your installation has prepared. The entry is for DLZSAM60 and the PSB names which that program will schedule (use): STBCUSR, STBCUCU. Also, remember that the JCL for the data bases must be included in the statements used to start CICS/VS. The JCL is the DLBL and extent statements used to define the customer, inventory, and secondary index data bases to the DOS/VS system.

Examples of the types of entries needed are:

### DFHFCT - CICS/VS File Control Table

```
DFHFCT TYPE=DATASET,DATASET=STDCDBP,ACCMETH=DL/I
DFHFCT TYPE=DATASET,DATASET=STDIDBP,ACCMETH=DL/I
DFHFCT TYPE=DATASET,DATASET=STDIX1P,ACCMETH=DL/I
DFHFCT TYPE=DATASET,DATASET=STDCX1P,ACCMETH=DL/I
DFHFCT TYPE=DATASET,DATASET=STDCX2P,ACCMETH=DL/I
```

### DFHPCT - CICS/VS Program Control Table

```
DFHPCT TYPE=ENTRY,TRANSID=DLZZ,                          X
       PROGRAM=DLZSAM60,TWASIZE=2048,                    X
       INBFMH=EODS,LOGREC=NO
```

### DFHPPT - Processing Program Table

```
DFHPPT TYPE=ENTRY,PROGRAM=DLZMAPS     MAPS FOR ONLINE PROG
DFHPPT TYPE=ENTRY,PROGRAM=DLZSAM60,SAMPLE ONLINE PROGRAM  X
       RELOAD=YES
```

### DLZACT - DL/I Online Nucleus Generation

```
DLZACT TYPE=PROGRAM,                                     X
       PGMNAME=DLZSAM60,         ONLINE SAMPLE PROGRAM    X
       PSBNAME=(STBCUSR,STBCUSU)
```

### DOS/VS Job Control Language for the Data Bases

```
// ASSGN SYS005,X'230'
// DLBL STDCX2C,'SAMPLE.CUSTDX2'
// EXTENT SYS005,111111
// DLBL STDCX1C,'SAMPLE.CUSTDX1'
// EXTENT SYS005,111111
// DLBL STDIX1C,'SAMPLE.INVDX'
// EXTENT SYS005,111111
// DLBL STDIDBC,'SAMPLE.INVEN'
// EXTENT SYS005,111111
// DLBL STDCDBC,'SAMPLE.CUST'
// EXTENT SYS005,'SAMPLE.CUST'
// EXTENT SYS005,111111
```

DLZSAM60 allows the terminal user to enter and query information about a customer order. The customer can have one or many individual locations placing orders. The information about customers and orders is kept in the customer data base while the information about the items the sample company sells is kept in the inventory data base. Using DL/I's facilities, the data bases are connected so that orders entered automatically cause updates to the inventory data base without the need to run some type of program to change inventory status due to orders received.

Each customer location can place one or many orders. The order entry terminal operator assigns an order number, order description and order date with each order received. The order itself consists of item numbers and quantities. Our sample company describes each item it sells, but requests that it be ordered by item number rather than by item name. Any one order is restricted to a maximum of five (5) order items. This restriction is caused by the way the DLZSAM60 application program accepts orders from the terminal. There is also a quantity restriction of 9,999 for any one item. Again, these restrictions are not caused by DL/I or the data bases. They simply reflect the method DLZSAM60 uses to process the data. These restrictions could be changed if the sample company decides that DLZSAM60 does not provide adequate function.

Order entry consists of placing an order for:

- a new customer and new location, or

- an existing customer and new location, or

- an existing customer and existing location.

The order entry requires an order number, order description, item numbers and quantity of each item desired.

After each order is entered, DLZSAM60 displays on the terminal the results of the order. That is, a listing of the customer, location, order and item information associated with this new order. For example, if the order data looks like this to our terminal operator:

```
customer number = 111111
location number = 002200
date = 1/11/77
order number = 232323
order description = '1977 first order'
items = 000100/55
        000200/200
        000300/157
```

and the operator places this data in the proper fields on the 3277 terminal screen, then DLZSAM60 will respond with a screen full of data describing the company name, address, contact, location name, address and contact, order date, description number, items ordered, item number, cost per item, amount ordered, amount shipped, amount back ordered and total cost for the order. This example assumes that the order is for an existing customer and existing location. Similar results are obtained when the order is for a new location or new customer. The terminal input, however, will increase. The action of entering an order automatically (through DLZSAM60) causes the inventory data to be updated to show the drop in stock or the need for back-ordering items.

DLZSAM60 also allows our sample company to query order information. They can look at an order, find the status of the items: amounts shipped or back ordered, cost per item, total cost of order, etc. They can also discover exact customer and location information associated with a specific order. They can query the orders

associated with a specific customer location and the locations associated with a specific customer. They can list this information by supplying the DLZSAM60 program with the information known about an order, customer or location such as: order date, customer name or number, location name or number.

## DLZSAM60 Screen Formats

The following examples show the 3270 screen formats presented by DLZSAM60. DLZSAM60 is activated by entering the transaction ID, (in this case, DLZZ). The screen formats are designed to allow data entry using a basic 3270 keyboard. No program function keys are used. Curser positioning to the proper input field is accomplished by the tab (→|) and backtab (|←) keys. After you key in the requested data, use the ENTER key to start processing. The program can be terminated at any time by pressing the CLEAR key.

```
        *DLZSAM60*  DLI/CICS/VS ONLINE SAMPLE PROGRAM

   THIS PROGRAM ALLOWS YOU TO ACCESS TWO DL/I DATA BASES:  CUSTOMER
AND INVENTORY.  THE TWO DATA BASES ARE LOGICALLY RELATED AND BOTH CAN
BE ACCESSED VIA SECONDARY INDEXES.

   DLZSAM60 ALLOWS YOU TO DISPLAY AND ADD CUSTOMER NAMES, LOCATIONS
AND ORDERS.  IT ALSO ALLOWS YOU TO DISPLAY AND UPDATE INVENTORY ITEMS
RELATED TO SPECIFIC CUSTOMER ORDERS.

   IN ORDER TO USE THIS SAMPLE PROGRAM, THE JOBSTREAM AS SUPPLIED ON YOUR
DL/I PID DISTRIBUTION TAPE MUST HAVE BEEN RUN SUCCESSFULLY.  IN ADDITION
ENTRIES MUST HAVE BEEN MADE TO THE FOLLOWING TABLES: DFHFCT, DFHPPT,
DFHPCT AND DLZACT.  (SEE DL/I GUIDE FOR NEW USERS)

     PROCESSING OPTIONS:

     1. CUSTOMER INQUIRY        2.  CUSTOMER ORDER ENTRY

PLEASE ENTER DESIRED OPTION -    THE 'CLEAR' KEY ALWAYS ENDS THE PROGRAM
```

This is the first screen that you will see after entering the transaction id, (we used DLZZ) to start the program. To enter the desired option, key a 1, for Customer Inquiry, or a 2, for Customer Order Entry, and press the ENTER key.

```
        *DLZSAM60*  DLI/CICS/VS ONLINE SAMPLE PROGRAM

   YOU HAVE CHOSEN THE CUSTOMER ORDER INQUIRY FEATURE.  PLEASE ENTER
ANY OF THE FOLLOWING INFORMATION KNOWN:

     CUSTOMER NAME: _

     CUSTOMER NUMBER:

     DATE OF ORDER: MONTH     DAY     YEAR

   ENTERING ONLY DATE OF ORDER WILL RESULT IN A DISPLAY OF UP TO 10
POSSIBLE ORDER CANDIDATES FOR SELECTION.  CANDIDATES WILL BE ORDERS
WHOSE ORDER DATE IS EQUAL TO OR LATER THAN THE DATE ENTERED.
```

This screen appears if you choose option 1, Customer Inquiry. You must fill in at least one of the three fields to identify a customer order. The fields are:

1. Customer Name - up to 25 characters
2. Customer Number - 6 digits

3. Date of Order - 2 digits for each (month, day, year)

If an incomplete field is entered the message, "PLEASE FILL IN FIELDS: NAME, NUMBER, OR FULL DATE", appears at the bottom of the screen. An order date that is later than the date of any order that is in the data base will cause the message "NO ORDERS ON OR AFTER THAT DATE. TRY AGAIN PLEASE.", to appear at the bottom of the screen.

```
        *DLZSAM60*  DLI/CICS/VS  ONLINE  SAMPLE PROGRAM
CUSTOMER NAME: COMPANY X INC.          NUMBER: 000001
STREET       : 10 MAIN STREET          CITY  : NEW YORK, NY 10010
CONTACT      : MR. JOHN SMITH
   LOCATION    :                       STREET  :
   CITY        :                       CONTACT :
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        LOCATION NAME                  LOCATION NUMBER
     1.  EASTERN REGION                000010
     2.  WESTERN REGION                000020

SELECT ONE CUSTOMER LOCATION. _
```

This screen appears if a valid customer name or order number was entered on the previous screen. (In this case, Company X was chosen.) To select a customer location, enter a number from the left of the list of locations associated with the customer (1 for EASTERN REGION, or 2 for WESTERN REGION). If an invalid customer location is entered, the message, "ERROR: SELECT ONE CUSTOMER LOCATION." appears on the screen.

```
        *DLZSAM60*  DLI/CICS/VS  ONLINE  SAMPLE PROGRAM
CUSTOMER NAME: COMPANY X INC.          NUMBER: 000001
STREET       : 10 MAIN STREET          CITY  : NEW YORK, NY 10010
CONTACT      : MR. JOHN SMITH
   LOCATION    : EASTERN REGION        STREET  : 69 BROAD ST
   CITY        : PHILADELPHIA, PA 11020  CONTACT : MR. JOHN DOE
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   ORDER DATE  NUMBER  DESCRIPTION            ITEMS      TOTAL COST
1. 01/29/77    100500  FIRST 1977 ORDER        40        $400.00

PLEASE SELECT CUSTOMER ORDER. _
```

This screen appears if you chose customer location 1, EASTERN REGION, on the previous screen. It lists the orders associated with the location chosen. You select an order to display by entering a number from the left of the list of orders associated with that customer location. In this case, there is only one customer order, order

number 100500. Up to five customer orders could be displayed on this screen. If an error is made in selecting the customer order, the message, "ERROR IN REPLY; SELECT ANY VALID ORDER.", appears on the bottom of the screen.

```
         *DLZSAM60*   DLI/CICS/VS  ONLINE  SAMPLE  PROGRAM

CUSTOMER NAME: COMPANY X INC.               NUMBER: 000001

STREET         : 10 MAIN STREET             CITY  : NEW YORK, NY 10010

CONTACT        : MR. JOHN SMITH

  LOCATION     : EASTERN REGION             STREET  : 69 BROAD ST

  CITY         : PHILADELPHIA, PA 11020   CONTACT : MR. JOHN DOE

-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

DATE: 29/01/77 NUMBER   100500    FIRST 1977 ORDER
TOTAL ITEMS IN ORDER:      40        TOTAL COST OF ORDER:      $400.00

   ITEM     DESCRIPTION             ORDER     SHIP     B/O         COST
1. 000100   INTEGRATED CIRCUIT        40       40       0       $400.00


PRESS ENTER KEY TO DISPLAY OPTION MAP.
```

This screen shows all the details about a specific customer, location, and order as requested in the previous sequence of screens. Up to five items will be displayed for each customer order on this screen. Pressing the enter key returns processing back to the option screen.

```
         *DLZSAM60*   DLI/CICS/VS  ONLINE  SAMPLE  PROGRAM
   THE COMPANY NAME OR NUMBER YOU HAVE ENTERED IS NOT LISTED IN THE
CUSTOMER DATA BASE.   BELOW IS A LIST OF THE CURRENT CUSTOMER NAMES
AND NUMBERS:

     CUSTOMER NAME               CUSTOMER NUMBER
1.   COMPANY K INCORPORATED      000006
2.   COMPANY L INC.              000005
3.   COMPANY N INC               000004
4.   COMPANY X INC.              000001
5.   COMPANY Y INC               000002
6.   COMPANY Z INC               000003


PLEASE MAKE CUSTOMER SELECTION _
```

This screen appears if the customer name or number you have entered does not match a valid data base entry. Up to 10 customer names can appear on this screen. Select a valid customer by entering the corresponding number from the left of the list of customer names.

```
     *DLZSAM60*  DLI/CICS/VS ONLINE SAMPLE PROGRAM

THE FOLLOWING IS A LIST OF ORDERS ON OR AFTER THE DATE ENTERED:

  ORDER DATE        ORDER NUMBER     CUSTOMER NUMBER
1. 03/10/77          100600           000002
2. 03/20/77          100722           000006
3. 05/10/77          100610           000004
4. 05/10/77          102050           000001
5. 09/20/77          100700           000003
6. 10/20/77          100705           000005


PLEASE MAKE ORDER SELECTION _
```

This screen appears if you have entered only an order date for your customer inquiry. Select a specific order by entering a number from the left of the list of order data. Up to ten customer orders can be displayed on this screen.

```
     *DLZSAM60*  DLI/CICS/VS ONLINE SAMPLE PROGRAM

 YOU HAVE CHOSEN THE CUSTOMER ORDER ENTRY FEATURE OF DLZSAM60.

1.  YOU CAN ENTER A COMPLETELY NEW CUSTOMER, LOCATION AND ORDER.
    THIS IS OPTION 1.  SIMPLY PRESS ENTER TO SELECT THIS OPTION.

2.  A NEW LOCATION AND ORDER FOR AN EXISTING CUSTOMER.  THIS IS OPTION 2.
    ENTER AN EXISTING CUSTOMER NUMBER HERE:      _

3.  A NEW ORDER FOR AN EXISTING CUSTOMER AND LOCATION.  THIS IS OPTION 3.
    ENTER CUSTOMER NUMBER ABOVE, LOCATION NUMBER HERE:

PLEASE MAKE YOUR SELECTION AND PRESS ENTER.
```

This screen appears if you select order entry mode, option 2, on the option screen. To select option 1, press the enter key. For option 2, enter a 6-digit customer number. Option 3 requires a 6-digit location number. If you use option 3, you must also fill in data for option 2. Press the ENTER key to start processing.

If you enter an incorrect customer or location number, DLZSAM60 will produce an error screen. An example of this error screen is included at the end of this chapter.

```
     *DLZSAM60*  DLI/CICS/VS ONLINE SAMPLE PROGRAM

  YOU MUST FILL IN ALL FIELDS IN ORDER TO UPDATE THE CUSTOMER AND INVENTORY

DATA BASES.  INVALID OR DUPLICATE INPUT WILL CAUSE AN ERROR SCREEN TO APPEAR.

CUSTOMER NAME:_ (up to 25 characters)    CUSTOMER NUMBER : (6 digits)

STREET       :  (up to 25 characters)    CITY  : (up to 25 characters)

CONTACT      :  (up to 25 characters)    LOCATION NUMBER : (6 digits)

   LOCATION      : (up to 25 characters)    STREET  : (up to 25 characters)

   CITY          : (up to 25 characters)    CONTACT : (up to 25 characters)

-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

ORDER DESCRIPTION: (up to 25 characters)    ORDER NUMBER: (6 digits)
ORDER DATE:  MONTH xx DAY xx YEAR xx  (2 digits each)
     ITEM NUMBER:    AMOUNT ORDERED:
1.  (6 digits)      (up to 5 digits)
2.       •               •
3.       •               •
4.       •               •
5.       •               •
```

This screen appears with all fields blank if you have selected order entry option 1. For option 2, the customer data is filled in by DLZSAM60. For option 3, the customer and location data is filled in by the program. You fill in all other fields as needed. You may enter up to five items for the order. If an order is not entered completely, the message, "PLEASE FILL IN ALL NEEDED FIELDS ON MAP.", appears on the bottom of the screen.

```
        *DLZSAM60*  DLI/CICS/VS ONLINE SAMPLE PROGRAM

CUSTOMER NAME: COMPANY X INC.            NUMBER: 000001

STREET          : 10 MAIN STREET         CITY  : NEW YORK, NY 10010

CONTACT         : MR. JOHN SMITH

   LOCATION      : EASTERN REGION          STREET   : 69 BROAD ST

   CITY          : PHILADELPHIA, PA 11020  CONTACT : MR. JOHN DOE


- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

DATE: 29/01/77 NUMBER   100500    FIRST 1977 ORDER
TOTAL ITEMS IN ORDER:     40       TOTAL COST OF ORDER:     $400.00

   ITEM     DESCRIPTION           ORDER    SHIP    B/O        COST
1. 000100   INTEGRATED CIRCUIT      40      40      0       $400.00


PRESS ENTER KEY TO DISPLAY OPTION MAP.
```

To verify the order entry, DLZSAM60 displays the above screen showing your order entry data after a successful data base update.

```
        *DLZSAM60*   DLI/CICS/VS ONLINE SAMPLE PROGRAM
DEBUG MAP:    LINK DISPLACEMENT =    XXXXXX
DL/I CALL:  GU     DL/I STATUS CODE:    GE
PCB ADDRESS:  24B248    PARM COUNT:    5
SSA1 = STSCCST *D(STQCCNO =000007)
SSA2
SSA3
SSA4
 REGISTER 7 =              REGISTER 10 =
 REGISTER 9 =              REGISTER 12 =
***NOTE: ALL SSA'S ARE SHOWN. ALL MAY NOT BE USED IN FAILING CALL.
PRESS ENTER TO RETURN TO INTRODUCTION MAP.
```

This screen appears when there has been an error in requesting DL/I services. Typically, this map appears when an incorrect location or customer number has been used during order entry or when an existing data base entry matches the one entered by the terminal operator. The valuable information on the map consists of the DL/I call parameters, the hex displacement into the DLZSAM60 program where the DL/I call was made, the storage address of the PCB used in the call, and several important general purpose registers used by DLZSAM60. Register 12 contains the address of the program's TCA (task control area), registers 10 and 7 are the program base registers, and register 9 contains the storage address of the output map itself.

DLZSAM60 uses a maximum of 4 SSAs in its calls to DL/I and all are displayed in the error map. The value of PARM COUNT minus 3 is the number of SSAs that were used in the failing call.

# Appendix A: DL/I System Installation and Batch Initialization

The DL/I system is distributed in machine-readable format as macro and Assembler source statements as well as preassembled object modules. Since the entire DL/I system is in object module format, there is no DL/I system generation. However, before application programs may be executed in a DL/I environment, the user must prepare for and install the DL/I system. To assist in this, the following topics are discussed in this chapter:

- Minimum machine requirements
- Deblocking the PID distribution tape
- Building the DL/I system
- Initialization of the DL/I batch system.

## Minimum Machine Requirements

The minimum machine requirements for DL/I is any IBM System supported by DOS/VS or DOS/VSE with a minimum real storage of 256K for a batch system or 512K for an online system in conjunction with CICS/VS. Data base storage files may be on 2314 Direct Access Storage Facilities, 2319 Disk Storage, 3330, 3340, or 3350 Disk Storage, or FBA device. In addition, two 9-track 2400 or 3400 series tape units and control units are required (if tape logging is used) as well as one 2821 Control Unit Model 1 (or equivalent), one 2540 Card Read Punch (or equivalent), and one 1403 Printer Model N1 (or equivalent).

## Building the DL/I System

The DL/I DOS/VS system is distributed as a DOS/VS BACKUP/RESTORE of DL/I private core image, relocatable, and source statement libraries. The distribution tape contains the following files:

File 1 - Null ⎱ Required by MSHP and RESTORE
File 2 - Null ⎰
File 3 - Backup of 3 DL/I Private Libraries containing basic code
- DL1 PRIVATE CIL - Core Image Library
- DL1 PRIVATE RL - Relocatable Library
- DL1 PRIVATE EBOOKS - Edited Macros (E.Books) plus Source for the following: C.DLIUIB, P.DLIUIB, R.DLIUIB, Y.HISTINFO, Y.INST
File 4 - Null
File 5 - A5746XX1. HISTORY. FILE (MSHP History File)
File 6 - Null
File 7 - Null
File 8 - Backup of DL/I Private Source Statement Library
- o DL1 PRIVATE A.OPT - Optional source material - A. Books for module and macros
File 9 - Null
File 10 - A5746OPT.HISTORY.FILE (Dummy MSHP History File for Optional SSL above in File No. 8)
File 11 - Null
File 12 - Online Sample Jobstream
File 13 - Tapemark
File 14 - Tapemark

The DL/I libraries can be restored by using the system RESTORE utility (DOS/VS Release 33 or later). Refer to the Program Directory document included with the distribution for library sizes and JCL for the Restore.

After the DL/I libraries have been successfully restored and DL/I has been tested, the libraries may be merged to permanent private libraries or to the system libraries.

The following modules may be placed in the SVA (shared virtual area) to improve performance. If they are placed in the SVA, they must reside in the system core image library.

| Module Name | Function |
| --- | --- |
| DLZDBH00 | Buffer handler |
| DLZDDLE0 | Load/Insert |
| DLZDHDS0 | Space management |
| DLZDLA00 | Call analyzer |
| DLZDLD00 | Delete/Replace |
| DLZDLR00 | Retrieve |
| DLZDXMT0 | Index maintenance |

### DOS/VS Supervisor Generation

The following parameters in the SUPVR and FOPT macros must be specified when generating the DOS/VS Release 34 Supervisor in order to run DL/I Version 1.4:

SUPVR

    AP=YES
    Required if logging is being performed in an online partition by either DL/I or CICS/VS or if the CICS/VS asynchronous loader option is used.

    PAGEIN=nn
    Required because DL/I and CICS/VS use the RELPAG macro. This is also required if the anticipatory paging feature of CICS/VS is used (ANTICPG parameter in the PCT). Set PAGEIN at least equal to the AMXT value in your CICS/VS system when using the ANTICPG feature.

FOPT

    AB=YES
    Both DL/I and CICS/VS use STXIT AB macros.

    IT=YES
    Required by CICS/VS.

    GETVIS=YES
    Forced because VSAM=YES is required by DL/I.

    OC=YES
    Required if CICS/VS is to support the central processor console as a terminal.

    PC=YES
    Both DL/I and CICS/VS use STXIT PC macros.

PCIL=YES
>    Required to install DL/I from the PID distribution
>    tape.

RELLDR=YES
>    Forced by VSAM=YES and GETVIS=YES.

TOD=YES
>    Required by CICS/VS.

VSAM=YES
>    Required by DL/I.

XECB=YES
>    Required for MPS support. A specification of YES
>    is sufficient for MPS as it requires ((NPARTS-1) * 4)
>    + 2 XECBs (4 per active MPS batch partition, plus
>    two additional for use in the CICS/VS partition).
>    If other programs are also using XECBs, you may
>    need to specify more XECBs than you would get
>    with the YES option. A specification of YES gen-
>    erates space for four XECBs per partition (NPARTS
>    * 4). The addition of the XECB parameter will add
>    about 1K to 2K to the size of your supervisor,
>    depending on the other supervisor options you
>    have specified.

### *DOS/VSE Supervisor Generation*

The following parameters in the SUPVR and FOPT mac-
ros must be specified when generating the DOS/VSE
supervisor in order to run DL/I Version 1.4 ICR or Ver-
sion 1.5:

SUPVR
>  MODE= {370|E}
>  >    Specifies whether the supervisor is generated for
>  >    S/370-mode or ECPS:VSE-mode. The default is E.
>
>  NPARTS= {3|n}
>  >    Default is 3; the minimum value is 2.
>
>  PAGIN=nn
>  >    Required because DL/I and CICS/VS use the
>  >    RELPAG macro. This is also required if the antici-
>  >    patory paging feature of CICS/VS is used
>  >    (ANTICPG parameter in the PCT). Set PAGEIN at
>  >    least equal to the AMXT value in your CICS/VS
>  >    system when using the ANTICPG feature.
>
>  TP= {BTAM|VTAM}
>  >    Required by CICS/VS. The specification of VTAM
>  >    automatically includes BTAM support.

FOPT
>  XECB=nn
>  >    Required for MPS support. Also, 'All-Partition
>  >    MPS Support' requires you to specify the number
>  >    of XECBs to be generated. The following formula
>  >    can be used to calculate the minimum number of
>  >    XECBs to be generated:

2+(2*(N+M))

where N is the number of partitions generated in
the supervisor and M is the maximum number of
concurrent MPS tasks. N and M can vary from 1
to 7.

As soon as MPS is started, 2 XECBs are defined for
each partition in the system plus 2 additional
XECBs. For example, 5 partitions generated and
MPS started would result in 12 XECBs *without* any
MPS processing in progress. Each active batch
MPS task requires 2 additional XECBs during their
execution.

Do not forget to consider the requirements of
other programs using XECBs. For example, the
use of VSE/POWER cross partition communication
macros GETSPOOL, CTLSPOOL, and PUTSPOOL will
require 2 XECBs.

## Relinking DL/I Modules

If any changes are made to DL/I by the user, that is,
DL/I modules are reassembled, the affected core image
modules must be relinked. A book containing the job
control statements to link all of DL/I is included in the
source statement library as A.DLZLNKBK. This may be
punched using SSERV.

Before relinking DL/I, ensure that the DOS/VS se-
quential access method (SAM) modules listed below (or
a superset of these modules) exist in a DOS/VS relocata-
ble library. These modules are required to execute the
DL/I system, including utilities, and, optionally, DL/I
for HSAM data bases. See *DOS/VSE Macro Reference*
for information about the following DOS/VSE SAM
modules:

IJJFCBZD - Device independent (DL/I)
IJFSZZWN - Tape (DL/I)
IJFUBZZN - Tape (DL/I)
IJFFZZZN - Tape (HSAM)
IJGQICZZ - DASD (DL/I)
IJGQOCZZ - DASD (DL/I)
IJGUICZZ - DASD (DL/I)
IJGUOCZZ - DASD (DL/I)
IJGFICZZ - DASD (HSAM)
IJGFOCZZ - DASD (HSAM)
IJFUZZZN - Tape (DL/I)

## CICS/VS-DL/I Release Dependencies

Some of the DL/I action modules reference fields in the
CICS/VS system areas of the TCA, PPT, etc. via DSECT.
The absolute displacement of these fields is not guaran-
teed by CICS/VS from release to release because they
are not intended for use by other than CICS/VS. There-
fore, it may be necessary to reassemble some DL/I
modules after a new release of CICS/VS is installed.
When installing a new release of CICS/VS, check the

documentation for that release to determine if displacement changes were made. The DL/I modules that need to be reassembled and cataloged after a CICS/VS system area DSECT change are:

    DLZOLI00  DL/I Online Initialization
    DLZSTP00  DL/I System Termination
    DLZRDBL1  DL/I Online Journalling
    DLZMPC00  MPS Master Partition Controller
    DLZBPC00  MPS Batch Partition Controller
    DLZMSTR0  MPS Start Transaction
    DLZMSTP0  MPS Stop Transaction
    DLZMPI00  MPS Batch Nucleus
    DLZODP    DL/I Online Interface
    DLZFTDP0  DL/I Formatted Task Dump

All the above modules except DLZODP must be cataloged to the core-image library. DLZODP must be reassembled and cataloged into the relocatable library as it is linkedited with the DL/I ACTs. The JCL statements required to reassemble and catalog these modules are shown in the following four examples:

***Example 1:*** Use these statements for modules DLZOLI00, DLZSTP00, DLZMPC00, DLZBPC00, DLZMSTR0, DLZMSTP0, and DLZFTDP0.

```
// JOB
// OPTION   CATAL
// EXEC     ASSEMBLY
   COPY     xxxxxxxx
   END
/*
// EXEC     LNKEDT
/&
```
where xxxxxxxx is the module name

***Example 2:*** Use these statements for module DLZMPI00.

```
// JOB
// OPTION   CATAL
// EXEC     ASSEMBLY
   COPY     DLZMPI00
   END      DLZMINIT
/*
// EXEC     LNKEDT
/&
```

***Example 3:*** Use these statements for module DLZRDBL1.

```
// JOB
// OPTION   CATAL
// EXEC     ASSEMBLY
   COPY     DLZRDBL1
   END      DLZRDBL1
/*
// EXEC     LNKEDT
/&
```

***Example 4:*** Use these statements for module DLZODP.

```
// JOB
// OPTION   DECK
// EXEC     ASSEMBLY
   COPY     DLZODP
   END
/*
/&
```

Note: The above DOS/VS job control statements may contain additional information. Refer to *DOS/VSE System Control Statements* for more detailed information concerning DOS/VS job control statements.

DLZOLI00 punches a phase card during assembly that changes its name to DFHSIDL when it is linkedited into the core-image library. After the above modules have been reassembled and cataloged to the appropriate libraries, reassemble and linkedit all ACTs.

Note: The CICS/VS modules DFHTBP2$ and DFHDBP2$ that are supplied in the CICS/VS starter system relocatable library contain DL/I DSECTS. If you use these modules with your CICS/VS system, you must ensure that they contain DL/I DSECTS for the same release as your DL/I system. If not, these CICS/VS modules must be reassembled.

## Initialization of the DL/I Batch System

A batch DL/I program is executed as a called program in a DOS/VS partition. The following DL/I utilities are executed as stand alone programs:

    DLZUCUM0 -  Data base change accumulation
    DLZUDMP0 -  Data base data set image copy
    DLZURUL0 -  HISAM reorganization unload
    DLZURRL0 -  HISAM reorganization reload
    DLZURG10 -  Data Base prefix resolution
    DLZLOGP0 -  Log Print
    DLZUACB0 -  Application Control Blocks Creation and Maintenance.

For all other DL/I utilities and for user-written application programs the DL/I initialization module DLZRRC00 is the program name executed by DOS/VS. The actual user-written DL/I program name, or the utility name, and, as required, the name of the PSB or DBD to be used with the program, the size of the data base buffer pool, buffer subpool sizes, subpool assignments, VSAM buffer options, storage layout control options, and whether this is a DL/I batch or utility initialization is passed to DL/I through either a parameter card on SYSIPT or directly from SYSLOG, depending on the UPSI byte setting.

The required control blocks and executable modules are loaded from a DOS/VS core image library (system or private), space for the buffer pool is obtained, and the buffer pool is initialized. The DL/I system log is opened (if applicable), the application program is loaded, and control is passed to the application program.

### *DL/I Parameter Information Requirements*
The following parameter information, beginning in column 1, must be entered from either SYSIPT or SYSLOG:

For data base reorganization or logical relationship resolution utility execution:

```
ULU,progname[,dbdname]
```

**Notes:**

1. For reload restart, ULR replaces ULU in the above example.
2. For HD Reorganization Unload and HD Reorganization Reload, buf, HDBFR=, HSBFR=, and TRACE= may optionally be entered.

For data base data set recovery utility execution:

```
UDR,progname,dbdname[,{buf}]
                      {1  }
    [,HDBFR= ][,HSBFR= ][,TRACE= ]
```

For data base backout utility execution:

```
DLI,DLZBACK0,psbname[,{buf}]
                      {1  }
    [,HDBFR= ][,HSBFR= ][,TRACE= ]
    [,ASLOG= ][,LOG=    ]
```

## *DL/I Initialization Job Control Language Requirements*

The job control statements required for utility program execution are shown for each utility in the section dealing with the particular utility. The job control statements required for batch application program execution are shown below.

| // UPSI | x0000xxx | The x values identify the desired DL/I function. |
|---|---|---|
| [// ASSGN<br>[// TLBL | SYS011,X'cuu']<br>LOGOUT    ] | These statements define the output log file. It must be tape and contain standard labels. LOGOUT is the symbolic name of the output file as specified in its DTF. If no output log file is desired, bit 6 of UPSI must be set to 1. These cards may then be omitted. |
| // ASSGN<br>// TLBL (Tape)<br>or<br>// DLBL (Disk)<br>// EXTENT | SYSnnn,X'cuu'<br>filename<br><br>filename<br>extent data | These statements define a data base file. Statements must be present for each file referenced by every data base referenced by the PSB. The parameters are explained as follows:<br>nnn - logical unit assignment of the file.<br>cuu - physical unit assignment of the file.<br>filename - symbolic name of the file (VSAM ACB name or SAM DTF name) to be processed. It must be the same as the DD1, DD2, or OVFLW parameter of the DATASET statement for the data base. |
| // EXEC | DLZRRC00,SIZE=xxxK | DL/I initialization module. Refer to *DL/I DOS/VS System/Application Design Guide* Chapter 7, for storage requirements. |
| **Note:** The above DOS/VS job control statements may contain additional information. Refer to the publication *DOS/VSE System Control Statements* for more detailed information concerning DOS/VS job control statements. | | |

# DL/I MPS Batch Partition Initialization

Starting an MPS batch partition requires an online DL/I partition being active and the master partition controller being initialized.

All data bases referenced by a batch program executing under MPS control must be defined in the CICS/VS partition and accessed through MPS. A program running under MPS control cannot access any data bases not known to MPS, that is, not defined in the CICS/VS partition.

Certain DL/I programs are restricted from running in the MPS environment, i.e. the data bases they access may not be shared across several partitions while these programs are executing. The programs in this catagory are:

- Utilities
- Programs loading a data base
- Programs using SHSAM or HSAM

In addition, any batch DL/I programs that modify the contents of the DL/I control blocks cannot run under MPS because the DL/I control blocks no longer exist in the batch partition.

## *DOS/VS UPSI Byte Settings for MPS*

| | | |
|---|---|---|
| Bit 0 | = 0 | Read parameter information via SYSIPT. |
| | = 1 | Read parameter information via SYSLOG. |
| Bits 1 - 4 | | Available for use by the application program. |
| Bit 5 | = 0 | Storage dump on set exit (STXIT) abnormal termination. |
| | = 1 | No storage dump on STXIT abnormal termination. |

Bits 6 - 7    Not used for MPS. Data base logging, normally controlled by UPSI bit 6, is controlled in the CICS/VS partition under MPS operation. STXIT linkage to DL/I for abnormal task termination, normally controlled by UPSI bit 7, is always active under MPS operation.

## DL/I MPS Parameter Information Requirements

The following information, beginning in column 1, must be entered from either SYSIPT or SYSLOG:

```
DLI,progname,psbname
```

progname
    specifies a one to eight alphameric character name of the application program to be executed.

psbname
    specifies a one to seven alphameric character name of the PSB as indicated in the PSB generation and referenced by the application program.

Any other parameters on the DL/I parameter statement are ignored. The parameter statement information is printed on SYSLST.

## DL/I MPS Initialization Job Control Language Requirements

The job control statements required for MPS batch application program execution are shown below.

| // UPSI | x0000x00 | The x values identify the desired DL/I function. |
|---------|----------|--------------------------------------------------|
| // EXEC | DLZMPI00, SIZE=xxxK | DL/I MPS initialization module. Refer to *DL/I DOS/VS System/Application Design Guide*, Chapter 7, for storage requirements. |

No ASSGN, DLBL, EXTENT, or TLBL statements are required to describe the data bases or DL/I log in the batch MPS job stream. This information is contained in the JCL for the CICS/VS partition.

**Note:** The above DOS/VS job control statememts may contain additional information. Refer to the publication *DOS/VSE System Control Statements* for more detailed information concerning DOS/VS control statements.

## Executing Batch MPS Programs

CICS/VS Release 1.4 or a subsequent release is required for DL/I Release 1.5. The parameter DLI=YES must be specified in the DFHSG TYPE=INITIAL macro to generate support for DL/I in a CICS/VS system. This is independent of whether MPS or PI (program isolation) will be used. There are no parameters for CICS/VS system generation specifically for DL/I MPS. DL/I PI requires that the CICS/VS system be generated to include support for the CICS/VS dynamic transaction backout facility.

For ease in maintenance of your CICS/VS system, you should try to use the preassembled CICS/VS programs provided in the CICS/VS starter system private core image library. DL/I support has been generated into the appropriate starter system CICS/VS programs with the exception of the CICS/VS dynamic backout program (DFHDBPI$) and the transaction backout program (DFHTBPI$) versions in the starter system core image library. To produce a version of each that supports DL/I, you must first merge the DL/I Release 1.5 private relocatable library into either the system relocatable library or the CICS/VS private relocatable library. Then you can linkedit the relocatable modules of the dynamic backout program and transaction backout program provided in the CICS/VS relocatable library (DFHDBP2$ and DFHTBP2$ respectively). Refer to the *CICS/VS System Programmer's Guide (DOS/VS)*, Appendix A, for more information on this subject.

The phase name of the // EXEC statement must be DLZMPI00. A SIZE parameter is not required unless your application program invokes some DOS/VS function that requires a partition GETVIS area. However, the use of the SIZE parameter is recommended as an operational standard. DL/I MPS does not require a partition GETVIS area in the batch MPS partition.

## Dynamically Scheduling MPS or Non-MPS Execution

The requirement that a different phase name be used on the EXEC statement to distinguish between MPS and non-MPS operation can cause operational difficulties. The operations staff must be aware of when CICS/VS is active and MPS is in operation and modify the JCL of batch DL/I jobs to specify the appropriate phase name on the EXEC statement.

An alternate approach would be to use a program to dynamically test to see if MPS was in operation or not and fetch the corresponding phase. The program can use the XECBTAB TYPE=CHECK macro to test for the presence of the XECB "DLZXCB00". If this XECB is currently defined (R15=0) then MPS operation is active and the program can fetch the phase "DLZMPI00". If MPS operation is not active (R15≠0) then the program should fetch the phase "DLZRRC00". Note that this technique will not work if logging is required as there is no way to dynamically assign the DL/I log device if non-MPS operation is required.

The JCL to execute a batch DL/I program in this environment would use an EXEC statement that specifies the XECB testing program's name, not DLZMPI00 or DLZRRC00. No other changes in JCL would be required.

An example program to do this for read-only (no logging) DL/I programs is given below.

```
DLZCTRL    CSECT
           BALR     R12,0             ESTABLISH BASE REG (R12)
           USING    *,R12             ...AND TELL ASSEMBLER
           OPEN     PRINTER,CONSOLE
           XECBTAB  TYPE=CHECK,XECB=DLZXCB00
           LTR      R15,R15           IS MPS ACTIVE?
           BNZ      NOMPS             ...NO
           LA       R2,=C'DLZMPI00'   YES, USE DLZMPI00
           MVC      IOAREA(L'MPSMSG),MPSMSG
           LA       R7,L'MPSMSG       SET UP MPS ACTIVE MSG
           B        OUTPUT            GO WRITE MSG & FETCH DL/I
NOMPS      EQU      *
           LA       R2,=C'DLZRRC00'   NOT ACTIVE,USE DLZRRC00
           MVC      IOAREA(L'NOMPSMSG),NOMPSMSG
           LA       R7,L'NOMPSMSG SET UP MPS NOT ACTIVE MSG
OUTPUT     EQU      *
           PUT      PRINTER           INDICATE CHOICE TAKEN
           PUT      CONSOLE           ...ON SYSLST AND SYSLOG
           CLOSE    PRINTER,CONSOLE
           FETCH    (R2)              FETCH APPROPRIATE PHASE
MPSMSG     DC       C'MPS ACTIVE - WILL EXECUTE DLZMPI00'
NOMPSMSG   DC       C'MPS NOT ACTIVE - WILL EXECUTE DLZRRC00'
PRINTER    DTFDI    DEVADDR=SYSLST,IOAREA1=IOAREAC,RECSIZE=81
IOAREAC    DS       0CL81
           DC       X'F1'
IOAREA     DC       CL80' '
CONSOLE    DTFCN    DEVADDR=SYSLOG,IOAREA1=IOAREA,BLKSIZE=80, X
                    RECSIZE=(7),RECFORM=UNDEF,MODNAME=DLZCONSL
           END
```

Note that since the data base I/O operations are being carried out in the CICS/VS partition for the batch MPS job, the DOS/VS job accounting information for the batch partition will not reflect any data base I/Os or associated CPU time for the data base call processing.

If the Performance Analyzer II FDP (5798-CFP) or similar program is being used to collect job accounting information for the CICS/VS partition, the data base I/Os and associated CPU time for the data base call processing will be charged to the DL/I batch partition controller task, CSDC.

# Appendix B: Controlling the DL/I Online System Environment

The online system environment can be controlled using system generation parameters to establish the initial configuration of the online system. Dynamic services (CICS master terminal function) can be used for adjusting the system configuration.

The DL/I online system can also be controlled in various ways. By specifying a unique DL/I online nucleus during CICS 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 Appendix D of *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.

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 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 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 nonasynchronous 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 *DL/I DOS/VS Diagnostic Guide,* SH24-5002, 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 due to an I/O error. The user may then perform off-line recovery procedure and then make the data base available again for online processing. Great care should be exercised when invoking the open/close functions due to the nonasynchronous 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 affect 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 Assembler language programs and have a special call format. The general structure of the calls 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 inter-face work area.

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 which consists 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 which consists 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

In order 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          INSURE NORMAL COMPLETION
           BNE      CMXTERR            IF NOT CHECK RETURN CODE
             •
             •
             •
SCHEDERR   DS       OH                 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'DMB1əəəD' 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          INSURE 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
           •
           •
           •
```

A number of terms and phrases used in the detailed explanation and description of DL/I are either new to most readers, or have new meanings. To improve the readability and understandability of this and other DL/I DOS/VS publications, the significant and important terms and phrases are defined here in alphabetic order for ease of reference. These definitions refer to Figure 9-1 which is a representative DL/I hierarchical structure. It is suggested that the reader become familiar with the terms that are defined here and refer to this section frequently during the initial use of DL/I.
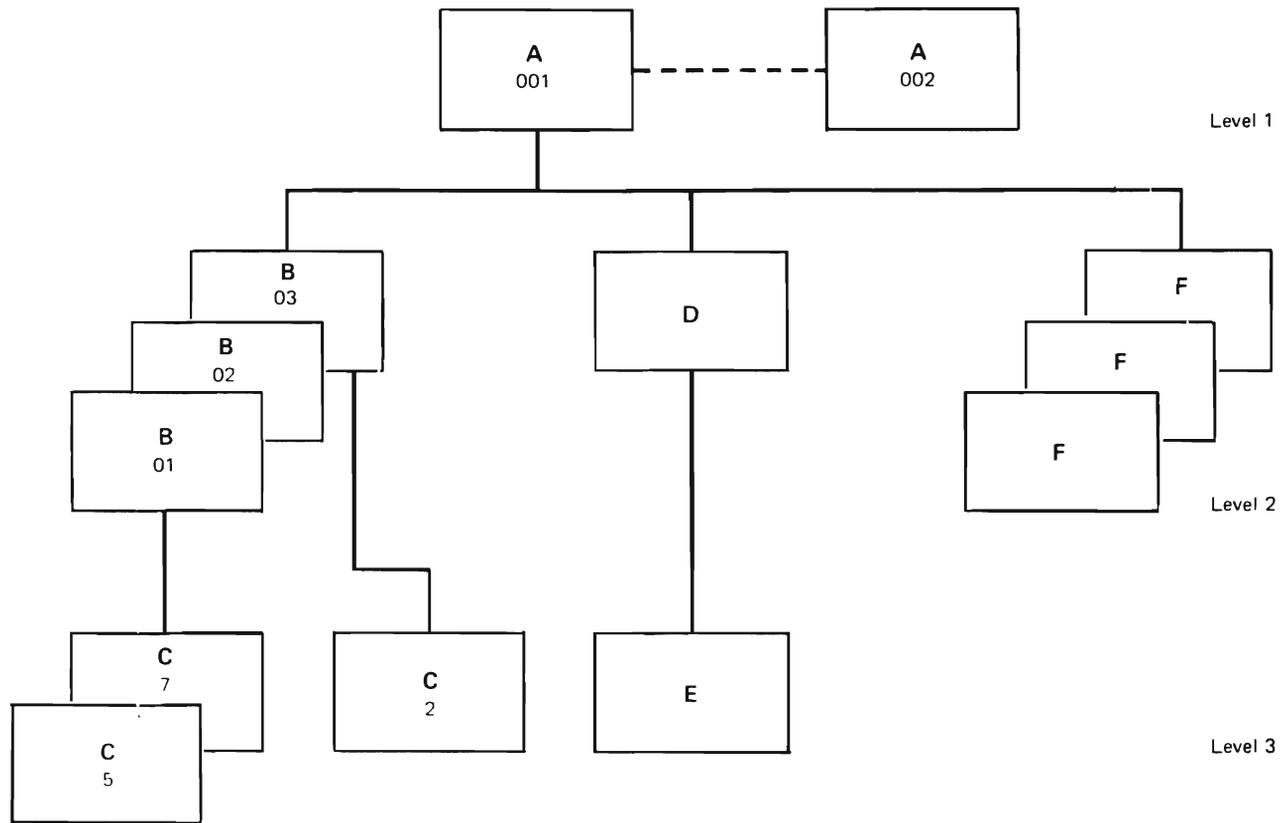


- Each block represents a segment
- The segment names are A through F.
- The numbers represent the sequence fields (keys)  If no number is present, the segment has no key.
- The lines connecting the segment blocks show the hierarchical paths.

Figure 9-1.  Representative DL/I Hierarchical Structure

**call.** The instruction in the COBOL, PL/1, or Assembler program that requests DL/1 services. For RPG II, see RQDLI command.

**child.** 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. Referring to Figure 9-1:

- 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.

**command code.** An optional addition to the SSA that provides specification of a function variation applicable to the call function.

**data base.** An organized grouping of records, where each record contains an organized grouping of segments, and each segment contains an organized grouping of data or information. A data base can usually be thought of as a file.

**data base administrator.** See *DBA*.

**data base definition.** See *DBD*.

**data management block.** See *DMB*.

**DBA.** data base administrator - The person in an installation who has the responsibility (full or part time) for technically supporting the use of DL/1. The normal responsibilities of the DBA are outlined in Chapter 1 of this manual.

**DBD.** data base definition - One DBD is generated by the DBA and cataloged in a core image library for each data base that is to be used in the installation. It defines the structure, segment keys, physical organization, names, access method, devices, etc. of the data base.

**DMB.** data management block - The DMB is created from a DBD by the application control blocks creation and maintenance utility and link-edited into a core image library by the DBA. Before an application program using DL/1 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 (by the DBA). The DMBs and the associated PSB are automatically loaded into main storage from the core image library at the beginning of application program execution (their loading is controlled by the parameter information supplied to DL/1 at the beginning of program execution).

**dependent segment.** Synonym for *child*.

**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 from top to bottom and from left to right. Referring to Figure 9-1, forward accessing of all the 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.

**key.** 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/1 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 exceptions being if there will never be multiples of a particular type or if a particular segment type will never have dependents.

**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 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 9-1:

- 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).

**logical.** When used in reference to DL/1 components, logical means that the component is treated according to the rules of DL/1 rather than physically as it may exist, or as it may be organized, on a physical storage device. For example, a logical DL/1 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 a different physical sequence than that by which they are retrieved logically for the application program by DL/1.

**MPS.** multiple partition support provides a centralized data base facility to permit multiple applications to access DL/1 data bases concurrently.

**multiple partition support.** See *MPS*.

**multiple SSA.** A series of SSAs included in a DL/1 call to identify a specific segment or path.

**parent.** 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 9-1:

- 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.

**parentage.** Establishment in a program of a particular parent as the formal beginning point for the use of the GNP or GHNP call functions. Parentage can only be established by issuing successful GU, GHU, GN, or GHN calls.

**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 four 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 9-1:

- 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 since it is a root segment.

**path call.** The retrieval or insertion of multiple segments in a hierarchical path in a single call, by using the D command code and multiple SSAs.

**PCB.** program communication block. Every data base accessed in an application program has a PCB associated with it. The PCB actually exists in DL/1 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/1 - The PCB names are defined under a pointer variable.
- RPG II - 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/1 results status code
4. DL/1 processing options
5. Reserved for DL/1
6. Segment name feedback area
7. Length of key feedback area
8. Number of sensitive segments
9. Key feedback area.

These fields are described in detail under "PCB-Mask" in Chapter 4 of this manual.

**PI.** program isolation is 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/1. 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 9-1:

- 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. Refer to "Multiple Positioning with DL/1 Calls" in Chapter 4.

**program communication block.** See *PCB*.

**program isolation.** See *PI*.

**program specification block.** See *PSB*.

**PSB.** program specification block. The DBA generates a PSB for each application program that uses DL/1 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. (See *DMB*.)

**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. The physical makeup of an SSA is fully described in "Segment Search Arguments" under "Calls to DL/1" in Chapter 4.

**record.** A data base record is made up of a unique root segment and all of its dependent segments. Referring to Figure 9-1: A-001, B-01, C-5, C-7, B-02, B-03, C-2, D, E, F, F, F constitute a data base record.

**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 RPG II program used to request DL/1 services.

**secondary indexing.** 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.

**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.** The segment name is assigned to each segment type by the DBA. Segment names for the different segment types must be unique within a data base. The segment name is used by the application programmer when constructing a qualified or unqualified SSA prior to issuing a call for a specific segment.

**segment search argument.** See *SSA*.

**segment type.** Same meaning as segment name. 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 the DBA). Referring to Figure 9-1, there are six different types of segments; A through F.

**sensitivity.** The DBA controls sensitivity to the various segments that constitute a data base on a program-by-program basis (when the PSB for each program is generated). 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. 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.** See *Key*.

**sequential processing.** Processing or searching through the segments in a data base in a forward direction (see *Forward*).

**SSA.** segment search argument. Describes the segment type, or specific segment within a segment type, that is to be operated on by a DL/1 call. The physical makeup of an SSA is fully described in "Segment Search Arguments" under "Calls to DL/1" in Chapter 4.

**status code.** Each DL/1 call returns a status code that reflects the exact result of the operation. The status codes are returned in field 3 of the PCB. The first operation that a program should perform immediately following a DL/1 call is to test the status code in the PCB to insure that the function called for was successful. The normally expected status codes associated with each call function are in Figure 4-3.

**synchronization point.** A logical point during the excution of an 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. Also referred to as sync point or synch point.

A synch point is created by a DL/1 CHKP call, a DL/1 TERM call, a CICS/VS synch point request, or end of task or program.

**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. The physical makeup of an SSA is fully described in "Segment Search Arguments" under "Calls to DL/1" in Chapter 4.

# Index

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

|  | Yes | No |
|---|---|---|
| • Does the publication meet your needs? | ☐ | ☐ |

• Did you find the material:

| | Yes | No |
|---|---|---|
| Easy to read and understand? | ☐ | ☐ |
| Organized for convenient use? | ☐ | ☐ |
| Complete? | ☐ | ☐ |
| Well illustrated? | ☐ | ☐ |
| Written for your technical level? | ☐ | ☐ |

• What is your occupation? _____

• How do you use this publication:

| | | | |
|---|---|---|---|
| As an introduction to the subject? | ☐ | As an instructor in class? | ☐ |
| For advanced knowledge of the subject? | ☐ | As a student in class? | ☐ |
| To learn about operating procedures? | ☐ | As a reference manual? | ☐ |

**Your comments:**

*If you would like a reply, please supply your name and address on the reverse side of this form.*

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)
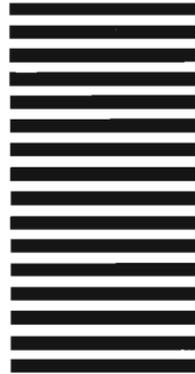
SH24-5001-2

**Reader's Comment Form**

Fold                                                        Fold

If you would like a reply, *please print:*

*Your Name* _____

*Company Name* _____ *Department* _____

*Street Address* _____

*City* _____

*State* _____ *Zip Code* _____

*IBM Branch Office serving you* _____

**IBM** ®

Cut or Fold Along Line

DL/I DOS/VS Guide For New Users    Printed in U.S.A.    SH24-5001-2

DL/I DOS/VS Guide For New Users   Printed in U.S.A.   SH24-5001-2